IoTignite

Nesnelerin İnterneti Platformu

IoT-Ignite ile Uçtan Uca Programlama

Industrial IoT & Edge Computing

ARDIC powered by

v 1.0

IoT-Ignite ile
Uçtan Uca Programlama

(version: 1.0)

YAZARLAR
Mehmet Ali SICAK
Uğur GELİŞKEN

İÇERİK GELİŞTİRME ve REDAKSİYON
Hakkı Yavuz ERZURUMLU
Haluk TÜFEKÇİ
Mert ACEL
Perihan MİRKELAM

KATKIDA BULUNANLAR
Afşin ÇELİK
Barış İNANÇ
Ceyhun ERTÜRK
Hüseyin BAŞHAN
Oğuz ÇAKIR
Soner UĞRAŞKAN

ve DİĞER EMEĞİ GEÇENLER
Adil KARAÖZ
Murat BİRBEN
Umut KOCASARAÇ
Şamil BEDEN


IOTignite

SUNUŞ

Her ülke içinde bulunduğu dönemin ihtiyaçlarını karşılamak için teknolojiye ve teknolojik ürünlere ihtiyaç duyar. Teknoloji üretip hem kendisi için kullanan hem de diğer ülkelere satan ülkeler olduğu gibi, kendileri üretmedikleri için teknolojik ürünleri başka ülkelerden almak zorunda olan ülkeler de var. Zira ihtiyaçlar beklemez ve mutlaka karşılanmalıdır.

ARDIÇ şirketi üç temel hedef üzerinde bundan on yıl önce hayatına başladı. Her şeyden önce "Eleştirmeden önce üret, - bu ülke için sen ne yaptın?" sorusuna cevap verebilmek için…

İkincisi ise, ülkemizin teknoloji alanında olan bir ihtiyacını karşılamak amacı ile yerli bir alternatifi en az yurt dışı eş değerleri kadar iyisini, hatta daha da iyisini masa üzerine koyabilmek için…

Son olarak da, geliştirdiğimiz söz konusu yüksek teknoloji tabanlı bu ürünleri yurt dışına satabilmek için…

Elbette bu üçüncü hedefin gerçekleşebilmesi, ikinci hedefteki masadan pozitif sonuçlar alınması ön koşulunu gerektiriyor.

Bu kitabın konusu olan IoT-Ignite platformu ile ilk iki temel kuruluş hedefimizi yerine getirdiğimizi gönül rahatlığı ile söyleyebiliyoruz. Üçüncü hedefimizi gerçekleştirebilmek için de var gücümüz ile çalışmaya devam ediyoruz.

Görevini yapmanın iç huzuru ile mutluyum. On yılda, günde yirmi dört buçuk saat çalışma sonucunda, **IoT-Ignite**'ın ortaya çıkmasında ve bu kitabın yazılmasında emeği geçen başta Haluk Tüfekçi, Hüseyin Başhan, Levent Babacan, Barış İnanç, Elif Çakmak ve tüm **Türk mühendis ve ekip arkadaşlarıma**, bu vizyonu paylaşan ve sahip çıkan, başta Hülya Kahveci ve Mehmet Aksayan olmak üzere tüm şirket kurucu ortaklarım ve yatırımcılarıma, kısacası ***yuvarlanan yürekleri yokuş yukarı durmayanlara*** teşekkür ederim.

Daha nice IoT-Ignite'lara…

Tunç Kahveci
Yönetim Kurulu Başkanı - Kurucu

ÖNSÖZ

IoT-Ignite platformu 10 yılı aşkın emek ve birikimin sonucu olarak gerçekleşmiş bir projedir.

IoT (Internet of Things - Nesnelerin İnterneti) yaygın olarak "*bir sensörden gelen verinin internet üzerinden buluta gönderilmesi ve burada gösterilmesi*" olarak algılanmaktadır ki bu tanım aslında IoT'nin en basit tanımı, alfabesinin ilk harfidir. Aslında, IoT'nin günümüzde teknolojinin dokunduğu her alanı adreslediğini söyleyebiliriz. Örneğin hepimizin kullandığı akıllı telefonlar aslında gelişkin IoT cihazlarıdır.

IoT'ye giriş olarak tanımlayabileceğimiz ilk kuşak çözümler, hem ülkemizde hem de tüm dünyada en çok referans alan tanım oldu. Maker (hobi geliştiriciler) dünyasının popülaritesinin artmasıyla birlikte Arduino benzeri geliştirme kartı edinen ve bununla bir-iki sensörden gelen bilgiyi okuyan, sonrasında bu veriyi İnternet ortamında bir sunucuya taşıyıp web sayfasında renkli grafiklerle gösterebilen çalışmalarla, ilk IoT platformu ve çözümleri piyasadaki yerlerini almış oldu.

Bir çözüm, o ana kadar çalışılmamış bir alanda bilgi edinilmesini sağlıyorsa ticari bir değer de taşıyabilir. Örneğin, bir fıçıdan akan içeceğin debisinin okunması ve o içecekten ne kadar satıldığının ölçülmesi gibi bir çalışma, içecek tedarikçisine stok tükenmeden sipariş talimatını otomatik veriyorsa "çözüm" olarak adlandırılabilir; çünkü işlevseldir ve ticari bir değer taşıyabilir. Küresel pazarda buna benzer şekilde çalışan 1000'e yakın IoT çözümü (platformu) olduğu bildiriliyor.

Bu tip çözümler ticari bir değer taşımakta ve basit çözümlerle daha fazla müşteriye ulaşmak için çözümlerini "platform" olarak pozisyonlamaktalar. Ancak bu tanım, platform olmaktan çok verilen çözümü adreslemektedir. Genel amaçlı platform ise, tanımını aşağıda verdiğimiz gibi daha karmaşık ve gelişkin bir mimari ve teknolojiyi gerektirmektedir.

Bu kitabın konusu olan IoT'nin, çalışma altyapı platformu olan "bulut"dan kısaca bahsederek devam edersek, kendisini organize edebilen ana katmanları ve fonksiyonel yapı taşları olan ve ölçeklenebilir olarak çalışabilen bir yapıdır. Bulut teknolojisi alfabesinin A'sı olan bu tanımdan sonra bu yapı çok derin ve karmaşık bir yöne doğru evrilir. Bulut üzerinde çalışması istenen sistem, çözüm ve uygulamaların; gerçek anlamda "bulut"un katma değerlerini kullanabilmesi için "bulut" üzerinde "cloud native"olarak kodlanması gerekmektedir. İstemci-sunucu (client – server) mimarisinde tasarlanmış klasik çözümün, fiziksel sunucu yerine herhangi bir bulut servisi sağlayıcıdan kiralanan sanal bir makina üzerinde çalıştırılması, arzu edilen verimi ve ölçeklenebilirliği sağlayamamaktadır.

IoT platformu nedir sorusuna bakmadan önce platform nedir sorusuna cevap bulmaya çalışalım. Platform, teknik olarak içinde birden çok ürünü oluşturmaya yarayacak parçaları bulundurur ve herhangi bir dikey çözüm yerine çözümler kümesinin ortak paydasını adresleyebilen bir yapıya sahiptir. Böylelikle, platform kullanarak özellikli bir çözümü oluşturmak, tek başına bir çözümü oluşturmaya nazaran çok daha kolay olur. Platform mimari felsefesi birçok çözüme hizmet verecek şekilde düşünülerek tasarlanmış olmalıdır.

Söylenene uygun olarak IoT platformu, nesnelerin internetinin ihtiyaç duyacağı temel yapı taşlarını barındırmak zorundadır. Bu temel yapı taşları şöyle sıralanabilir: iletişim katmanı, çekirdek IoT fonksiyonları, analiz ve uygulama katmanı, dış dünya ile iletişimi sağlayacak zengin arayüz katmanı.

İletişim katmanı IoT platformunda nesneler ile platform arasındaki iletişiminin sağlandığı ve yönetildiği katmandır. Bu katmanda, değişik iletişim protokollerinin (TCP, HTTP, MQTT, XMPP, vb.) desteklenmesi gerekir. Bunun yanında daha karmaşık konularda da destek verebilmesi gerekir;

örneğin, nesnelerin bir güvenlik duvarı (firewall) ve NAT (network address translation) arkasında olması durumunu yönetebilmesi, nesne çevrimdışıyken (offline) gönderilmiş bir komutun saklanıp doğru bir kuyruklama mekanizmasıyla nesne çevrimiçi (online) olduğunda gönderilmesi gibi senaryoları karşılayabilmesi de beklenir. Nesnelerin sayısı on binlere, yüz binlere ulaştığında, iletişim katmanı bu yüksek çift yönlü trafiği performans kaybı yaşamaksızın yönetebilmelidir.

Çekirdek IoT fonksiyonları başlığı altında şu temel yapıları sıralayabiliriz; nesnelerin uzaktan ayarlanabilmesi, nesnelerin her birinin benzersiz bir tanım ile ayrıştırılabildiği altyapısının olması, bu nesnelerden gelen bilginin kayıpsız saklanması ve veri saklama yapısının elastik ve ölçeklenebilir bir yapıda olması, nesneler ile bulut arasında güvenli iletişimin sağlanması, nesnelerin üzerinde koşmakta olan yazılımların gerektiğinde uzaktan ve güvenilir bir şekilde güncellenebilmesi ve elbette binlerce, yüz binlerce nesnenin sisteme otomatik olarak güvenli bir şekilde dahil olabilmesi… Tüm bu özellikleri sağlayacak mekanizmaların aynı zamanda istenen çoklukta nesnenin dahil edilebileceği, ölçeklenebilir bir yapı içinde geliştirilmiş olması vazgeçilmezdir.

Analiz ve uygulama katmanını belki de bir IoT platformunun sağladığı fayda modelinin en öne çıktığı katman olarak düşünebiliriz. Burası toplanan bilginin işlendiği ve sonuç veya rapor ürettiği altyapı katmanıdır. Veri analiz edilir, görselleştirilir, karmaşık olay işlemeden, yapayzeka katmanından geçer. Veri bu fonksiyonlardan sonra sistem üzerinde bir eylemi tetikleyebilir. Burada karmaşıklığı fazla fonksiyonları işaret ediyoruz; yüz binlerce nesneden gelen bilginin gerçek zamanlı veya gerçek zamana yakın şekilde işlenmesi ve karar verilebilmesi gerçek anlamda ölçeklenen bir yapıyı gerektirir.

Bunların yanı sıra, edge-computing (kenarda bilgi işleme), nesnelerin internetinin yükselen kavramı olarak ortaya çıkıyor. Nesneler topladıkları veriyi platform bulutuna göndermeden önce kendilerine fiziksel olarak yakın bir yere (örneğin bir gateway'e veya gateway görevi üstlenmiş herhangi bir cihaza) gönderir. Bulut üzerinde gerçekleşecek veri analizi ve hatta karmaşık olay işleme gibi bazı işlemler bu katmanda yapılabilir. Böylelikle anlık veri, büyük veri denizine eklenmeden önce sonucunu üretip görevini tamamlayabilir. Hatta nesne ile platform arasında internet bağlantısı olmadığı durumlarda bile veri kayıt altına alınabilir, edge-computing ile veri eş zamanlı olarak işlenebilir ve önceden tanımlanmış karar mekanizmaları yerel olarak işletilebilir. Yapay zeka teknolojisinin bu katmanda yapılabiliyor olması, örneğin eğitilmiş veri setinin bu katmanda kullanılabilmesi ve gelen veri üzerinden sonuç (inference) çıkarılması gibi fonksiyonların da bu katmana taşınması giderek gereklilik halini alıyor.

Elbette edge-computing, IoT platform mimarisinin çok daha yetkin özelliklere sahip olmasını da gerektirmektedir. Edge compute dediğimizde nesneye yakın, işlem yapabilen bilgisayar katmanından söz ediyoruz; bunu aynı zamanda dağıtık bir bilgisayar mimarisi olarak da tanımlamak mümkün. Üzerinde değişik algoritma ve servislerin koştuğu sahaya yayılmış binlerce cihazın yönetilebilmesi, üzerinde çalışan servislerin güncel ve doğru çalışmasının sağlanması da gerekmektedir.

Bunun yanında, edge-compute katmanında nesnelerin doğrudan bu katmana bağlanabilmesi için, gerekli tüm iletişim protokollerini desteklenmesi (örneğin nesnelerin IoT platformuna olan bağlantısında olduğu gibi MQTT, TCP vb.), bu katmanın kendisinin otomatik nesne keşfi, güvenli kayıt ve iletişim yapabilecek alt yapıya sahip olması gerekir. Aynı zamanda bu bilgilerin IoT bulutunda yansımasının oluşturulması gibi arka planda çok önemli ve karmaşık işlevlerin de sağlanması gerekir.

Dolayısıyla edge-compute destekleyen IoT platformunda yukarıda söz ettiğimiz, IoT işlevlerine ek olarak, bu katmanı da hassasiyet ile yönetebilecek gelişmiş başka bir katmana daha ihtiyaç vardır; edge-computing platform ve servis yönetim katmanı. Bu katman belki de sahada mini sunucu da diyebileceğimiz binlerce, on binlerce bilgisayarı, bunların üzerinde koşan servisleri, bu mini

sunuculara bağlanmış sensörleri, erişim düzenekleri (actuator) de bu katman üzerinden sorunsuz olarak yönetebiliyor olmalıdır.

Arayüz katmanı ise geliştirici ve son kullanıcıların görsel olarak platform ile iletişiminin sağlandığı (HMI-Human Machine Interface) çoğunlukla web tabanlı arayüzlerden oluşur.

Bunun yanında başka sistemler veya platformlar ile IoT platformunun iletişim kurmasını sağlayan iletişim metotlarının da olması gerekir. Günümüzde bu tip iletişimi sağlayan ana iletişim metotları arasında REST-API'leri anabiliriz. Bu API'ler ile dış dünyadaki sistemin IoT platformundan istekte bulunarak sonuçlarını platformdan alabilir. Benzer şekilde IoT platformu da dış dünyadaki sistemlere çağrıda bulunabilir; örneğin sistemde bir olay gerçekleştiğinde IoT platformu SMS (short message service) gönderebilen bir sistem ile iletişime geçerek SMS gönderebilir. Buna ek olarak, dış dünyaya olay (event) gerçekleştiği zaman bunu anlık olarak aktarabilecek arayüz tiplerinin de olması gereklidir; günümüzde en çok kullanılan mekanizma, pub/sub (publisher/subscriber) iletişim modelidir. Bu model üzerinden IoT platformu dış sistemlere onların önceden belirttikleri olay tiplerini anlık olarak bildirilebilir.

Buraya kadar anlatılan işlevlere sahip bir IoT platformunu "full-stack IoT platform" (uçtan uca tam katmanlı nesnelerin interneti platformu) olarak tanımlıyoruz. En başta belirttiğimiz, globalde anılan 1000'e yakın IoT platformunu bu tanımla değerlendirdiğimizde ise sayının iki basamaklı bir sayıya zor ulaştığını görüyoruz.

Endüstri 4.0, günümüzde en çok konuşulan konuların başında geliyor. Konunun ticari ve sosyo-ekonomik boyutlarını bir kenara bırakıp teknik gereksinimler boyutuna bakacak olursak, Endüstri 4.0 için temel IoT platform mimarisinin edge-computing katmanlı full-stack IoT platform mimarisi olduğunu mutlaka belirtmemiz gerekir.

IoT platformunda olması gereken işlevlerin hepsinin doğal olarak ölçeklenebilirliği ise vazgeçilmez bir ön koşuldur. Bu nedenle bulut alt yapısı üzerinde koşan IoT platformunu oluşturan servislerin de "cloud-native" bir şekilde yazılmış olması gerekir.

Bu kitabın içeriğinde kullanılan ve servisleri tanıtılan IoT-Ignite platformu yukarıda tanımlanan "full-stack IoT platform" özelliklerine sahip bir IoT platformudur ve sözü geçen tüm katman ve işlevlere sahip bir yapıda tasarlanmıştır. Üzerinde birden fazla servis multi-tenant (çoklu kiracı) yöntemi ile çalışıyor ve yüz binlerce uç noktaya hizmet veriyor.

IoT-Ignite platformu edge-compute katmanında (mini sunucular) gömülü Linux işletim sistemini desteklemekle birlikte Android işletim sistemini de geniş çapta destekliyor. IoT-Ignite Agent adı verilen Android uygulaması Google Play'den indirilebilir. IoT-Ignite agent için aslında bir uygulama demek doğru sayılmaz; onlarca servis ve uygulamacığın bir arada bulunduğu bir framework demek daha doğru olur.

Bunun yanı sıra, Android üzerinde, IoT-Ignite agent ile birlikte çalışan ve kitabın değişik bölümlerinde örneklerle gösterilen birçok servis ve uygulama da var. Bunları kullanarak kolaylıkla Android üzerinde çalışan bir edge-gateway oluşturmak mümkün. Bu kitapta bunu gösteren onlarca örnek sistem paylaşılıyor.

Bu kitabın içeriğinde PilarOS (diğer adı TRDROID) adında bir işletim sistemine atıfta bulunuluyor. PilarOS, Google'ın Android Open Source Project (AOSP) projesinin güncel şeklini temel alıp ARM ve x86 tabanlı işlemciler için derlenmiş sürümüdür. ARDIÇ tarafından derlenen bu işletim sistemi, başka bir üreticinin "binary code"unu bulundurmaz ve IoT-Ignite platformunun içinde tanımlanan edge-gateway işlevini de içerir.

PilarOS'un içinde ARDIÇ tarafından geliştirilen AFEX ismi verilen bir çerçeve katman daha bulunur. AFEX (Android Frame Work Extensions), AOSP'de bulunan Android SDK ve API'lerine

ek olarak, 1500 üzerinde ek API arayüzü sağlamaktadır. Android işletim sisteminin boot-loader, kernel seviyelerinden en üst seviyede uygulama katmanlarına kadar tüm katmanlarının kontrol edilebilmesinin yanı sıra ek güvenlik yapılarını da sunmaktadır.

Android işletim sistemini IoT dünyasında edge-compute platformu olarak kullanan bilindik iki şirket var: birisi ARDIÇ diğeri de beklendiği gibi Android işletim sistemi sahibi olan Google Inc.

Google, 2016 yılının sonunda AndroidThings adını verdiği bir işletim sistemini duyurdu. Bu işletim sistemi standart Android işletim sisteminin tersine açık kaynak olarak yayınlanmadı. İçeriği tamamen kapalı kaynak ve Google'ın desteklemeye karar verdiği belli donanımlar üzerinde çalışır binary şeklinde yayınlandı. AndroidThings ile PilarOS yapıları gereği birbirlerine benziyor ve IoT dünyasında isterleri karşılayabilecek şekilde tasarlanmışlardır.

AndroidThings'in duyurulduğu 2016 Aralık ayında, ARDIÇ'ın geliştirmiş olduğu PilarOS işletim sistemi yaklaşık bir senenin üzerinde sahada ve müşteri kullanımındaydı. Günümüzde her iki işletim sistemi, (PilarOS ve AndroidThings) temel işlevsellik açısından birbirine paralel şekilde evriliyorlar ve her ikisinin de kendisine ait güçlü yönleri var.

Diğer yandan IoT-Ignite platformu ve bu platformun parçası olan IoT-Ignite agent AndroidThings cihazlarını da destekliyor. (AndroidThings kapalı bir işletim sistemi olduğundan IoT-Ignite agent'in işlevselliği PilarOS'ta olduğu gibi geniş bir yelpazede değil, Google'ın SDK'sının yetenekleri ile sınırlıdır).

IoT-Ignite platformu üzerinde geniş bir yelpazede çözüm oluşturmak mümkün. Sunacağınız servisler başta örneği verilen boş fıçı anlama servisinden, karmaşık Endüstri 4.0, "connected vehicle" servis alanlarına kadar geniş bir hizmet ağını kapsayabilir.

Geliştirme ortamında geliştiriciler için birçok örnek uygulama, bu kitapta da üzerinden geçilen örnek kullanım senaryolarına yer veriliyor. https://devzone.iot-ignite.com üzerinden geliştiricilerin kolaylıkla kendi servislerini geliştirebilmeleri için tasarlanmış IoT-Ignite platformunun sınırlı yeteneklerinin sunulduğu arayüze ulaşılabiliyor. Bu arayüz tamamen maker'ların veya servis geliştiricilerin kısa yoldan kendi servislerini hızla geliştirmesini sağlamaktadır.

Diğer yandan, IoT-Ignite kurumsal arayüzü (https://enterprise.iot-ignite.com), IoT-Ignite platformunun çok daha kapsamlı servislerine erişim sağlıyor. Bu arayüz ise kurumsal yaklaşımla; ölçeklenebilir ve sahada binlerce uç nokta ve servisin yönetilebilmesi için tasarlandı. Bu kitapta her iki arayüzün kullanımı da örneklerle gösteriliyor.

IoT'nin ne olduğunu detaylı olarak örnekleyen ve IoT-Ignite ile uçtan uca bir IoT platformunda neler yapılabileceği gösteren ve Türkiye'de bir ilk olan bu kitabın geliştiricilerin ufkunu açması ve dünyada IoT konusunda öncü bir konuma gelmemize faydasının olması bizim için büyük bir gurur kaynağı olacaktır.

Bu kitapta kullanılan uygulamaların, geliştirme ortamlarının ve SDK'lerin güncel sürümleri:

Adı	**Sürümü**	**Link**
Uygulama		
IoT-Ignite Agent	0.8.42	https://play.google.com/store/apps/details?id=com.ardic.android.iotigniteagent
SPA* *Android 5.0 ve daha alt versiyonlar için kullanılabilir.*	1.0.6	https://download.iot-ignite.com/ServicePlatformApp/
HwNodeAppTemplate	1.1	https://github.com/IoT-Ignite/android-library-hwnodeapptemplate
IoT Ignite Demo App	0.8.12	https://download.iot-ignite.com/DemoApp/
Dynamic Node Example	0.8.24	https://download.iot-ignite.com/DynamicNodeExample/
sample-mqtt-NodeToCloud	1.0.0	https://github.com/IoT-Ignite/sample-mqtt-NodeToCloud
SDK		
IoT-Ignite SDK	0.8.2	https://devzone.iot-ignite.com/knowledge-base/iot-ignite-device-sdk/
mqtt-client-wrapper	1.0.0	https://github.com/IoT-Ignite/java-mqtt-client
Geliştirme Ortamı		
Android Studio IDE	3.0 ve üzeri	https://developer.android.com/studio/
Gradle	4.1 +	Android Studio ile gelecektir.
Arduino IDE	1.6.5 +	https://www.arduino.cc/en/Main/Software
İşletim Sistemi		
PilarOS	20160927_HDMI 20160927_VGA	https://download.iot-ignite.com/Pilaros-rpi3/

Tabloda yer alan araçların güncel versiyonlarına ve indirme bağlantılarına *www.iot-ignite.com/docs/IoT-Ignite-Book-v1/AppReference* adresinden erişebilirsiniz.

İÇİNDEKİLER

BÖLÜM 1

IoT ve IoT 2.0

IoT Nedir ve Yaşam Döngüsünü Kavramak

IoT, Internet of Things ifadesinden elde edilen bir kısaltmadır. Türkçe karşılığı Nesnelerin İnterneti olan bu yaklaşım, fikir olarak eskilere dayanan (1970'ler) ancak uygulama olarak yeni yeni gelişen bir teknoloji yaklaşımıdır. Amacı da elektronik cihazların internet üzerinden haberleşmesini, kontrol edilmesini ve belirli bir görev için kullanılmasını sağlamaktır. Bu yeni bir teknoloji değil, var olan teknolojilerin kullanılarak elektronik cihazların uzaktan kontrolünü sağlayan bir yaklaşım modelidir.

Internet of Things kavramı ilk defa Kevin Ashton tarafından 1999 yılında kullanılmıştır. Çalıştığı kurumun yöneticilerinin dikkatini RFID teknolojilerine çekmeyi amaçlayan bir sunumunda bu tabiri kullanmıştır. O zamanlar için internet teknolojisi sunumun amacına hitap ettiği ve bir bakıma elektronik cihazları birbirine bağladığı için Internet of Things kavramını kullanmıştır. O zamandan beri kullanılan bu kavram 2005 yılında The International Communication Union (Uluslararası İletişim Birliği) tarafından hazırlanan Internet of Things isimli bir raporda resmi olarak kabul edildi. Bu raporda Internet of Things için aşağıdaki gibi bir açıklama yapılmıştır.

"Bilgi ve iletişim teknolojileri dünyasına yeni bir boyut eklendi: herkesin her yerden herhangi bir cihazdan bağlantı sağlayabileceği bir kavram. Bağlantıların çoğalması beraberinde dinamik bir ağın yani Internet of Things'in oluşmasını sağlamaktadır."

Bu rapor hakkında ayrıntılı bilgiye erişmek için şu linki ziyaret edebilirsiniz:

https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf

Birçok analiste göre, Internet of things önümüzdeki on yılın en etkili teknolojisi olarak değerlendirilmektedir. Internet of Things, yaşamımız ve alışkanlıklarımızı doğrudan etkileyen ve değiştiren teknolojik bir kavramdır. IoT birbirine bağlı akıllı telefonlar, tabletler ve bilgisayarlardan daha fazlasıdır. Kısaca bu kavram nesnelerin birbirine bağlı olduğu ve aynı zamanda internete bağlandığı bir ekosistemdir. Internet of Things potansiyel olarak internete bağlanabilen, veri ve bilgi alış verişi yapabilen her donanımı içerebilir. Bu teknoloji yaklaşımının dinamikleri arasında, daha az enerji tüketimiyle birlikte veri işleme gücü artan işlemciler ve elektronik cihazların ucuza mal edilmesini gösterebiliriz.

IoT kavramını, bir ürünün daha kaliteli olması ya da bir hizmetin daha iyi gerçekleşmesi amacıyla ihtiyaç duyulan tüm sistemleri sağlamak için ağ, algılama, büyük veri (big data) ve yapay zeka teknolojisini kullanan gelişmiş bir otomasyon sistemi olarak da tanımlayabiliriz. IoT ile bir sistemin daha berrak ve şeffaf bir şekilde çalışması sağlanır. Bu şeffaflık ürün veya hizmetin daha kaliteli olmasını da beraberinde getirecektir. Örneğin, bir fabrika düşünün, amacınız bu fabrikada aralıksız bir şekilde üretim yapmak olsun. Sürekli çalışan bir fabrikanın arıza tespitini yapmak kolay olmasa gerek. Ayrıca makinelerin sürekli çalışması zaman içinde yıpranmalara ve dolayısıyla ürünlerin kalitesiz olmasına neden olacaktır. Eğer IoT tabanlı ve içerisinde yapay zeka ve büyük veri olan bir sistem kullanırsanız, çalışan makinenizin arızası daha ortaya çıkmadan önce arıza tespiti yapılabilir, hatta arıza belirli bir seviyeye ulaştığında sistem sizin adınıza arızalı olan parçanın siparişini verebilir. Böyle bir sistemde hem ürünlerin kalitesi hem de fabrikaların sürekli çalışması sağlanmış olur. Bu örnek IoT yaklaşımının endüstride ne kadar etkili olduğunu göstermek adına çarpıcı bir örnektir.

Şimdi de IoT kavramı ile birlikte kullanılan ancak bu kavramdan farklı olan bazı terimler hakkında kısaca bilgi verelim. Bu terimler şunlardır:

- Machine to Machine (M2M)
- Industrial Internet of Things (IIoT)
- Web of Things (WoT)
- Internet of Everything (IoE)
- Industry 4.0

Bu kavramlar hakkında tek tek bilgi verip Internet of Things arasındaki farkları belirtelim.

Machine to Machine (M2M)

Ağ cihazlarının insan eliyle yardım almadan bilgi alışverişinde ve eylemde bulunmasını sağlayan bir teknolojidir. İşlemler makineden makineye ya da point-to-point olarak gerçekleşir ve arada insan faktörü yoktur. Burada iletişimi sağlamak için RFID, Wi-Fi ve hücresel ağlar kullanılır. M2M'de bağlı cihazlar için herhangi bir standart bulunmuyor şu an için mevcut teknolojiler kullanılmaktadır.

M2M ve IoT arasındaki farklılıklar aşağıdaki gibidir.

M2M	**IoT**
Noktadan noktaya iletişim cihaz içine gömülmelidir.	Cihazlar IP ağlarını kullanarak iletişim kurar. Ayrıca farklı iletişim protokollerini destekler.
Cihazlar arası veri dağıtımı genellikle hücresel veya kablolu ağlar ile sağlanır.	Veri dağıtımı bulutta barındırılan bit katman ile sağlanır.
Bağlı cihazlar internet bağlantısına güvenmezler.	Çoğu durumda cihazlar aktif bir internet bağlantısı isterler.
Sınırlı entegrasyon.	Sınırsız entegrasyon.

IoT ve M2M arasında farklılıklar olsa da M2M ile gelen MQTT protokolünü IoT uygulamalarında kullanabiliyoruz. Hatta IoT-Ignite platformu da MQTT protokolünü desteklemektedir.

Industrial Internet of Things (IIoT)

M2M'den farklı olarak sadece makineler arasındaki bağlantıları değil, insan arabirimlerini de içeren bir teknolojidir. Yıllardır endüstride kullanılan otomasyon teknolojisi bu kavram altında yer almaktadır. IIoT'in arkasındaki felsefe, akıllı makinelerin insanlardan daha doğru, tutarlı bir şekilde veri yakalama ve iletişim kurmakta daha iyi olmasıdır.

IIoT ve IoT teknolojilerinin aynı veya farklı olduğu üzerine birçok tartışma yapılmış. Ancak zamanla bu iki kavram arasındaki farklılıklar veya benzerlikler kendini yavaş yavaş göstermiştir. Genel anlayış, IoT'nin daha geniş ve daha soyut bir kavramı ifade ettiğini, IIoT'in ise bir ağda bulunan makinenin kendisini veya ortamı izleyen ve kontrol eden, uzaktaki bir uygulama altyapısıyla iletişim kurmak için ağı kullanan aygıtlar olarak ifade ettiğidir.

IoT ile IIoT arasındaki en büyük fark ise; IoT'nin sanal ve gerçek dünyayı birleştirmesi, IIoT'in ise bir sistemde bulunan makinelerin bir ağ üzerinde iletişim kurmasıdır. Yani IoT, IIoT'ye göre daha geniş bir kavramdır ve evrenseldir. IIoT ise daha çok yerel uygulamalarda kullanılmaktadır. Buna örnek olarak bir fabrikada bulunan otomasyon sistem veya sistemleridir.

Web of Things (WoT)

Web of Things, gerçek dünyadaki nesnelerin, World Wide Web'in bir parçası olmasına izin veren yaklaşımlar, yazılım mimari stilleri ve programlama kalıplarını tanımlamak için kullanılan bir kavram olarak karşımıza çıkmaktadır. Nesneleri web üzerinden birleştirmeyi hedefleyen bu teknoloji yalnızca yazılım mimarisi üzerine yoğunlaştığı için diğer kavramlara göre kapsamı oldukça dardır. Ağ üzerinde bulunan ve nesneleri birbirine bağlayan bir uygulama katman olarak da ifade edilebilir.

Web of Things, var olan web standartlarını kullanarak nesneleri birbirine bağlamaya çalışmaktadır. Web standartlarını kullanarak farklı cihazları web'e bağlamak, sistemler ve uygulamalar arasındaki entegrasyonu sağlamak daha kolaydır.

IoT ve WoT arasındaki ilişki diğer kavramlara kıyasla açık ve nettir. Çünkü IoT kavramı, WoT kavramını da kapsamaktadır. IoT bir sistemde HTTP protokolü, yani web standardı kullanılarak nesneler arasındaki haberleşme veya nesnelerin Cloud ile olan iletişimleri sağlanmış olur. En önemli fark ise WoT'nin sadece bir uygulama katmanı olması, içerisinde fiziksel katman veya donanımların bulunmamasıdır. Ancak IoT sistem olarak fiziksel ve uygulama katmanlarından meydana gelmektedir.

Internet of Everything (IoE)

IoT kavramından önce, bu sistem farklı şekillerde isimlendiriliyordu. Internet of Everything bu kavramlardan biridir ve IoT kavramından önce Cisco tarafından kullanılmıştır. Cisco'nun bu kavramı çıkarmasının temel sebebi IoT'nin gücünü geç fark etmesi ve kaçırdığı gemiyi durdurmak için IoT kavramını genişleterek kullanmasıdır. Aynı durum Edge Computing içinde geçerlidir. Zira Cisco bu sistemi biraz değiştirerek Fog Computing adı altında lisanslamaya çalıştı. IoT kavramına alternatif getiren tek şirket Cisco değildi. Intel firması da başlangıçta IoT sistemlerini Embedded Internet (Gömülü İnternet) olarak ifade etmişti.

IoE ve IoT'nin odak noktası nesnelerin internetidir. Ancak IoE'de, IoT'den farklı olarak dört katman bulunmaktadır. Bu katmanlar IoE konsepti içinde zorunlu olarak yer almaktadır. Zaten Cisco'nun amacı IoT farklı bir şekilde lisanslamak olduğu için bu gibi kriterlerin olması lisans için bir nebze olsun zorunludur.

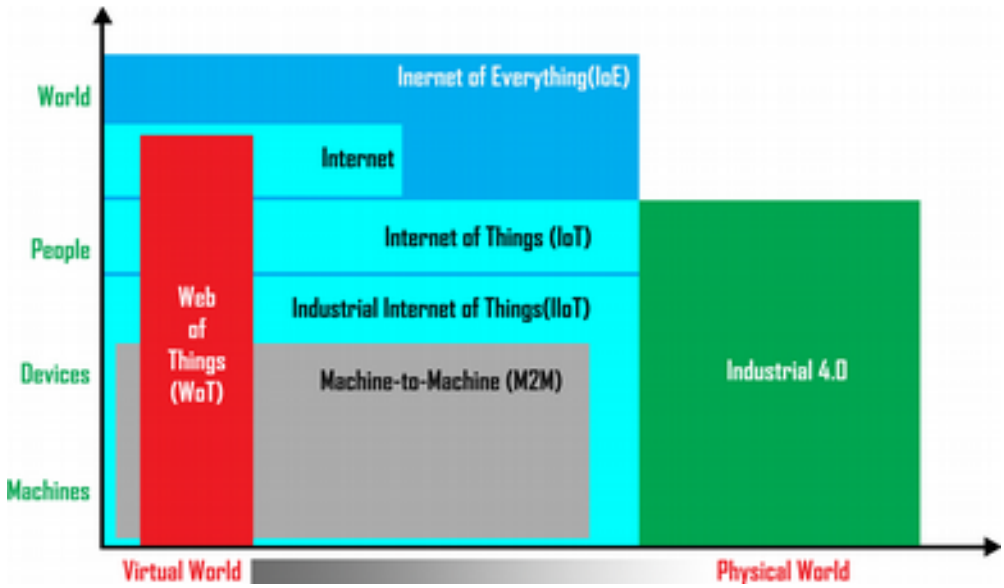
IoE'de yer alan katmanlar; İnsanlar (People), Veri (Data), Süreç (Process) ve Nesneler (Things) şeklindedir. Bu kavram IoT'ye göre daha geniş bir kavramdır. Cisco bunu lisanslarken, Things katmanını IoT olarak lisanslamıştır. Buradan IoT kavramının, IoE içinde saklanmaya çalıştığını görebiliriz.

Industry 4.0

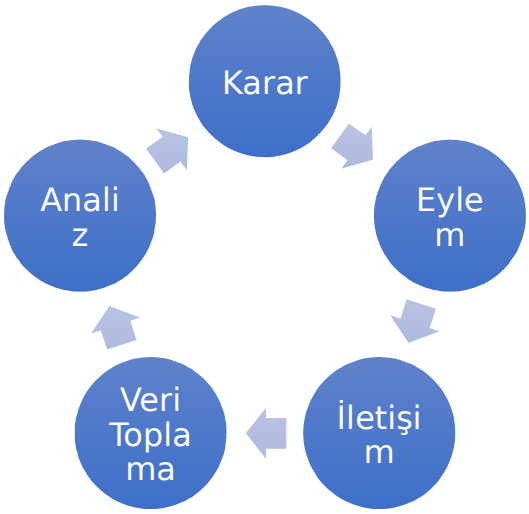
IoT ve Endüstri 4.0 arasındaki en önemli fark, yaş farkıdır. IoT kavramı 1999'da ilk kez kullanılırken, Industry 4.0 2011 yılında Alman hükümetinin bir girişimi olarak kullanıldı. Bu terim 4.Endüstriyel Devrimi olarakta anılmaktadır. Bundan dolayı Endüstriyel devrimlerini incelemekte fayda var.

- Endüstri 1.0: Bu süreçte, su ve buhar gücü kullanımı ile çalışan mekanik sistemler yer almaktadır. Bunların üretildiği dönem 1. Endüstriyel Dönem olarak anılmaktadır (1750-1830 yılları arasında). Ayrıca üretimin makineleştiği dönemdir.
- Endüstri 2.0: Elektrik enerjisinin üretilmeye başlandığı dönemi ifade etmektedir (1870 yılında başladı). Üretimin serileştiği dönemdir. Telgraf ve telefon gibi icatlar bu dönemde üretilip kullanıldı.
- Endüstri 3.0: Elektronik ve bilişim teknolojilerinin birlikte kullanıldığı dönemdir (1969 yılında başladı). Üretimin sayısallaştığı ve otomasyon sistemlerinin oluşturulduğu dönemdir. Kişisel bilgisayarların üretimi bu dönemde gerçekleşti.
- Endüstri 4.0: Bu dönemde internetin üretimde kullanılması ağır basmaktadır. Bundan dolayı Endüstri 4.0, IoT kavramını da kapsayan daha geniş bir kavramdır. IoT ile arasındaki diğer önemli fark da Industry 4.0'ın devlet veya akademik tabanlı olarak devam etmesidir.

IoT ve diğer kavramlar arasındaki farkları görmek ve aralarındaki ilişkiye daha iyi anlamak için aşağıdaki grafiği inceleyiniz.



Bilmemiz gereken bir diğer kavram da IoT yaşam döngüsüdür. Yaşam döngüsü doğrusal olmaktan ziyade dairesel bir yaklaşıma sahiptir. Çünkü amaç sistemi sürekli takip etmek ve elde edilen verilere göre eylemde bulunmaktır. Örnek bir yaşam döngüsü aşağıdaki gibidir.



Yaşam döngüsü birçok platformda farklı şekilde sizlere sunulabilir. Ancak genel olarak bir IoT yaşam döngüsü yukarıdaki gibidir. Yaşam döngüsünü incelerseniz, burada aslında bir başlangıç noktası bulunmadığı görülür. Verilen Analiz, Karar, Eylem, Veri Toplama ve İletişim bloklarından herhangi birisi IoT yaşam döngüsünü başlatabilir. Ancak genel kanı, oluşturulan bir IoT sisteminde durağanlık olmadığıdır. Yani yukarıda verilen yaşam döngüsü aktif olduğu anda durmadan çalışmaya devam eder.

Burada verilen evreleri fikir vermesi için kısaca açıklayalım.

Veri Toplama

Sensör veya çevre birimlerinden veri alınmasını sağlayan evredir. Bu evrede sadece veriler alınır ve ihtiyaç halinde sistemin veritabanına ve buluta kayıt edilir. Ya da herhangi bir kayıt işlemi yapılmadan doğrudan analiz evresine geçilerek analiz işlemleri yapılır.

Analiz

Sensör veya çevre birimlerinden alınan ham verilerin analizini yani işlenmesini ve kullanılabilir verilere dönüştürülmesini sağlayan evredir. Büyük Veri sistemleri bu evrede IoT yaşam döngüsünde etkisini göstermektedir. Büyük Veri ile belirli bir kurala bağlı olmayan verilerin analitik olarak analiz işlemi gerçekleştirilir.

Karar

Analiz evresinde oluşturulan yapılandırılmış verileri kullanarak karar mekanizmalarının başlatılmasını sağlayan evredir. Bu evrede, Analiz aşamasında elde edilen veriler ile sistem ya da insanlar tarafından belirlenen sınır verileri arasında bir kıyaslama işlemi yapılır. Kıyaslama işlemi herhangi bir eylemin gerçekleşmesini sağlamak için kullanılır.

Eylem

Karar aşamasında yapılan kıyaslamalara göre herhangi bir eylemde bulunmayı sağlayan evredir. Bu aşamada evimizdeki klimayı çalıştırmak bir eylem olduğu gibi klimayı kapatmak da bir eylemdir. Önemli olan burada bir işlemin veya hareketin gerçekleşmesini sağlamaktır.

İletişim

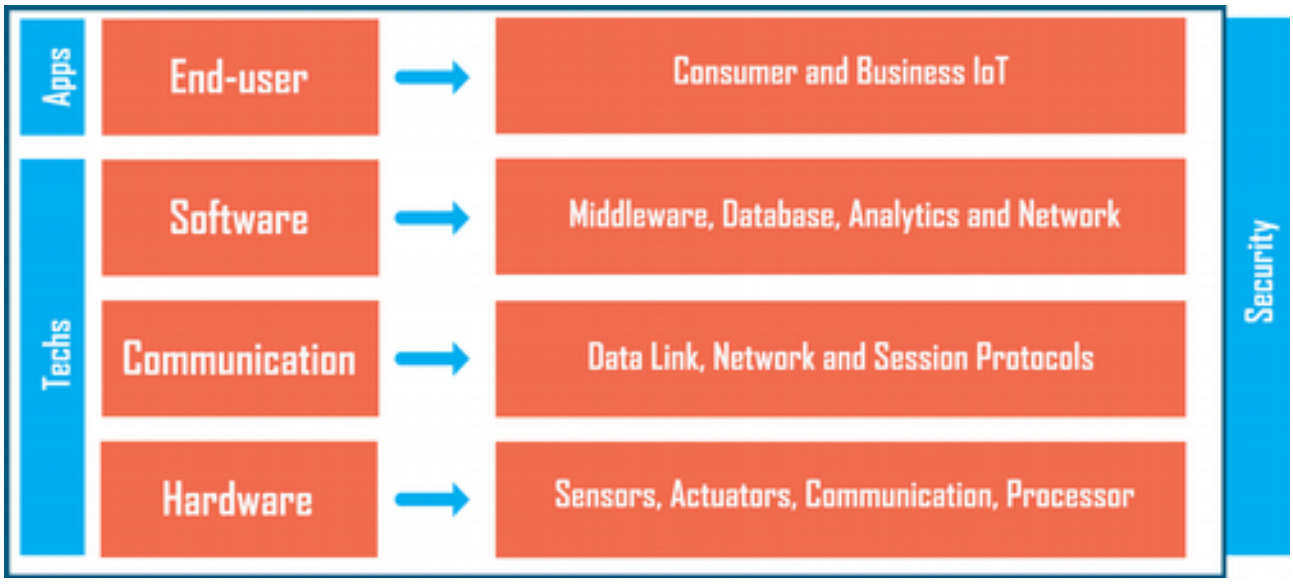
IoT bir sistemde iletişim en önemli kavramlardan biridir. Hatta iletişimin olmadığı bir yerde tam olarak IoT'den söz edilemez. İletişim evresinde birbirinden farklı ve uzakta bulunan cihazların haberleşmesi sağlanır.

IoT Teknoloji Katmanları

IoT Tabanlı bir sistem dört katmandan meydana gelmektedir. Bu katmanlar ayrıca IoT sistemi geliştirmek için ihtiyacımız olan temel gereksinimleri belirlemeyi de sağlamaktadır. IoT bir sistemde olması gereken temel katmanlar ve kullanılan teknolojiler dört başlık altında ele alınabilir.

- Yazılım (Software)
- İletişim (Communication)
- Donanım (Hardware)
- Güvenlik (Security)

Öncelikle bu katmanlar arasındaki ilişkiyi bir şekilde gösterelim ve daha sonra bunları açıklayalım.



Donanım katmanında IoT sistemi için kullandığımız sensör ve işlemciler yer alır. Burada ihtiyacımız olan donanımlar ve bunların yardımcı ve tamamlayıcı türevleri kullanılır.

İletişim katmanında, donanımlar arasında veri trafiğini sağlayan ve genellikle düşük band desteği bulunan iletişim protokolleri bulunur. Örneğin; radyo sinyalleri, Wifi, NFC veya Bluetooth gibi teknolojiler bu katmanda yer alır.

Yazılım katmanında ise veri depolama, veriler için analitik işlemler, front end işlemler ile ağ ve cihaz yönetimini gerçekleştiren yazılımlar yer alır.

Güvenlik, ayrı bir katman olmayıp her katmanda sağlanması gereken bir gereksinimdir. Buradaki temel amaç; yazılım, donanım ve iletişim katmanlarında sistemin güvenliğini sağlamaktır. Güvenlik katmanında yazılım ve donanım gibi bileşenlere ihtiyacımız vardır.

Tüm bu katmanların bir araya gelmesiyle birlikte IoT sistem veya sistemleri meydana gelir. Yazılım, donanım, iletişim ve güvenlik işlemlerinin yapılmasını sağlayan IoT sistemler için her dört katmanın oluşturulması gerekiyor. Bunlara ek olarak End-User, yani son kullanıcı uygulamalarını

da ekleyebiliriz. Çünkü geliştirilen IoT sistemlerinin temel amacı son kullanıcılara hizmet sağlamaktır. Örneğin bir bankanın işlemlerinden yararlanmak için geliştirilen mobil uygulama buna iyi bir örnektir.

Güvenlik katmanı yazılım, donanım, iletişim ve son kullanıcı katmanlarını saran ve her zaman için olması gereken bir katmandır. IoT bir sistemin şu an için en büyük handikabı şüphesiz güvenliktir.

IoT ve IoT 2.0 Farklılıkları ve Evrim Süreci

IoT ve IoT 2.0 arasındaki farklılık 2015 yılında gerçekleşen ITU Telecom World etkinliğinde Ryutaro Kawamura tarafından hazırlanan bir sunumda ele alınmıştır. IoT 2.0 ile birlikte IoT kavramı daha ileriyi taşınmakta ve insan kararlarının arabuluculuğu olmadan sistemlerin geliştirilmesi hedeflenmektedir. IoT 2.0'da tüm işlemler makineler veya elektronik cihazlar ile sağlanmaktadır. Bu işlemler için de yapay zeka sistemlerinin geliştirilmesi gerekiyor. Tabii bunların geliştirilmesi için daha çok zamana ihtiyaç olduğundan dolayı şimdilik insan faktörü ve IoT birlikte devam edecektir. Ancak IoT derken aslında amacın elektronik cihazların iletişimi olduğunu belirtmekte fayda var.

ITU Telecom World 2015 etkinliğindeki sunuma erişmek için şu linki ziyaret edebilirsiniz:

https://www.ituaj.jp/00_sg/20151012_TW15/slide/3_NTT.pdf

IoT şu an için bulut sistemlerini kullanarak sensörlerden gelen veriler üzerinde işlem yapabilmektedir. Ancak bu sunumda IoT 2.0'ın Edge Computing (Uçta İşleme - Uç Bilişim) veya Cisco'nun yeni tabiriyle Fog Computing (Sis Bilişim) teknolojisini kullanacağından bahsedilmektedir. Bu teknolojiden biraz bahsedersek. Gelecekte çok büyük miktarda IoT sensörleri IP adreslerini kullanarak internete bağlanacaktır. Hatta Cisco'nun tahminlerine göre 2020 yılında 50 milyar cihazdan oluşan devasa bir IoT ağı oluşacaktır. Bu cihazlardan gelen veriler şimdilik bulut sistemlerinde analiz edilmektedir. Ancak bulutun yetmediği durumlarda sis bilişim teknolojisini kullanacağız. Sis bilişim sistemi, anlık veri işlemenin çok önemli olduğu ve hemen o anda bazı eylemlerin yapılması gerektiği durumlarda kullanılmaktadır. IoT ile kullanılan bulut sistemlerindeki gecikme bu sayede ortadan kalkacaktır. Sis bilişim sistemi ve bulut bilişim birlikte kullanılmaktadır. Ancak anlık verilerin işlenmesinde sis bilişim etkin olarak kullanılmaktadır.

Bulut ve sis arasındaki farkları anlamak için aşağıdaki tabloyu inceleyelim.

	Bulut	**Sis / Uç**
Gecikme	Fazla	Az
Sunucu ile istemci arası mesafe	Fazla	Az
Güvenlik açıkları	Belirsiz	Belirli
Veri iletimine saldırı	Yapılabilir	Yapılma ihtimali düşük
Cevaplama süresi	Dakikalar, günler hatta haftalar	Milisaniye
Ağdaki Konumu	Internet	LAN/WAN

Yukarıdaki diyagramı incelerseniz, Fog veya Edge teknolojisinin Cloud ile IoT cihazları arasında olduğunu görebilirsiniz. Ayrıca bu teknoloji bir önbellekleme sistemi olarak ta tanımlanmaktadır. Edge ve Fog'u aynı katmanda göstersek bile her iki sistem arasında çok küçük farklılıklar var.

Edge teknolojisi her ne kadar Cisco tarafından Fog adı altında lisanslanmış olsada aslında ARDIC Technology firması IoT-Ignite platformunda bu teknolojiyi yıllardır kullanmaktadır. IoT-Ignite, Gateway, yani ağ geçidi cihazlarını kullanarak Fog teknolojisinde gerçekleştirilen anlık kontrollerin yapılmasını sağlamaktadır. Gateway olarak Android cihazlar, Raspberry Pi, bilgisayarınız veya bu işlem için özel olarak tasarlanmış cihazları kullanabilirsiniz. Ancak ağırlıklı olarak Android yüklü cihazların kullanıldığını görmekteyiz. Bunu da Android'in yaygın olmasına bağlayabiliriz. Kavram olarak Gateway, Node ve Thing gibi anahtar kelimelere sahip olan bu platformu kullanarak IoT uygulamalarını Fog seviyesinde geliştirebiliriz. Bunlarla ilgili bilgileri bu bölümün sonunda ayrı bir başlık altında inceleyeceğiz.

Yukarıda IoT-Ignite platformunun logosunu görmektesiniz. 2012 yılında geliştirilen IoT-Ignite yapı olarak, 2015 yılında Cisco, Dell ve Intel'in birlikte kurduğu OpenFog konsorsiyumu ile aynı amaca hizmet etmektedir. Tek farkı bu düşünceyi ARDIC firmasının üç yıl önceden geliştirmesidir. Böyle bir sistemin yerli bir firma tarafından kurulması gerçekten IoT ve ülkemiz için çok önemlidir. Kitabımızda ele aldığımız bu platformun önemini belki bu şekilde daha iyi aktarmış oluruz sizlere. Çünkü birçok büyük firma Gateway seviyesinde Edge bilişimi kullanmadan önce ARDIC, bunu tüm insanlığın hizmetine sunmayı başarmıştı.

IoT-Ignite ile ilgili demo uygulamanın örnek bir görüntüsünü aşağıdaki gibidir.

Daha sonra ayrıntılarına yer vereceğimiz bu platformu kullanarak veri yönetiminin kolay bir şekilde yapıldığı IoT sistemlerini geliştirebilirsiniz.

Şimdi de IoT'nin evrim sürecini bir tablo içinde sizlere sunucağız. Burada başlangıç olarak 1999 tarihini ele alacağız ancak IoT'nin tarihi kablosuz iletişim kadar eskidir. Çünkü IoT'de amaç elektronik cihazların birbirine bağlanması vardır. Ancak evrimsel sürece baktığımızda meydana çıkan veri miktarındaki fazlalık ve bağlı cihaz sayısındaki artış IoT kavramını IoT 2.0'a kadar gelmesini sağlamıştır. İlk zamanlar RFID teknolojisi ile başlayan bu yaklaşım zaman içinde meydana gelen BigData, Cloud Computing ve Fog Computing gibi teknolojiler ile IoT 2.0 olarak önümüze çıkmaktadır.

1999	Internet of Things kavramı ilk defa Kevin Ashton tarafından 1999 yılında kullanılmıştır.
2005	1999 yılından beri kullanılan bu kavram 2005 yılında The International Communication Union (Uluslararası İletişim Birliği) tarafından hazırlanan Internet of Things isimli bir raporda resmi olarak kabul edildi. Bu rapor ile birlikte IoT evrensel olarak kabul edilmiştir.
2006	Bu tarihten itibaren IoT Avrupa tarafından tanındı ve Avrupa'da ilk IoT Konferansı düzenlendi.
2008	Cisco'ya göre IoT bu iki tarihte doğdu. Çünkü bu tarihte internete bağlanan cihaz sayısı ilk kez dünyadaki insan sayısını geçti. 2010'da bu rakam 12.5 milyar cihaza ulaştı.
2010	Çin Başbakanı Wen Jiabao, 2010 yılında meydana gelen ve yaklaşık 12 gün süren trafik sorununa çözüm bulmak için IoT teknolojisini Çin için kilit bir endüstri olarak nitelendirdi. 2010 yılından itibaren IoT'ye büyük yatırımlar yapılmaya başlandı.
2011	İnternete bağlanan cihaz sayısının hızla artamısıyla beraber Ipv4 protokolü yetersiz kalmaya başladı. Bunun için IPv6 protokolü geliştirildi. Bu şekilde IoT önündeki en büyük engellerden biri daha ortadan kalkmış oldu. Yani IPv6 aslında IoT önündeki engelleri ortadan kaldırmak için oluşturuldu. Bu protokol ile 2^{128} adet cihaz internete bağlanabilecek. Bu tarihten itibaren meydan gelen önemli gelişmelerden bazıları da şunlardır: Cisco, IBM ve Ericsson gibi firmalara konuyla ilgili geniş eğitim ve pazarlama girişimleri üretmeye başladı. Arduino ve diğer birçok donanım platformu bu dönemde olgunlaştı.
2012	**ARDIC Tehcnology firması IoT-Ignite platformunu geliştirdi. Ayrıca bu sistem ITU Telecom World 2015 etkinliğinde ele alınacak olan IoT 2.0 standardına da sahiptir. Temel amacı Edge veya Fog denilen sistemi kullanarak Gateway seviyesinde IoT sistemler oluşturmayı sağlamaktır.**
2015	ITU Telecom World 2015 etkinliğinde IoT ve IoT 2.0 arasındaki farklılıklar ilk kez ele alındı. IoT, insanlar ve internet bağlı cihazlar arasında kurulurken, IoT 2.0'da ise tüm işlemler makineler veya elektronik cihazlar ile sağlanmaktadır. Bu tarihte OpenFog konsorsiyumu Cisco, Dell ve Intel tarafından kullanılarak IoT 2.0'a olan yolculuk daha da hızlandı. Fog veya Edge teknolojisinin amacı gerçek zamanlı analiz ve eylemleri gerçekleştirmeyi sağlamaktır.

IoT-Ignite için Bilmemiz Gereken Bazı Temel Kavramlar

Bu sistemin nasıl çalıştığını anlamak adına IoT-Ignite ile gelen bazı temel kavramları bilmemizde fayda var. Bunlar şu şekildedir:

Gateway

Ağ geçidi olarak Türkçe'ye çevrilen bu kavram IoT-Ignite ile iletişim kuran cihazlardır. IoT sisteminde bulunan veriler bu cihazlar ile buluta gönderilir. Gateway olarak; Android yüklü cihazlar,

evimizdeki bilgisayarımız veya Raspbery Pi gibi cihazlara kullanabileceğimiz gibi, sadece bu işlemi yapmak için geliştirilmiş cihazları da kullanabiliriz. Örneğin; Dell Edge Gateway gibi.

IoT-Ignite, Gateway olarak Android'e daha çok önem vermektedir. Ancak diğer bahsettiğimiz cihazlarıda rahatlıkla kullanabilirsiniz. Aşağıda verilen resim IoT-Ignite platformunun Gateway ekleme sayfasından bir görüntüdür.

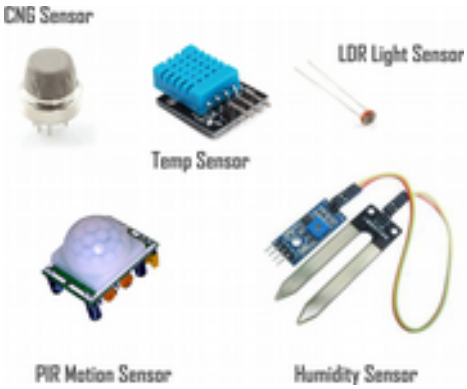
Node

IoT sistemde bulunan sensörlerden veriyi alıp Gateway yani ağ geçidine gönderen cihazlardır. Gateway cihazlar, Node'tan gelen verileri buluta gönderir. Node olarak kullanabileceğimiz aygıtlar oldukça fazladır. Arduino, NodeMCU, Raspberry Pi vb.

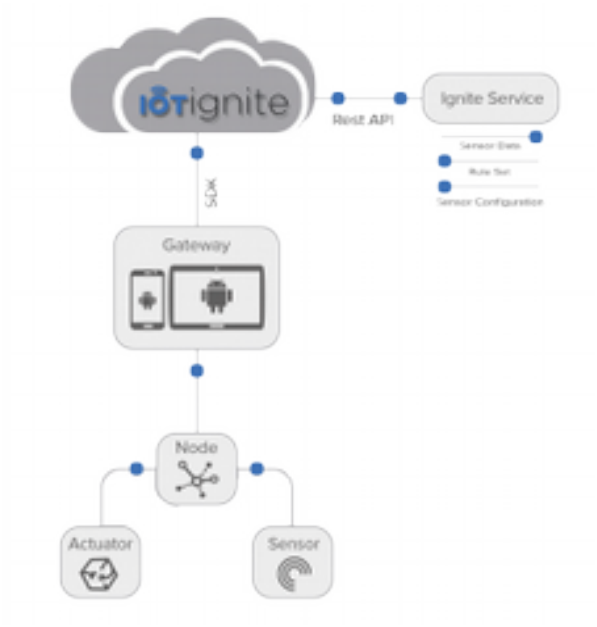
Thing

Node cihazlara bağlanan sensörlere Thing denir. Sensörler sıcaklık, nem, ışık şiddeti, gaz, mesafe vb. verileri ölçmek için kullanılan cihazlardır. Bunların meydana getirdiği bir ekosistemi IoT-Ignite ile rahatlıkla yönetebiliriz.

Aşağıda verilen resimde IoT-Ignite ile kullanabileceğimiz bazı sensörleri görmektesiniz.



Bu kavramların meydana getirdiği bir IoT ekosistemi aşağıdaki gibidir.



Bu ekosistemi incelediğimiz zaman Node cihazların sensör ve actuator cihazlardan verileri alıp Gateway cihazlara gönderdiğini görmekteyiz. Node ile Gateway arasında veri trafiği Bluetooth, Wireless veya kablolu bağlantılar ile sağlanmaktadır. Ayrıca isterseniz hücresel ağları, RFID ve NFC teknolojilerini hatta Wifi-Direct teknolojisini de kullanabilirsiniz. Gateway, Node cihazlardan gelen verileri buluta göndermektedir. Bu şekilde aynı ekosistemi paylaşan IoT uygulamaları ortak bir platforma sahip olmakta ve veri bütünlüğü sağlanmaktadır.

IoT-Ignite'ın Özellikleri

ARDIC firmasının geliştirdiği IoT-Ignite ile ilgili bilinmesi gerekene genel özellikler aşağıdaki gibidir.

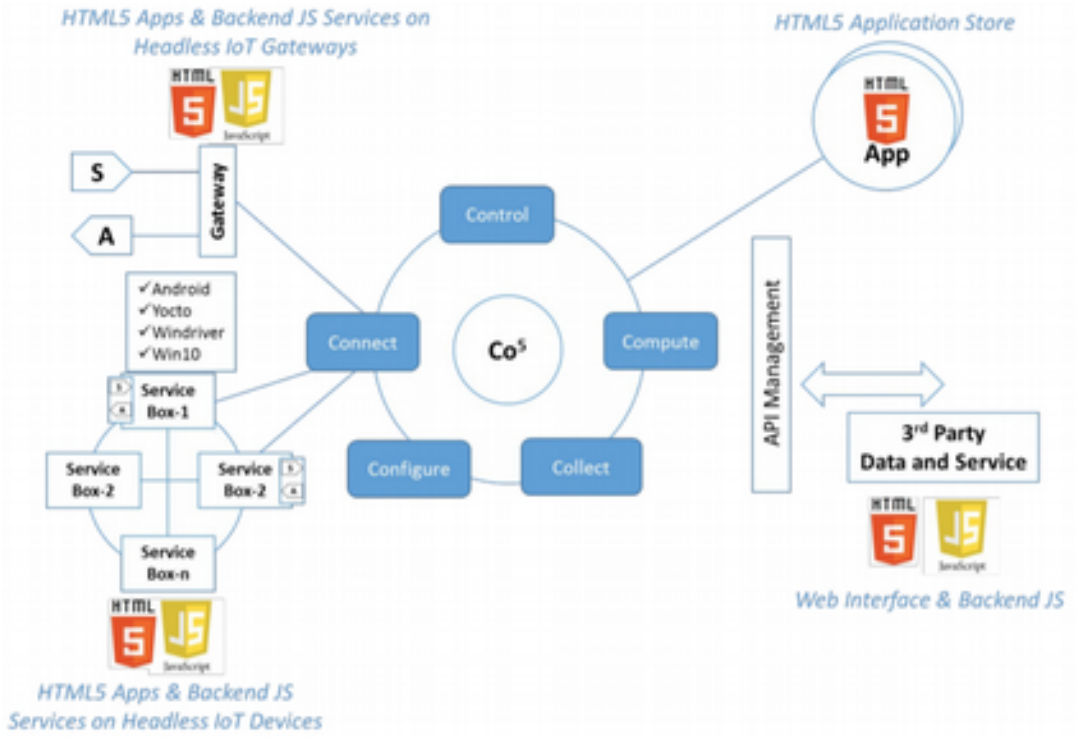
- Güvenilir,
- Ölçeklenebilir,
- Dağıtılmış bir platform olması,
- Çoklu ortam özelliği,
- Güvenli ağ bağlantısı,
- Kullanım istatsitikleri,
- Arabuluculuk,
- ve Karmaşık Olay Takibi gibi zengin özelliklere sahip olarak gelmektedir.

IoT-Ignite platformu yukarıdaki özelliklerinden ve desteklerinden dolayı geleceği parlak bir PaaS girişimi olarak yaşamına devam etmektedir. Bu platformun üç temel direği ise şunlardır;

- **CO**[5] : Connect (Bağlantı), Configure (Yapılandırıma), Collect (Veri Toplama), Control (Kontrol) ve Compute (Hesaplama) işlemlerini temsil eden bir kısaltmadır. IoT-Ignite API'leri servis sağlayıcılarının ağlarında bulunan kenar cihazlara kolayca bağlamalarını, yapılandırmalarını, yönetmelerini ve izlemelerini sağlar. Ayrıca veri toplama, erişim ve analiz gibi işlemleri yaparken güvenli ve dağıtılmış bir platform üzerinde faaliyetlerini gerçekleştirir.

- **Rich User Interface & Experience (Zengin Kullanıcı Arayüzü ve Deneyimi)**: IoT-Ignite kullanıcı araçları, HTML5 tabanlı olduğundan hem kullanıcı hem de servis sağlayıcıları için zengin bir kullanıcı arayüzüne sahip olup bu sayede daha konforlu bir deneyim sağlamaktadır.
- **Ease of Integration (Entegrasyon Kolaylığı)**: Farklı kurumların ürettiği 3. partilerde dahil olmak üzere kolayca IoT-Ignite ile entegre edilebilir. Bu şekilde IoT-Ignite platformunun bir parçası haline getirilebilir.

Burada verilen bilgiler IoT-Ignite'ın temel yapı taşlarını bilmemizi sağlamaktadır. Bu bilgilerden yola çıkarak IoT-Ignite ve HTML5 teknolojisi arasındaki ilişkiyi aşağıdaki şekilde gösterebiliriz.



IoT-Ignite'nin Desteklediği Server/Client Sağlayıcıları

IoT-Ignite platformu hem istemci hem de sunucu taraflı uçtan uca setlerini desteklemektedir. Client yani müşteri tarafında bulunan sensörler ve aktuatörler doğrudan veya gateway tabanlı bir bağlantıyla Ignite platformuna bağlanabilmektedir. Doğrudan bağlantı IoT-Ignite zengin özellik kümesi için kolay bir arabirim sağlarken, gateway bağlantısı aynı teknolojiyi ve hizmetleri müşteriye en yakın noktada sağlar.

IoT-Ignite platformu, servis sağlayıcıların yeteneklerini zenginleştirir. Ayrıca geniş API'leri, müşteri veri izolasyonu, servisler, uygulamalar ve kenar cihazlar arasındaki güvenlik, müşteri kullanımı için istatistikler ve para kazanma özelliklerini desteklemekle birlikte bulut tabanlı özelliklere kolay erişim sunmayı sağlar.

IoT-Ignite ile aşağıdaki servislerden faydalanabiliriz.

- IoT-Ignite edge (kenar) yönetimi için CO^5 API'lerini destekler,
- Sunucu ve uygulamalar arasında güvenli iletişim sağlar,
- API yönetimi.

IoT’in Avantajları ve Dezavantajları

IoT ile elektronik cihazlar veya makinelerin haberleşmesi ve kontrolü pratik bir şekilde yapılmaktadır. Bu pratikliğin avantajları ve dezavantajları da şüphesiz ki bulunmaktadır. Bunları maddeler halinde sıralayalım.

Avantajları

- Paradan, enerjiden ve zamandan tasarruf etmesi.
- Yaşam kalitesini arttırması.
- Machine-to-Machine ile makineler arasında haberleşmeyi sağlaması. Bu işlemi yaparken de belirtilen kurallara göre makine veya elektronik aygıtların kontrolünün yapılmasını sağlar.
- Kritik durumların kontrolü ve gözetlenmesinin daha denetimli olarak yapılmasını sağlaması.
- Geleceğin en büyük hayalleri arasında bulunan akıllı şehir gibi sistemlerin geliştirilmesini sağlaması.

Dezavantajları

- **Gizlilik/Güvenlik:** Gizlilik ve güvenlik IoT’in en tartışmalı konularından biri olduğu için öncelikle bunu ekledik. Her türlü işlemin takip ve kontrolünün sadece makineler aracılığıyla yapılması beraberinde bazı olumsuzlukları da getirmektedir. Örneğin bu sistemleri kullanarak evinizde bitmek üzere olan bir ürünün siparişini daha siz fark etmeden sistemin sipariş etmesi güzel. Ancak yapılan alışverişlerin şifrelenmesi ve gizlilik gerektiren alışverişlerin diğer kişilerce görülmemesi için güvenlik sistemlerine ihtiyaç vardır.
- **Karmaşa:** IoT ile geliştirilen sistemler karmaşık olduğu için bu gibi sistemlerde istenmeyen bazı çıktılar oluşabilir. Örneğin ihtiyacınız olan bir siparişin hiç verilmemesi veya ihtiyaçtan fazla sipariş verilmesi gibi durumlar.
- **Kontrol:** Diğer bir dezavantaj da teknolojinin bir araç olarak hayatımıza daha da yerleşmesi hatta hayatımızı kontrol etmesi gösterilebilir.

Burada verilen bilgileri baz aldığımızda en büyük avantaj şüphesiz hayatı kolaylaştırması ki teknolojinin temel amacı bu zaten. En büyük dezavantajı da gizlilik ve güvenlik gibi en temel insan ihtiyaçlarının tehlikeye düşmesi gösterilebilir. Böyle bir sistemde evinizin herhangi bir köşesinde kamera bulundurmak ve bunu internete bağlamak ziyadesiyle tehlikeli durumlara yol açabilmektedir.

IoT Uygulamalarının Zorlukları

Bir önceki başlıkta IoT uygulamalarının hayatımıza katkılarından veya zararlarından bahsettik. Bu başlık altında ise bu tür uygulamaları geliştirirken bizleri ne gibi zorluklar bekliyor bundan bahsedeceğiz. Öncelikle IoT bir sistem hem cihaz türü hem de cihaz sayısı olarak geniş bir ekosisteme sahiptir. Çeşitli ve bir o kadar da farklı cihazın beraber bir sistem içinde çalışması kolay bir işlem değildir. Bu tür uygulamaları geliştirirken bizi bekleyen bazı olası zorluklar bulunmaktadır. Bunları bilmek, önümüzü görmek açısından oldukça önemlidir.

Bu tür uygulamaları geliştirirken bizleri bekleyen zorluklar şunlardır:

Heterojen Veri Akışı

IoT bir sistemde veri akışı format, hız ve yapı olarak heterojen olma eğilimindedir. Doğal olarak böyle bir sistemin verileri doğrulayan ve farklı veri tiplerini destekleyen bir yapıda olması gerekiyor. IoT bir sistemin öncelikle heterojen yapıda bulunan bu verileri desteklemesi hatta bunlar

üzerinde analitik işlemler yapabilmesi gerekiyor. Heterojen veri akışı ve farklı veri tipleriyle çalışabilmek için Büyük Veri sistemlerine ihtiyacımız vardır.

Veri Kalitesi

IoT bir sistemde gürültü veya eksik veri gibi durumlar verileri değiştirmekte ve sistem çapında bir belirsizliğe neden olmaktadır. Ayrıca veri kalitesi çevresel koşullara göre değişebilmektedir. Veri akışını etkileyen gürültü veya güvenilir olmayan sensörlerden alınan verilerin kalitesini belirlemek için istatistiksel ve olasılık tabanlı yaklaşımlar oluşturmamız gerekiyor. Bu yaklaşımlar veri akışını bozan gürültü veya sensörlerde gelen verilerin kalitesini ölçmede kullanılır.

Gerçek Zamanlı İşlem

IoT bir sistemde veri akışının hızlı olması ve gerçek zamanlı veri analizi çok önemlidir. Gerçek zamanlı işlem yapmak, gelecek için en önemli zorluklardan biridir. Özellikle 2020 yılında 50 milyardan fazla cihazın internete bağlanacağı hesaba katılırsa bu cihazlardan gelen verilerin analizi oldukça önem arz ediyor. IoT sistemleri, Büyük Veri ekosisteminin bir parçası olan veri akışı platformlarını kullanarak büyük fayda sağlayabilir. Bir diğer çözüm de Cisco'nun Fog Computing adı altında patentini aldığı Edge Computing sistemleri de gerçek zamanlı işlem yapmak için kullanılabilir.

Gizlilik ve Güvenlik

IoT sistemde bulunan veriler, özellikle kişisel verilerin işlenmesi gereken durumlarda sıkı güvenlik önlemlerine ve gizlilik protokollerine ihtiyaç vardır. Kişisel verilerin güvenliği son kullanıcıların bu sisteme olan güvenlerini oluşturmak adına oldukça önemlidir. Bundan dolayı gizlilik ve güvenlik en hassas konular arasındadır. IoT bir sistemde bu gibi verilerin şifreli ve güvenli veri depolama teknikleri kullanılarak korunması gerekmektedir.

Verilerin Doğru Anlaşılması

IoT bir sistemde verilerin heterojen ve büyük miktarda olması sistemin bu verileri yanlış olarak değerlendirmesine neden olabilir. Bu sadece IoT için değil, veri madenciliği için de önemli bir konudur. Herhangi operasyonel bir eyleme geçmeden önce buradaki verilerin doğru şekilde anlaşılması için test işlemlerinin kapsamlı olarak yapılması gerekmektedir. Verilerin doğru anlaşılması için klasik veri madenciliği teknikleri kullanılabilir.

IoT'nin Kullanım Alanları ve Etkileri

IoT'nin kullanım alanları oldukça geniş ve geleceği büyük çapta etkileyeceği de bariz bir gerçektir. Çünkü teknolojinin kullanımı ile birlikte hem hayat kalitesi artmakta hem de para ve enerjiden büyük oranda tasarruf edilmektedir. Bir de teknolojik cihazların haberleşmesi ve cihazların yapay zeka gibi bir teknoloji ile desteklenmesi hayatı daha da yaşanır kılacaktır.

IoT'nin etki alanı, evindeki enerji kullanımını azaltmak isteyenlerden işlemlerini hızlandırmak isteyen büyük organizasyonlara kadar geniş bir yelpazeye yayılmaktadır. Böyle bir teknoloji hem bireyler, hem şirketler hem de devletlerin işlemlerin yapmasını sağlamak için büyük kolaylıklar sağlamaktadır.

Burada IoT'in kullanım alanlarını başlıklar halinde inceleyelim.

Anlık Gözlemleme

Anlık gözlemleme alanında IoT uygulamaları iki kategoride ele alınabilir.

- **Güvenlik**: Devletler anlık gözlemleme sistemlerini kullanarak şehir içinde meydana gelmesi muhtemel olayları anında önleyebilir veya meydana gelen bir suçun temel unsurlarını bu sistem ile izleyebilir. Böyle bir sistemde birçok suç meydana gelmeden önlenebilir. Hatta yapay zeka gibi bir sistemle bazı anormal davranışlar analiz edilerek hırsızlık veya gasp vakaları önlenebilir.
- **Üretim**: İnsan nüfusundaki patlama ve beraberinde artan ihtiyaçlar, fabrikaların daha fazla üretim yapmasına neden olmaktadır. Yıl boyunca çalışan bir fabrikada üretilen ürünlerin piyasaya sürülmeden önce hatalı olup olmadığını kontrol etmek için anlık gözlemleme sistemleri kullanılabilir. Özellikle en temel ihtiyaçlardan olan gıda ve giyim sektöründe bu gibi hataların önüne geçmek firmaların hem nakliye masraflarını azaltır hem de müşteri memnuniyetini korumayı sağlar.

Pazarlama ve Reklam

Pazarlama ve reklam alanında IoT özellikle, şirket veya kuruluşların müşteri ihtiyaçlarını veya tercihlerini daha iyi analiz etmesini ve bunlara yanıt vermesini kapsar. Şu anda bu alanda IoT uygulamalarının olduğunu görmekteyiz.

Reklamda tek boyutlu bir uyum stratejisi yerine kişiselleştirilmiş reklamların sunulması daha etkilidir. Çünkü ihtiyacımız olmayan veya ilgimizi çekmeyen bir ürünün reklamı ne kadar kaliteli olursa olsun bizi o ürünü almaya itmeyecektir. O yüzden genel olarak yapılan reklamların etkisi oldukça zayıftır. Ancak web sitelerinde kullanıcının gezdiği ürünlerin muadillerini kullanıcıya sunmak alışverişi tetikleyen en önemli unsurlardan bir tanesidir.

Akıllı Ev Sistemleri

Akıllı ev sistemleri reklamcılıkta olduğu gibi kişiselleştirilmiş uygulamalar geliştirmeyi sağlamak için kullanılır. Tabi bu sadece evler için değil iş yerimizdeki ofisler için de geçerlidir. Böyle bir sistemde evimizi veya ofisimizi kendimize göre özelleştirebilir ve ihtiyaçlarımız için düzenleyebiliriz. Bunun en büyük avantajı evimizdeki ve ofisimizdeki konforu ideal bir seviyeye çıkarmasıdır. Bu şekilde memnuniyet, verimlilik, sağlık ve güvenliğimizi geliştirebiliriz.

Ofis alanını kendimize göre özelleştirmek yaptığımız işin kalitesini arttıracaktır. Aynı durum evimiz için de geçerlidir. Örneğin, ofis veya evimize girdiğimizde aydınlatma ve sıcaklığı kendi tercihimize göre ayarlamak, bazı cihazların otomatik olarak başlatılmasını sağlamak bu alan için örnek verilebilir.

İmalat Uygulamaları

İmalat veya üretim en zorlu ve bir o kadar da en riskli sektörlerin başında gelmektedir. Çünkü üretilen bir malın satışının ne olacağı tam bir muamma. Gerçi yapılan bazı piyasa araştırmaları ile küçük de olsa bir tahminde bulunabiliriz. Ancak dediğimiz gibi imalat en riskli analizlerin başında gelmektedir.

IoT uygulamaları ile ürünlerin piyasaya çıkmasında, öncesinde veya sonrasında yapılan geleneksel pazar araştırmaları için ihtiyaç duyulan zaman ve kaynağın önemli ölçüde azalması sağlanır. Ayrıca risk tahmini ile ilgili daha güvenilir ve daha ayrıntılı bilgiler bulunduğu için yeni bir ürünün satışında meydana gelecek herhangi bir risk üretimden önce kolaylıkla tespit edilebilir. Diğer bir faydası da uygun olmayan ürün dağıtımını kontrol etmek ve ürün iadelerini önlemeyi sağlamaktır. Çünkü üretim sırasında meydana gelen veya gelebilecek hataların önceden fark edilmesi; IoT bir sistemde anında tespit edilebilir ve bu şekilde üretilen bir ürün dağıtıma çıkarılmadan geri dönüşüme alınabilir.

Enerji Uygulamaları

Teknolojinin yükselişi, beraberinde enerji maliyetlerini de arttırdı. Bundan dolayı tüketiciler enerji tüketiminin azaltılması veya kontrolü için bazı çözümler talep etmektedir. IoT ile hem aygıt düzeyinde hem de bir evin tüm sistemi boyunca enerji kullanımı analizi yapılabilir. Bu analizle birlikte gerekli optimizasyon işlemlerini de sağlamaktadır. Örneğin, ihtiyaç duyulmayan lambaların otomatik olarak kapatılmasını sağlamak buna en güzel örnektir. Böyle bir sistemde IoT ile odada veya ofiste herhangi bir hareketlilik algılanmadığı zamanlarda enerji tüketimine yol açan tüm cihazların kapatılması sağlanabilir.

IoT ile gelen bir diğer yenilik de, eski cihazlar veya hatalı sistem parçalarından kaynaklanan sorunlu tüketimler rahatlıkla tespit edilebilmesidir. Geleneksel yöntemlerde bu gibi hataları tespit etmek oldukça maliyetli ve ciddi zaman kaybına neden olmaktadır.

Enerji uygulamaları için ev ve ofisleri örnek verdik ama şüphesiz enerji konusunda küçük, orta ve büyük ölçekli işletmeler oldukça muzdariptir. Meydana gelen ve istenmeyen enerji atıkları işletmeleri maliyet konusunda oldukça zorlamaktadır. Büyük işletmeler enerji konusunda nispeten daha profesyonel davranabiliyorlar. Çünkü enerji tasarrufu için ihtiyaç duydukları sistemleri rahatlıkla temin edebiliyorlar. Ancak aynı durum özellikle küçük ölçekli işletmelerde mümkün olamıyor. Enerji tasarruf sistemleri küçük işletmeler için oldukça maliyetli olduğu için bu gibi sistemleri işletmelerinde kuramıyorlar. IoT özellikle küçük ve orta ölçekli işletmelerin kanayan bu yarasına ilaç olmaktadır.

Tabii IoT'in enerji alanındaki faydaları bunlarla sınırlı değil. IoT ile sunulan enerji analizi beraberinde sistemin daha güvenli olmasını sağlamaktadır. Enerjinin en büyük dezavantajı içinde bulunduğu sistemi yok etme ihtimalidir. Enerjideki aşırı yüklenmeler bazı durumlarda yangın gibi büyük felaketlere neden olmaktadır. IoT, tüketimin ötesinde, sistemin aşırı yüklenmesini veya tıkanmasını da engeller. Ayrıca, sistem performansına ve kararlılığa, kesinti, hasar gören ekipman ve yaralanmalar gibi kayıplara karşı da ciddi oranda korum sağlar.

Sağlık Uygulamaları

IoT ile birlikte tıbbi araştırmalar, bakım ve acil bakım gibi işlemler önemli ölçüde daha etkili olarak gerçekleştirilebilir. Hasta bakımı, hasta takibi, cihaz verilerinin toplanması ve daha birçok işlemin bir ağ çatısı altında bir araya toplanması tıbbi işlemlerin yükünü azaltmakla birlikte ayrıntıya daha fazla hassasiyet ve olaylara daha hızlı tepkiler verilmesini de sağlar.

Örneğin, acil bakıma gelen bir hastanın girişi yapıldıktan sonra hastayla ilgili tüm tıbbi kayıtların çekilmesi ve bu kayıtların doktora veya hemşireye gösterilmesi tedavi işlemlerinin daha hızlı ve zararsız bir şekilde yapılması sağlanır. Ayrıca yoğun bakımda bulunan bir hastanın anlık yaşam fonksiyonlarını cihazlardan alan ve alınan verileri analiz edip herhangi kritik bir süreçte bunu doktor veya hemşireye ileten bir sistem birçok hastanın hayatının kurtarılmasını önemli ölçüde etkiler.

Gerçek Uygulama: Majör depresif veya klinik depresyon hastalığı ile mücadele etmek adına Takeda Pharmaceuticals ve Cognition Kit Limited firmaları Apple Watch uygulaması geliştirmektedir. Bu uygulama bilişsel işlevleri izlemek ve değerlendirmek için verileri aktif ve pasif olarak almayı sağlayan akıllı saat uygulamasıdır. Böyle bir uygulama dünyada yaklaşık 350 milyon insanın depresyon hastalığına yakalandığını düşünürsek gerçekten faydalı bir uygulamadır. Bu gibi hastaların hem tedavi sırasında hem de tedavi dışında takibi ve anormal bir durumun tespiti gibi işlemlerede akıllı saat uygulaması hayat kurtarmayı hedeflemektedir.

Taşımacılık Uygulamaları

Taşımacılık uygulamaları kişisel araçlar, ticari araçlar, trenler ve ticari taşımacılığı kapsayan genel bir ifadedir. Burada trafik kontrolü, park etme, yakıt tüketimi ve daha birçok ulaşım unsuru hakkında insanlara yardımcı olan sistemler tasarlanması hedeflenmektedir.

Taşımacılıkta, tıkanıklığı yönetmek, kazaları azaltmak, park işlemini yapmak ve yakıt tüketimini azaltmak temel hedefler arasındadır. IoT ile tüm trafik gözlem noktalarında bulunan cihazlar vasıtasıyla trafik akışının izlenmesi ve analiz edilmesi sağlanır.

Burada verilecek en iyi uygulamalardan biri akıllı trafik işaretleridir. Akıllı trafik işaretlerinin geliştirilmesi sürücülerin daha iyi bilgilendirilmesini sağlar. Bu şekilde trafiğin tıkanması ve kazaların önüne geçilmesi sağlanmış olur. Yine akıllı trafik lambalarının geliştirilmesi ile birlikte araç sayısının fazla olduğu yollarda trafiğin rahatlatılması sağlanabilir. Geleneksel trafik lambalarında her lambanın açık kalma süresi önceden belirlenmiştir. Ancak IoT sistemli trafik lambaları trafiğin yoğunluğuna göre bu süre üzerinde otomatik olarak değişiklik yapabilmektedir.

Gerçek Uygulama: Dünyanın en uzun süren trafik sıkışıklığı 2010 yılında Çin'de meydana geldi. Yaklaşık 100 km boyunca meydana gelen bu sıkışıklık 12 günde ancak aşılabildi. Benzer bir durum 2015 yılında yine Çin'de gerçekleşti. 50 şeritlik yolun 20 şeride düşmesi ve bu durumun tatil dönüşüne denk gelmesi tekrar trafiğin kilitlenmesine neden oldu. Çin'in en büyük problemlerinden olan trafik sıkışıklığından bir görüntü:

Çin, yukarıdaki görüntüden kurtulmak adına Büyük Veri ve Yapay Zeka teknolojilerini kullanan bir IoT sistemi geliştirdi. Bu sistem belirli günlerde veya belirli saatlerde trafiği yönlendirmekte, bazı yolları kapatıp sürücüleri belirli yollara yönlendirmekte, trafik ışık sürelerini değiştirerek trafiğin tıkanmasını engellemektedir. Geliştirilen bu sistem sayesinde Çin'de aşırı derece araba olmasına rağmen trafiğin rahat bir şekilde akması sağlanmaktadır.

Eğitim Sistemi Uygulamaları

Eğitimin en iyi şekilde ve verimli olarak yapılabilmesi için eğitmen veya öğretmenin olabildiğince işine odaklanması gerekir. Bir okulda öğretmenin işi sadece eğitim vermekle sınırlı değil. Eğitimle

birlikte büro, yönetim ve idari işleri de bulunmaktadır. Bu gibi işlerin olabildiğince eğiticiden uzak olması eğitim için bir ihtiyaçtır. IoT tabanlı eğitim uygulamaları, eğitmen veya öğretmenlerin daha iyi bir eğitim vermesini sağlar. Özellikle yönetim ve idari işlerdeki görevlerinden kurtardığı için eğitmenlerin eğitime daha iyi odaklanması sağlanır. Bu gibi sistemlerde örneğin öğrencinin derste olup olmadığını kontrol etmek gibi işlemler IoT ile sağlandığı için eğitmenin büro işleriyle uğraşarak eğitimden kopması önlenmiş olur.

IoT, öğretmenlere güçlü eğitim araçlarına kolay erişim olanağı sağlar. Her öğrenci için en iyi yöntemi belirleyerek öğrenci seviyesine ve ihtiyaçlarına göre materyal oluşturulması hedeflenir. Hatta onların seviyesine göre sınav sorularının hazırlanması gibi uç işlemler de yapılabilir.

Eğitimin etkili olmasında eğitimcilerin eğitime odaklanmasıyla birlikte verilen eğitimin birey tabanlı olması da etkilidir. IoT ile birlikte, öğrencilerin ihtiyaç duydukları bilgilere erişimi sağlanır ve eğitimin birey tabanlı olması gerçekleştirilir. Böylece her öğrenci eğitime aktif olarak katılabilir. Bu şekilde eğitimin kalitesi de artmış olur.

Gerçek Uygulama: California, Richmond'daki bir okulda öğrencilerin varlığını izlemek için RFID teknolojisi kullanılmaktadır. RFID çipleri kimlik kartlarına yerleştirilerek öğrencilerin sınıfta hatta okul sınırları içinde olup olmadığı kolaylıkla tespit edilmektedir.

Kamu Uygulamaları

Kamu uygulamaları olarak şehir planlaması, yönetimi ve güvenlik ile ilgili uygulamaları örnek olarak verebiliriz. IoT ile bir şehrin planlama ve yönetimi ile ilgili karmaşık yönlerinin pratik bir şekilde analiz edilmesi sağlanır. Örneğin şehir planlamasını etkileyen nüfus artışı, imar, haritalama, su temini, ulaşım, gıda talebi, sosyal hizmetler ve arazi kullanımı gibi çeşitli faktörlerin incelenmesini olabildiğince basitleştirir. Bu alanlarda elde edilen bilgileri toplar ve yeni nüfusa göre şehrin yaşanabilir hale getirilmesi için gereken bilgileri bizlere sunmayı sağlar.

IoT ile özellikle devlet eliyle yavaş bir şekilde yapılan kamu faaliyetlerinin daha hızlı ve doğru bir şekilde yapılması sağlanır. Yavaş yapılan işlemlerin çoğunda önceden öngörülemeyen faktörler oldukça etkilidir. IoT ile, öngörülemeyen faktörler daha gerçekleşmeden tespit edilerek bunlara çözüm sunulması sağlanır.

Diğer bir kamu uygulaması da şüphesiz güvenliğin sağlanmasıdır. Ulusal tehditler çok çeşitli ve karmaşıktır. IoT ile silahlı kuvvetlerin sistem ve servisleri daha da güçlendirilir. Zengin denetim ve gözlem için ucuz ve performanslı cihazlarla sınırların daha iyi korunmasını sağlar. IoT ile genellikle birkaç birime ve sayısız kişiye yayılmış koruma görevleri otomatik olarak gerçekleştirilir. Bu durum beraberinde hem hızı hem de doğruluğu getirmektedir.

Gerçek Uygulama: Bu kategoride geliştirilen uygulama şüphesiz New York'ta bulunan akıllı çöp kutularıdır. Bunların temel çalışma prensibi çöp toplayıcılarına çöpün toplanması gerektiğini yani dolu olduğunu haber vermesidir. Çöp servisi bu şekilde sadece gerekli görülen yerlere gitmekte ve yakıt tasarrufu sağlamaktadır.

Tüketici Uygulamaları

Tüketiciler IoT'nin veri analizinden kişisel ve mesleki olarak yararlanmayı sağlar. IoT ile geliştirilen bir tüketici uygulamasının temel amacı tüketicinin yaptığı iş ve işlemleri takip etmeyi sağlamak ve bu işlemlere göre kullanıcının yaşam kalitesini, güvenliğini ve bilgilendirmesini sağlamaktır.

Örneğin IoT ile geliştirilen bir tüketici uygulaması evinizi size göre hazırlayabilir. Evin sıcaklık ve nem gibi çevresel koşullarını sizin için yeniden düzenleyebilir. Acil durumlarda uygun kişilere alarm vererek koruyucu önlemler alabilir. Evinizde veya ofisinizde meydan gelen bir arızayı tespit edebilir ve bunu size iletebilir. Hatta tamirciye mesaj (SMS) bile atabilir.

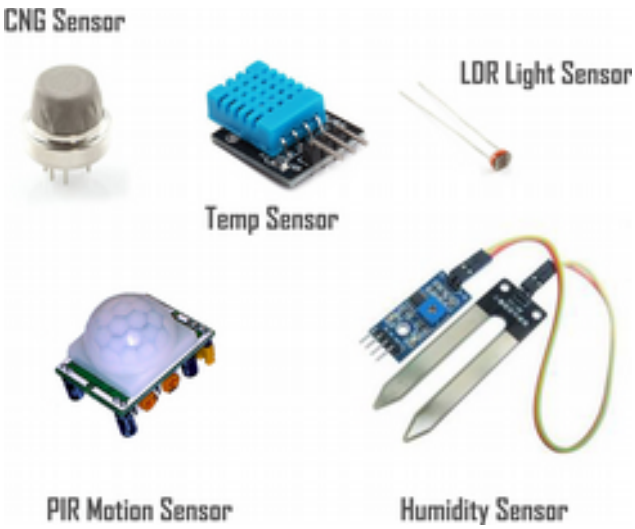
IoT Donanımları ve Open Hardware Kavramı

IoT tabanlı bir sistemde temel amacımız veri toplama, iletişim ve uzaktan kontrol olduğu için bu işlemleri yapabilen donanımlara ihtiyaç vardır. Çünkü veri toplamadan herhangi bir analiz işlemini yapamayız ya da iletişim modülleri olmadan uzakta bulunan başka bir cihazı kontrol edemeyiz. Neler olup bittiğini belirlemek ve bu bilgileri iletmek için geliştirdiğiniz sistemin bu tür işlemleri yapabilen donanımlara ihtiyacı vardır.

Bu gibi amaçları yapmak için ihtiyacımız olan donanımları şu şekilde sıralayabiliriz.

- Sensörler
- İşlemciler
- İletişim Modülleri
- Geliştirme Kartları
- Sunucular

Donanımlar bunlarla sınırlı değil. Zaten sensörler tek başına geniş bir yelpazede donanım desteğini tanımlamaktadır. Sensörlere örnek olarak; konum tespiti için GPS, hareket tespiti veya hareketi ayırt etmek için ivmeölçer, sıcaklığı ölçmek için sıcaklık sensörlerini verebiliriz. IoT sistemlerde kullanılan sensörlerden bazıları aşağıdaki gibidir.

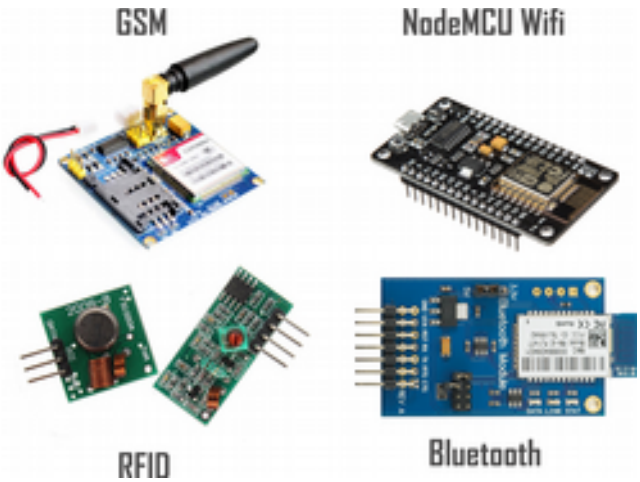


IoT sistemlerde sensörler ile çevreden veriler alınır, alınan verilerin miktarı olukça fazla olduğu için bunların analizinin yapılması gerekir. Analiz işlemi ne kadar hızlı yapılırsa o oranda gerçek zamanlı eylem de bulunulması sağlanmış olur. Bunun için güçlü işlemcilere ihtiyacımız var. Büyük Veri sistemleri bu güçlü işlemciler aracılığıyla veri analizini çok kısa bir zaman için gerçekleştirebilir.

Sunucular derken aslında Cloud, yani bulut bilişimi kastediyoruz. Çünkü IoT bir sistemde veri bütünlüğünü veya doğruluğunu sağlamak, farklı cihazlar arasında veri trafiğini sağlamak için bulut sistemlerine ihtiyaç vardır. Bulut sistemleri sunuculardan meydana gelen ve güçlü işlemcilere sahip olan bilgisayar sistemleridir.

Sunucuların merkezde olduğu bulut sistemler ile IoT içinde bulunan diğer cihazlar arasında veri transferi yapmamız gerekiyor. Bunun için iletişim modüllerine ihtiyacımız var. İletişim modülleri ile yakın veya uzakta bulunan aygıtların kontrolü sağlanır.

Bazı iletişim modelleri şunlardır:



Açık donanım (open hardware) veya açık kaynak donanım (open source hardware) kavramına gelirsek, söz konusu donanımın herkes tarafından incelenmesi, modifiye edilmesi, oluşturulması ve dağıtılması için lisanslanan bir donanımın tasarım özelliklerinin herkese açık olmasını ifade eder.

Open software kavramı yazılım için kullanılırken, open hardware kavramı fiziksel aygıtlar için kullanılır. Geliştirilen bir yazılım açık kaynak ise kaynak kodları herkes tarafından görülebilir ve üzerinde istenen değişiklikler kolaylıkla yapılabilir. Aynı durum açık kaynak donanımlar için de geçerlidir.

Açık kaynak donanımlar için herkese açık olan temel bileşenler şunlardır:

- Donanımın kaynak kodu,
- Tasarım özellikleri,
- Bilgisayar destekli çizim dosyaları (CAD),
- Şemalar,
- Mantık tasarımları

IoT sistemlerde kullanabileceğimiz açık kaynak donanımlardan bazıları şunlardır.

- Arduino
- Beagle Board
- Flutter Wireless
- NodeMCU
-

Şimdi bu donanımlar hakkında kısaca bilgi verelim.

Arduino

Kullanımı kolay, yazılımı dayanan açık kaynak kodlu bir prototipleme platformudur. Arduino, düşük güç tüketimini sağlayan bir mikro işlemciye sahip olup kullanıcıya donanım üzerinde tam kontrol sağlayan açık kaynak bir geliştirme kartıdır. Arduino ile ister IoT ister DIY (Do-It-Yourself, Kendin Yap) gibi projeler geliştirebilirsiniz. Arduino kartınızı programlamak için Arduino IDE yazılımını kullanmanız gerekiyor. Bu yazılım Processing tabanlı bir yazılımdır. Kartı programlamak için kullandığımız Arduino programlama dili is Wiring tabanlıdır.

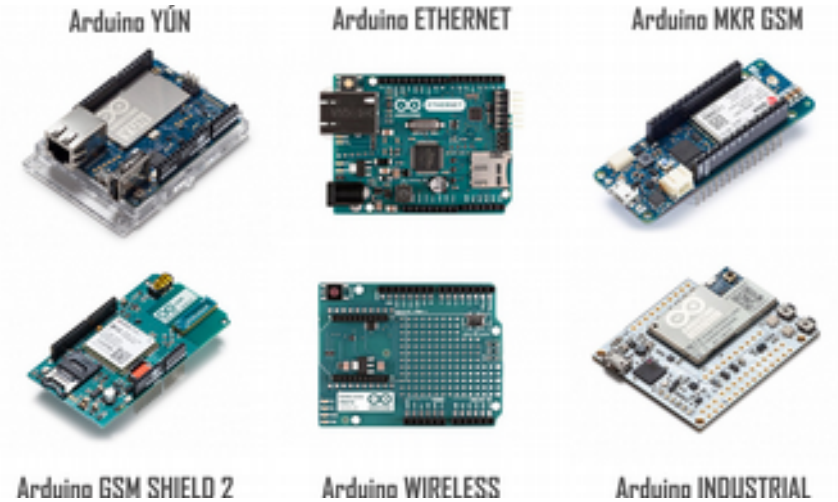
Arduino, basit elektronik projelerden bilimsel projelere kadar binlerce çalışmada kullanılmıştır. Programlamanın basit ve kolay olmasından dolayı öğrenciler, amatörler, programcılar, öğretmenler ve daha birçok kesim tarafından tercih edilmektedir.

Arduino kartlarının birçok çeşidi vardır. Genel olarak kartların tümünde benzer bileşenler olmasına rağmen, kullanılan mikrodenetleyici modeli, giriş/çıkış pinleri, çalışma gerilimleri gibi bazı küçük farklılıklar bulunmaktadır.

Arduino'nun resmi web sitesindeki bilgilere göre geliştirilen her model belirli bir amaca hizmet etmektedir. Bundan dolayı modellerin kategorilere bölündüğünü görmekteyiz. Arduino'nun resmi web sitesinde geliştirilen ürünlerin tam listesini aşağıdaki resimde inceleyebilirsiniz.

https://www.arduino.cc/en/Main/Products

Arduino kartlarından bizi ilgilendiren kategori Internet of Things kategorisidir. Burada bulunan modeller nesnelerin interneti için geliştirilen modellerdir. Aşağıda bu modellerden bazılarının görselini sizlerle paylaştık.



BeagleBoard

BeagleBoard, Arduino'nun aksine kredi kartı büyüklüğünde bir bilgisayardır. 2018 Ocak itibarıyla en son versiyonu BeagleBorad-X15'tir. İşletim sistemi olarak Linux (Arduino, Debian ve Ubuntu) kullanmaktadır. Arduino'ya kıyasla daha fazla işlem yapma kapasitesine sahiptir. Bu kartın üzerinde düşük güç tüketimini sağlayan bir işlemci ve 157 adet GPIO pini bulunmaktadır. Ayrıca USB3.0, SATA, Kamera, SD kart, HDMI, Ethernet ve daha birçok port desteği ile gelmektedir. Dahili Wifi modülüne sahip olduğu IoT uygulamalarında kullanılabilir. Mini bir bilgisayar olduğu için Arduion'dan ziyade Raspberry Pi3'e benzemektedir.

BeagleBorad-X15'in örnek bir görüntüsü aşağıdaki gibidir.

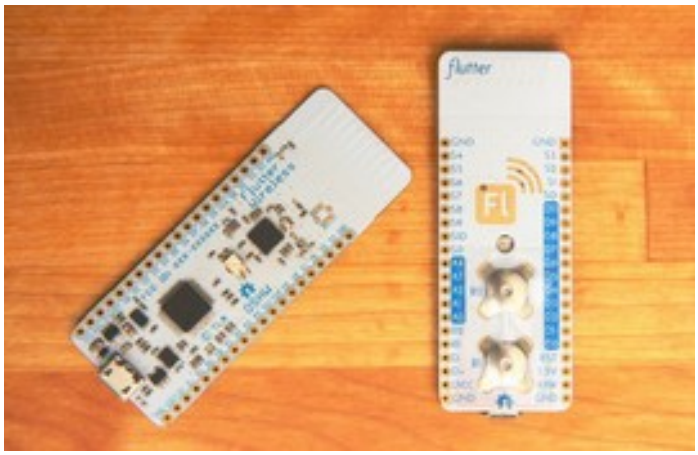
Flutter Wireless

Arduino tabanlı olarak tasarlanan ve temel amacı, diğer cihazlar arasında kablosuz bağlantı oluşturmaktır. Bu işlemi yapmak için ağ protokollerine sahiptir. Flutter'ı önemli kılan en önemli özelliği ARM tabanlı olması ve dahili bir şifreleme yöntemine sahip olmasıdır. Özellikle robotik projeler, sensörler arasında kablosuz bağlantı kurulması gereken durumlar ve tüketici elektroniği için kullanılabilir.

Bu kartın genel özellikleri aşağıdaki gibidir:

- Güçlü ARM işlemcisine sahip olması.
- Uzun menzilli ve güçlü bir kablosuz iletişim sağlaması.
- Dahili pil şarjı desteğinin bulunması.

- Dahili bir güvenlik yongasına sahip olması.



NodeMCU

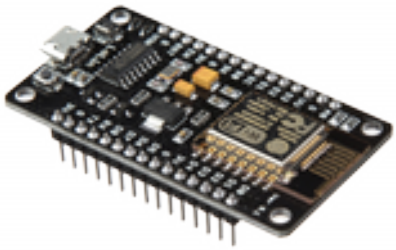
NodeMCU madeni paradan biraz büyük, açık kaynaklı, interaktif, programlanabilir, maliyeti düşük bir Wifi modülüdür. Kablosuz iletişimi sağladığı için IoT projeleri geliştirmek için kullanılır. Arduino IDE ile programlanır. Programlama dili olarak Arduino dilini veya kendine özgü geliştirilen Luo script dilini kullanabilirsiniz.

NodeMCU ile şu işlemleri gerçekleştirebiliriz:

- Kablosuz ağlara bağlanabilir,
- Arduino benzeri IO erişimi sağlayabilir,
- HTTP istemci/sunucu oluşturabilir,
- PWM tabanlı işlemler oluşturabilir,

Bunlarla beraber Bootstrap'ı kullanarak NodeMCU modülünü kontrol edebilen web arayüzleri tasarlayabilirsiniz.

NodeMCU'nin örnek bir görüntüsü aşağıdaki gibidir.



IoT Geliştirici Donanım Kit'leri

SDK'ler (Software Development Kit – Yazılım Geliştirme Kiti) yazılım geliştirmek için ihtiyacımız olan bileşenleri sağlayan yazılımlardır. Bu yazılımları bilgisayarınıza indirerek istediğiniz bir programlama dili için (Örneğin Java) yazılım geliştirebilirsiniz. SDK'ler geliştireceğimiz yazılım için sınıf, metot veya paketleri sağlar. Bunları kullanarak yazılım geliştirebiliriz. Benzer durum donanım geliştirme için de geçerlidir. Donanım tabanlı bir proje geliştirirken bazı geliştirme kitlerine ihtiyacımız var.

IoT tabanlı sistem veya sistemleri geliştirmek için geliştirici donanım kitlerini kullanırız. Geliştirici kitlerin sayısı yukarıda değindiğimiz açık kaynak donanımlardan dolayı oldukça fazladır. Açık kaynak lisansı altında geliştirilen bu donanımları kullanarak IoT sistemleri geliştirebiliriz.

Burada örnek verebileceğimiz donanım kitlerinden bazıları şunlardır.

- Raspberry Pi
- Arduino
- NodeMCU
- NXP
- Intel Edison

Arduino ve NodeMCU hakkında bilgi vermiştik. Burada diğer geliştirme kartları hakkında kısa bir bilgilendirme yapalım.

Raspberry Pi

Raspberry, düşük maliyetli kredi kartı büyüklüğünde olan bir bilgisayardır. Herhangi bir bilgisayar monitörü veya TV'ye bağlanabilir ve üzerindeki USB bağlantıları ile fare ve klavye gibi çevre birimlerini kullanabilir. Her yaştan insanın bilgisayar kullanmasını sağlayan küçük ve yetenekli bir cihazdır. Scratch ve Python gibi dilleri kullanarak programlama yapabilirsiniz. Mini bir bilgisayar olmasına rağmen standart bir masaüstü bilgisayarın yapması gereken her türlü işlemi rahatlıkla yapabilir. Örneğin; internette arama yapmak, yüksek çözünürlüklü video izlemek, paket programları kullanmak ve hatta oyun oynayabilirsiniz.

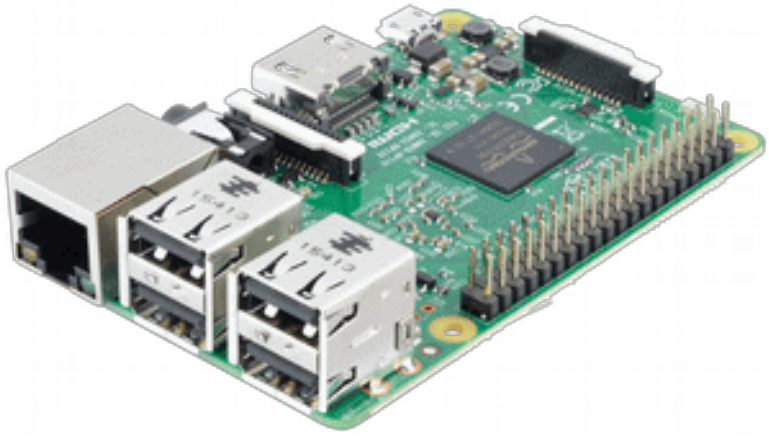
Raspberry Pi üzerinde bulunan genişleme pinleri (GPIO) sayesinde dış dünya ile iletişime geçebilir. Bu özelliğinden dolayı birçok projede kullanılmıştır. Temel geliştirilme amacı dünyanın her yerindeki çocukların bilgisayarların nasıl işlediğini öğrenmesini sağlamak ve programcılığa ilgilerini çekmektir.

Raspberry Pi 3, en son üretilen Raspberry versiyonudur. Raspberry'nin resmi işletim sistemi Raspian olmakla birlikte aşağıda verilen işletim sistemlerini de desteklemektedir.

- Ubuntu Mate
- Snappy Ubuntu Core
- Windows 10 IOT Core
- OSMC
- Libreleec
- Pinet
- Risc OS

Bunlara ek olarak Google firmasının geliştirdiği Android Things işletim sistemini de verebiliriz. Android tabanlı olan bu işletim sisteminin amacı IoT projeleri geliştirilmesini sağlamaktır. Görüleceği üzere Raspberry Pi gelecek vadeden bir projedir.

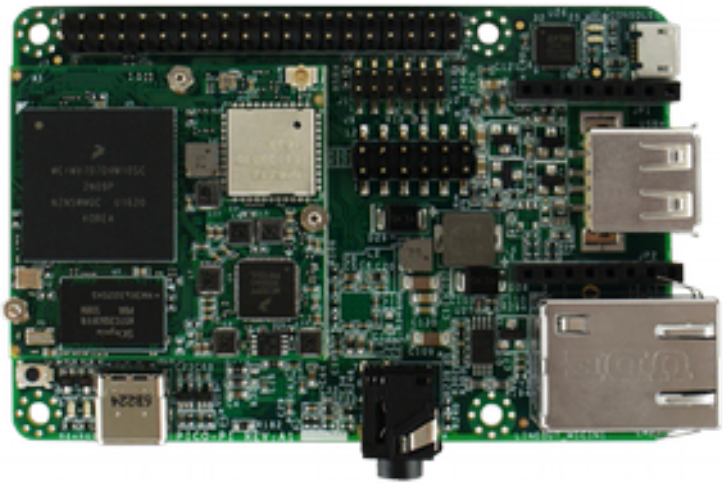
Raspberry Pi3'ün örnek bir ekran görüntüsü aşağıdaki gibidir.



NXP

Android Things işletim sisteminin desteklediği geliştirme kartlarından bir diğeri NXP'dir. Düşük güç tüketimi ve yüksek performans gerektiren IoT uygulamalarına hitap eden bir geliştirme kartıdır. Üzerinde ARM Cortex A7 işlemcisi, 512 MB RAM ve dahili Wifi modülü bulunan bu kart 4GB'lik dahili bir hafızaya sahiptir. Ayrıca USB 2.0 ve Ethernet port'larına da sahiptir. IoT uygulamaları geliştirmek için genişleme pin'lerini kullanan bu kart IoT tasarımları oluşturmak için ileri bir teknoloji sunmaktadır. Google tarafından Android Things için kullanılması bu kartın yaygınlaşmasında önemli bir adımdır.

NXP i.MX7D modelinin örnek bir görüntüsü aşağıdaki gibidir.



Android Things işletim sisteminin desteklediği diğer NXP modelleri şunlardır:

- NXP Pico i.MX6UL
- NXP Argon i.MX6UL

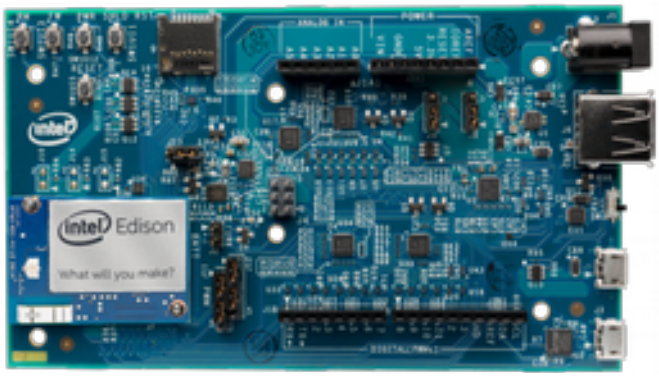
Intel Edison

Intel Edison kartı isminden anlaşılacağı üzere Intel tarafından geliştirilmiştir. Bu modül, güçlü hesaplama performansının yanı sıra Wifi ve Bluetooth bağlantılarına sahip bir IoT platformu sağlar.

Android Things ilk çıktığı zaman bu karta destek verirken kısa bir süre sonra desteğini çekti. Bu kartın diğer özellikleri şunlardır:

- 500MHz Intel Atom işlemcisi
- 1GB RAM
- 4GB Dahili Hafıza
- Arduino uyumlu
- Bluetooth Low Enegry desteği
- Cihazlar arası ya da cihaz ve bulut arasında veri transferini sağlaması gibi birçok özelliği bulunmaktadır.

Intel Edison'un örnek bir görüntüsü aşağıdaki gibidir.



Yazılım Teknolojisi ve Protokoller

IoT sistemleri genel amaçlı ve birçok farklı platforma hitap edeceği için burada kullanılan yazılım veya yazılımlar; gömülü sistemler, ortak sistemler ile ağ ve eylem gerektiren temel alanlara hitap etmelidir. IoT yeni bir teknoloji değil teknolojinin nasıl kullanılacağını gösteren bir yaklaşımdan ibarettir. Bu yaklaşımda birçok farklı cihaz bir araya gelerek ortak bir amaç için beraber çalışmaktadır. Bundan dolayı bu yaklaşımın yazılım yelpazesi de oldukça geniştir. Çünkü IoT ile geliştirilen bir sistemde veri toplama, cihaz entegrasyonu, gerçek zamanlı analitik işlemler ve iletişim gibi eylemler yapılacağı için her eylem için hem farklı donanımlar hem de farklı yazılım dilleri kullanmak gerekiyor.

IoT bir sistemde bulunması gereken yazılımlar kategori olarak aşağıdaki gibi olmalıdır.

Bu kategoriler hakkında kısaca bilgi verelim.

Middleware

İşletim sistemi ve bu işletim sistemi üzerinde çalıştırılan uygulamalar arasında yer alan yazılımlara Middleware, yani arayazılım denir. Ara yazılımlar kullanıcıya görünmeyen gizli bir çeviri katmanına sahip olup temel amacı dağıtılmış uygulamalar için veri iletişimini ve yönetimini sağlamaktır. Özellikle kullanıcı arayüzüyle veritabanı ve bilgi işlem sistemleri arasında veri trafiğinin hızlı bir şekilde yapılmasını sağlamak için geliştirilmiştir. Ara yazılımların IoT'de temel

kullanılma amacı eylemlerin gerçek zamanlı yapılmasını sağlamasıdır. Gerçek zamanlı işlem yapmasını da veri dağıtım vizyonuna borçludur. Çünkü temel amacı karşılıklı veri alışverişinden ziyade veri trafiğini gerçek zamanlı gerçekleştirmektir. Diğer bir faydası da dijital dünyada öngörü sistemleri için var olan sistemlerle mobil uygulamalar gibi etkileşimli sistemler arasında bağlantıyı sağlayarak kurumsal bir bütünlük ve katman sağlamaktır.

Arayazılımları kullanabileceğimiz bazı uygulamalar aşağıdaki gibidir:

- İstemci uygulamasını kullanan bir kişinin yetki sahibi olup olmadığını kontrol etmek.
- Herhangi bir sorun veya problem meydana geldiği zaman veritabanının bozulmamasını sağlamak.
- Mesajlardan meydana gelen bir kuyruğu yönetmede kullanılır. Özellikle aradaki bağlantı gevşek olan durumlarda mesaja bağlı işlem veya eylem gerçekleştirmeyi sağlar.
- Web tarayıcılarından gelen isteklere cevap vermeyi sağlayan bu uygulamalarda istek sonuçlarını tarayıcıya göndermek için kullanılır.
- İstemci programlarının dağıtık bir sistemde bulunan diğer hizmet veya sunucuları bulmasını sağlamak için kullanılır.

Veritabanı

IoT'de başarının elde edilebilmesi uygulamanın sahip olduğu ve ürettiği verilere dayalıdır. Bununla birlikte IoT uygulamaları için bir veritabanını incelemek, geleneksel veritabanlarından oldukça farklı kriterler gerektirir.

IoT veritabanı yönetim sistemlerini bazı zorluklar getirmektedir. Bu zorlukları şu şekilde sıralayabiliriz.

Geleneksel veritabanlarında ihtiyaç duyulduğu zaman veri alımı sağlanır. Ancak IoT sistemlerde bu işlem gerçek zamanlı ve sürekli yapılır. Gerçek zamanlı büyük miktarda veri alımı IoT ile gelen en büyük zorluklardan biridir.

Geleneksel yöntemlerde işlemler veya eylemler belirli aralıklarla yapılır. IoT sistemde olaylar sabit aralıklarla değilde akış devam ettikçe gerçekleşir.

IoT bir sistemde, kurumsal uygulamalarda karşılaşına verilerden daha fazla veri ile çalışmak gerekir.

Bu zorluklardan dolayı klasik veritabanları IoT için uygun değildir. Bunun yerine Büyük Veri sistemlerini kullanmak gerekiyor.

Veri Çözümleme (Analytics)

Veri çözümleme kavramı verilerin analiz işlemlerinin yapıldığı ve bu gibi işlemleri yapan yazılımların bulunduğu bir kategoridir. Tabi burada sadece veri analizi değil veri toplama ve veri dağıtımı gibi işlemler de yapılır. Analytics yaşam döngüsü hakkında ayrıca bilgi vereceğiz.

Veri çözümleme kategorisinde çok farklı işlevlere sahip olan yazılımlara ihtiyaç vardır. Ancak bu kategoride bulunan en yaygın yazılım Büyük Veri yazılımlarıdır. Büyük Veri ile verilerin depolanması, analizi ve dağıtımı gibi işlemler yapılır. Ayrıca veri toplama işlemi için kullanacağımız yazılımlar IoT sistemde bulunan geliştirme kitlerine de bağlıdır. Örneğin: Arduino için Arduino Programlama dili, Raspberry için Java veya Python gibi yazılım dillerini kullanmamız gerekebilir.

IoT bir sistemde veri çözümleme işlemleri çok önemlidir. Çünkü bu sistemlerde büyük miktarda veri ile çalışmak gerekiyor. Bundan dolayı bunların toplanması, depolanması, analizi ve dağıtımı için güçlü yazılımlara ve donanımlara ihtiyaç vardır.

Yukarıdaki şemada IoT bir sistemde bulunan veya bulunması muhtemel bazı aygıtlar gösterilmektedir. Bu aygıtlardan veri almak veya bunları anlık yönetmek için IoT veri çözümleme sistemlerine ihtiyacımız vardır.

Ağ

IoT, yani nesnelerin interneti ekosisteminin meydana gelmesi için güçlü ağ sistemlerine ve protokollerine ihtiyacımız vardır. Tüm elektronik cihazlar güçlü yazılım ve donanımlara sahip olsa bile aralarında bir ağ olmadığı zaman buna kesinlikle IoT diyemeyiz. IoT'nin meydana gelmesi için mutlaka cihazların birbirine bağlı olması gerekir.

Cihazların haberleşmesi, veri alışverişi ve kontrolü gibi işlemlerin yapılması için cihazların bağlanması gerekiyor. Burada kablolu veya kablosuz bağlantı yapılabilir. Ancak esas olan nesnelerin internet üzerinden birbirine bağlı olmasıdır. Burada çok farklı ağ teknolojileri ve ağ protokolleri bulunmaktadır. Örneğin, TCP/IP, MQTT veya HTTP gibi protokoller veri iletimi için kullanılabilir. Ayrıca sunucu işletim sistemi gibi yazılımlar da bu kategoride yer alır. Ağ protokolleri için ayrıntılı bilgi vereceğimiz için şimdilik burada bırakıyoruz.

IoT Analytics (Analiz) Yaşam Döngüsü

IoT Analiz yaşam döngüsü, veriler ile çalışırken takip edilmesi gereken aşamaları içerir. Nesnelerin internetinde veriler üzerinde çalışarak işlem yapmak önemli olduğu için bu işlemlerin belirli aşamalara ayrılması önem arz etmektedir.

Analiz işlemi için takip edilmesi gereken aşamalar şunlardır.

- Veri Toplama (Data Collection)
- Veri Analizi (Data Analysis)
- Veri Dağıtımı (Data Deployment)

IoT Analytics Lifecycle

IoT Veri analizi yaşam döngüsünde, öncelikle sensör veya çevre birimlerinden verileri alınır, alınan veriler analiz edilir ve kullanılacak forma dönüştürülür. Daha sonra oluşturulan veriler kontrol edilecek cihazlara dağıtılır. Bu adımları takip ettiğimizde buna IoT Analytics yaşam döngüsü denir.

Şimdi bu aşamaları tek tek açıklayalım.

Data Collection (Veri Toplama)

Bu aşamada sensör, eyleyici (actuators) veya konum gibi veriler çevre birimlerden toplanır. Toplanan verilerin kaynak ve format olarak doğrulanması işlemleri yapılır. Bu şekilde verilerin bütünlük, doğruluk ve tutarlılık bakımından geçerli olup olmadığı kontrol edilir.

Data Analysis (Veri Analizi)

Bu aşamada veri akışının yapılandırılması, verilerin depolanması ve nihayetinde kayıt edilmesi gibi temel işlemler yapılır. Ayrıca veri madenciliği, sınıflandırma, kümeleme ve kural belirleme gibi en önemli ve IoT için vazgeçilmez işlemler burada gerçekleşir. Bu teknikleri kullanarak ham verilerin uygulanabilir bilgiye dönüştürülmesi sağlanır.

Data Deployment (Veri Dağıtımı)

Bu aşamada daha önceki iki aşamada yapılan işlemlerden sonra elde edilen verilerin kullanılarak eylem dönüştürülmesi sağlanır. Ayrıca IoT verilerinin grafiksel olarak kullanıcıya gösterilmesi burada yapılır. Bunların dışında IoT ile elde edilen bilgilerin farklı uygulamalar arasında kullanılması da yine bu aşamada gerçekleşir.

Veriyi Bilgiye Dönüştürüp Toplama

Veri, kendi başına anlamı olmayan ham gerçeklik olarak tanımlanır. Veriler nicel ve nitel olarak elde edilebilir. Sayısal bir değer bildiren ve ölçüm veya sayım yoluyla elde edilen veriler nicel, sayısal bir değeri olmayan veriler nitel olarak ifade edilir. Verilerin tek başına bir anlam ve işlevi bulunmamaktadır.

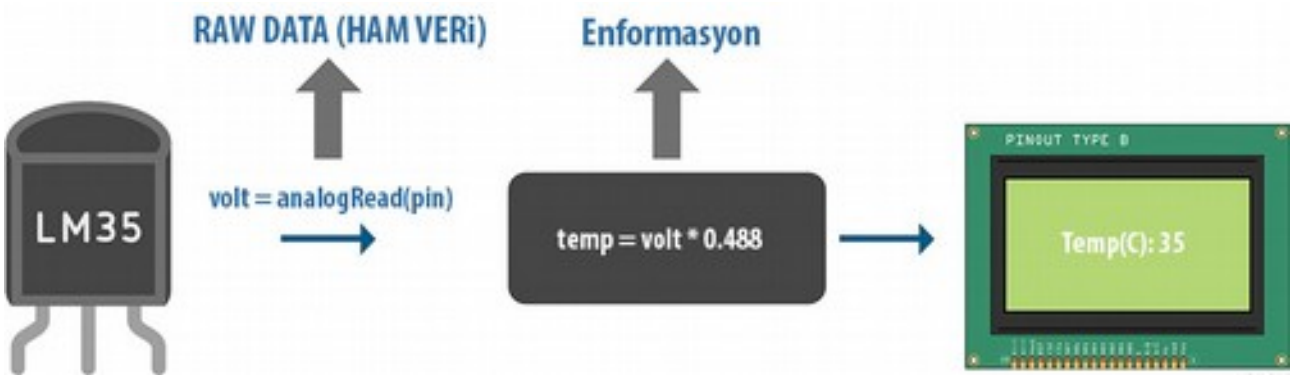
Enformasyon ise belirli bir amacı gerçekleştirmek için düzenlenmiş veri formu veya verinin anlam kazanmış hali olarak tanımlanır. Verilerin, enformasyona dönüşmesi için bazı işlemlerden geçmesi gerekiyor. Bu işlemler, sorgulama, depolama, düzenleme veya özetleme şeklinde yapılabilir.

Gateway aygıtlar kısa sürede ve yüksek miktarda verileri çevre birimlerden veya sensörlerden sağlamaktadır. Ancak alınan bu verilerin büyük bir kısmı veya tamamı ham, yani işlenmemiş verilerdir. IoT bir sistemin temel amacı veriyi baz alarak geniş ölçekte bir sistemi kontrol edebilmeyi sağlamaktır. Alınan ham verileri bir sistemi kontrol etmek için doğrudan kullanamayız. Bunun için elde edilen verilerin sistem için uygun formata dönüştürülmesi gerekiyor.

Enformasyon kavramını anlamak için LM35 sıcaklık sensörünü ele alalım. LM35 maliyeti uygun ve yüksek kaliteli bir sıcaklık sensörüdür. Bu sensör sıcaklık ölçümü için analog bir çıkış üretir. Analog

çıkış ham veri olarak kabul edilir. Çünkü bu veriyi alıp kullanıcıya doğrudan gösteremeyiz ve bir sistemi kontrol edemeyiz. Diyelim ki bu analog veriyi kullanarak evinizdeki klimayı kontrol edeceğinizi varsayalım. Öncelikle bu veriyi enformasyona yani bu uygulama için sıcaklık değerine dönüştürmeniz gerekiyor.

Aşağıda verilen resimde LM35 sensöründen alınan ham verinin nasıl enformasyona dönüştürüldüğünü görebilirsiniz.

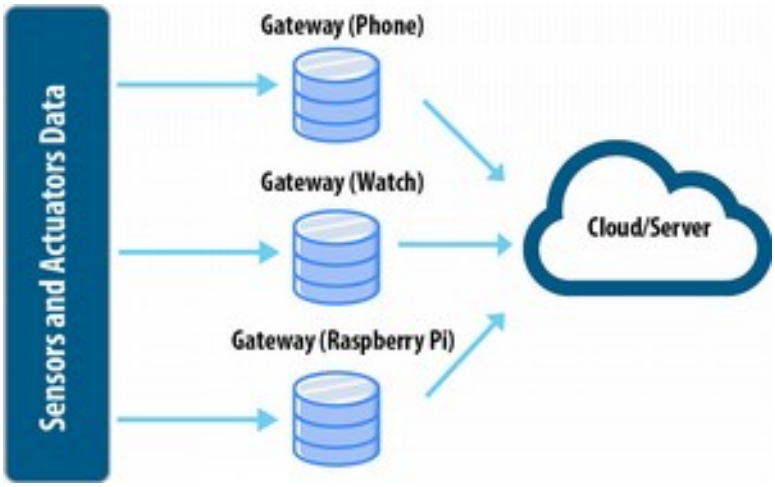


LM35'ten alınan ilk veri volt olup ham veridir. Burada çevreden ölçülen sıcaklık verisi bulunmaktadır. Bunu elde edebilmesi için 0,488 (Bu değer değişkenlik gösterebilir) gibi bir sabitle çarpmamız yeterlidir. Bu işlemden sonra sıcaklık değeri yani enformasyon elde edilir ve dijital ekranda kullanıcıya gösterilir.

Burada sadece LM35'i ele aldık ama IoT sisteminde kullanılan diğer sensörlerin büyük bir çoğunluğu çevreden ham verileri toplar. Ham verileri enformasyona dönüştürmek için kullandığımız sensör ve sensörlerin ham verilerini enformasyona dönüştürmek için üretici ile irtibata geçmeli veya datasheet'leri incelememiz gerekmektedir.

Veri Depolama

IoT bir sistemde bulunan elektronik cihaz veya sensör gibi donanımlardan elde edilen verileri tekrar tekrar kullanabilmek için verileri depolamamız gerekiyor. Veriler, IoT'nin birçok yerinde depolanabilir. Gateway Devices, yani ağ geçidi cihazları dediğimiz ve veri süzme, analitik, güvenlik ve daha birçok fonksiyona sahip cihazlarda da veri kaydı yapılabilir. Örneğin akıllı telefon, akıllı saat, otomobil veya herhangi bir donanım depolama birimi olarak kullanılabilir.



Farklı cihazlara veri kaydı yapmak mümkün olmakla birlikte bizi asıl ilgilendiren depolama alanı şüphesiz Cloud, yani bulut sistemleridir. Bulut bilişim dediğimiz bu teknolojinin amacı IoT bir

sistemde veri bütünlüğünü ve doğruluğunu sağlayıp, sistem genelinde meydana gelecek karmaşayı engellemektir. Bir diğer amacı da elde edilen ham verilerin işlenerek sistem genelinde kullanılacak bir formata dönüşmesini sağlamaktır. Yani sadece depolama değil verilerin işlenmesi de burada yapılır.

Her ne kadar Gateway cihazlarda veri depolama yapılabileceğini belirtsek de, bu cihazların CPU gücü ve depolama alanındaki sınırlar verilerin bulut sistemlerinde depolanmasını kaçınılmaz kılmaktadır. Bulut sistemleri hem güçlü sunucular ile desteklenmektedir hem de IoT bir sistemde veri depolama tek hedef değildir. Amaç veriler üzerinde analiz işlemlerini de yapabilmektir. Bundan dolayı güçlü işlemcilere hatta veri analizi yapabilen Büyük Veri sistemlerine ihtiyacımız var. Bulut bilişim bu gereksinimleri sağlayabilecek bir kapasiteye sahiptir. Gateway, yani kenar (Edge) cihazlar genel olarak büyük veri işleme sistemlerine sahip değildir. Hatta böyle bir sistemin bu cihazlarda olması zorunlu değildir. Ancak şunu da unutmamak gerekir ki, bulut sistemleri verileri alabilmek için Gateway cihazlara ihtiyaç duyarlar. Örneğin akıllı telefonu bir Gateway olarak kullanarak çevrede bulunan sıcaklık, nem veya yağmur sensörü gibi donanımlardan verileri alıp bulut sisteme gönderebiliriz. Sensörler kendi verilerini doğrudan gönderemedikleri için şimdilik akıllı telefon, akıllı saat veya Raspberry gibi ağ geçidi cihazları IoT bir sistem için kaçınılmazdır.

Gateway aygıtların IoT bir sisteme en büyük katkısı kısa sürede yüksek miktarda veri üretebilmeleridir. Bulut sistemleri de elde edilen muazzam veri trafiğini sağlayan hatta bu cihazlardan gelen verileri alabilecek güçlü veritabanı sistemlerine sahiptir.

Gerçek Zamanlı Analiz ve Yorumlama

Gerçek zamanlı analiz ve yorumlama IoT bir sistemde işlemlerin anında yapılabilmesi için en önemli konulardan biridir. Analiz, ham verilerin işlenmesi ve ihtiyacımız olan verilerin ayrıştırılması olarak tanımlanabilir. Yorumlama ise, analiz sonucunda elde edilen verilere göre herhangi bir eylemde bulunmayı sağlamaktır. Yorumlama yani bir eylemin gerçekleşmesi için verilerin gerçek zamanlı analizi çok önemlidir. IoT bir sistemin gelecek için en büyük handikaplarından birisi şüphesiz eylemlerin gecikmesidir. IoT tabanlı akıllı şehir sistemlerinde böyle bir gecikme işlemleri hızlandırmaktan ziyade hem yavaşlatacak hatta daha kötüsü şehirde bir karmaşanın meydan gelmesine neden olacaktır. Bu gibi durumların önüne geçmek ve işlemlerin anlık olarak yapılması için güçlü IoT analiz sistemlerine ihtiyacımız var.

Şu an için gerçek zamanlı analiz ve yorumlama için bulut sistemlerini kullanıyoruz. Bulut sistemleri büyük verilerle başa çıkacak Büyük Veri veritabanı sistemlerine sahiptir. Bu sistemler verilerin hızlı bir şekilde analiz edilmesini sağlamaktadır. Ancak asıl problem bu işin gerçek zamanlı yapılmasıdır. Sadece bulut sistemlerle devam edilirse, işlemlerin anlık olarak gerçekleşmesi mümkün değildir.

Cisco, 2020 yılında 50 milyardan fazla cihazın internete bağlanacağını tahmin etmektedir. Bu kadar cihazdan gelen milyarlarca terabaytlık verinin analiz işlemi sadece bulut sistemlerine bırakılamaz. Bunun için Edge Computing denilen yeni bir teknoloji geliştirilmiştir. Edge'nin amacı verilerin anlık analizini ve gerçek zamanlı işlem yapılmasını sağlamaktır. Cisco bu teknolojiyi Fog Computing adı altında lisansladı. Kitabımızda ele aldığımız IoT-ignite platformu da bir Fog girişimidir. Tabi bu platformda Gateway aygıtları kullanılmaktadır. Ancak Gateway ve Fog aynı işlemi yapmaktadır.

IoT-Ignite platformunu ele alırsak, bu platformda Gateway denilen ağ cihazları ile alınıp buluta gönderilmektedir. Yapı olarak Fog ve Edge'ye benzediği için Iot-Ignite platformunda gerçek zamanlı analiz ve yorumlama yapılmaktadır.

Kablosuz Ağ Protokolleri

Ağ teknolojilerinin IoT projelerinde etkin bir şekilde kullanılması beraberinde bazı protokollerin kullanılmasını kaçınılmaz kılmaktadır. IoT sistemleri ile geliştirilen bir proje standart protokolleri ve ağ teknolojilerini kullanır. Protokolleri kullanarak IoT projesinde bulunan donanımların haberleşmesi veya veri iletişimi sağlanır.

IoT-Ignite platformu ile çalışırken bu protokol veya protokollerden bazılarını uygulamalarımızda kullanacağız. Örneğin Android (Gateway) bir telefonun Bluetooth üzerinden NodeMCU (Node) kartında bulunan sensörlerden veri alıp gönderebileceğiz. İsterseniz, bu işlem için Wi-Fi teknolojisini de kullanabilirsiniz. Bu tercih tamamen size kalmış. Önemli olan bu protokolleri kullanarak veri trafiğini sağlamak ve nihayetinde verileri buluta gönderip kayıt etmeyi ve gerektiğinde kullanabilmeyi sağlamaktır.

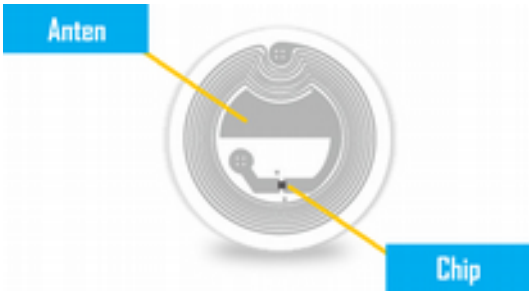
Şimdi IoT ile birlikte kullanılan kablosuz ağ protokoller hakkında bilgi verelim.

NFC

ISO/IEC tarafından 2003 yılında bir standart olarak kabul edilen NFC, iki elektronik cihazın haberleşmesini sağlayan bir ağ teknolojisidir. Bu teknolojide veri iletimi kısa mesafede, yüksek frekansta ve düşük bant genişliğinde gerçekleşir. NFC'nin uygulama alanı oldukça geniştir. Bunlara örnek olarak; ulaşım, akıllı posterler, cihazlar arasında veri iletimi, para transfer işlemleri veya ödeme işlemlerini verebiliriz.

NFC, RFID teknolojisinden farklı olarak veri iletimini kısa menzilde ve daha düşük bant aralığında gerçekleştirir. NFC ile veri iletişimi yapılabilmesi için iki elektronik cihazın arasındaki mesafenin maksimum 5 cm olması gerekiyor. Zaten bundan dolayı Near Field Communication, yani Yakın Alan İletişimi olarak adlandırılmaktadır. İletişimin yakın bir mesafede yapılma zorunluluğu güvenlik için oldukça önemlidir. Çünkü bu gibi iletişim protokollerinde ödeme gibi işlemler yapılırken güvenliğin sağlanması oldukça önemlidir. İki cihaz arasındaki mesafe bir nebze olsun güvenliğin sağlanmasında oldukça etkilidir. Bu iletişim teknolojisinde frekans olarak 13.56 Mhz'lik bir frekans kullanılır. Bu değer oldukça yüksek bir frekanstır.

NFC teknolojisi RFID'in özel bir alt kümesi olarak geliştirilmiştir. Özellikle veri alışverişinin güvenli bir şekilde yapılması için geliştirilmiştir. NFC cihazları hem okuyucu hem de NFC etiketi olma özelliğine sahiptir. Piyasada bu işlevi gerçekleştiren en önemli aygıt Akıllı Telefonlardır. Akıllı telefonlarda bulunan NFC aygıtları hem okuyucu hem de etiket olarak çalışabilmektedir.



NFC Tag'nın yapısı oldukça basit ve yukarıdaki gibidir. Burada bir anten ve chip görmekteyiz. Anten bakır veya alüminyumdan meydana gelmektedir. NFC okuyucular anten aracılığıyla etikete bir sinyal gönderir. Etiket kendisinde bulunan biricik bilgileri okuyucunun antenine gönderir. Anten gelen veriyi alarak dijital ortama aktarır.

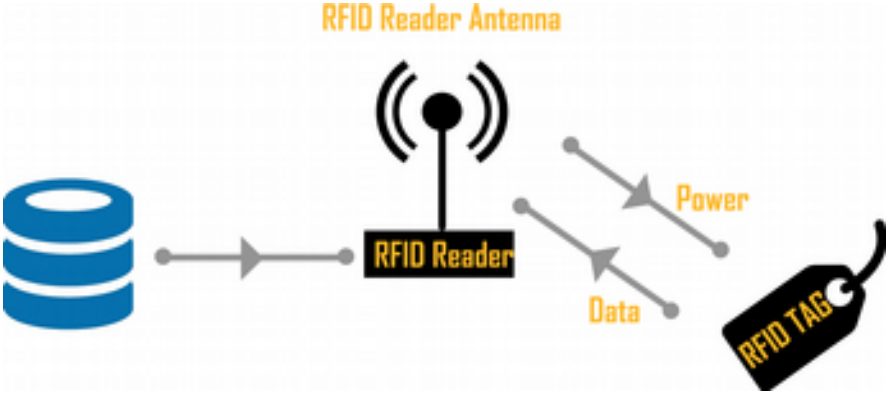
RFID

Radio Frequency Identification, yani Radyo Frekanslı Tanımlama olan RFID, nesneleri veya objeleri tanımlamak için kullanılan bir teknolojidir. Nesneleri tanımak için radyo dalgaları kullanılır. Tanımlama veya tanıma işlemini yapabilmek için nesneye bir RFID etiketi eklenir. Bu etikete yazılan veri okunarak nesne hakkındaki kimlik bilgilerine kolaylıkla erişim sağlanabilir. Örneğin bu teknoloji kullanılarak California, Richmond'taki bir okulda öğrencilerin kimlik kartlarına yerleştirilen RFID çipleri ile öğrencilerin derste olup olmadığı kontrol edilmektedir. Hatta öğrencinin okul sınırları içinde olup olmadığı bu şekilde izlenebilmektedir.

NFC teknolojisi yakın alan iletişimi için kullanılırken, RFID teknolojisinde verilerin uzaktan iletimi söz konusudur. Her iki teknolojide sektörde "tag" denilen etiketler kullanılır. Etiketlere yazılan veri okunarak takip, ödeme vb. işlemler rahatlıkla yapılabilir.

RFID tabanlı bir sistemde en azından birer adet etiket, okuyucu ve antene ihtiyaç vardır. RFID okuyucular anten aracılığıyla etikete bir sinyal gönderir. Etiket kendisinde bulunan biricik bilgileri antene gönderir. Anten gelen veriyi bilgisayar ortamına dijital olarak gönderir. Destek yazılımlar sayesinde bu veriler okunarak işlemlerin yapılması sağlanır.

RFID teknolojisini kullanan bir sistemin çalışma yapısı aşağıdaki gibidir.



RFID teknolojisinde etiketler aktif ve pasif olmak üzere ikiye ayrılır. Aktif etiketlerin kendi güç kaynakları olduğu için 100 metreye kadar sinyal yayma gücüne sahiptirler. Bu etiketler özellikle konum tespiti gibi geniş alana yayılan işlemleri yapmak için tercih edilir. Pasif etiketlerde durum biraz daha farklı. Bu etiketlerin kendi enerji kaynakları bulunmadığı için enerji ihtiyacınız antenden gelen elektromanyetik enerjiden karşılarlar. Alınan bu enerji etiket içinde bulunan verinin okunması ve antene iletilmesinde kullanılır.

Bluetooth

Bluetooh teknolojisi 2.4 GHz bandında radyo dalgaları ile çalışan ve temel amacı farklı cihazlar arasında veri iletişimini sağlayan kablosuz bir teknolojidir. Oldukça düşük enerji ile çalışan bu teknoloji özellikle cep telefonları ile standart bir teknoloji haline gelmiştir. Bu teknolojide iki cihaz arasında 100 metreye kadar bir mesafede veri iletimi yapılabilmektedir.

2007 yılında Bluetooth 2.1 ile birlikte enerji tasarrufu 5 kat daha iyileştirildi. 2009 yılında çıkan Bluetooth 3.0 ile veri iletim hızı 24 Mbps'ye kadar yükselerek kullanıcıların dosya paylaşımı ve oyun oynamaları daha konforlu bir seviyeye yükselmiştir.

Bluetooth 4.0 ile birlikte güç tüketimi daha da azaltılarak, fitness, kalp atışı ve adım sayacı gibi donanımların telefonlarda kullanılması sağlanmıştır. IoT ile ilgilenenlerin özellikle low-energy

protokolü olarak bildiği en önemli Blutooth sürümü 4.0'dır. Çünkü bu sürümde enerji tüketimi daha da azalarak IoT sistemlerinde kullanılabilecek bir hale gelmiştir.

Bluetooth.com'daki açıklamaya göre 6 Aralık 2016 yılında yayınlanan Bluetooth 5.0 versiyonu, özellikle IoT uygulamalarına hitap etmektedir. Bu sürümün en önemli özelliği duvarlardan daha etkili geçebilmesi ve ev, ofis gibi iç mekanlarda daha etkin kullanılabilmesidir. Bluetooth 5.0 ile birlikte menzil mesafesi 4 katına, veri iletim hızı 2 katına ve bağlantısız veri yayın kapasitesi 8 katına çıkarılmıştır. Bunları yaparken de düşük güç tüketimini sağlaması IoT için vazgeçilmez bir protokol haline gelmesini sağlamıştır.

Wireless

Ev ve ofis ortamında Wifi bağlantısının yaygın bir şekilde kullanılması Wireless teknolojisinin IoT içinde tercih edilmesinde en büyük etkene sahiptir. Bu ağ protokolü ile büyük verilerin işlenmesi ve iletilmesi hızlı bir şekilde gerçekleşir. Şu an için kullanılan en yaygın Wifi standardı 802.11n standardıdır. Verilerin iletimi hızlı bir şekilde yapılmasına karşın enerji tüketiminin fazla olması IoT projeleri için en büyük dezavantajdır. Wifi bağlantısının kullanıldığı projelerde güç tüketimini azaltmak için kablosuz bağlantı dinleme modunda bırakılmaktadır.

Radio Signal

Radyo sinyalleri ile düşük oranlı ağlar oluşturmak için ZigBee, Z-Wave veya Thread gibi radyo protokolleri kullanılır. Bu teknolojiler düşük güç tüketimiyle birlikte yüksek oranda verim sağlamaktadır. Küçük cihazlar için yerel ağlar kurulurken bu tip radyo protokollerini rahatlıkla kullanabiliriz.

ZigBee, Bluetooth benzeri, kişisel alan ağlarının oluşturulması için kullanılan ve bu işlemi yaparken düşük güç tüketimi sağlayan bir iletişim protokolüdür. Temel amacı gündelik cihazların bir ağ içerisinde yönetilmesini ve kontrol edilmesini sağlamaktır.

Z-Wave, düşük enerjili bir RF iletişim teknolojisidir. Başta geliştirilme amacı, lamba ve sensör gibi ürünler için ev otomasyonu oluşturmayı sağlamaktır. 100Kbits/s gibi veri hızlarına sahip küçük veri paketlerini güvenilir ve düşük gecikmeli olarak iletmek için tasarlanmıştır. Bu radyo protokolü 1GHz altındaki bantta çalışır ve 2.4 GHz aralığında Bluetooth ve ZigBee gibi diğer kablosuz teknolojilerden kesinlikle etkilenmez. Z-Wave ile 232 adet cihaza kadar kontrol sağlanabilmektedir.

THREAD

Thread, IPv6 tabanlı ve ev otomasyonu ortamını hedefleyen bir ağ protokolüdür. Bu protokol Bluetooth veya ZigBee gibi bir protokol değildir. IP veri aktarımını kullanan kablosuz bir ağ protokolü olduğu için Wifi bağlantısını tamamlayan bir ağ protokolüdür. Bu protokol özellikle IoT projeleri için geliştirilmiştir. Özellikle tüketici uygulamaları, ev içi ve ev çevresindeki cihazlar için tasarlanmıştır. Wifi'ye göre en büyük artısı veri iletimini yaparken düşük güç tüketimi sağlamasıdır.

LTE-A

Hücresel bir iletişim protokolü olan LTE-A ile daha uzun mesafelerde çalışacak IoT uygulamalarının geliştirilmesi sağlanır. LTE-A ile büyük verilerin yüksek hızda iletimi sağlanır. Tabii bunun yerine GSM veya 3G gibi teknolojileri de kullanabiliriz. Ancak uzak veri iletişiminin gerekli olduğu yerlerde 4G yani LTE-A bandını kullanmak IoT uygulamasının daha hızlı işlem yapmasını sağlayacaktır.

LTE A

Büyük verilerin bu yolla gönderimi hem pahalı hem de yüksek güç tüketimine neden olmaktadır. Ancak daha küçük verilerin hızlı iletimi gibi durumlarda bu protokolü kullanmamız işlemlerin daha hızlı yapılmasını sağlayacaktır. Özetle uzak mesafelerde veri almak veya cihazları kontrol etmek için LTE gibi hücresel ağ teknolojilerinden faydalanmak gerekiyor.

Wifi Direct

Wifi Direct önceleri Wifi P2P olarak adlandırılan ve iki adet Wifi istemcisine sahip cihaz arasında kablosuz bağlantı oluşturarak dosya ve veri transfer işleminin yapılmasını sağlayan ağ kurma işlemidir. Bu yöntemde kablosuz bağlantı kullanılarak Bluetooth'ta olduğu gibi veri iletiminin yapılması sağlanır. Ancak veri iletimi Bluetooth'a göre daha hızlı ve verimli bir şekilde gerçekleşmektedir. Bunu yaparken internet bağlantısına da yerel ağ bağlantısına da ihtiyacınız yok. Çünkü burada internet üzerinden veri iletimi yapılmıyor, aynı ortamda bulunan iki cihaz arasında haberleşme sağlanıyor.

Bu yöntemin en büyük avantajı büyük verilerin hızlı bir şekilde ve gecikmeye izin vermeden iletilmesini sağlamaktır.

Veri Protokolleri

IoT ekosisteminde ele alınması gereken konulardan bir diğeri de veri dağıtımı protokolleridir. Yukarda ele aldığımız kablosuz ağ protokolleri verinin iletildiği yollar veya kanallardır. Burada ise veri dağıtılırken hangi şekilde paketlenir bunlardan bahsedeceğiz.

Aradaki farkı en iyi şekilde anlamak için şöyle bir benzetme yapabiliriz. Bir kargo şirketinde ürünlerin dağıtımı için kullanılan kara veya hava yolları kablosuz ağ protokollerini temsil ederken, kargo aracı ise veri dağıtım protokollerini temsil eder.

Veri dağıtım protokollerinin en çok kullanılanları şunlardır:

- MQTT
- CoAP
- AMQP

Şimdi bunlar hakkında kısaca bilgi verelim.

MQTT

Machine to Machine (M2M) kavramı, Industrial Internet of Things (IIoT) kavramı içinde gelişen önemli bir bileşendir. M2M, veri ve bilgi alışverişinde makinelerin birbirleriyle nasıl konuştuğu üzerine odaklanır. Diğer bir deyişle, M2M teriminde, insan eylemleri olmaksızın gerçek zamanlı veri alışverişini sağlayan tüm teknolojilere ve kablosuz ağlara atıfta bulunuyoruz. Makineler bu yöntem ile kendi başına diğer makinelere veri aktarımı yapabilir. Bundan dolayı M2M'de aşağıda verilen uygulamalar geliştirilebilir.

- Telemetri (bir sistemin uzaktan kablolu ya da kablosuz izlenmesi)
- Gerçek zamanlı hata bildirimi
- Uzaktan makine durumunu denetleme
- Gerçek zamanlı veri toplama

Yukarıdaki işlemleri gerçekleştirmek için MQTT protokolü kullanılır. MQTT ile IoT uygulamaları geliştirebiliriz. Bu protokolün çalışma mantığını bilmek bundan dolayı çok önemlidir. Şimdi bu protokol hakkında bilgi verelim.

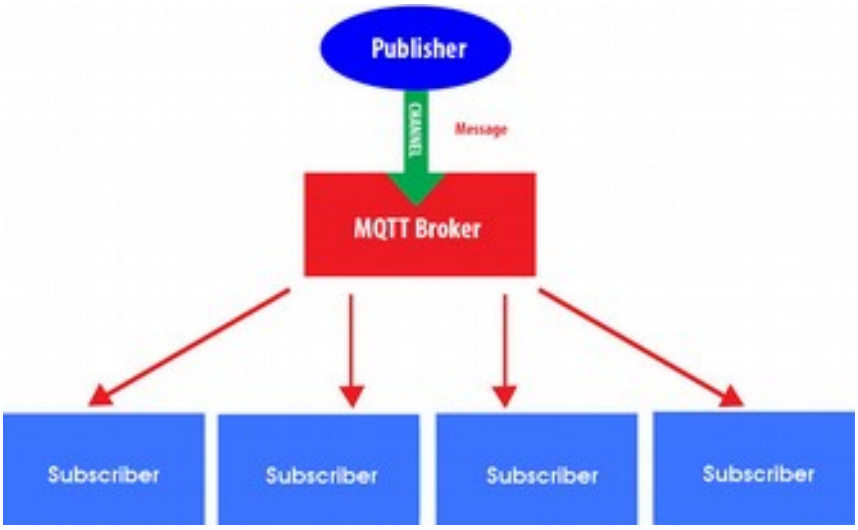
Message Queue Telemetry Transport (MQTT), 1999 yılından mesaj tabanlı geliştirilen bir protokoldür. IoT'de, M2M yani makineler arasında veri alışverişi için kullanılan en yaygın protokoldür. Bu protokolün kullanımı ve uygulanması çok kolaydır. Özellikle ağ bant genişliği kısıtlamaları olan durumlarda bu protokolden faydalanabiliriz. Bu satırları yazarken MQTT'nin son sürümü 3.1.1'di. Birçok IoT bulut platformu bu protokolü benimsemiştir.

MQTT'de ağın güvenli olmadığı veya zayıf olduğu durumlarda mesajın ana hedefe ulaştırılması tek hedef olarak icra edilir. MQTT'de de HTTPS'de olduğu gibi hem SSL hem de kimlik doğrulama mekanizmaları vardır.

MQTT'nin önemli rol oynadığı birkaç yaygın kullanım örneği aşağıdaki gibidir.

- Telemetri
- Bildirim Sistemleri
- Akıllı Evler

MQTT, mesaj tabanlı bir protokoldür. Diğer bir deyişle veri alışverişi sürecine katılan makineler mesaj gönderip alabilirler. Ayrıca publisher/subscriber (yayın/abone) modelini temel alarak çalışır. Yayıncı/Abone modelini anlamak için aşağıdaki resmi inceleyiniz.



Yukarıdaki şemada görüleceği üzere MQTT'de mesajın iletimi sürecinde üç ana bileşen bulunmaktadır.

Publisher (Yayıncı)

Bu bileşen sistemde bulunan ve mesaj üreten bir cihazı temsil etmektedir. Diğer bir deyişle bilginin kaynağıdır.

MQTT Broker

Yayıncının hizmet verdiği abonelere mesajın gönderilmesini sağlar. Burada gönderilen mesajlar bir dizi halinde abonelere iletilir.

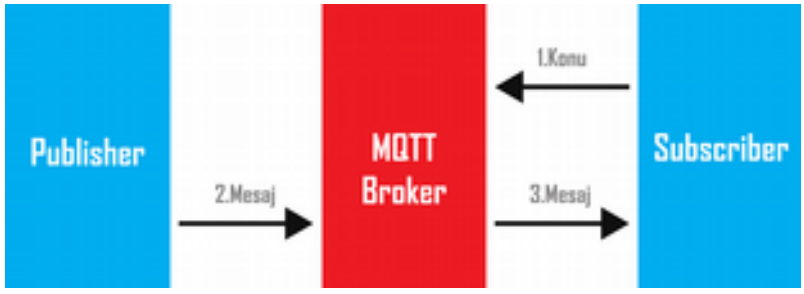
Subscriber (İstemci)

Yayıncıdan mesaj almak isteyen istemcileri temsil etmektedir. MQTT'nin yayıncıdan aldığı mesajlar buraya iletilir. Burada da elektronik cihaz bulunmaktadır.

MQTT'nin çalıştığı bir sistemde çok sayıda yayıncı ve abone bulunabilir.

Yayıncıdan gelen mesajı alacak olan aboneleri filtrelemek için MQTT bir konu (topic) kullanmaktadır. Bu konu yayıncı ve istemci arasındaki bir kanalı temsil etmektedir. Burada kullanılan konu bilgisi UTF-8 dizgesi olarak temsil edilir.

Şimdi de yayıncı ve abone arasındaki mesaj alışverişinin nasıl olduğunu inceleyelim. Öncelikle aşağıdaki resmi inceleyiniz.



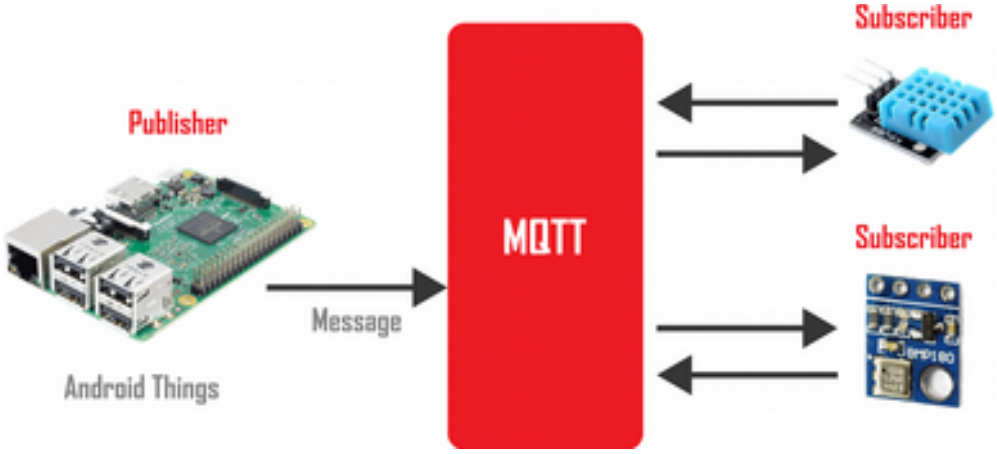
Şekilde incelediğimizde önemli olan adımların şunlar olduğu görülür.

- Öncelikle istemci veya abone mesaj almak istediğini MQTT'ye bildirir.

- Yayıncı konuyla ilgili bir mesaj yayınlar.
- MQTT yayıncıdan gelen mesajı alıp ilgili istemciye gönderir.

MQTT'yi kullanarak bu şekilde mesaj göndermenin en önemli faydası yayıncı, aboneye bir mesaj gönderdiği zaman o sırada abonenin bağlı olması veya aktif olması zorunlu değildir. Bu gibi durumlarda mesaj bloke edilmez. Bağlantı sağlandığı anda mesaj hemen iletilir.

MQTT kullanılan bir IoT sistemi aşağıdaki gibi olabilir.



CoAP

CoAP, bir belge transfer protokolüdür. Bu yönüyle HTTP protokolüne benzer. Ayrıca özel bir web transfer protokolü olarak da ifade edilmektedir. Ancak HTTP'den farklı olarak CoAP, IoT'de bulunan kısıtlı ağlar ve cihazlar için geliştirilmiştir. Akıllı enerji ve bina otomasyonu gibi M2M uygulamaları için bu protokolü özellikle kullanmanızı tavsiye ediyoruz. Çünkü temel geliştirilme sebebi budur.

CoAP protokolü, sadece UDP üzerinde çalışmaktadır. Yani TCP üzerinde bu protokol çalışmamaktadır. CoAP protokolü, IoT üzerinde bulunan node'lar ile çalışmak için 10KB'a kadar düşük RAM ve 100KB'lık kod alanına sahip mikrodenetleyiciler ile çalışacak şekilde tasarlandı.

IoT için en büyük problemlerden biri olan güvenlik problemini de çözmeye çalışan bu protokol oldukça güvenlidir. Ayrıca XML ve JSON gibi verilerle uyumlu olarak çalışabilmektedir. HTTP'ye benzediği için server/client modelini takip etmektedir. Yani GET, PUT, POST ve DELETE gibi metotlar ile istemciler sunucudan istekte bulunabilir. Son olarak datagram tabanlı olduğu için SMS ve diğer paket tabanlı iletişim protokolleri üzerinde kullanılabilir.

AMQP

AMQP, bir uygulama katmanı protokolüdür. Çok sayıda mesajın iletimini sağlamak ve iletişimi doğrulamak için tasarlanmıştır. Mesajın iletimini garantilemek için akış kontrollü ve mesaj odaklı bir iletişim sağlar. Bu protokolü kullanmanın iki önemli sebebi güvenirlik ve birlikte çalışabilirliktir.

AMQP, mesaj iletimi için aşağıdaki özelliklere sahiptir;

- Mesajlaşma tipi ile ilgili geniş imkanlara sahip olması.
- Güvenirlik.
- Konu tabanlı yayın ve abone mesajlaşma.
- Esnek yönlendirme.

Bu protokol birçok firma tarafından kullanılmaktadır. Bunlardan en önemlileri şunlardır:

- JP Morgan, günde bir milyar mesajı işleme koymak için bunu kullanmaktadır.
- NASA Nebula Cloud Computing için bu protokolü kullanmaktadır.
- Google firması karmaşık olayları işlemek için bu protokolden faydalanıyor.

Bu protokoller dışında aşağıdaki protokolleri de kullanabilirsiniz.

- HTTP
- REST
- XMPP
- STOMP

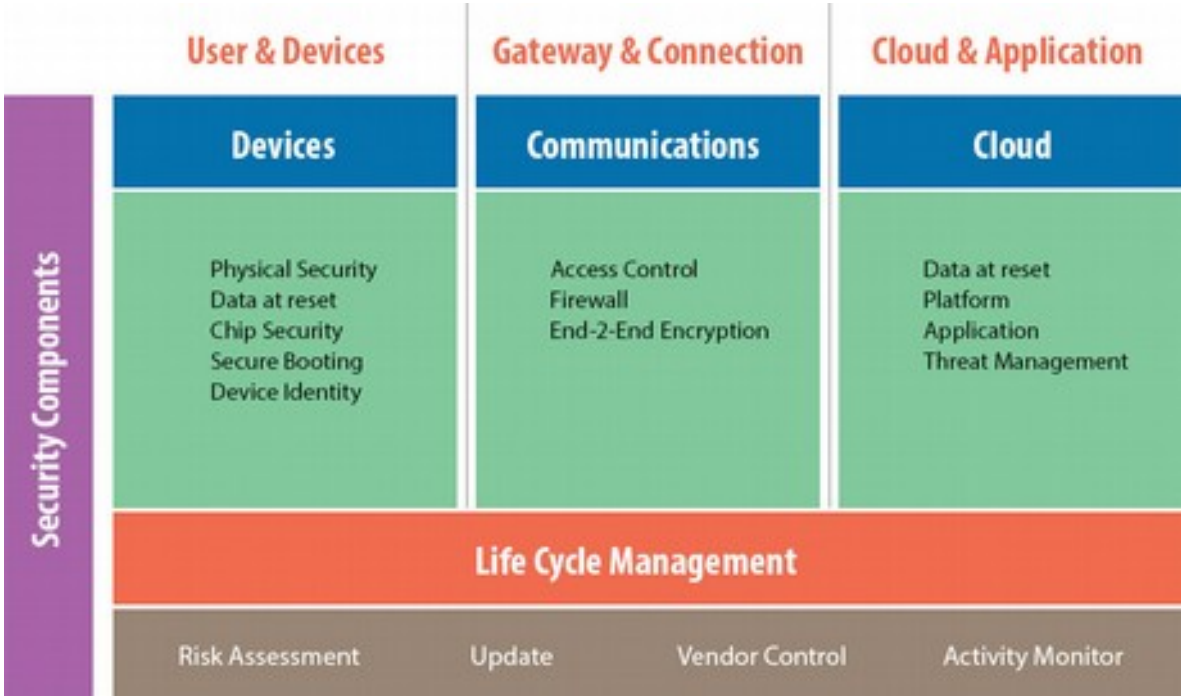
IoT'de Güvenlik Yaklaşımları

IoT ile birlikte elektronik cihazların herhangi bir ağ veya internet üzerinden haberleşmeleri, veri alış verişi yapmaları sağlanmaktadır. Cihazların bu şekilde haberleşmeleri, güvenlik açıklarına ve kişisel bilgilerin tehlikeye girmesine neden olmaktadır. IoT ile gelen belki de en önemli sorun budur. Hayatı kolaylaştırması ve bulut sistemlerin kullanılarak herhangi bir yerden elektronik cihazların kontrol edilmesi, evinizdeki kamera veya mikrofonun faydadan çok zarar vermesine sebep olabilmektedir.

IoT sistemlerinde güvenlik açıklarının meydana gelebileceği dört farklı katman vardır. Iot-analytics.com'a göre bu katmanlar şu şekildedir:

- Cihaz
- İletişim
- Bulut
- Yaşam Döngüsü Yönetimi

End-to-end, yani uçtan uca IoT uygulamaları tasarlarken bu katmanlar için geliştirilen donanım, yazılım, iletişim gibi uygulama bölümlerinde güvenliğin olabildiğince sağlanması gerekiyor. IoT uygulamaları farklı teknolojilerin bir araya gelmesiyle zincirleme bir bağ ile meydana gelir. Her bir zincir halkasının sağlam bir güvenlik yaklaşımına dayanması sistemde zayıf halkaların ortadan kaldırılmasını ve dolayısıyla güvenliğin tesisini sağlar.



Yukarıdaki şemada katmanlar arasındaki ilişkiyi net olarak inceleyebilirsiniz.

Şimdi bu katmanları tek tek açıklayalım.

Cihaz

IoT uygulamaları için güvenliğin ilk adımı Device, yani kullanıcı kontrolünde veya kullanıcı ile etkileşimde olan cihazların güvenli olmasını sağlamak gelir. Kullanıcı güvenliğini tehlikeye atacak ilk katman burasıdır.

Bu katman için aşağıda verilen güvenlik önlemlerinin alınması gerekiyor.

- Veri giriş güvenliği,
- Donanım ve yazılım güvenliği,
- Güvenli önyükleme,
- İşlemci veya yonga güvenliği,
- Cihaz erişimi için güvenlik anahtarı veya kimliği tanımlamak.

Bu katmana örnek olarak akıllı telefonları verebiliriz. Bankacılık, alışveriş gibi birçok işlemi yaptığımız telefonlarda ihtiyaç olan durumlarda kredi kartı bilgilerimizi girebiliyoruz. Bu gibi hassas ve önemli olan verilerin başkaları tarafından ele geçirilmemesi için birçok güvenlik önlemine ihtiyacımız var. Örneğin yukarıdaki listede güvenli önyükleme gibi bir korumada, sadece doğrulanmış yazılımın aygıtta çalışması sağlanır. Doğrulanmamış yazılımlar bu sayede cihazımızda çalışamayacaktır. Ayrıca cihaz erişimi için belirlenen şifreler de cihaza herkesin erişimini engelleyecektir. Son kullanıcının sürekli etkileşim halinde olduğu bu katman için geliştirilen cihazların güvenliği IoT uygulamaları için hayati öneme sahiptir.

İletişim

İletişim katmanı IoT uygulamalarında verilerin güvenli bir şekilde iletildiği veya alındığı ortamların güvenliğini ifade eder. Hassas veriler özellikle fiziksel, ağ veya uygulama katmanlarında iken saldırıya hedef olabilir.

Bu ağ katmanlarından gelebilecek saldırılar şu şekildedir.

- Fiziksel Katman; Wifi veya Ethernet üzerinden.
- Ağ Katmanı; Ipv4 veya IPv6 üzerinden.
- Uygulama Katmanı: MQTT veya WebSocket üzerinden.

Buradaki üç katmanda aşağıda verilen güvenlik önlemlerinin alınması oldukça önemlidir.

- Erişimlerin kontrol edilmesi.
- Veri akışını kontrol eden Firewall gibi güvenlik duvarlarının kullanılması.
- End-to-End (Uçtan uca) Şifreleme.

Bulut

Bulut katmanında, cihazlardan gelen verilerin kullanılarak yeni eylemlerin veya işlemlerin yapılabilmesi için verilerin bir ölçek temelinde analiz edildiği ve yorumlandığı katmandır. Ayrıca verilerin depolanmasını veya IoT sisteminde yer alan diğer cihazların veri gereksinimlerini karşılamayı da sağlar. IoT sistem veya sistemlerinin bir bütünlük içinde olmasını sağlayan bu katman için güvenli ve etkin bulut hizmetlerinin sunulması gerekiyor. Aksi bir durumda sistemin bütünlüğü ciddi şekilde bozulabilir.

Bulut katmanı için aşağıdaki güvenlik önemlerinin alınması gerekiyor:

- Bulutta depolanan verilerin saldırılara kolayca hedef olmaması için verilerin kesinlikle şifrelenmesi gerekiyor.
- IoT sisteminde bulunan bir buluta diğer bulutlardan erişim sağlanması veya bazı uygulamaların bulut hizmetinden faydalanması için doğrulama veya tanımlama işlemlerinin yapılması sağlanmalı.
- Dijital sertifika veya kimlik doğrulama gibi önlemler alınmalıdır.

Yaşam Döngüsü Yönetimi

Bu katmanda IoT uygulamasının güvenliğini güncel tutmak için her bir IoT bileşeninin gözetim ve denetim altında tutulması sağlanır. Yani burada yapılan temel işlem donanım, yazılım veya güvenlik önlemlerinin güncel olup olmadığını kontrol etmektir.

Bu katman için aşağıdaki güvenlik önlemlerinin alınması gerekiyor:

- Sistem genelinde meydana gelen etkinliklerin izlenmesini sağlamak. Anormal durumları loglamak.
- Düzenli güvenlik yamaları oluşturmak ve güvenlik önlemlerinin güncel kalmasını sağlamak.
- Güvenlik için IoT cihazlarının uzaktan güvenli olarak bakımını sağlamak.

Burada verilen her katman IoT güvenliği için birer halkayı temsil etmektedir. Halkalar ne kadar güvenli ve sağlam olursa güvenlik de o nispette sağlam olacaktır.

Şimdi de IoT güvenliği arttırmak için kullanılması gereken önemli teknolojileri başlıklar halinde ele alalım.

Ağ Güvenliği

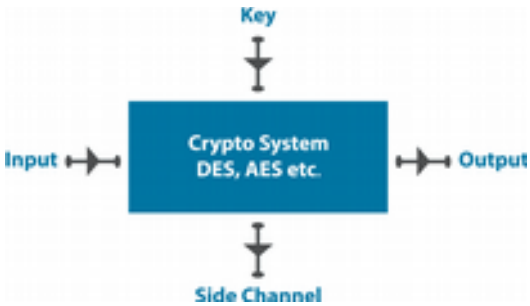
Ağ bağlantısı kablolu veya kablosuz yapılabilir. Kablolu bağlantı kablosuz bağlantıya göre oldukça güvenli. Ancak IoT sistemlerinde kablosuz bağlantının daha yaygın kullanıldığını görmekteyiz. Kablosuz bağlantının kullanıldığı IoT sistemlerinde güvenliği güçlendirmek için kablosuz ağ güvenliğinin sağlanması gerekmektedir. Kablosuz ağlarda birçok farklı protokolün kullanılması en büyük güvenlik açığı olarak ele alınmaktadır. Veri iletiminin ağrılıklı olarak kablosuz yapılacağı dikkate alındığında bu teknolojideki güvenlik önlemleri IoT için vazgeçilmez olmaktadır.

Yetkilendirmeler

IoT cihazlarının kullanılabilmesi için yetkilendirmelerin yapılması gerekiyor. Yetkilendirme işlemi beraberinde şifreli erişimi ve kimlik doğrulamayı da getirmektedir. IoT sistemleri birçok kullanıcının kullanımına sunulmuş olabilir. Böyle bir durumda anonim yani herkese açık bir erişimden ziyade yetkilendirme yapmak ve bu yetkilendirmeler arasında hiyerarşiye dikkat etmek sisteminizin daha güvenli olmasını sağlayacaktır. Yetkilendirme işlemi, iki faktörlü kimlik doğrulama (Bankacılık işlemlerinde ilk şifreden sonra akıllı telefona gelen ikinci bir şifre gibi), biyometrik veya dijital sertifikalar şeklinde yapılabilir.

Yan Kanal Saldırıları (Side Channel Attacks)

Verilerin şifrelenmesi veya kimlik doğrulama yani kullanıcı girişi sırasında yapılan saldırılara yan kanal saldırıları denir. Şifreleme için kullanılan gizli anahtarların zayıf bir işlemciden alınmasını hedefleyen bu yöntem özellikle şifreleme anında gerçekleşir. Gizli anahtarların alınıp şifrelerin çözülmesi mantığını kullandığı için esasen bir tersine mühendislik uygulamasıdır.

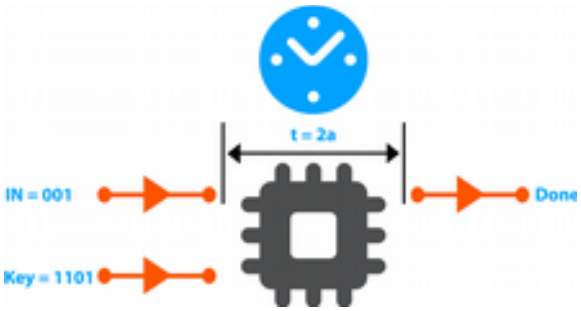


Tipik yan kanal saldırıları; Timing (Zamanlama) ve Power Analysis (Güç Analizi) saldırılarıdır. Şimdi bunlar hakkında kısaca bilgi verelim.

Timing (Zamanlama) Saldırısı

Zamanlama saldırısı, bir saldırganın, bir bilgisayarın veya ağ sisteminin güvenlik açıklarını, sistemin farklı girdilere ne kadar sürede yanıt vereceğini inceleyerek keşfetmesine olanak sağlayan bir güvenlik açıklamasıdır. Zamanlama özellikleri şifreleme anahtarına bağlı olarak değişir, çünkü farklı sistemler farklı girişleri işlemek için biraz farklı zaman alır.

Zamanlama saldırıları, OpenSSL kullanan akıllı kartlar ve web sunucuları gibi cihazları hedeflemek için de kullanılır. Paul Kocher isimli kriptocu, bu yöntemi kullanarak RSA şifresini kırmadan özel şifre çözme anahtarlarını bu yöntemle ele geçirebildi.



Power Analysis (Güç Analizi) Saldırısı

Yan kanal saldırılarının en güçlü ve yaygın modellerinden bir tanesi olan bu saldırı, bir şifreleme işlemi sırasında bir sistemin güç kaynağı tarafından sızdırılan bilgileri kullanarak şifre kırma işlemini gerçekleştirir. Güç analizi saldırıları, anlık güç tüketimi ile şifreleme uygulamasının dahili durumu arasında bir ilişki bulmaya çalışır. IoT sistemimizde bulunan donanımlara bu şekilde yapılan saldırılar ile AES şifresinin kırılması sağlanabilir. Bundan dolayı bu gibi saldırılara karşı donanımları güvence altına almamız gerekiyor.

Güvenlik Analizi ve Tehdit Algılama

Tehditlerin algılanabilmesi için iyi bir güvenlik analizine ihtiyacımız vardır. IoT üzerinde yapılan her bir güvenlik analizi gelecekte gelmesi muhtemel tehlikelerin engellenmesinde veya algılanmasında hayati öneme sahiptir. Burada dikkat edilmesi gereken husus; tehlikelerin algılanabilmesi için yapay zeka sistemlerine ihtiyacımız olduğudur. Çünkü analiz sonucunda elde edilen veriler baz alınarak gelecekte olması muhtemel tehditlerin daha gelmeden engellenmesi bu şekilde sağlanabilir.

Arabirim Koruması (API)

IoT sisteminde bulunan aygıt veya cihazlara erişim sağlayabilmek için uygulamalara, dolayısıyla arayüzlere ihtiyacımız var. Arayüzler veya arabirimler ile IoT sisteminde bulunan donanımları kontrol edebilir, veri trafiğine müdahale edebilir hatta sisteme zarar dahi verebiliriz. Arayüzden meydana gelecek tehditleri ortadan kaldırmak ve engellemek için daha önce de bahsettiğimiz üzere veri alışverişi yapması gereken cihazların kimliğini doğrulama ve yetkilendirme yeteneği gereklidir. Bu sayede sadece yetkili cihazlar veri trafiğini yönetebilir ve sistemi yönlendirebilir. Aksi durumda herkese açık olan bir sistem birçok güvenlik açığının oluşmasına neden olacaktır.

Yazılım Güncellemeleri

Siber saldırı veya tehdit olarak tanımladığımız, sisteme zarar vermeyi hedefleyen tüm faaliyetler durağan olmayıp, yeni yöntemler kullanılarak geliştirilmektedir. Yani saldırılar tek bir yöntemle değil farklı yöntemlerle tekrar tekrar denenmektedir. Antivirüs sistemlerini geliştiren şirketlerde olduğu gibi geliştirilen yeni saldırı yöntemlerini tespit edip bunlara karşı yazılımsal önlemler alınması gerekmektedir. Sistemi kurup güncellemeler yapmadığınız zaman IoT sistemi ilaç olmaktan ziyade zehir olabilmektedir. Çünkü güvenlik açıklarının olduğu bir sistemi kimse kullanmak istemeyecektir.

IoT sistemlerde yazılımın önemi oldukça önemlidir. Ayrıca siber saldırıların büyük çoğunluğu sistemde yüklü olan yazılım veya yazılımlara yapılmaktadır. Bundan dolayı sistemde bulunan yazılımların güncel olmasına özellikle dikkat edilmelidir.

Şifrelenmiş Veriler

IoT sistemleri ağ tasarımı ele alındığında End-to-End, yani uçtan uca sistemleri olarak tanımlanmaktadır. Ayrıca sistemde bulunan cihazlar genellikle birbirinden uzak olmakta ve iletişim kablosuz bağlantılar ile sağlanmaktadır. Veri trafiği de cloud, yani buluttan aygıta veya aygıttan buluta olacak şekilde akmaktadır. Böyle bir sistemde giden ve gelen verilerin şifrelenmesi bilgi güvenliği açısından oldukça önemlidir. Veriler aygıttan buluta akarken, kablosuz bağlantı ortamına bırakılmadan önce kesinlikle şifrelenmelidir. Bu kural tersi durum için de geçerlidir. Verilerin güvenliğinde en son geliştirilen AES veya RES şifreleme algoritmalarının kullanılması sistem güvenliğiniz ve dolayısıyla veri güvenliği için çok önemlidir.

Sistem Geliştirme

IoT bir sistem, yazılım ve donanım bileşenlerinden meydana gelmektedir. Bu bileşenler yapı olarak heterojendir. Yani her bir yazılım ve donanım farklı bir amaca hizmet etmek için tasarlanmıştır. Basit sistemlerin güvenlik açıklarını kapatmak birkaç güncelleme veya donanım iyileştirmesi ile mümkünken bu durum IoT'de böyle değildir. Çünkü karmaşık sistemlerin bir araya geldiği IoT'de güvenlik açıklarını kapatmak için sürekli bir sistem geliştirme aktivitesi yapılması gerekir.

IoT sistemleri uçtan uca olduğu için her bir düğüm veya bileşenin sürekli bir denetime tabi tutulması, ortaya çıkan sonuçlara göre de bileşenin yazılım veya donanım olarak geliştirilmesi gerekir. IoT'de güvenlik açıklarını kapamak karmaşık olduğu için geliştiriciler bu konuda hala uygulamalar yapmaktadırlar. Burada esas olan hem donanım hem de yazılımın güvenli sistemlerde kritik bir öneme sahip olduğunu bilmektir.

IoT-Ignite Platformuna Genel Bakış

IoT-Ignite, ARDIC firmasının geliştirdiği projelerden biridir ve temel amacı IoT uygulamalar geliştirmeyi sağlamaktır. Ayrıca cihazların Ignite servislerine ve diğer cihazlara bağlanmasını, veri ve etkileşimleri güvence altına almasına, cihaz verilerinin işlenip eyleme geçilmesini ve uygulamaların çevrimdışı iken bile cihazlarla etkileşime geçmesini sağlamaktadır.

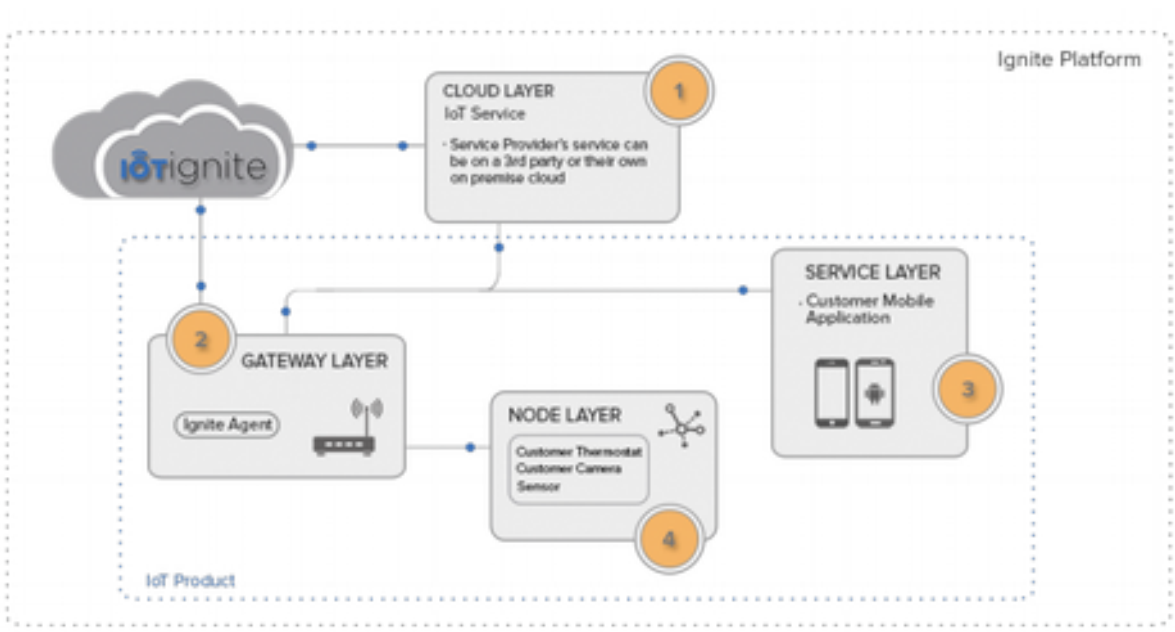
IoT-Ignite, IoT işlemlerini gerçekleştirmek için Platform as a Service (PaaS) olarak tasarlanmıştır. Bundan dolayı IoT dünyasına yeni bir yaklaşım getiriyor. Projenin amacı IoT'nin yaygın kullanımını sağlamaktır. Temel olarak kullanılan Gateway cebimizde bulunan akıllı telefonlardır. Özellikle Android yüklü cihazlara hitap etmektedir. Tasarımın arkasında modern ve eşsiz mimariler kullanılan bu sistem IoT uygulamaları için tüm imkanları sağlamaktadır.

Bu platformu önemli kılan sebepler bunlarla sınırlı değil. Bizim için önemli olan diğer iki sebebi şöyle açıklayabiliriz:

- Öncelikle yerli bir firmanın girişimi sonucu geliştirilmesi ve Türkçe dil desteğinin olması,
- İkinci olarak, Cisco, Dell ve Intel gibi firmaların 2015 yılında kurduğu OpenFog konsorsiyumu ile IoT işlemlerini Gateway seviyesinde gerçekleştirme girişimlerini, ARDIC firmasının 2012 yılında bu hizmeti IoT-Ignite ile insanlığa sunmasıdır. Bu durum ülkemiz için şüphesiz bir gurur kaynağıdır.

IoT-Ignite oldukça kullanışlı ve kolay bir platformdur. Sistem bizlere 5 Gateway'e kadar ücretsiz kullanım sağlamaktadır. Ayrıca veri yönetimi oldukça basittir.

Bu sistemin temel bileşenleri aşağıdaki gibidir.



Bunlar hakkında ayrıntılı bilgiler vereceğiz. Ancak böyle bir mimariye sahip olduğunu bilmeniz şimdilik yeterlidir.

BÖLÜM 2

IoT-Ignite

IoT-Ignite Hakkında

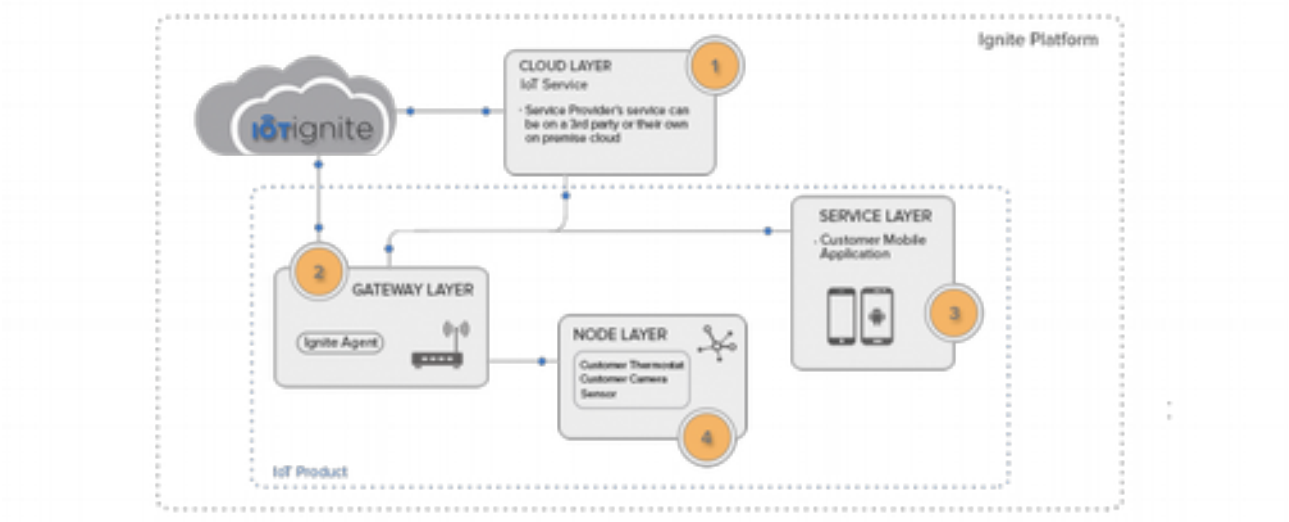
IoT-Ignite, cihazların Ignite Servisi'ne ve diğer cihazlara bağlanmasını, veri ve etkileşimleri güvence altına alarak, cihaz verilerini işleyip gerekli işlevlerin çalıştırılmasını ve uygulamaların çevrimdışıyken bile cihazlarla iletişime girebilmesini sağlayan bir IoT platformudur.

IoT-Ignite, bir Internet of Things projelerinin geliştirilmesi için etkili bir şekilde tasarlanmış, kullanımı kolay ve esnek bir hizmet platformudur (PaaS). IoT-Ignite, IoT dünyasında yepyeni yaklaşımlar getirerek geleceğin şekillenmesinde büyük bir rol üstlenmektedir.

IoT-Ignite'ın yaklaşımı; IoT'nin dar alanda odaklanmasını aşarak, daha geniş bir platformu uçtan uca yönetmeyi amaçlar. En güçlü IoT Kiti, cebinizdeki akıllı telefonlardır. Tasarımının arkasında modern bir mimari ile en iyi uygulamalar yer almaktadır. Kullanımı kolay kullanıcı arabirimleri ile geliştiriciler için en rahat ve en kolay geliştirme ortamını sağlar, üstelik bunu da ücretsiz yapar. Çoklu platform desteği ile istediğiniz kadar cihazı yönetebilmenin rahatlığını ve keyfini IoT-Ignite ile yaşayabilirsiniz.

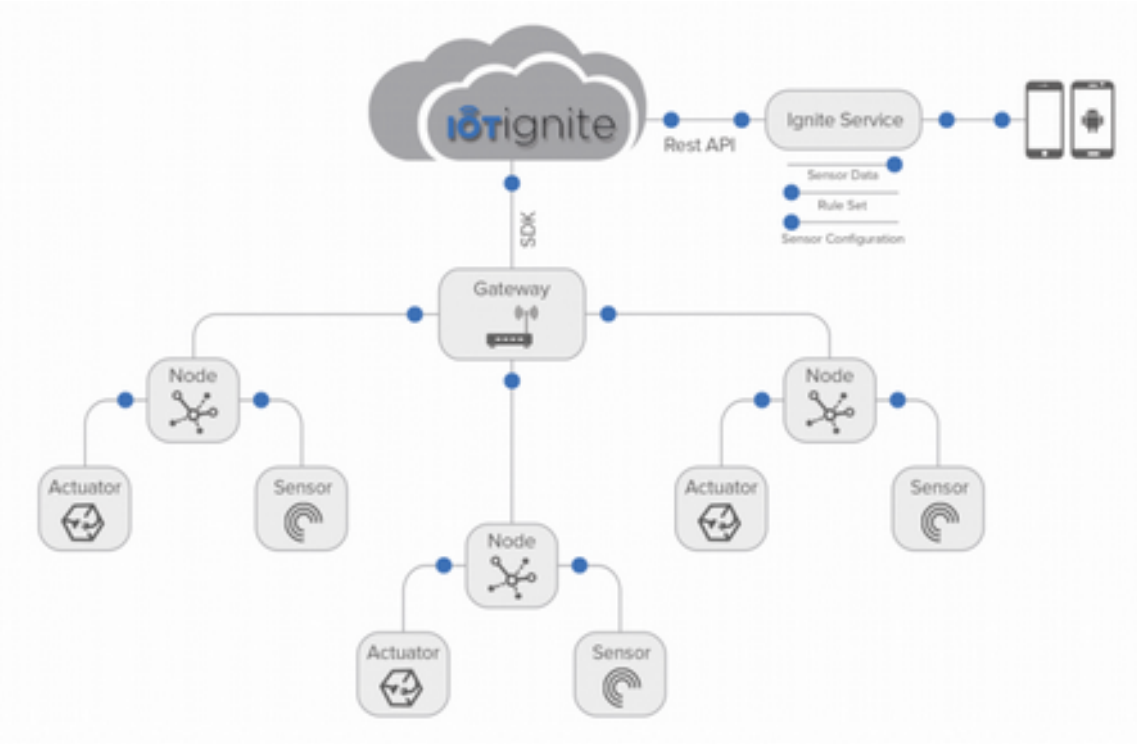
Servis Katmanları

Aşağıda yer alan şemada, IoT-Ignite platformunun dört temel katmanı; **Bulut**, **Ağ Geçidi, Düğüm** (**Node)** ve **Hizmet** katmanları yer almaktadır. Şimdi her bir katmanı kısaca tanımak amacıyla inceleyelim, ilerleyen bölümlerde her bir katman detaylıca anlatılacaktır.



Bulut Katmanı

Ignite-Cloud, IoT-Ignite platformunun **uçtan uca** hizmet etkinleştirici katmanıdır. Bulut katmanı, cihazlar (gateway, ağ geçici) veya harici kaynaklar tarafafından oluşturulan olaylara (events) dayalı basitten karmaşık kurallara (rules) geçiş yapma yeteneğine sahiptir.



- **Uçtan Uca Yaşam Döngüsü Yönetimi**
  IoT-Ignite, tüm hizmet yaşam döngüsü (life cycle) yönetimini geliştirme, uygulama, büyük ölçekli işlem yönetimi sağlar.
- **Cihaz (Gateway & Nodes)**
  IoT-Ignite; geliştirme, dağıtım, büyük ölçekli işlem yönetimini yönetir.
- **Kural Motoru (Engine)**
  IoT-Ignite Cloud'ta, değişen ihtiyaçları karşılamak için her zaman yeniden yapılandırılabilen bir kural (rules) altyapısı vardır. Hizmetiniz, toplanan verileri analiz eden Bulut katmanı ve belirlenen kurallara göre bir uyarı veya işlem yürütmek için kurallar dizisi oluşturabilir.

  Örneğin Gateway'e bağlı cihazlardaki (Edge Devices) sıcaklık sensörü ile alınan ortam sıcaklığı belirli bir derecenin üzerine çıkarsa yangın söndürme sistemini aç. Veya eğer sıcaklık belirli bir değerin üzerine çıkarsa yangın departmanına uyarı mesajı (sesli, yazılı) gönder.
- **Analitik**
  IoT-Ignite, Gateway'lerden ve Gateway'lere bağlı Node'lardan gelen her sensör verisini ve log kayıtlarını saklar, verilerini analiz edebilmenizi, gerçek zamanlı veya geçmişe dönük olarak verileri takip edip incelemenize olanak sağlar. Bununla birlikte PubSub sistemi ile verilerinizi başka Cloud platformlarına aktarıp senkronize edebilir, daha gelişmiş analizler yapabilirsiniz.
- **Entegrasyon ve REST API**
  IoT-Ignite platformu; 3. parti Bulut, Sosyal Ağlar ve Mesajlaşma Hizmetleri gibi her kesimden kullanıcıya açık bulut servisleri ile entegrasyon için REST API'leri sunar. Servisiniz bir 3. parti bulut kullanıyorsa, IoT-Ignite'ın REST API'larını kullanarak entegre edebilirsiniz. Örneğin; Twitter'da bir uyarıyı tweetlemek için bulutta bir kural belirlenebilir.

Gateway (Ağ Geçidi) Katmanı

IgniteCloud, IoT-Ignite platformunun uçtan uca hizmet etkinleştirici katmanıdır. IoT hizmetlerine bağlantı kurulabilmesi, yönetim, denetim ve veri taoplama olanağı sağlamak amacıyla **IgniteAgent** yazılımı kullanılır. Gateway'ler ve Gateway'lere bağlı diğer cihazlar, Cloud servisleriyle birçok iletişim protokolleri kullanarak hızlı, güvenli ve verimli bir şekilde iletişim kurar. Gateway ve cihazların sensörlerle algıladığı çevre şartlarına göre ihtiyaçları karşılamak adına, istenilen her zaman için yeniden yapılandırılabilen, Bulutta çalışan bir Rule Engine (kural motoru, sihirbazı) bulunmaktadır. Hizmet ve servisleriniz, toplanan verileri analiz eden Cloud katmanı ve tanımlanan kurallarla veya kurallar zinciri ile çok daha karmaşık durum analizi yapabilir, Gateway çevrimdışı olsa bile cihazlar kendi kendisini daha önce tanımlanan kurallara göre yönetebilir.

x86 mimarisi içeren bir cihaz veya **Open Hardware** (Rasperry Pi) aygıtları, IoT-Ignite platformu projelerinizde bir Gateway olarak kullanılabilir. Bu cihazlara **PilarOS-ARDIC Endüstriyel Android** versiyonun kurulması yeterlidir. PilarOS, buluta bağlanabilmek ve cihazı bir Gateway'e dönüştürecek bütün bileşenleri barındırır.

Ayrıca Gateway olarak kullanmak için bir Android cihazınızı (akıllı telefon, tablet) da kullanabilirsiniz. Android cihazları bir Gateway olarak kullanmak fikri IoT-Ignite tarafından ortaya atılmış ve başarılı bir şekilde de ürünleştirilerek servis edilmiştir. Herhangi bir Android işletim sistemli cihazı Gateway olarak kullanabilmek için cihaza **IgniteAgent**'ın yüklenmesi gerekmektedir. IgniteAgent, **Google Play Store**'da ve **http://iotapp.link**'te yer almaktadır. Bu konuyla ilgili ilerleyen bölümlerde çalışacağız.

Hizmet Katmanı

Eğer hizmet veya servisinizin bir Mobil Uygulama'ya ihtiyacı olduğunu düşünüyorsanız IoT-Ignite SDK ile ihtiyaçlarınızı karşılayacak seviyede arabirimler geliştirebilirsiniz. **IoT-Ignite SDK**; IgniteCloud ile bağlantı kurmak, kimlik doğrulaması, mesaj alışverişi, veri aktarma ve cihaz (edge gateway) işleme, birim saklama ve güvenlik sağlar.

Düğüm (Node) Katmanı

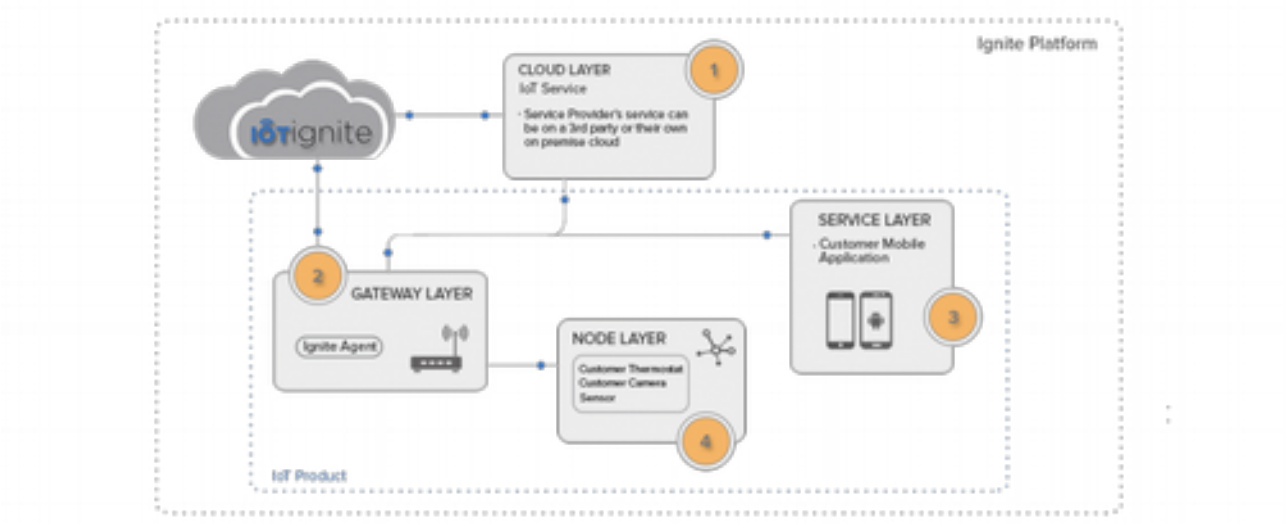
Node; Open Hardware olarak bilinen Arduino, NodeMCU gibi cihazlardır. Bunlar, çevrelerinden sensörler (sensör ve/veya actuator) ile verileri toplayıp IgniteCloud ile çift yönlü iletişim kurabilir. Node'lar için Eclipse, Arduino Sketch, Arduino IDE'ler ile sensörler (sensor: sadece veri gönderir, actuator: veri gönderir veri alır) için modüller geliştirebilirsiniz.

Servis Kavramı

Kısaca incelemiş olduğumuz IoT-Ignite katmanlarını anlayıp birbiri ile nasıl ilişkileri olduğunu kavramak gerekir. Genel olarak kabul görmüş ve IoT platformlarının dört temel bacağı bunlardır.

IoT-Ignite platformunu uçtan uca anlatan bu kitapla, orta seviyede bir yazılım geliştiricinin Android Gateway kullanarak çeşitli Node'larla nasıl bir IoT Servis'i oluşturacağını adım adım göreceğiz.

Tekrar IoT-Ignite mimarisine göz atalım...

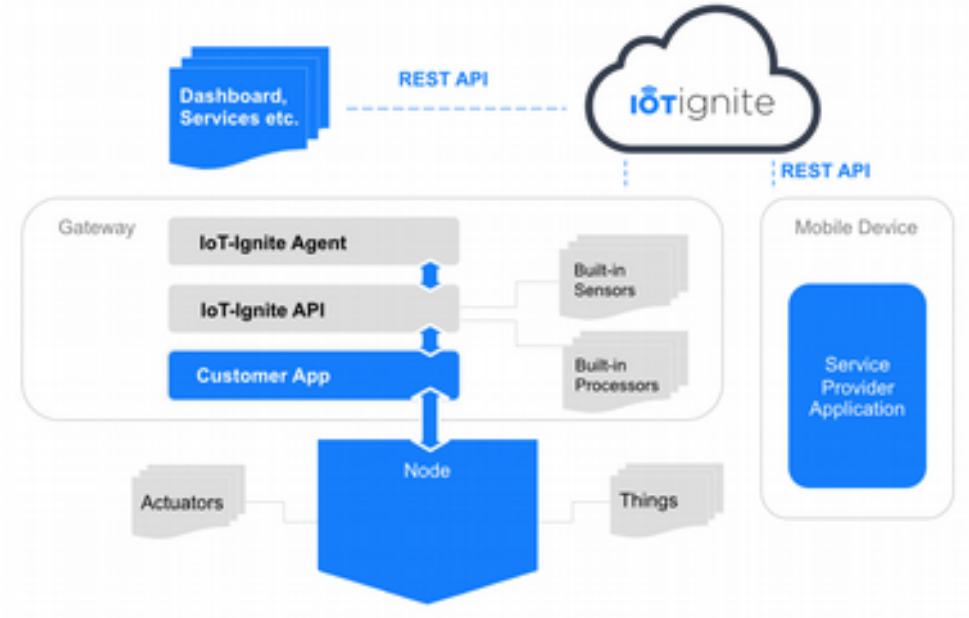


Bir IoT Servis'i temelde iki kısım olarak ele alınmalıdır. Bunlar; ağ geçidi ve bulut katmanlarıdır. Node ve Servis katmanları, aslında Gateway katmanının alt katmanları olmaktadır.

Gateway katmanında ister tek bir cihaz, isterseniz birden fazla cihaz yönetilebilir. Yönetimden kasıt; cihazlardan veri alınması (sensörlerden alınan veriler), cihazlara veri (yapılandırmalar, kurallar, dosyalar, uygulamalar gibi...) gönderilmesi, cihazın tek bir uygulamayı çalıştırdığı Kiosk moda getirilmesi, yeniden başlatılması gibi fonksiyonlar düşünülebilir. Eğer tek bir cihaz kullanılacaksa, o cihaz hem çevreden verileri toplayan bir algılayıcı, hem de verileri buluta göndermeden önce bir önişlemden geçirip daha temiz ve özet veriyi buluta gönderme aracı görevini üstlenmektedir. Eğer birden fazla cihaz ile çalışılacaksa, bir Android Gateway'e diğer cihazlar ağ üzerinden bağlanabilir ve yine tek bir Gateway üzerinden buluta bağlanılabilir. Bu yöntem, bulutla olan iletişim trafiğini hem azaltır, hem de enformasyon haline getirilmiş verinin işlenebilmesine ve daha doğru kararlarla cihazların yönetilebilmesine imkan tanır.

Gateway katmanında yer alan diğer alt cihazlar ve sensörler Node olarak tanımlanır ve Node katmanını oluşturur.

Gateway'in bulut ile bağlantısını kurmak, Node'ları Gateway'e bağlamak/kaldırmak, Node'larda yer alan sensörleri tanıtmak ve veri tiplerini yapılandırmak, hatta sensörlerden gelen verilerin anlık ve geçmişe dönük olarak gösterimini sağlayıp analiz edilebilmesi de Servis katmanında yer alır. Servis katmanında IgniteCloud API'ları kullanılarak kendi servisinize ait özel uygulamalar (mobil, masaüstü) geliştirerek servisiniz ihtiyaçlarınız doğrultusunda yönetebilir, çeşitli online (Cloud tarafında çalışan) veya offline (Gateway tarafında çalışan) kurallar ile cihazlarınızın davranışlarını yönetebilirsiniz. ,



En basitinden şöyle bir sistem düşünelim...

Müşterilerimize sunacağımız IoT servisimizin ismi: “Akıllı Kapı” olsun.

İlk olarak Gateway ve alt katmanı olan Node katmanına bakalım...

Normal bir apartman dairesi değil de, daha büyük bir ev düşünelim. Bu evin sokak kapısı, evin giriş kapısı, hatta her bir odanın kapısını akıllı hale getirmek ve müşterinizin bir Android uygulaması ile kapılarını açıp kapatabilmesini, saat 12:00 ile 06:00 arasında da otomatik olarak kapıların kendi kendine kapanmasını sağlayan bir hizmet ile yönetilsin. Her bir kapı üzerinde yer alan Rasperry Pi’lar birer Node olmaktadır. Rasperry Pi’ların verilerini yöneten de bir Gateway’imiz olsun ve tüm Node’lar bu Gateway’e TCPI/IP protokolü üzerinden bağlı olsun. Gateway üzerinde yer alan bir saat, anlık olarak sistem saatini okuyabilir ve Node’lara 0 (false) sinyali göndererek kapıların kapalı konuma gelmesini sağlayabilir. Bu sistem, çevrimdışı çalışan bir kuraldır (rule). Müşterinize de sunmuş olduğunuz bir Android uygulama ile de kişi evindeki kapıları bir liste halinde görebilir, istediği kapıyı da uygulama üzerindeki bir switch buton ile açıp kapatabilir. Bu da Servis katmanını olur. Servis katmanı ile Android APP’den gelen sinyaller önce Ignite Cloud’a gönderilir, orada işlenir ve müşterinin Gateway’ine gönderilir. Gateway’de de ilgili Node’un ilgili Sensör’üne (kapıyı açıp kapatan sensör, kilidi açıp kapatan bir servomotor) 1 (true) veya 0 (false) verilerini gönderir.

İşte en basitinden bir servis sunma mantığı bu senaryodaki gibidir. Şimdi de servisinizi hazırlarken kullanacağınız geliştirme ortamına, yani, **IoT-Ignite Devzone**’a başlayalım.

BÖLÜM 3

Devzone
Ücretsiz IoT Geliştirme Platformu

Devzone Nedir

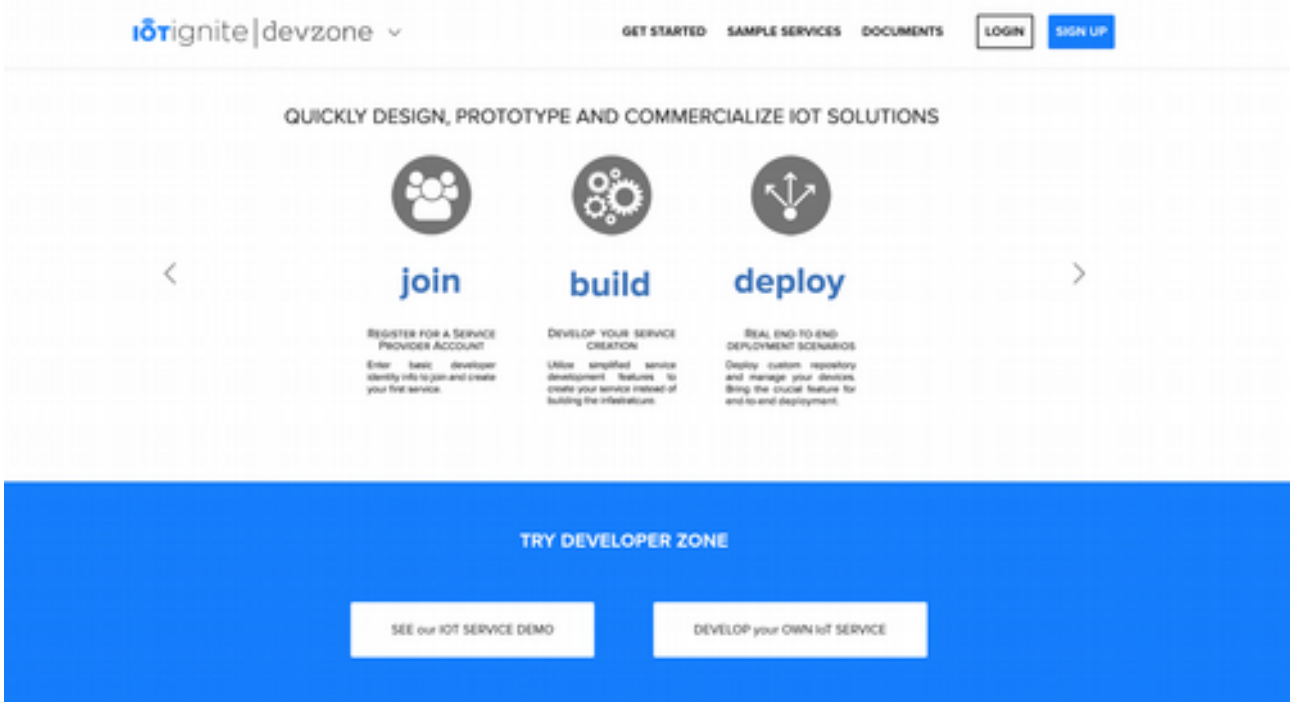
Devzone (Development Zone, Geliştirme Bölgesi) bir IoT-Ignite ürünüdür. IoT platformu için hizmet geliştirmek isteyen tüm geliştiricilere açık bir geliştirme ortamıdır, ücretsizdir.

Devzone'a üye olarak, IoT-Ignite platformunda Gateway ve Node'larınızı register (lisanslama) yapabilmeniz için ücretsiz 5 adet lisans hesabınıza tanımlanır. Bu lisanslar ile Android cihazlarınızı, Arduino, Raspberry Pi 3, NodeMCU veya MQTT kullanarak cihazlarınızı lisanslayabilir ve bir Gateway'e dönüştürebilirsiniz. Sonrasında ise lisansladığınız cihazlar ile kendi servisinizi geliştirme ortamında istediğiniz gibi kolayca geliştirebilirsiniz. Cloud ve/veya Gateway Rule'lar kullanarak cihazlarınızın çeşitli çevresel şartlarda nasıl davranışlar sergileyeceğini tanımlayabilirsiniz.

Devzone, iki ayrı kısımdan meydana gelmiştir. Bunlar; Demo ve Development alanlarıdır.

Devzone'a aşağıdaki bağlantıdan ulaşabilirsiniz.

https://devzone.iot-ignite.com



Demo, adından da anlaşılacağı üzere IoT-Ignite'ı hızlıca tanıyabilmek ve neler yapabildiğini basitçe anlayabilmeniz için IoT geliştiricilerinin kendi Android akıllı telefonları ile sanal sensörler üzerinden simülasyon yapabilmesini sağlar.

Development kısmı ise, artık doğrudan gerçek sensörler ile IoT uygulama geliştirilmesine olanak tanır. Toplamda 5 adet Gateway lisanslayabilir, her bir Gateway'in detaylarını görebilir, içinde yer alan Built-in Processors (Yerleşik İşlemciler), Built-in Sensors (Yerleşik Sensörler) Virtual Sensors (Sanal Sensörler) ve geliştiricinin kendi tanımladığı sensör gruplarını da görebilmenizi, bu sensörleri yapılandırabilmenizi, Cloud / Gateway kuralları tanımlayabilmenizi, çeşitli Debug (Hata Ayıklama / Log) araçları ile analiz işlemleri yapabilmenizi ve veri akışını izleyip raporlayabilmenizi sağlar.

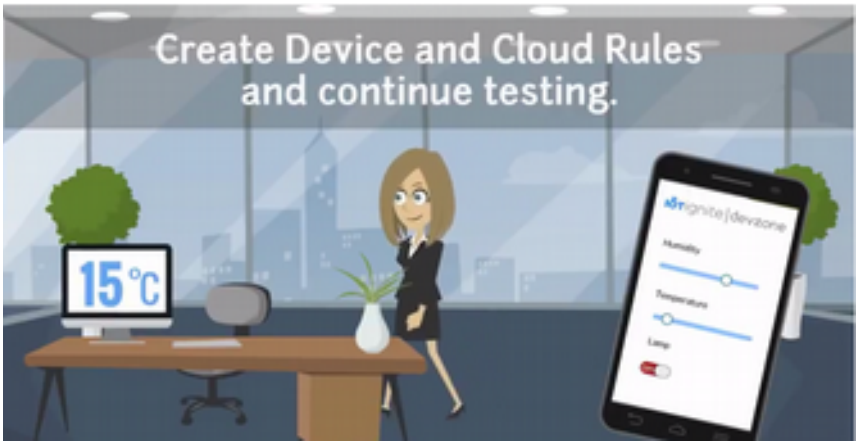
İlk olarak Demo adımı ile başlayalım...

Adım Adım Demo

IoT geliştiricilerinin Android işletim sistemli akıllı telefonları ile sanal sensörler üzerinden IoT-Ignite ve IoT Cloud'u tanıyabilmeleri ve test edebilmeleri amacıyla izledikleri ve birebir uygulayabildikleri bir süreçtir.

Demo Senaryosu ve Akış Hakkında

Demo akışında, ilk işimiz olarak Devzone'a ücretsiz olarak üyeliğimizi gerçekleştirmemiz gerekiyor. Üyeliğimiz ile birlikte otomatik olarak IoT-Ignite'ta hem son kullanıcı (end user) hem de sistem kullanıcısı (system user) hesaplarınız otomatik olarak tanımlanır. Sistem kullanıcısı hesabınıza da ücretsiz olarak 5 adet lisans tanımlanır. Demo akışına başladığımızda ilk olarak bir tanıtım videosu bizi karşılar ve süreç hakkında bir sunum yapar.



Sonrasında ise Android cihazınıza IoT Agent yazılımının yüklenmesi gerekmektedir. Yükleme işleminden sonra IoT Agent yazılımı açılır ve bu yazılım içinde akıllı telefonunuzun kamerasını kullanarak bir QR Code okutulması istenir. Böylelikle cihazınız lisanslanmış olur. Kısa bir süre (en fazla 1 dakika) sonunda da Demo akış ekranında bir uyarı penceresi ile cihazınızın lisanslandığı bilgisi görüntülenir. Hemen ardından otomatik olarak bir sonraki aşamaya geçilir. Cihazınıza yüklediğiniz IoT yazılımı, Virtual Demo APP adında bir uygulama indirir. Virtual Demo APP, Android telefonunuzda sanal sensörler oluşturur. Bunlar; Humidity (Nem), Temperature (Sıcaklık) ve Lamp (Lamba) sensörleridir. Her üç sensör de sanal veri üretmek için kullanılır. Sanal veri üretmek için sensörleri tanımlayan araçları kullanabilirsiniz. Örneğin Humidity altında yer alan Slider (Sürgü)'ı sağa doğru çekerseniz, nem miktarının arttığını ekranda göreceksiniz. Buraya kadar normal, ancak IoT Agent yazılımı, nem sensöründen aldığı veriyi anlık olarak IoT Cloud'a gönderir. Gönderilen veri, yine anlık olarak Demo ekranında görülmektedir. Aynı şekilde sıcaklık ve nem sensörü (sadece aç/kapat yapılabilir) verileri hem cihaz ekranında hem de demoyu yaptığınız bilgisayar (veya ne kullanıyorsanız) ekranında görülecektir. Ayrıca Lamba sensörü, dışarıdan da veri okuyabilir durumdadır. Yani çift yönlü iletişim halindedir. Demo ekranında yer alan switch butonuna tıklayarak, Android cihazınızdaki sanal lambayı açıp kapatabilirsiniz.

Bir sonraki adımda da işin en zevkli kısmı olan Rules, yani Kurallar kısmına geçilir. Bu alanda hem Cloud hem de Gateway taraflı olarak kurallar oluşturabileceğiniz basit bir araç yer almaktadır. Tanımlayacağınız kurallara göre akıllı telefonunuz çeşitli tepkiler verecektir (ekranda uyarı mesajı çıkması, zil çalması gibi).

Son adımda ise, servisinizde yer alan Cloud ve Gateway kurallarının özeti ile birlikte Demo akışı süresince sanal sensörler ile üretmiş olduğunuz verilerinin geçmişe dönük analizini ayrı ayrı grafiklerle görebileceksiniz. Bu aşamada Demo akışı için kullanmış olduğunuz lisansınızı silebilir, IoT-Ignite ile bağlantısını koparabilirsiniz.

> Devzone; IoT-Ignite Platformu'nun, geliştiricilerin IoT uygulamalarını geliştirirken kullanabileceği araçların bir kısmını sağlamaktadır. Ancak çok daha fazla sayıda Gateway ile çalışılmak istendiğinde Devzone yetersiz kalmaktadır, zaten asıl görevi de bu değildir. Devzone, bir başka IoT-Ignite ürünü olan EHUB (Enterprise Hub)'ın Lite versiyonudur. EHUB; çok daha fazla kullanıcı, servis ve Gateway'in yönetilmesi için, gelişmiş araçlar ve paneller sunar. EHUB'ı ilerleyen bölümlerde inceleyeceğiz.

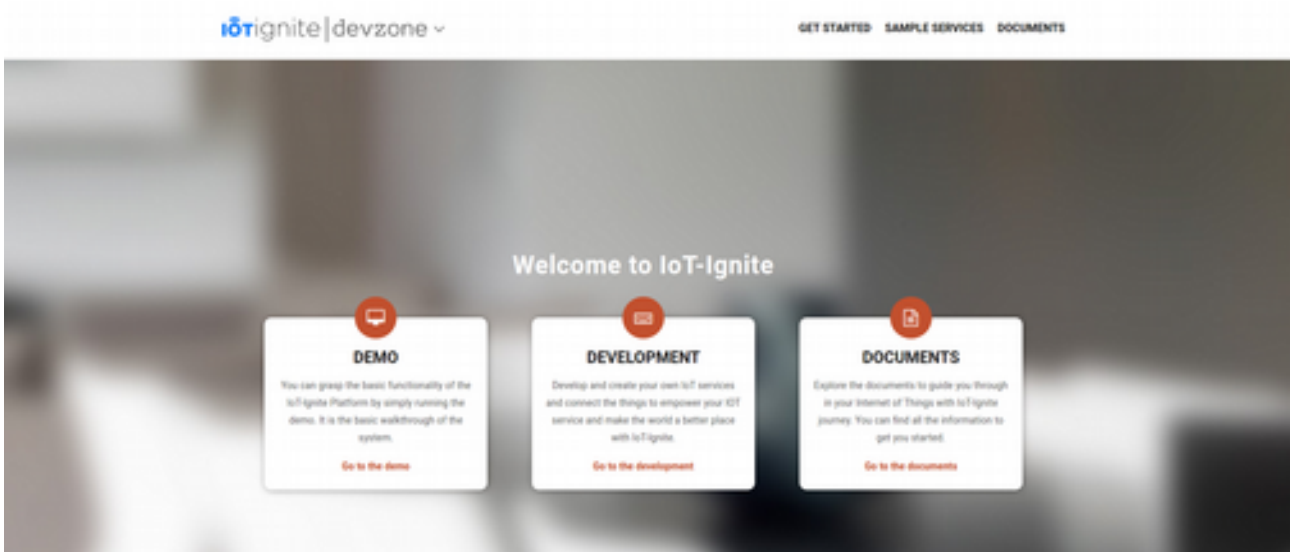
Android Akıllı Telefonun (Gateway) Platforma Kaydedilmesi

Demo akışımızda ilk adımımız IoT-Ignite platformunda bir üyelik hesabı açmaktır. Aşağıdaki bağlantıya girin.

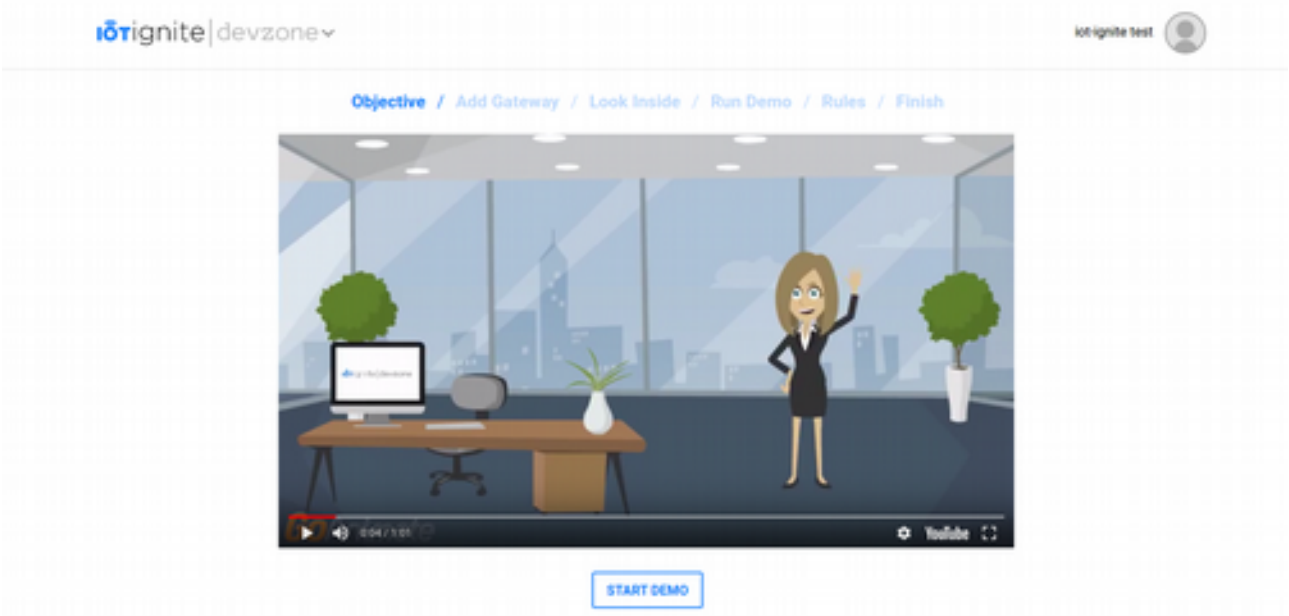
https://devzone.iot-ignite.com

SEE OUR IOT SERVICE DEMO (IoT Servis Demo'muzu Görün) butonuna tıklayın ve açılan sayfada da **Don't have an account?** (Hesabınız yok mu?) Butonunu tıklayarak üyelik formu doldurup **SIGN UP** (Kaydol) butonu ile de üyelik işleminizi tamamlayın.

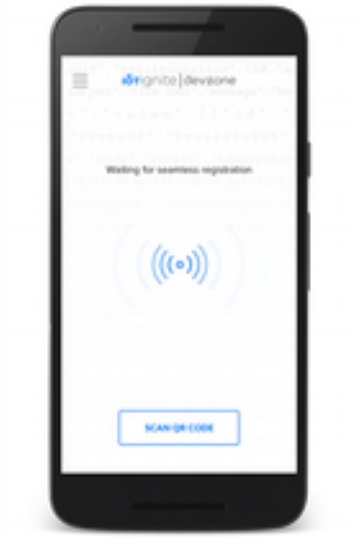
Üyelik işlemi tamamlandığında, aşağıdaki gibi bir ekran ile karşılaşacaksınız. **DEMO**'yu seçip ilerleyin.



DEMO’ya başladığınızda sizi **Objective** (Amaç) sayfası karşılayacak, burada DEMO akışı ile ilgili kısa bir video yer almaktadır. Video’yu izleyip DEMO süreci hakkında bir fikir edinebilirsiniz.



Video’nun altında yer alan **START DEMO** (Demo’ya Başla) butonu ile **Add Gateway** (Ağ Geçidi Ekle) sayfasına ilerleyin.

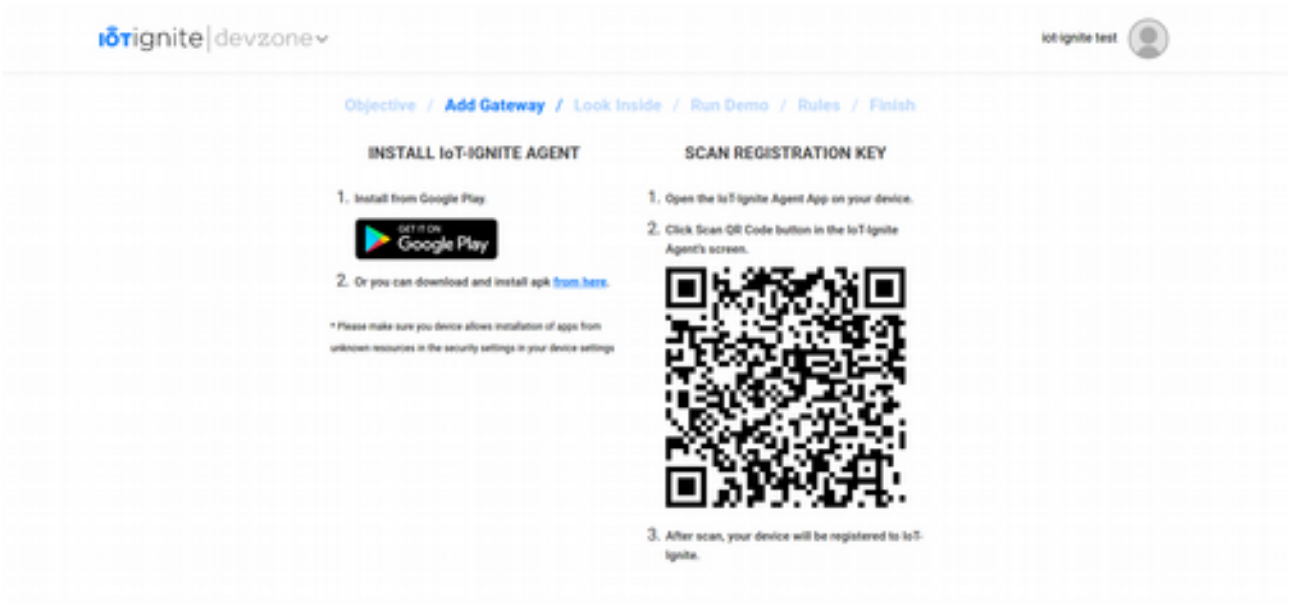


Objective sayfasında Android işletim sistemli akıllı telefonunuzu IoT-Ignite’a nasıl kaydedeceğiniz ve lisans alacağınız hakkında bir yönerge ile karşılaşacaksınız.

Yapmanız gereken ilk işlem **IoT-Ignite Agent**’ı akıllı telefonunuza yüklemek olacaktır. İster **Google Play**’den, isterseniz de manuel olarak **APK** dosyasını kendiniz **IoT-Ignite sunucularından** indirip telefonunuza yükleyin.

İkinci işlem ise; akıllı telefonunuza yüklediğiniz IoT-Ignite Agent programını açıp, ekranda sizin için özel olarak tanımlanmış QR Code’u okutmak olacak. Okutma işlemini yapmak için IoT-Ignite Agent’ı açtığınızda direkt olarak karşınıza çıkacak olan **Scan QR Code** (QR Kodu Tara) butonunu tıklayın ve kamerayı ekrandaki QR koda tutun.

Kod okunur okunmaz cihazınızdaki bilgiler IoT-Ignite Cloud’a gönderilecek ve cihazınız kaydolacaktır.



Lisanslama işlemi tamamlandığında (Yaklaşık 5-10 saniye kadar sürebilir), Android cihazınız artık bir Gateway'e dönüşecektir. Cihazınızın ekranında hesap bilgileriniz ve Gateway'e dönüşen cihazınızın birtakım bilgileri görülecektir.

Lisans tamamlandığında, otomatik olarak **IoT-Ignite Demo** programı indirip kurulacaktır. Bu program, içinde **temperature** (sıcaklık), **humidity** (nem) ve **lamp** (lamba) sanal sensörlerini barındırmaktadır.

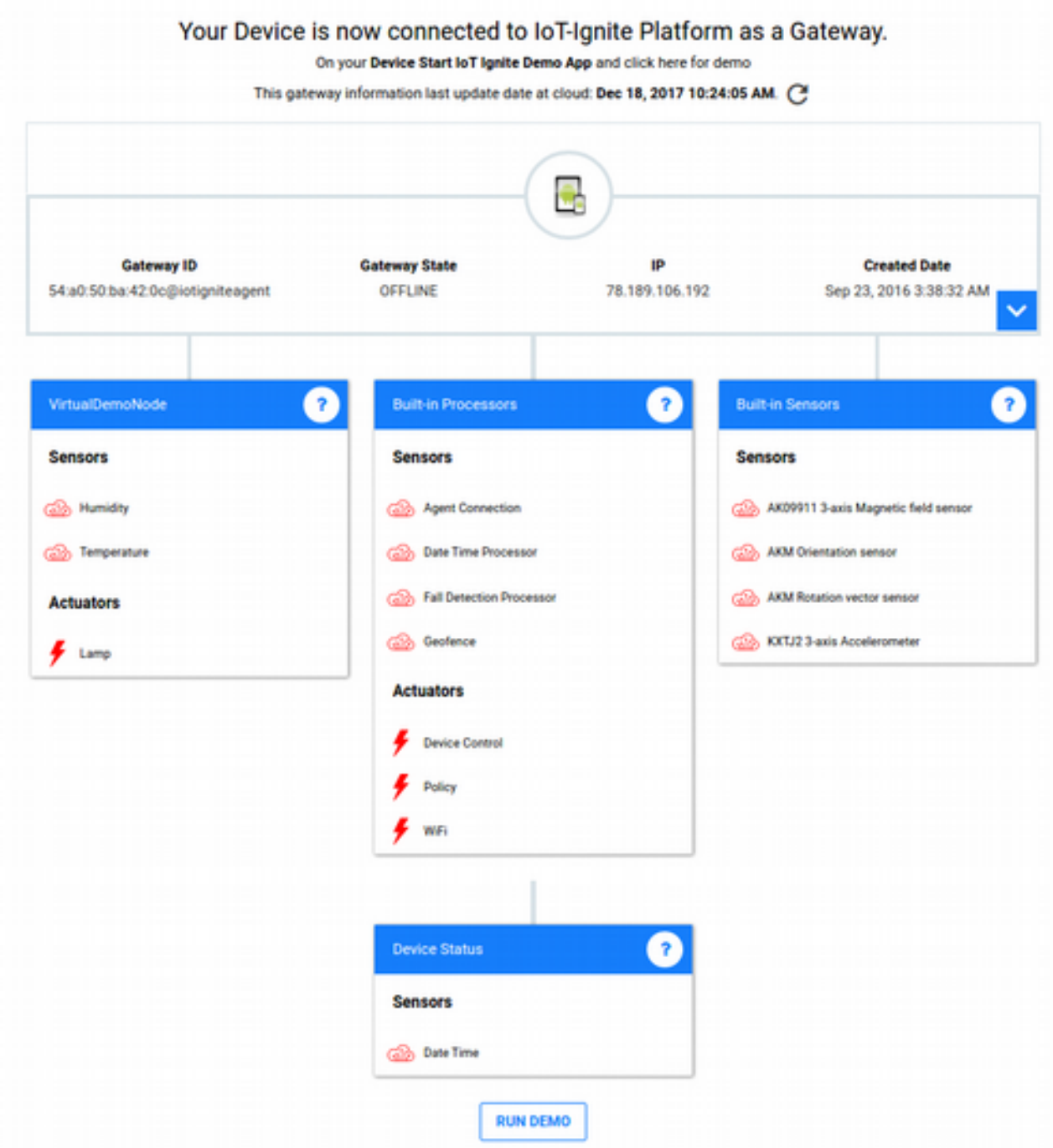
IoT Ignite Demo uygulamasında yer alan bu sensörlerdeki değerler gerçek değildir, sadece simülasyon yapabilmeniz için varsayılan değerlerle açılmıştır.

Şimdi bir de Gateway'de gerçekten var olan sensörleri görelim...

Cihazdaki Sensörlerin ve Actuatör'lerin Listelenmesi

Demo sayfasındayken üst menüden **Look Inside** (İçine Bak) butonuna tıklayın ve bir sonraki aşamaya geçin.

Look Inside sayfasındayken, aşağıdakine benzer bir ekran ile karşılaşacaksınız. Bu ekranda Android Gateway'in en son hangi tarihte sensör bilgilerini (hem sensör listesini, hem de çevrimiçi/çevrimdışı bilgileri) gönderdiği gösterilmektedir. Sensörler VirtualDemoNode, Built-in Processors, Built-in Sensors ve Device Status grupları altında toplanmaktadır. IoT-Ignite Demo App'de yer alan sensörler VirtualDemoNode altında yer almaktadır. Eğer daha sonraları yeni sensörler ve sensör grupları tanımlayacak olursanız, bu listede yine görebilirsiniz.

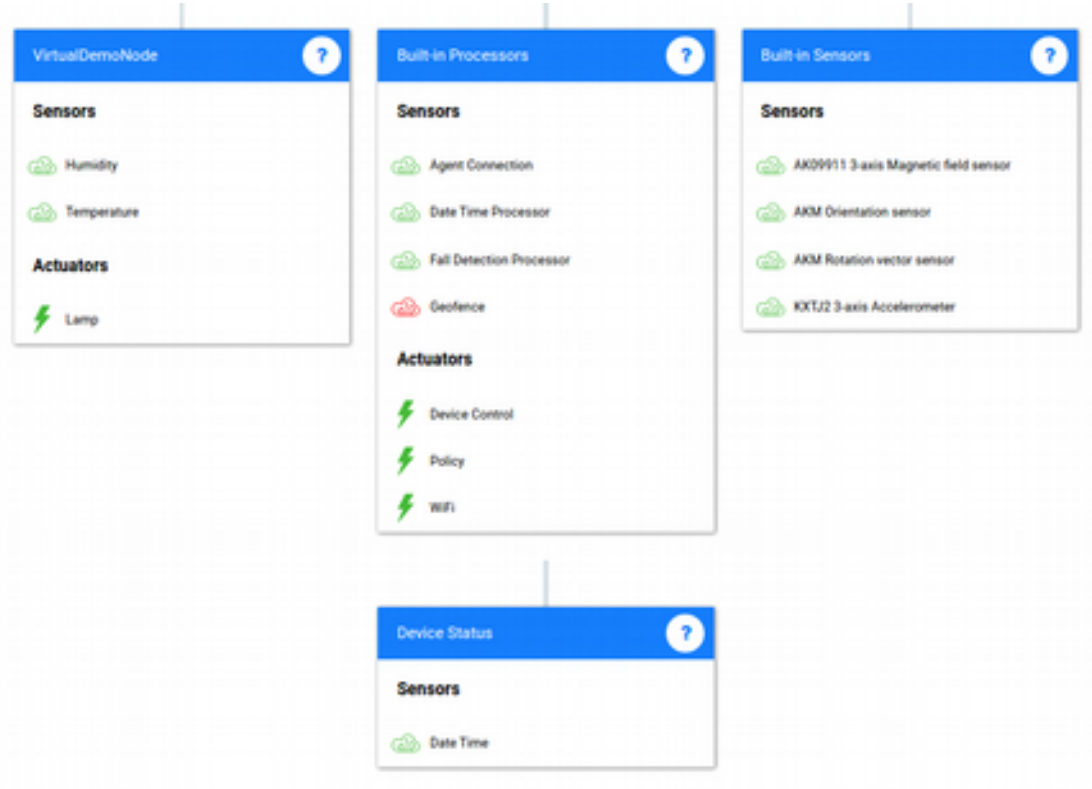


Ek olarak; Gateway'ın ID'sini (o gateway'e ait biricik numara), ağ durumunu, IP'sini, pil durumunu ve hafıza durumlarını da görebilirsiniz. Eğer Gateway'in yeniden envanter verilerini göndermesini istiyorsanız, ekranın sağ tarafında yer alan **Refresh** ikonuna tıklayarak Gateway'e bir

istek gönderebilirsiniz. Listede yer alan sensörlerden veri okuyabilmemiz için **ONLINE**, yani çevrimiçi olmaları gerekmektedir. Yukarıdaki ekran görüntüsüne bakıldığında kırmızı renkli görülmektedirler. Bunun anlamı, o an sensörler OFFLINE, yani çevrimdışı demektir.

Özellikle VirtualDemoNode altında yer alan sensörlerin çevrimiçi olabilmesi için Android cihazınızda ilk olarak IoT-Ignite Agent, sonrasında da Virtual Demo App'in açılması gerekmektedir. Eğer sensörler hala çevrimdışı görünüyorsa, cihaza envanter verisini tekrar göndermesi için istek gönderin. Yine de bir sorun yaşıyorsanız, Android cihazınızın internet bağlantısını kontrol edin.

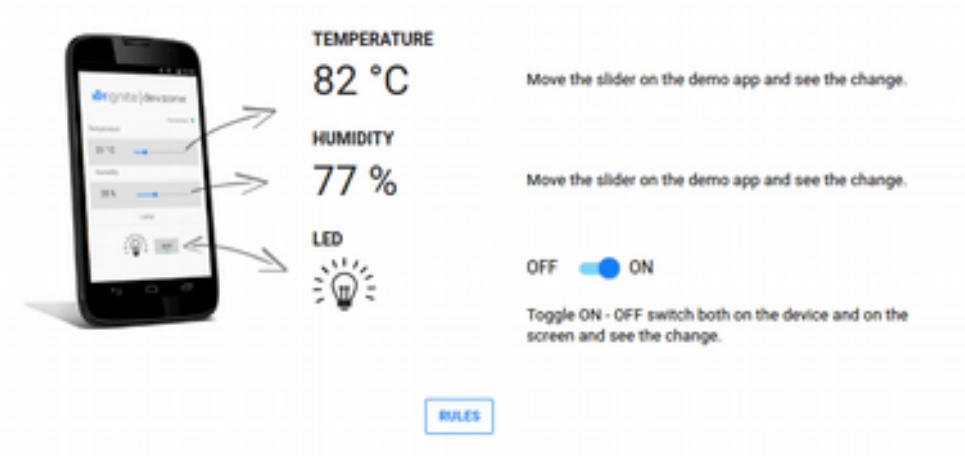
Cihaza yeniden istek gönderdikten hemen sonra aşağıdaki gibi yeşil renkli olarak sensörlerimizi görebiliriz.



RUN DEMO (Demo'yu Çalıştır) butonuna tıklayıp bir sonraki aşamaya geçelim...

Sanal Sensörlerin (Lamp, Temperature, Humidity) Gerçek Zamanlı Değerlerini Görmek

Run Demo ekranına geldiğimizde, Android cihazımızdaki sanal sensör değerlerini göreceğiz. Cihazınızdaki sanal sensör verilerini değiştirdiğinizde (sürgüleri sağa-sola kaydırarak veya switch butonunu açıp kapatarak), gerçek zamanlı olarak ekranda da veriler değişecektir.



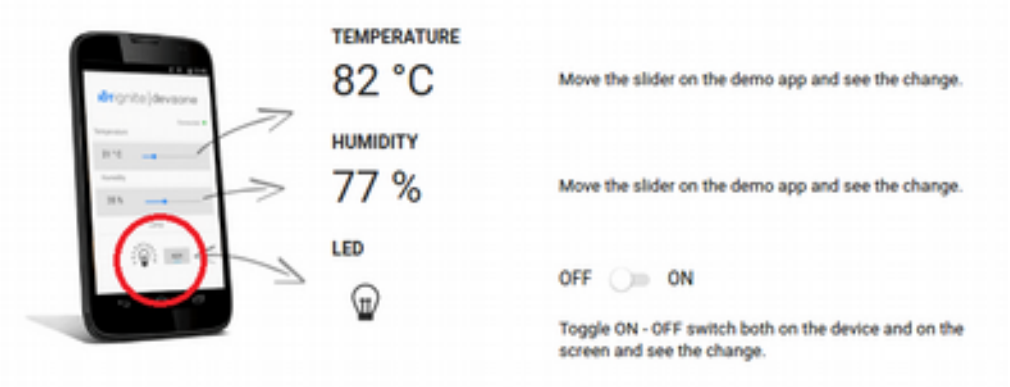
Sensör verilerinin her birinin değişmesinde, sensörün o anki verisi, sensör ismi ile birlikte IoT Cloud'a Gateway üzerinden gönderilir. Cloud'ta ise Websocket bağlantısı kurulur. Demo ekranında arkaplanda otomatik olarak bu Websocket'e bağlantı kurulur ve Gateway'inizden gelecek herhangi bir sensör verisi dinlemede beklenir. Eğer sanal sensörlerden herhangi birinden bir veri değişikliği bildirilirse, o veri arayüzde gösterilir.

Kullanıcı < > Demo App < > IoT-Ignite Agent < > Ignite Cloud < > Devzone Demo Page

Ek olarak, belki dikkatinizi çekmiştir; Look Inside sayfasında iken VirtualDemoNode altında Temperature ve Humidity birer Sensor, Lamp ise Actuator olarak gösterilmiştir.

Temperature ve Humidity birer sensördür, yani çevreden sıcaklık ve nem değerlerini belirli bir değer aralığında okur. Lamp ise bir eylem gerçekleştirir, çevreden veri almaz. Açık veya kapalılık durumu vardır, açık ve kapalı (true/false) değerleri vardır. Cihazınızdan Lamb sensörünü elinizle açabilir ve kapatabilirsiniz. Eğer bu işlemi elinizle yapabiliyorsanız, **Remote** (uzaktan) olarak **IoT-Ignite Cloud** üzerinden de yapabilirsiniz. Uzaktan Lamp sensörünü de yine Run Demo ekranında rahatlıkla yapabilirsiniz. Ekranda yer alan **OFF-ON switch** butonunu değiştirdikçe, Gateway'e "*Lamp actuator'ünü aç/kapat*" şeklinde emirler gider. Komutu alan Gateway, bu eylemi gerçekleştirecek olan sensöre true/false değerlerini iletir. Sonuç olarak da lamba açılır veya kapanır. Bu eylemi kendiniz de ekran üzerinden test edebilirsiniz.

Aşağıdaki ekran görüntüsünde, Lamp için OFF değeri gönderilmiş ve lambayı temsil eden ikonun ışıkları kaybolmuştur (Aşağıdaki şekil ile karşılaştırınız).



Sensörlerimizi incelediğimize göre artık Gateway'imizin davranışlarını kontrol etmek için **Cloud** ve **Gateway Rules** (Kurallar) oluşturmaya başlayalım. **RULES** butonuna tıklayın ve bir sonraki aşamaya geçin.

Sensörler için Kurallar Tanımlamak ve Kuralları Aktif-Pasif Yapmak

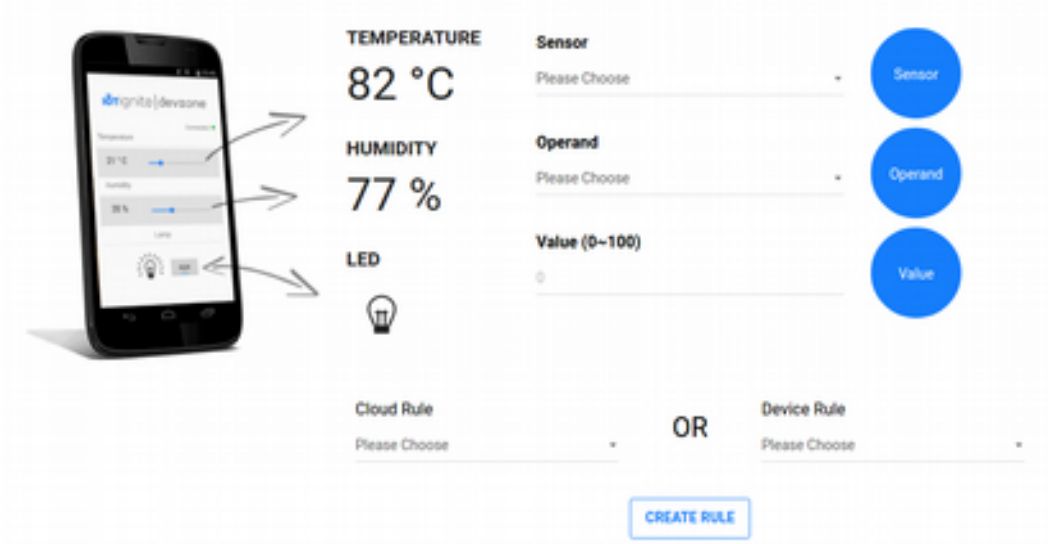
Öncelikle Rules (Kural) nedir, tanımlayalım...

Kurallar; belirli koşulların sağlanması durumunda belirli işlemlerin yapılması için tanımlanan ifadeyi belirtir. Kurallar sadece bir koşula bağlı olabileceği gibi, birden fazla koşula da bağlı olabilir. Ayrıca sağlanan koşul ile bir veya birden fazla eylem de gerçekleştirilebilir.

Demo akışımızda, sanal sensörlerimizi kullanarak birtakım basit eylemler gerçekleştiren kurallar tanımlayabiliriz. Kurallar iki farklı ortam için tanımlanır. Bunlar; Cloud ve Gateway.

Cloud kuralları; adından da anlaşılacağı gibi bulut üzerinde çalışır. Gateway'lerden gelen verileri alan Cloud sistemi, ilgili kural koşullarını sağladığında bunların eylemlerini yerine getirir. IoT-Ignite Cloud üzerinden Gateway ile ilişkilendirilmektedir. Bu kuralın çalışabilmesi için Gateway ile Cloud arasında network bağlantısının aktif olması gerekmektedir.

Gateway kurallarında ise; tanımlanan kural Cloud'ta çalışmaz, direkt olarak o kural tanımı Gateway'e Push (Gönderilir) edilir. Böylece Gateway ile Cloud arasındaki ağ bağlantısı kopmuş olsa bile, o kurala ilişkin şartlar gerçekleştiğinde kural kendiliğinden aktif olacaktır. Özellikle yoğun Gateway veri trafiğinin yaşandığı veya ağ bağlantısının sorunlu olduğu çalışma koşullarında Gateway'lerin bu kurallara göre karar verebilmeleri, IoT dünyası için büyük bir yenilik ve kritik bir gelişme olmuştur.



Yukarıdaki şekilde, Rules sayfasında yer alan **Kural Tanımlama Sihirbazı** görülmektedir. Oldukça basit olan bu araç ile aşağıdaki sıraya göre kural tanımlanır.

Sensör	**Operand (Operatör)**	**Values (Değerler)**	**Rules (Kurallar)**	
			Cloud Rule	**Gateway Rule**
Temperature Humidity Lamp	Less Than (Küçüktür), Equal (Eşit), More Than (Büyüktür)	0~100	Mail (E-Posta), Message (Mesaj)	Ring (Zil) Led Off (Lamba kapat) Led On (Lamba aç)

Bir kural tanımlanırken, tabloya göre soldan sağa doğru ilerleyecek şekilde mantık kurulur. İlk olarak bir sensör seçilir. İkinci adımda bu sensörün o anki değerinin başka bir değer ile karşılaştırma işlemi seçilir. Seçimler arasında küçüktür, eşittir ve büyüktür mantıksal operatörleri vardır. Seçilen sensör verisinin hangi değer ile mantıksal ilişki kurulacağı değeri de bir sonraki aşamada tanımlıyoruz. Humidity ve Temperature için 0-100 arası değer tanımlanırken, Lamp sadece açık ve

kapalı değerleri vereceği için 0 (kapalı) ve 1 (açık) değerleri karşılaştırma değeri olarak kullanılabilir. Mantıksal sorgu eğer geçerli ise; kural türü olarak belirlemiş olduğunuz Cloud veya Gateway kurallarından birini çalıştıracaktır (İlerleyen konularda göreceğimiz Devzone geliştirme ortamında daha karmaşık kurallar da tanımlanabilecek). Cloud kuralları arasında mail (e-posta) ve message (mesaj) bulunmaktadır. Mail; üyelik hesabı açarken tanımladığınız e-posta adresine veya belirttiğimiz herhangi bir e-posta adresine bir uyarı mesajı gönderecektir. Message de; Gateway'in, yani Android cihazınızın ekranında bir açılır pencerede uyarı mesajı yazdıracaktır. Dikkat edin; e-posta ve mesaj içeriği jenerik olduğundan, Cloud ortamında bu metnin düzenlenebilmesi gerekmektedir. Bu nedenle Cloud Rule olarak kullanılmış olması mantıklıdır. Gateway kurallarında ise ring, Gateway'de 5-10 saniye kadar bir uyarı sesi çalar. Led on ve Led off da, sanal sensörlerden Lamp'ı açıp kapatır. Bir kural örneği vermek gerekirse; "Temperature değeri 80'den yukarıda olursa Lamp sensörünü aç" gibi bir ifade oluşturmaktadır.

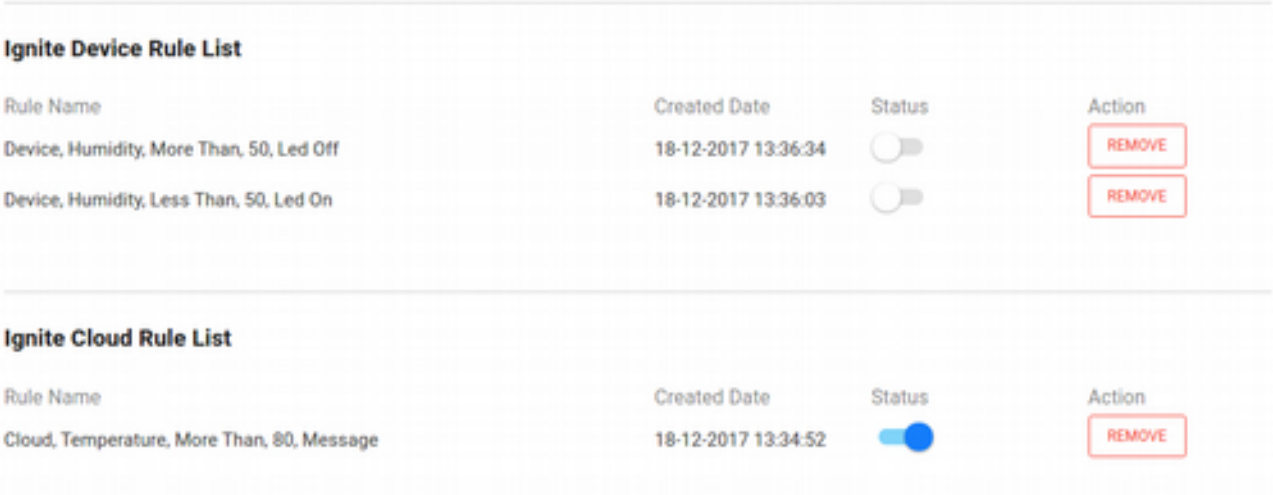
Rules ekranında, açılır menülerden kolaylıkla seçimlerinizi yapıp CREATE RULE (Kural Oluştur) butonuna tıklayıp kuralınızı rahatlıkla oluşturabilirsiniz. Örnek olması amacıyla aşağıdaki kuralları tanımlayalım.

1 adet Cloud Rule:

- Temperature (Sensör) değeri, 80'den (Values) büyük olursa (Greater Than) Message Cloud Rule'unu çalıştır.

2 adet Gateway Rule:

- Humidity (Sensör) değeri, 50'den (Values) büyük olursa (Greater Than) Led Off Gateway Rule'unu çalıştır.
- Humidity (Sensör) değeri, 50'den (Values) küçük olursa (Greater Than) Led On Gateway Rule'unu çalıştır.



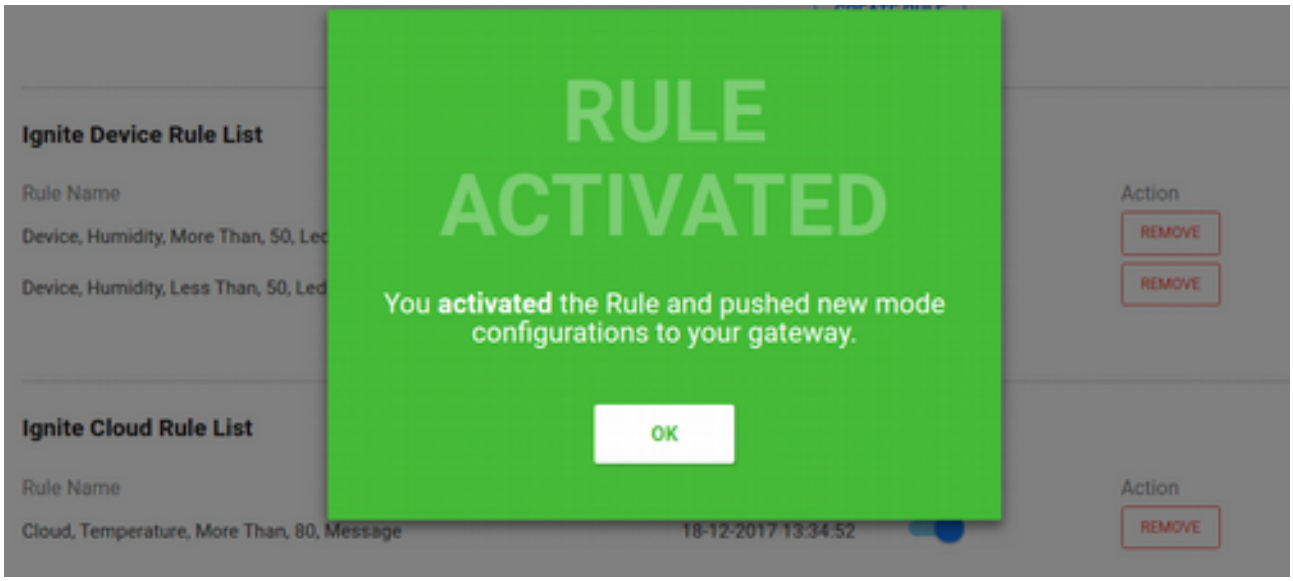
Bu kuralları oluşturduğumuzda, kurallarımız yukarıdaki şekilde görüldüğü gibi listelenecektir. Yukarıda yazmış olduğumuz ifadelere göre seçtiğimiz ayarlarla otomatik oluşturulan bir **Rule Name** (Kural İsmi) ile listeleniyor. Kuralların yanında oluşturulma tarihleri ile birlikte **Aktif/Pasif** yapma switch butonu ve o kuralı sileceğiniz **REMOVE** (Sil) butonları yer almaktadır.

Bir Cloud Rule oluşturduğunuzda, otomatik olarak **Status**'u **ON** durumundadır, yani aktiftir. Gateway Rule oluşturduğunuzda ise **OFF** durumu ile oluşturulur, çünkü henüz Gateway'e **Push** edilmemiştir.

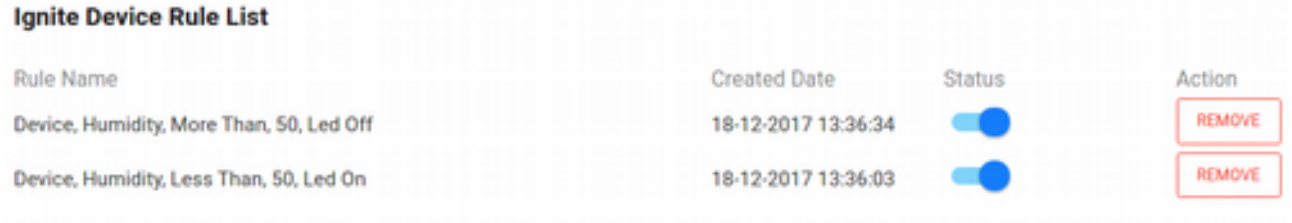
Bu aşamada Status'leri değiştirmeden Android cihazınızdaki sensör değerlerini değiştirin. Şu an sadece bir adet Cloud Rule'umuz çalışacaktır. Temperature değerini 80'den yukarıya getirdiğinizde ekranda aşağıdaki mesajı göreceksiniz.

Yukarıdaki şekilde, Android cihazda bir Message kuralı ile gösterilen mesaj yer almaktadır. Varsayılan olarak "*IoT-Ignite Cloud rule Triggered.*" mesajı gösterilmiştir. **OK** butonuna tıklanarak bu mesaj kapatılabilir.

Tekrar Rules ekranına dönelim ve **Ignite Gateway Rule List**'ten diğer **Gateway** kurallarının da **Status** değerlerini **ON** yapalım. Bu işlemi yaptığınızda, her bir kural için ekranda aşağıdaki gibi bir bilgi mesajı çıkacaktır. Bu mesaj, seçtiğiniz Gateway kuralı aktif edilerek Gateway'e yapılandırma ayarlarının gönderildiğini bildirmektedir. Tekrar pasif hale getirdiğinizde de Gateway kuralının pasif yapılıp tekrar Gateway'e gönderildiğini belirtecektir. Yani bir Gateway kuralı ilk oluşturulduğunda o anda Gateway'de tanımlı değildir, mutlaka aktif edilerek PUSH edilmesi gerekir. Kural tekrar pasif hale getirildiğinde, yine bu durumun da Gateway'e bildirilmesi gerekir. Eğer o an Gateway'in ağ bağlantısı yoksa, Gateway çevrimiçi olduğunda bu yapılandırmaları alacaktır. Gateway, yapılandırma aldığında, Android cihazın ekranında "*Updating device policies*" mesajı kısa bir süre görüntülenecektir.



Her iki Gateway kuralını da aktif ettik.

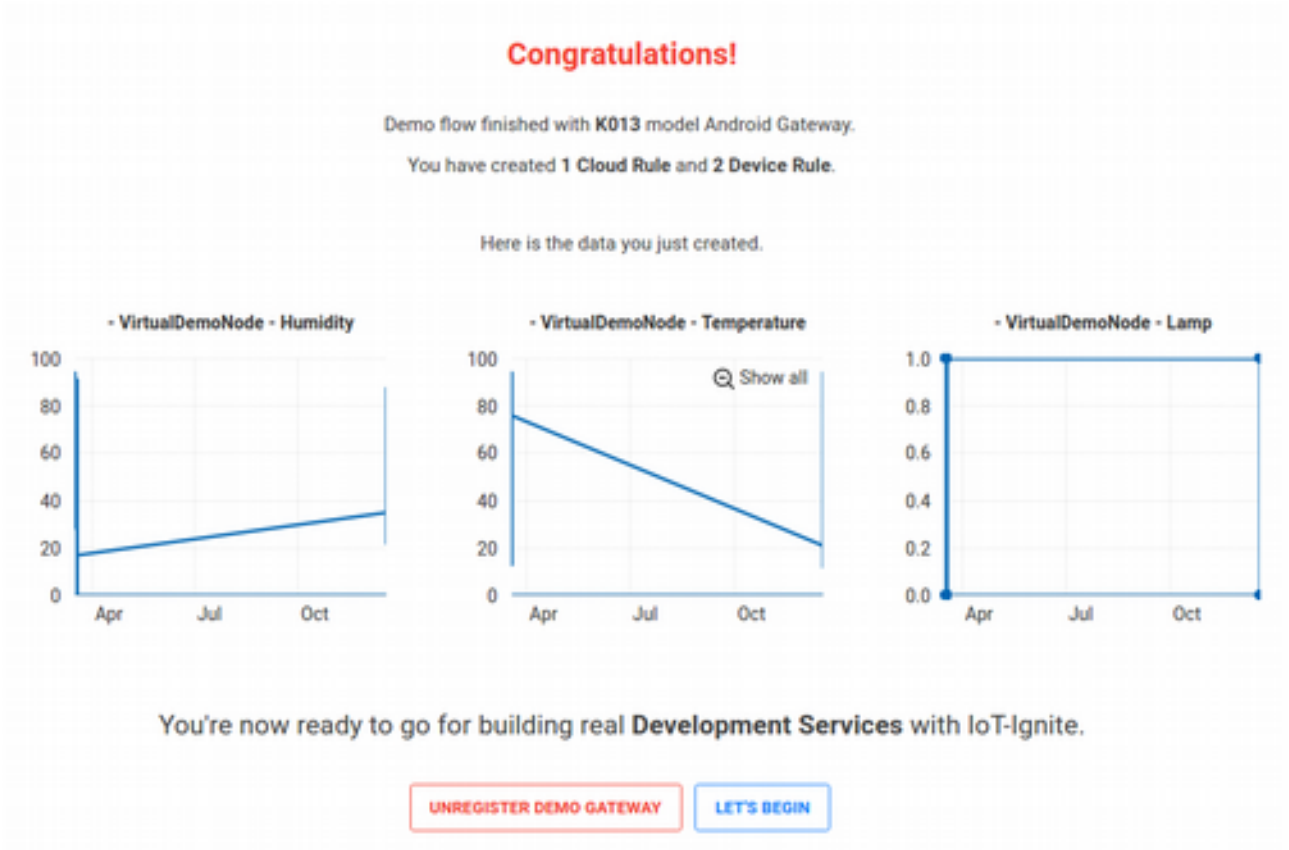


Tekrar Android cihazımıza dönelim ve Humidty değerini 50'den yukarı ve aşağı olacak şekilde birkaç kere değiştirelim. Humidity değeri 50'den yüksek olursa lamba kapanacak, 50'den düşük olursa da lamba açılacaktır.

Evet, Demo akışının sonuna geldik. Şimdi son olarak Demo neler yaptığımızın analizine bakalım. Üst menüden **Finish** butonuna tıklayın...

Raporlama ve Analiz

Finish ekranına geldiğimizde aşağıdaki şekilde görüldüğü üzere bir raporlama ekranı bizi karşılayacaktır. Bu ekranda, Demo akışımızın bittiğine dair bir mesaj ile birlikte kaç adet Cloud, kaç adet de Gateway Rule oluşturduğumuz görülmektedir. Hemen altında da üç adet grafik görülmektedir. Bunlar; Humidity, Temperature ve Lamp sensörlerimizden gelen verilerin geçmişe dönük verilerini göstermektedir.



Demo akışı süresince kullanmış olduğunuz Android cihazınızın IoT-Ignine Cloud ile bağlantısını koparmak, yani lisansı kaldırmak için **UNREGISTER DEMO GATEWAY** (Demo Ağ Geçidinin Kaydını Kaldır) butonuna tıklamanız yeterli olacaktır. İşlem tamamlandığında Android cihazınızda yine bir bildirim alacaksınız ve **IoT-Ignite Agent** uygulamasını açtığınızda tekrar **QR Code** okutma ekranı yeniden lisanslama işlemi için hazırda bekleyecektir.

Demo akışını ve analizi de tamamladığınızda, **LET'S BEGIN** (Hadi Başlayalım) butonuna tıklayarak, **Devzone Geliştirme Ortamı**'na yönlendirileceksiniz.

Devzone Geliştirme Ortamı

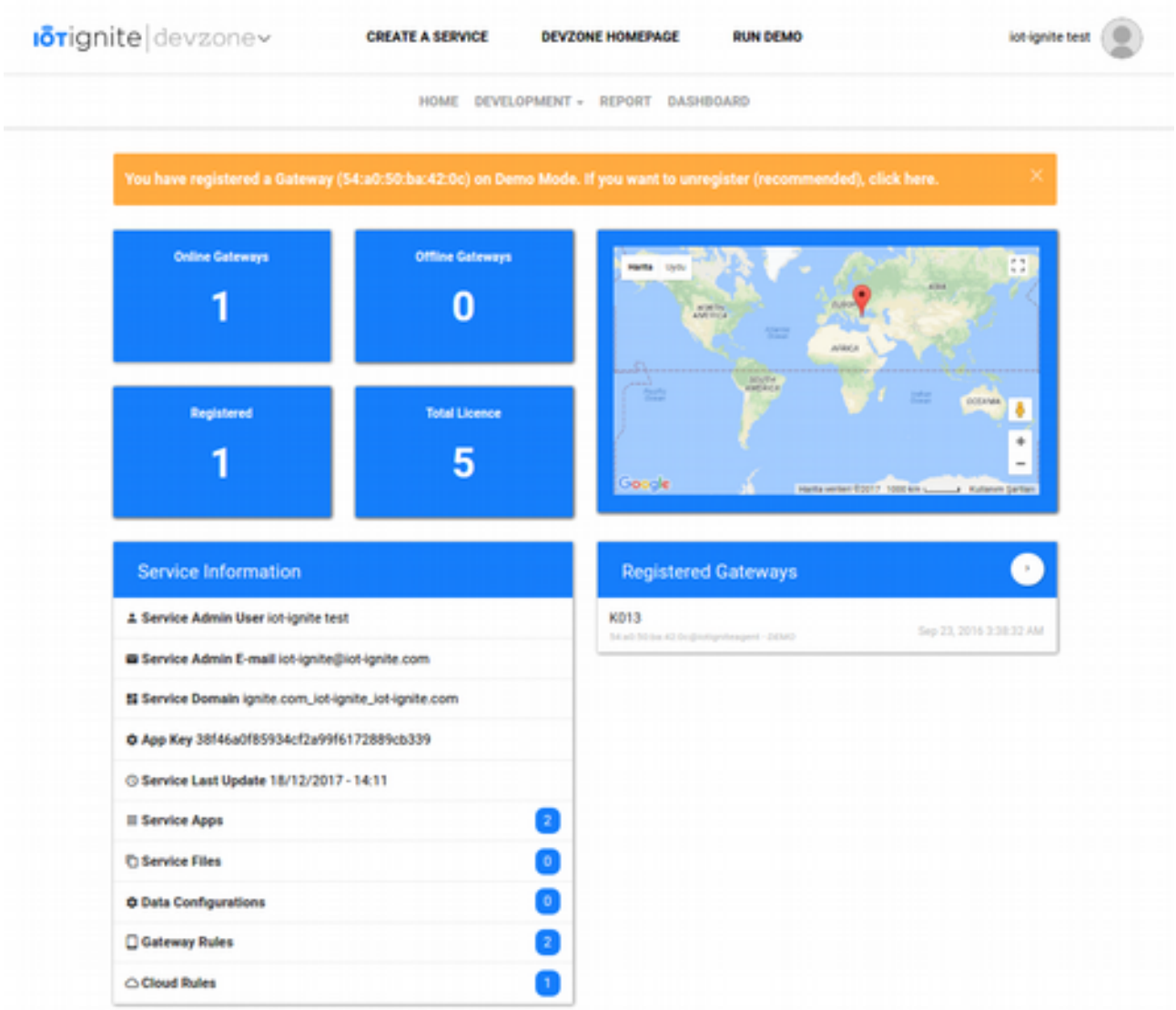
Devzone, **Dev**elopment **Zone** (Geliştirme Bölgesi) kelimelerinden türetilmiş, IoT-Ignite platformunu etkin bir şekilde kullanabilmeleri için geliştiricilere açık olan ücretsiz bir araçtır. Devzone, direkt olarak bir servis veya ürün geliştirmeye yönelik değildir, sadece geliştiricilerin kendilerine tanımlanmış olan 5 adet lisans ile bir veya birden fazla cihazı kullanarak IoT uygulamalar ve servis geliştirme sürecinde pratik yetenekler kazanması amacı ile kullanılmaktadır. Geliştiriciler Devzone'da yeterince pratik kazandıktan sonra, daha da gelişmiş olan IoT-Ignite ürünü olan **EHUB** aracı (Devzone'da yapabildiklerinize ek olarak çok daha fazla özellikler barındıran, tamamen ürün ve servis geliştirmeye yönelik servis geliştirme ve yönetim aracı) ile pratikte öğrendiklerini artık gerçek anlamda kullanmaya başlayabilir.

Dashboard Ekranının İncelenmesi ve Raporun Analiz Edilmesi

Devzone'a aşağıdaki URL'den giriş yapabilirsiniz.

https://devzone.iot-ignite.com/dpanel

Devzone'a giriş yaptığınızda, aşağıdaki gibi bir **Dashboard** ile karşılaşacaksınız.



Kısaca Dashboard'ta yer alan öğeleri inceleyelim...

Eğer Demo senaryosunda lisanslı bir cihazınız varsa, bu cihazın lisansını kaldırıp kaldırmayacağınız ile ilgili bir bilgi mesajı turuncu renkli kutuda en üst kısımda gösterilir. Kutu içinde yer alan **click here**'e (buraya tıkla) tıklanarak lisans kaldırılabilir. Demo senaryosunda lisansladığınız cihazınızın Modu, **DEMO**'dur. Devzone'da varsayılan olarak lisanslanan cihazların modu da **DEVELOPMENT** olmaktadır. Bu nedenle Devzone'da DEMO modunda bir cihazın olması pek de anlamlı değildir. DEVELOPMENT dışında daha farklı modlarla da geliştirme yapılabilmekte, hatta sonraki kısımlarda göreceğimiz gibi kullanıcı kendi modunu da tanımlayabilmektedir.

> **Policy (Politika) ve Mode (Mod) Nedir?**
>
> Cihazın kullanıcı tarafından hangi ayarlarının yapılabileceğini, hangilerinin değiştirilemeyeceğini, sistemde hangi uygulamaların kurulup/kurulamayacağını, uygulamaların hangi izinlere sahip olabileceğini, cihaza hangi arabirimlerden erişilebileceğinin, vs. kuralları Policy'yi (Politika) oluşturur. Mode ise içinde bir veya birden fazla policy bulunduran, bunun dışında uygulama, sertifika, içerik (content), servis (Uygulama Marketi, İçerik Marketi, Uzaktan Yönetim ve Loglama, vb.) ve ayarların (wifi, e-mail, vpn, bookmark, vb.) bulunmuş olduğu tanımların bütünüdür.
>
> Bir cihaz aynı anda birden fazla politikayı içeren bir mod alsa da sadece tek bir politikayı aktif

> halde tutabilir. Bir mod'un birden fazla politikayı içinde bulundurması farklı durumlarda bu politikalar arasında geçiş yapmasını sağlamaktadır. Örneğin: Okul ağında bulunan bir öğrenci cihazı sadece okul uygulamalarına izin veren bir cihaz politikasına geçirilebilir, okul ağından çıktıktan sonra da ev politikasına geçerek eğlence uygulamalarını da cihazın kullanımına açabilir.

IoT-Ignite'a kaydolan her geliştirici için ücretsiz **5 lisans** hakkı tanındığından bahsetmiştik. Şu an Demo'da bir cihazımız olduğu için **Registered** (Kayıtlı) adlı mavi kutucuk içerisinde 1 yazmaktadır. **Total Licence** (Toplam Lisans) ise 5 olarak sabit kalmaktadır. Gateway bir ağa bağlı olduğu için **Online Gateways** (Çevrimiçi Ağ Geçitleri) kutucuğunda da 1 görülmektedir. Çevrimdışı bir Gateway olmadığı için de **Offline Gateways** (Çevrimdışı Ağ Geçitleri) kutucuğunda 0 yazmaktadır. Hemen sağ tarafta yer alan mini haritada ise, o an hangi Gateway'ler çevrimiçi ise haritada işaretli olarak gösterilmektedir (Eğer Gateway'in GPS desteği varsa).

Haritanın hemen altında ise Registered Gateways (Kayıtlı Ağ Geçitleri) yer almaktadır. Bu alanda, kayıtlı olan Gateway'ler model ismi, tenant (kiracı) ismi, aktif olan Mod ve lisanslanma tarihi bilgileri ile listelenmektedir. **K013** model isimli, **54.....0c@iotigniteagent.com** tenant isimli, **DEMO** modunda ve **23 Eylül 2016**'da lisanslanmış cihaz görülmektedir.

> **Tenant, System User ve End User Nedir?**
>
> IoT-Ignite'ta lisanslanmış her bir Gateway, aslında bir müşteri/kiracı (tenant) hesabı altında tanımlanır. Tenant altında gateway'ler dışında o tenant'a ait sistem kullanıcıları, uç kullanıcılar (end-user), çözüm içinde kullanılacak uygulama ve içerik dosyaları, konfigürasyon bilgileri, kurallar, mod ve politikalar, kullanılan servisler vb. bulunur. Tenant, bir ana sistem kullanıcısının yönetimi altındadır.
>
> Devzone'da üyelik hesabı açtığınızda, IoT-Ignite platformunda sizin adınıza öncelikle bir tenant tanımlanır, bu tenant altında hem System User hem de End User hesapları otomatik olarak aynı isimde açılır. Lisanslamış olduğunuz her bir Gateway ise End User'ın hesabına bağlı olur.

Sol alt kısımda ise **Service Information (Servis Bilgisi)** başlığı altında servisinize ait bilgilerin özetini görebilirsiniz. Bunlar;

- **Service Admin User**: Servisi başlatan yöneticinin hesabı, ad ve soyadı.
- **Service Admin Email**: Servisi başlatan yöneticinin e-posta adresi.
- **Service Domain**: ignite.com_ prefix (ön ek) ve yönetici e-posta adresi ile birleştirilip türetilerek tanımlanmış olan etki alanı.
- **App Key**: Geliştirme yaparken kullanılacak olan, hesaba tanımlı özel bir anahtar.
- **Service Last Update**: Servisin en son güncellenme tarihi.
- **Service Apps**: Serviste uygulama havuzunda hazır olarak bekleyen uygulama (.apk) sayısı.
- **Service File**: Serviste dosya havuzunda hazır olarak bekleyen dosya (resim, metin, ses vb) sayısı.
- **Data Configuration**: Serviste yer alan veri yapılandırmaları sayısı.
- **Gateway Rules**: Serviste yer alan Gateway kuralları sayısı.
- **Cloud Rules**: Serviste yer alan Cloud kuralları sayısı.

Adım Adım Servis Oluşturma

Bir IoT geliştiricisi olarak, IoT-Ignite Cloud platformundan hizmet satın aldığımızı düşünelim ve bu altyapıyı kullanarak müşterilerimize akıllı kapı zili satmak gibi bir girişimimiz olduğunu varsayalım. Bu durumda biz hem bir geliştirici hem de girişimci konumunda oluruz. IoT-Ignite ile başlatmış olduğumuz üyelik hesabı ile birlikte otomatik olarak bir de servis başlatmış oluruz. Servisimizin ismi, iotignite.com_ prefix'i ve e-posta hesabımızın birleşimi ile otomatik olarak tanımlanmıştır. Bu servis adı altında Gateway'lerimizi lisanslayabiliriz. Her bir Gateway, müşterilerimize satacağımız akıllı kapı zili ürününde Cloud ile haberleşecek olan Arduino, Raspberry Pi, Android veya MQTT olabilir, tercih size kalmış. Her bir kapı zilinin bizim kullanmış olduğumuz servisimize bağlı olması gerekmektedir. Servisimiz ile de ürünlerimizi yönetebiliriz. Şimdi adım adım bir servis nasıl oluşturulur ve nasıl yönetilir görelim...

Servis Hizmetini Başlatmak

İlk olarak servis ismimizi düzeltmekle başlayalım...

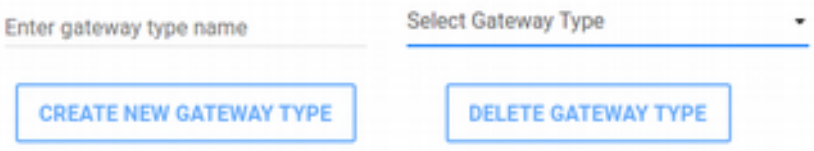
Servis İçin İsim Tanımlama

Devzone'da üst menüden **CREATE A SERVICE** (Bir Servis Oluştur) butonuna tıklayıp **Begin** (Başlangıç)'a tıklayın. Aşağıdaki gibi bir ekran ile karşılaşacaksınız.

Gördüğünüz üzere varsayılan olarak tanımlanmış ignite.com_iot-ignite_iot-ignite.com isminde bir servisimiz var. Bu isimle bir servis oluşturup müşterilerimize sunduğumuzda onlara da anlamsız gelecektir. Bu alana daha anlamlı bir servis ismi girin ve **UPDATE** (Güncelle) butonuna tıklayarak servisinizin adını güncelleyin veya bu işlem zorunlu olmadığı için direkt olarak **Gateway** sayfasına geçiş yapıp cihazlarınızı lisanslayın. Örneğimizde **IOTIGNITE** ismi girildi. Güncelleme işleminden sonra hemen Gateway eklemeniz için bir sonraki adıma yönlendirileceksiniz.

Çoklu Servisler için Servis Modu Oluşturmak ve Silmek

Daha önce de bahsetmiştik, Demo senaryosunda DEMO, Devzone'da ise DEVELOPMENT modları ile geliştirme yapılmaktadır. Bu modlar ve modlara ait yapılandırmalar, üyelik hesabınız ile birlikte otomatik olarak oluşturulmaktadır. Eğer bu modların dışında kendiniz de yeni bir mod oluşturmak istiyorsanız (Daha sonraları bu modu yapılandırmanız gerekecek), yine Begin sayfasında iken yapabilirsiniz. **DO YOU NEED A MULTI GATEWAY SCENARIO IN YOUR SOLUTION?** (Çözüm Senaryonuzda Daha Fazla Gateway'e mi İhtiyacınız Var) butonuna tıklayın. Hemen altında aşağıdaki gibi bir form belirecek.



Sol tarafta yer alan **Enter gateway type name** (Ağ Geçidi Tipi İsmini Girin) alanına yeni bir mod ismi girip **CREATE NEW GATEWAY TYPE** (Yeni Ağ Geçidi Oluştur) butonuna tıkladığınızda yeni bir mod tanımlanacaktır.

Sağ tarafta ise, servisinizde tanımlı olan modlar yer almaktadır. Devzone'a üye olduğunuzda 3 adet mod varsayılan olarak gelmektedir. Bunlar; **DEMO**, **DEVELOPMENT** ve **ANDROIDTHINGS**'tir. Ekleyeceğiniz diğer modlar da bu listede görülecektir. Eğer bir mod silmek isterseniz, **Select Gateway Type** (Ağ Geçidi Türü Seç) listesinden modu seçip, **DELETE GATEWAY TYPE** (Ağ Geçidi Türünü Sil) butonuna tıklamanız yeterlidir.

Gateway (Android, Raspberry PI3, PilarOS, MQTT) Kaydetmek

Devzone'da IoT-Ignite platformuna 4 farklı tipte cihazı Gateway olarak tanımlayabiliriz. Bunlar; Android işletim sistemli akıllı telefon/tabletler, Raspberry PI3, IoT-Ignite alt yapısı ile geliştirilmiş PilarOS işletim sistemli cihazlar ve MQTT (Sanal Gateway) olmak üzere 4 grup altında toplanabilir. Şimdi sırasıyla her birinin IoT-Ignite'a nasıl lisanslanacağını ve lisansların nasıl kaldırılacağını görelim.

Gateway Lisanslama

Gateways sayfasındayken REGISTER A GATEWAY (Bir Ağ Geçidi Kaydet) butonuna tıklayın. Aşağıdaki şekilde görüldüğü gibi bir arayüz ile karşılaşacaksınız.

Açılan arayüzden, lisanslama işleminde kullanacağınız donanım türüne tıklayın ve her biri için özel lisanslama ekranlarına giriş yapıp inceleyin. Gateway lisanslama aşamalarını sırası ile inceleyelim....

Android Telefon veya Tablet'i Kayıt Etmek

Demo akışındaki lisanslama süreci ile aynıdır. Lisaslanacak olan Android işletim sistemli akıllı telefon veya tabletinize IoT-Ignite Agent yazılımını yükleyip, lisanslama ekranında karşınıza çıkan QR kodu taratmanız yeterlidir.

Raspberry PI 3 Üzerinde PilarOS ile Kayıt Etmek

PilarOS; ARDIC tarafından geliştirilmiş olan IoT-Ignite yönetim API'larını ve Iot-Ignite Cloud erişim bileşenlerini barındıran Android tabanlı geliştirilmiş bir mobil işletim sistemidir. PilarOS'u Raspberry Pi 3'e yükleyerek kurulum işleminizi başlatabilirsiniz.

Kurulum işlemi için ilk olarak minimum **8 GB microSD** gerekmektedir.

Kurulum işlemi basamakları şunlardır:

- PilarOS işletim sistemi IoT-Ignite sunucularından indirilir.

https://download.iot-ignite.com/Pilaros-rpi3/

İndirilen imaj dosyası **zip** formatındadır. Öncelikle sıkıştırılmış dosyayı açınız (**7zip** programını kullanılabilirsiniz). IMG dosyası microSD Card'a yazdırılır.

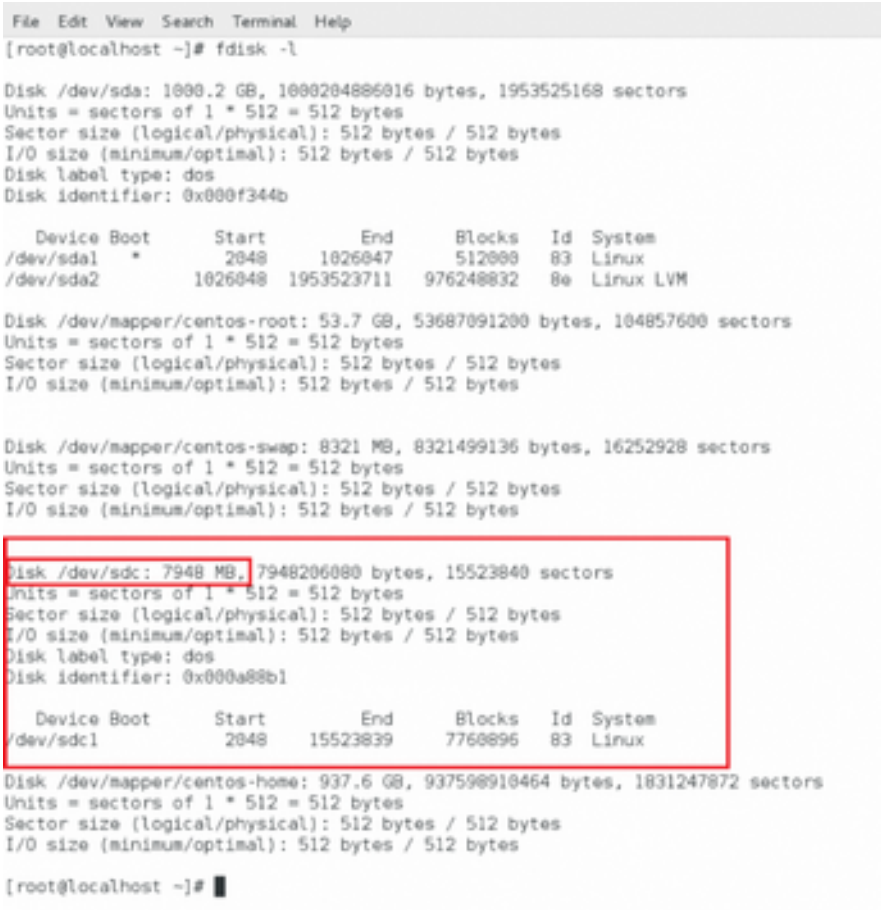
Yazdırma işlemini eğer **Windows** işletim sisteminde yapıyorsanız **Win32 Disk Imager** programını kullanabilirsiniz. microSD Card'ı sisteminizin kart okuyucusuna takın ve Win32 Disk Imager programını **Run as Administrator** (Yönetici olarak aç) ile açın. Programın sağ üst köşesinde yer alan **klasör** ikonuna tıklayıp, dosya gözatıcısı ile PilarOS imajını bulun (Dosya gözatıcısında dosya filtreme türünü ***.*** yapın) ve açın. Ardından **Device (Cihaz)** listesinden de microSD Card'ı bulun ve seçin. Son olarak **Write** (Yaz) butonuna tıklayın. Yaklaşık olarak 5 dakika kadar sürecektir. Yazdırma işlemi tamamlanınca **Exit** ile programdan çıkın, microSD Card'ı sisteminizden çıkarıp Raspberry Pi 3'ün kart okuyucusuna takın. Raspberry Pi 3'ü tekrar başlatın. Otomatik olarak PilarOS ile boot edecektir. Bu işlem de yine 5 dakika kadar bir süre alacaktır.

Eğer Linux işletim sisteminde yapıyorsanız, indirdiğiniz PilarOS zip dosyasını, aşağıdaki **unzip** komutu ile ile dışarıya çıkarın. Örneğin;

```
unzip Pilaros-N-rpi3_20160927_HDMI_8GB.img.zip
```

microSD Card'ı bilgisayarınızın kart okuyucusuna takın. Ardından **fdisk** komutunu kullanarak microSD Card'ın cihaz ismini öğrenin.

```
fdisk -l
```



Ardından dd komutu ile yazdırma işlemine başlayın. **dd** komutunun formatı aşağıdaki gibidir.

```
dd if=<hedef> of=<img>
```

Örneğin;

```
dd if=/home/eozen/Desktop/Pilaros-N-rpi3-20160927_HDMI_8GB.img of=/dev/sdc
bs=4096
```

Yazdırma işleminin tamamlanmasını bekleyin. Yaklaşık 5 dakika sürecektir. İşlem tamamlanınca belliği boşaltmak için **sync** komutunu kullanın.

İşlem tamamlanınca, microSD Card'ı sisteminizden çıkarıp Raspberry Pi 3'ün kart okuyucusuna takın. Raspberry Pi 3'ü tekrar başlatın. Otomatik olarak PilarOS ile boot edecektir. Bu işlem de ilk seferinde 5 dakika kadar bir süre alacaktır.

Lisanslama adımlarına devam edelim...

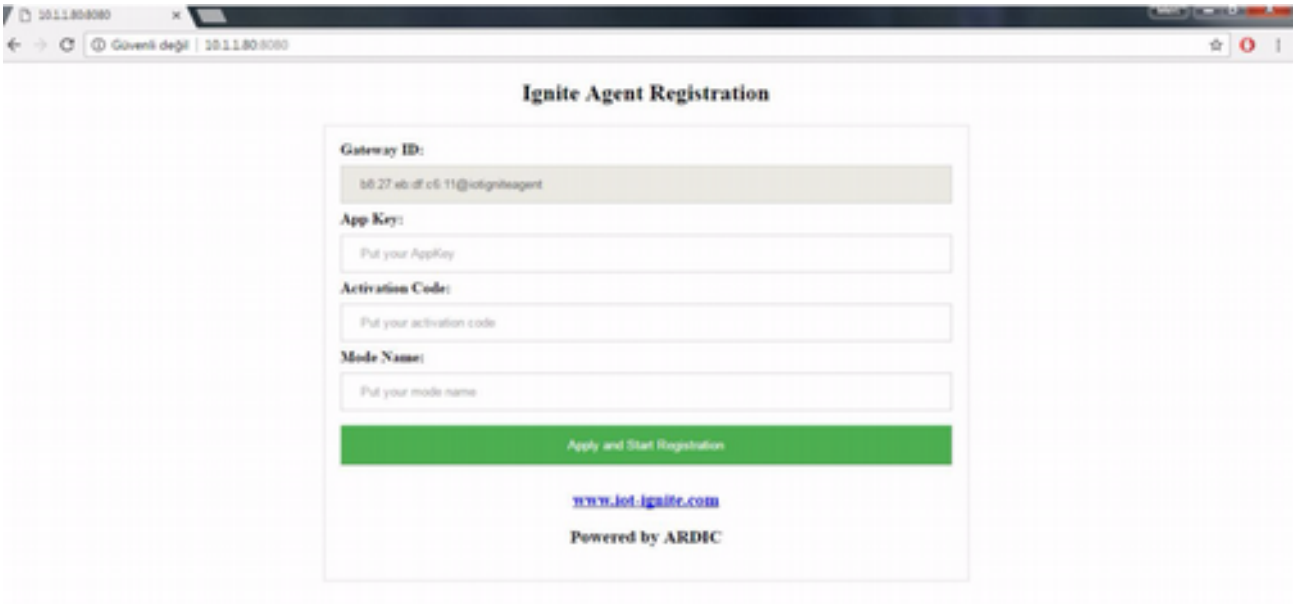
- Raspberry Pi 3 lisanslama işlemi.

Lisanslama işlemi iki farklı metotla yapılabilir. Birincisi; Service Provider APP'i akıllı telefonunuza veya tabletinize indirin, kurun. (Service Provider APP, Android 5.0 ve Android'in daha alt sürümlerinde kullanılabilir. Cihazının Android 6.0 veya üzeriyse diğer metodu tercih edin.)

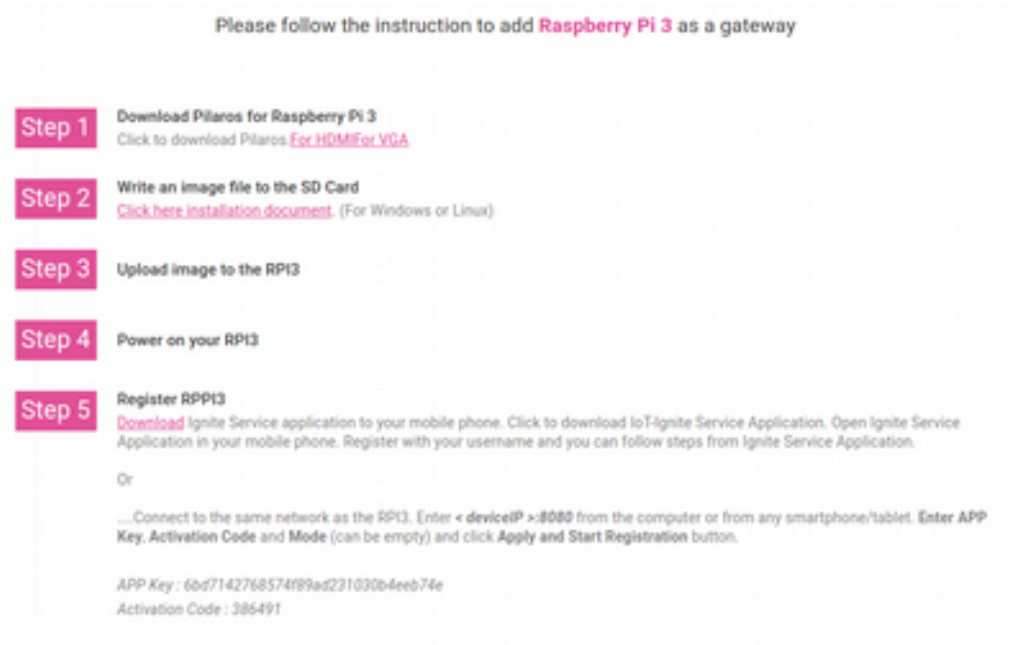
https://download.iot-ignite.com/ServicePlatformApp/

Daha sonra uygulamayı açın, Devzone'a kaydolduğunuz hesap ile giriş yapın ve kurulum adımlarını izleyin. Kurulum aşamasında; Raspberry Pi 3 ilk açıldığında kendisini Hotspot olarak açar. Aynı ağda bulunan cihazlar Service Provider APP ile ağda taranır ve bulunur. Bulunan cihazlara da lisans bilgisi gönderilir.

Veya bilgisayarınızdan veya akıllı telefon/tablet üzerinden gateway_ip_adresi:8080 URL'ine girin. Karşınıza aşağıdaki gibi bir form ekranı gelecek.



Burada lisanslama ile ilgili 3 adet bilgi istenmektedir. Bu bilgiler **APP Key (Uygulama Anahtarı), Activation Code** (Aktivasyon Kodu) ve **Mode** ismi'dir. APP Key, Devzone Dashboard'unda yer almaktaydı, hatırlayın. Activation Key ise bir End User'a Gateway'i lisanslamak için kullanılan eşleştirme anahtarıdır. Devzone'da 5 adet lisansımız vardı, hatırlayın. Devzone'a üye olduğumuzda hesabımıza bir de End User tanımlanmaktaydı. Bu 5 lisans, aynı **End User**'a tanımlanmıştır. Devzone'da lisansladığınız her Gateway, bu End User üzerinde görülecektir. Mod ismi boş bırakılabilir (boş bırakılırsa varsayılan olarak DEVELOPMENT modu atanır) veya yine **Begin** adımında tanımladığınız bir mod ismi girilebilir. **APP Key** ve **Activation Code** bilgileri **Gateway** sayfasındayken **REGISTER A GATEWAY**'e tıklayıp açtığınız arayüzde **Raspberry Pi 3**'e tıkladığınızda karşınıza gelecektir. Adım adım gösterilen kurulum yönergelerinden **5. adım** altında yer almaktadır.



Endüstriyel Gateway'ler (Dell, Gigabyte…) Üzerinde PilarOS ile Kayıt Etmek

Endüstriyel Gateway'ler de aynı Raspberry Pi 3'te yaptığımız kurulum işlemleri gibidir. Aradaki tek fark; bu Gateway'lerde PilarOS'un hazır olarak yüklü olarak gelmesidir.

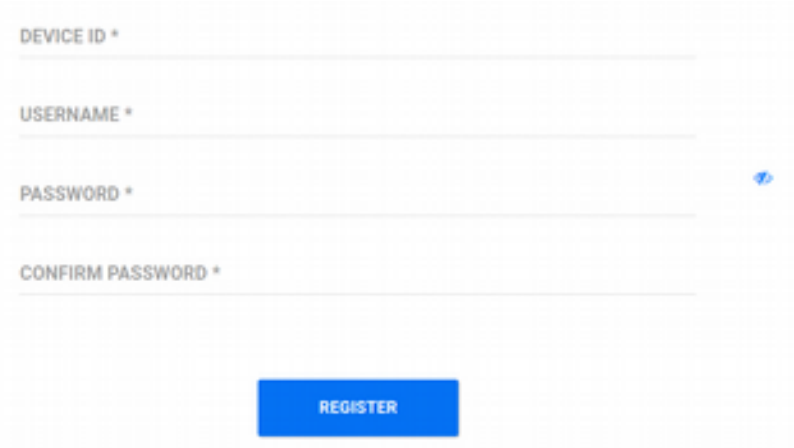
MQTT Virtual Gateway (Linux, NodeMCU…) Kayıt Etmek

MQ Telemetry Transport veya MQTT, düşük güçlü aygıtları bağlamak için bir çözüm olarak popülerlik kazanan, birçok platformda rahatlıkla kullanılabilen bir telemetri mesajlaşma protokolüdür. MQTT başlangıçta pub/sub mimarisine dayalı iletişimler için tasarlanmış olsa da genel amaçlı "makineden makineye" (M2M) iletişim protokolü olarak kullanılabilir. Kısıtlı cihazlar ve düşük bant genişliği, yüksek gecikme süresi veya güvenilmez ağlar da yaygın olarak kullanılır. Son derece basit ve hafif mesajlaşma protokolüdür.

IoT-Ignite, MQTT için Cloud ile rahatlıkla iletişim kurulabilmesi için **JAVA8** ile geliştirilmiş bir **MQTT Client** sunmaktadır. Bu Client ile özel sensörler ve özel aracılar tanımlanabilmektedir.

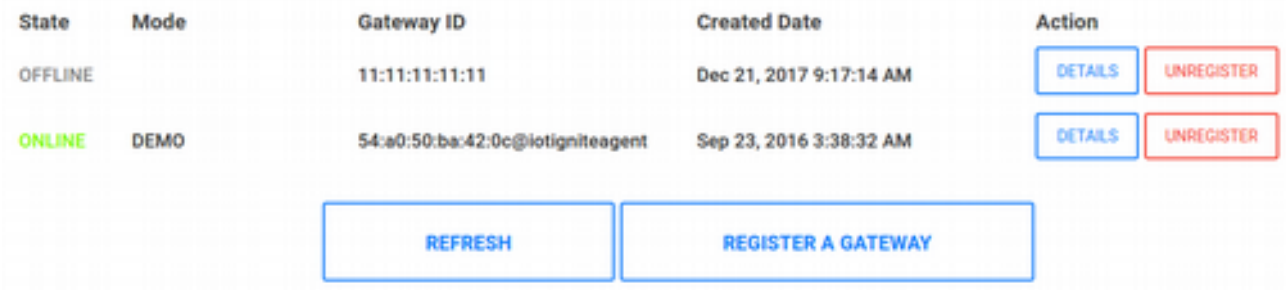
IoT-Ignite'ta MQTT bağlantısı kurabilmeniz için bir kayıt oluşturmanız gerekmektedir. Bu kayıt için bir cihaz ID'si ile birlikte, kullanıcı adı ve şifre tanımlanmaktadır. Tanımladığınız bu bilgiler, MQTT Client'ı kullanırken **mqtt.properties** dosyası içinde yazılarak yapılandırılmaktadır. İlerleyen konularda detaylıca göreceksiniz.

MQTT lisanslama işlemi için yine **Gateway** sayfasındayken **REGISTER A GATEWAY** butonuna tıklayıp, açılan arayüzden **MQTT (Virtual Gateway)**'i seçin. Karşınıza aşağıdaki gibi bir form çıkacak, bu formu doldurun ve **REGISTER** butonu ile işleminizi tamamlayın.



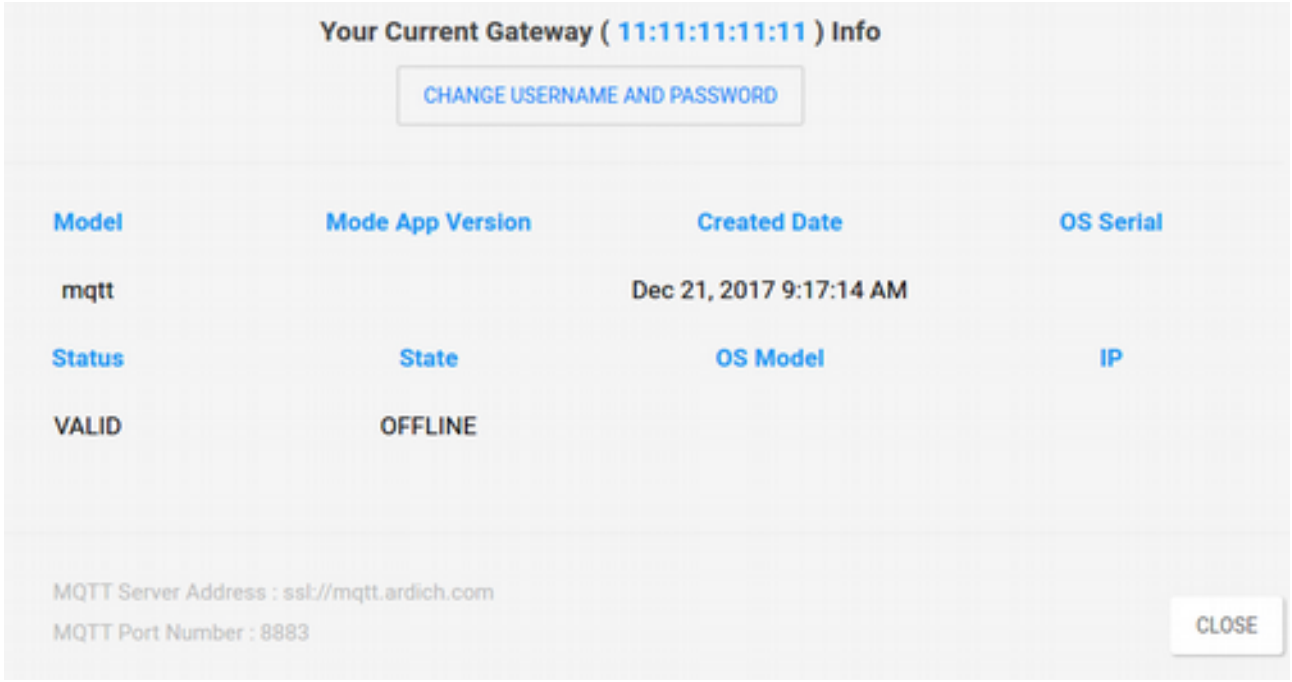
Kayıtlı Gateway'in Bilgilerini Görüntülemek

Lisansladığınız Gateway'lere dair birtakım bilgileri Gateways sayfasındayken, listede yer alan Gateway'in yanında yer alan **DETAILS** (Detaylar) butonuna tıklayarak açılan arayüzde görebilirsiniz.



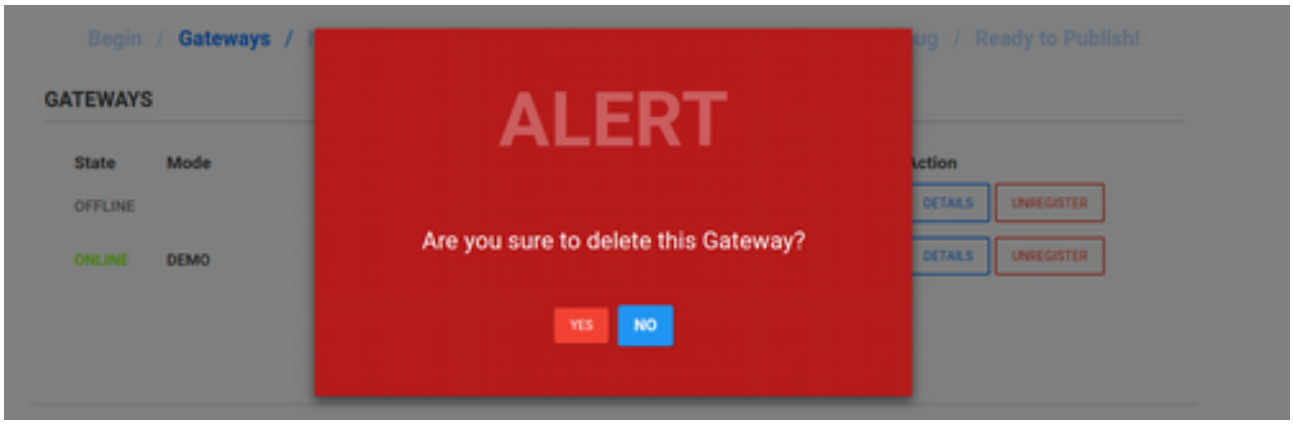
Gateway verilerinin güncellenmesi için de **REFRESH** (Yenile) butonuna tıklayabilirsiniz. Örneğin ONLINE olan Android Gateway'in detaylarına bakarsak aşağıdaki gibi bilgileri görürüz.

Ek olarak MQTT olarak kaydedilmiş Gateway'lerin detay ekranında, o MQTT'i kaydederken girmiş olduğunuz kullanıcı adı ve parola bilgilerini güncelleyebilirsiniz. Bunun için **CHANGE USERNAME AND PASSWORD** (Kullanıcı Adı ve Parolayı Değiştir) butonuna tıklayıp açılan formda bilgilerinizi güncelleyip tekrar kaydedin.



Kayıtlı Gateway'i Kaldırmak

Lisanslanmış olan bir Gateway'i kaldırmak için, yine **Gateways** sayfasındayken ilgili Gateway'in hemen yanında bulunan **UNREGISTER** (Kaydı Sil) butonuna tıklayıp, açılan onay penceresinden de **YES** (Evet)'i tıklamak yeterlidir. Gateway'in ağ bağlantısı varsa, anında Gateway'de de lisans bağlantısı kopacaktır ve tekrar lisanslama arayüzüne dönecektir. Eğer Gateway'in o an ağ bağlantısı yoksa, tekrar ağa bağlandığında lisansı kalkacaktır.



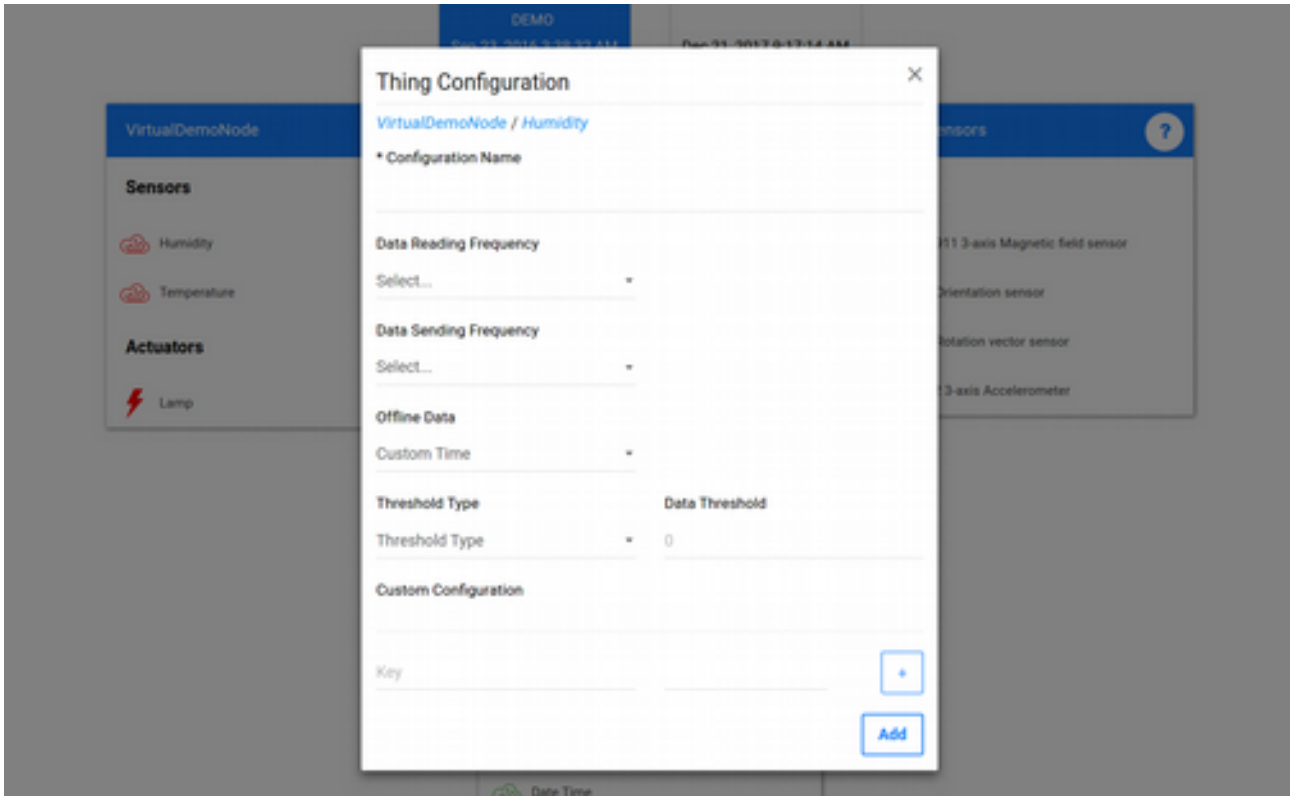
Kayıtlı Gateway'lerdeki Sensör ve Actuator'ların Listelenmesi, Yönetilmesi ve Özel Veri Yapılandırmaları

Demo senaryosunda da görmüş olduğumuz **Nodes** sayfasıyla, Gateway'in içindeki sensörleri görebiliriz. Devzone'da Nodes sayfasında sensörleri görmenin dışında, her bir sensörün **veri yapılandırmalarını** yapabiliriz.

Genel Sensör ve Actuator'ler İçin Yapılandırma Yapmak ve Yapılandırmayı Gateway'e Push Etmek

Bir sensörü yapılandırmak için listede yer alan sensörü fare ile tıklayıp veri yapılandırma arayüzünü açmamız gerekir. Birçok sensör için yapılandırma ayarları standarttır, sadece Geofence ve Date-Time sensörleri için farklı yapılandırma ayarları bulunmaktadır (Birazdan göreceğiz).

Aşağıdaki örnek arayüzde, Humidity sanal sensörü için açılan veri yapılandırma arayüzü görülmektedir.



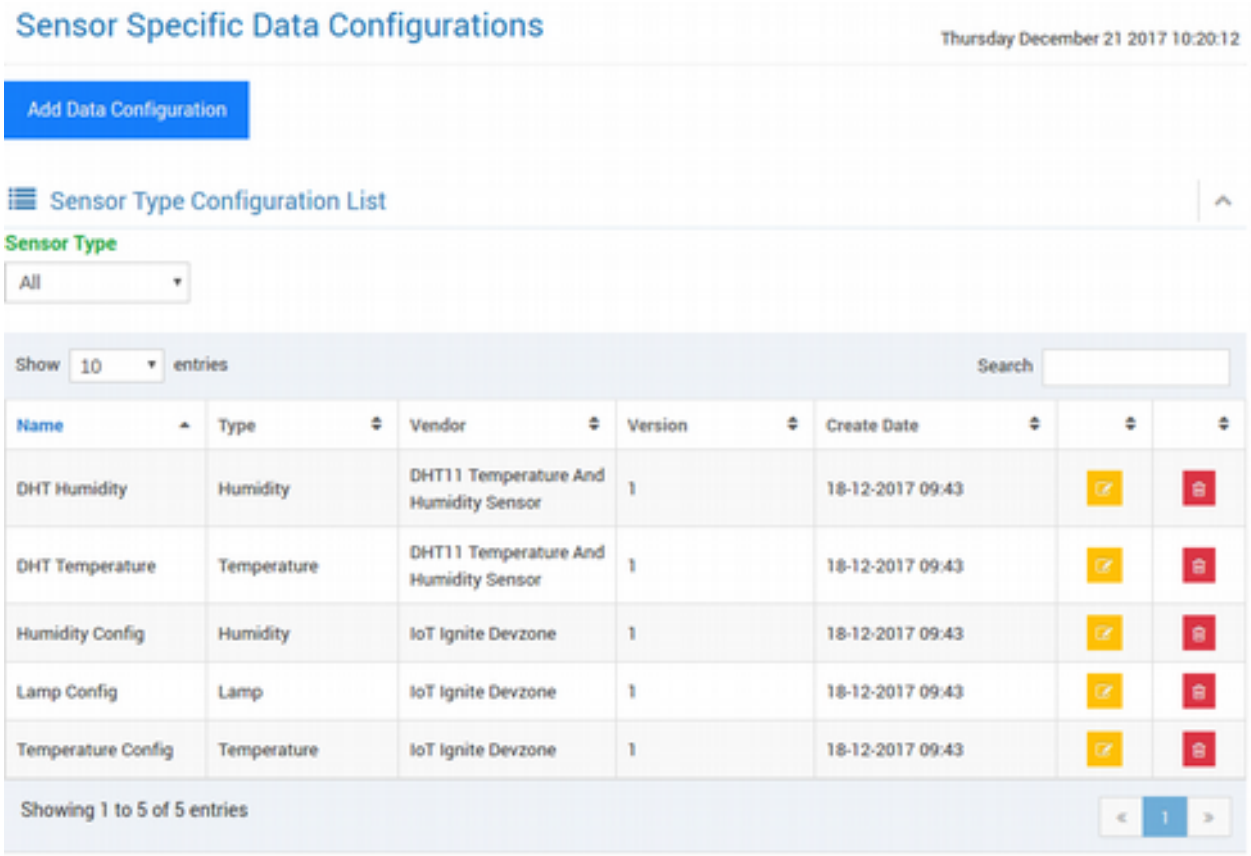
Bu arayüzde yapılan veri yapılandırması, aslında doğrudan belli bir sensör için yapılandırma değildir. Yapılan yapılandırma, bir veri yapılandırma şablonudur. Bunun anlamı şudur; yapılandırdığınız veri yapılandırma şablonunuzu, başka Gateway'lerdeki sensörler için de doğrudan uygulayabilirsiniz. Ancak Devzone'da yapılan veri yapılandırmaları, kayıt işlemi ile birlikte doğrudan Gateway'e gönderilmez. İlerleyen adımlarda göreceğimiz Gateway Services sayfasında, veri yapılandırmalarımızı aktif edip, istediğimiz Gateway'lere push edeceğiz.

Sensörlerin IoT Cloud'a veri gönderebilmeleri için yapılandırılmaları gerektiğini biliyoruz. Ama belki farketmişsinizdir. Demo akışında (DEMO modu) gördüğümüz Humidity, Temperature ve Lamp sensörleri için bir veri yapılandırma ayarları yapmadığımız halde anlık olarak veri gönderebiliyorlardı.

Devzone'a kayıt yaptığınızda, otomatik olarak bu 3 sensör için veri yapılandırması **sensör bazında** otomatik olarak tanımlanmaktadır. Bu yapılandırmaları Devzone'da değil, EHUB'ta görebiliriz (*Sensor Data Configurations* > *Sensor Specific Data Configurations*). Aşağıdaki görüntüde varsayılan ayarlarla tanımlanmış olan yapılandırmalar yer almaktadır.

EHUB'a Devzone hesabınız ile girebilirsiniz.

Listede gördüğünüz üzere 5 adet veri yapılandırması yer almaktadır. Humidity için tanımlanan sensör veri yapılandırma detayına baktığımızda aşağıda yer alan ayarları görebiliriz.



Bu arayüz, aynı Devzone'da ilk gördüğümüz sensör veri yapılandırma arayüzüne benzemektedir. Ama Humidity sensörüne tıklayıp açılan yapılandırma ayarlarında bu daha önceden tanımlanmış ayarları göremedik. Bunun sebebi; EHUB'ta varsayılan olarak gelen veri yapılandırma ayarları **Sensör ID** bazında, Devzonde'ki veri yapılandırma ayarları da **Node ID + Sensör ID** bazındadır. Yani EHUB'taki veri yapılandırma ayarlarına göre sensörün türü (Type) Humidity olan bütün sensörlerde o varsayılan ayarlar geçerli olacaktır. Devzonde ise VirtualDemoNode Node ID içinde

yer alan Humidity Sensör ID'li sensör için yapılandırma yapılır. Devzone'da yapacağımız yeni yapılandırma, önceki yapılandırmayı bozmayacaktır, ancak Gateway'e yapılandırma ayarını push ettiğimizde, yeni yapılandırma ayarı eskisini ezecektir. Şimdilik EHUB ile bir işimiz yok, sadece Devzone'da neden sanal sensörler için yapılandırma ayarları görünmediği halde sensör veri akışının sorunsuz çalıştığını anlatabilmek için gösterdik.

https://enterprise.iot-ignite.com/v3/data-config

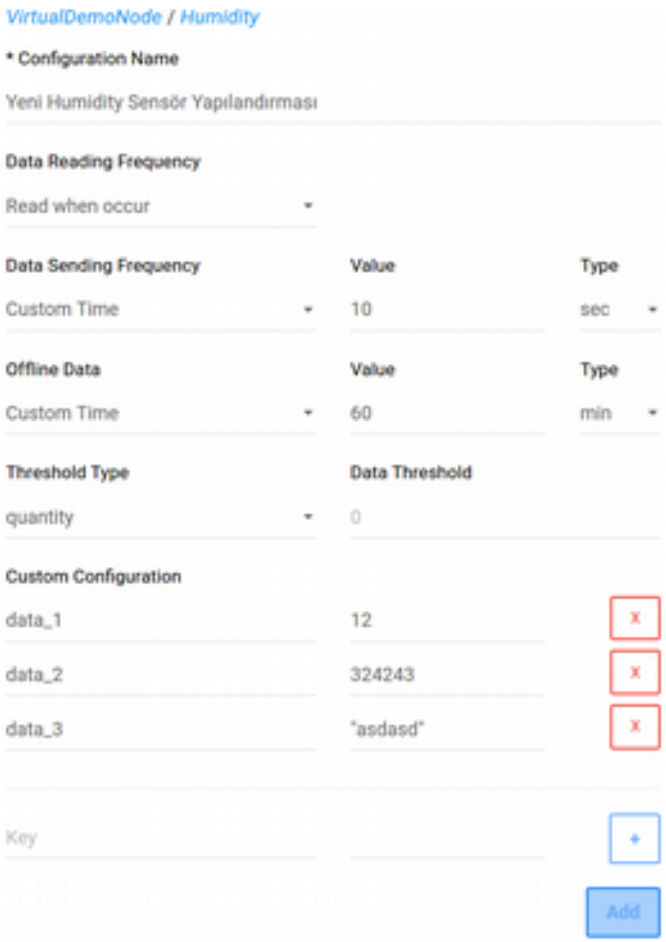
Tekrar Devzone'a dönelim ve Nodes sayfasından Humidity için veri yapılandırma ayarlarını inceleyerek yeniden yapılandıralım...

Genel yapılandırmalarda 5 adet parametre ve bu parametrelerin de opsiyonları bulunmaktadır, kısaca inceleyelim..

- **Configuration Name**: Node ID + Sensor ID olarak tanımlanan sensör veri yapılandırması için bir isim.
- **Data Reading Frequency**: Veri okuma sıklığı ayarları.
  - Do not read: Sensörden veri okunmaz.
  - Read when occur: Sensörde yeni bir değer üretildiğinde veri okunur.
  - Custom time: Özel olarak bir zaman periyodu belirtip, veri okuma sıklığı tanımlanabilir.
- **Data Sending Frequency**: Veri gönderme sıklığı ayarları.
  - Do not send: Sensörden veri okunsa bile IoT-Ignite Cloud'a göndermez.
  - Send when occur: Sensörde yeni bir değer üretildiğinde veriyi IoT-Ignite Cloud'a gönderir.
  - Custom time: Özel olarak bir zaman periyodu belirtip, IoT-Ignite Cloud'a veri gönderilir.
- **Offline Data**: Çevrimdışı durumdayken verilerin ne yapılacağı ayarları.
  - Do not keep: Eğer Gateway çevrimdışıysa, üretilen sensör verilerini saklama.
  - Custom time: Özel olarak bir zaman periyodu belirtilip, sensör verileri saklanır. Gateway tekrar çevrimiçi olduğunda bu belirtilen zaman süresince tutulan veriler topluca IoT-Ignite Cloud'a gönderilir.
- **Threshold Type**: Veri eşiği ayarları. Burada yeni veri ile bir önceki verinin karşılaştırılması/farkı eşik olarak kabul edilir.
  - Quantity: Miktar. Veri eşiği en az belirtilen miktar kadar değişirse, IoT-Ignite Cloud'a veri gönderilir.
  - Percentage: Yüzde. Veri eşiği en az belirtilen yüzde kadar değişirse, IoT-Ignite Cloud'a veri gönderilir.
- **Data Threshold**: Veri eşiği türü olarak seçilen opsiyonun değeri sayısal tanımlanır.

Bu 5 ayardan farklı olarak en alt kısımda Custom Configuration (Özel Yapılandırma) bulunmaktadır. Bu yapılandırma türünde Key-Value (Ad-Değer) şeklinde JSON formatı ile özel yapılandırma ayarlarını sensörlere gönderebilirsiniz. + butonuna tıkladıkça, yeni Key-Value alanları açılacaktır, istenilen kadar değer tanımlanabilir.

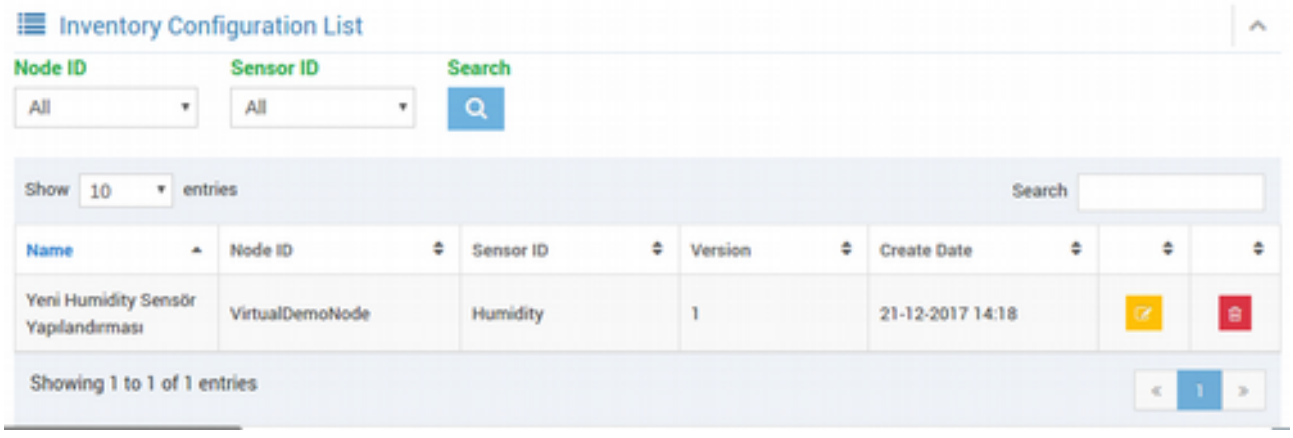
Aşağıda, örnek bir yapılandırma görülmektedir.



Tanımladığımız yapılandırmaya göre sensörün verisi değiştiği her zaman veri okunacak ancak 10 saniyede bir gönderecek. Gateway çevrimdışı olduğunda da 60 dakika veriyi saklayacak. Özel yapılandırmada da data_1, data_2 ve data_3 key'lerini ve değerlerini tanımladık.

Add (Ekle) butonuna tıklayıp eklediğimizde arayüz kapanacaktır. Sensörlerin listelendiği alanda ise Humidity sensörümüzün sağ tarafında bir **check işareti** belirecektir. Bunun anlamı; bu sensör için yeni bir sensör veri yapılandırması yapılmış demektir.

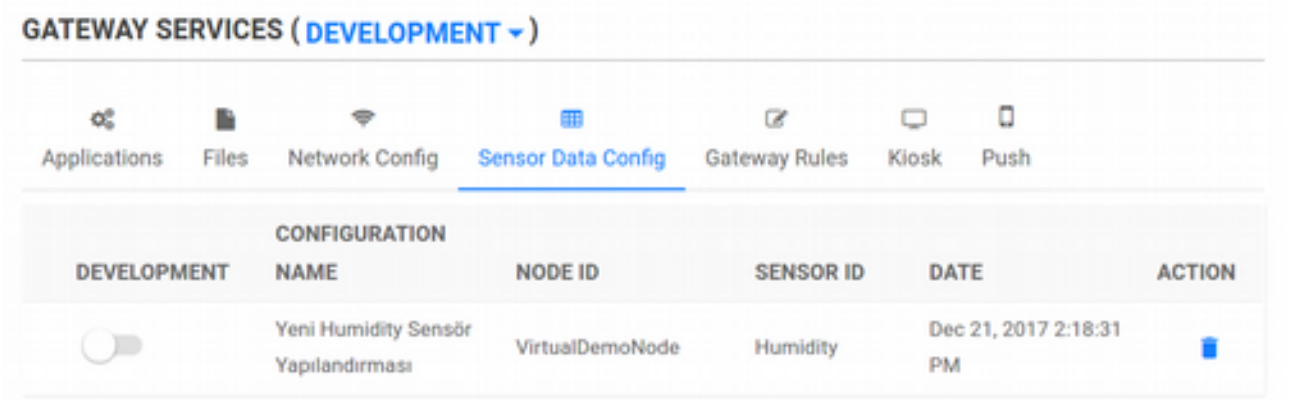
Eklediğimiz bu yapılandırmanın EHUB'ta görüldüğüne bakabilirsiniz. İlerleyen konularda aynı işlemleri EHUB'ta nasıl yapabileceğinizi detaylıca göreceksiniz.



Evet, eklediğimiz yeni sensör veri yapılandırması **Inventory Configuration List**'te (Yapılandırma Listesi Envanteri) yerini aldı. Artık istediğimiz Gateway'e bu yapılandırmayı gönderip, o Gateway'in sensöründen verileri okuyabiliriz.

Unutmayın, Nodes sayfasında tanımlanan yapılandırmalar, sadece envanterde tutulur (EHUB'ta Inventory Configuration List). Yapılandırmanın geçerli olabilmesi için Gateway'e push edilmesi gerekir. Bunu da Devzone'da **Gateway Services** (Ağ Geçidi Servisleri) sayfasından yapabiliriz.

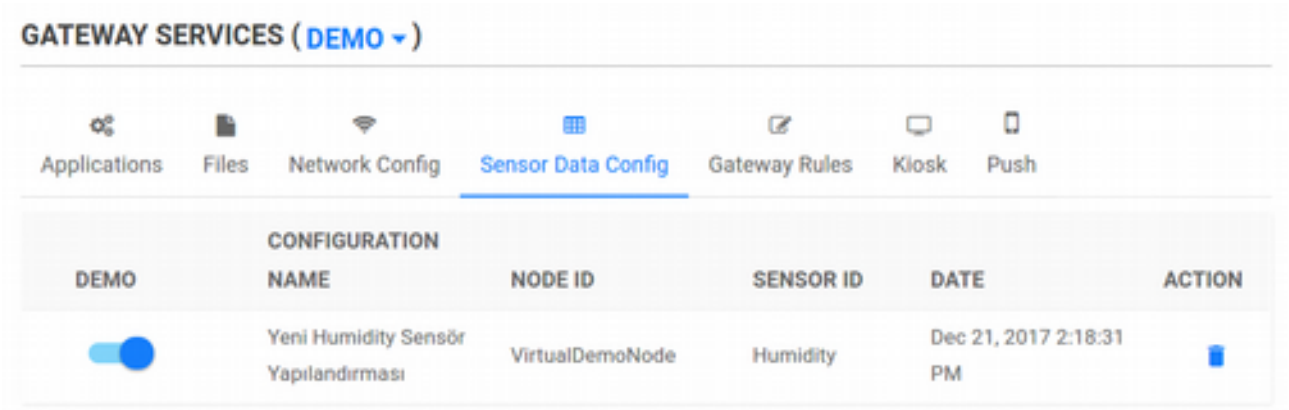
Gateway Services sayfasına girelim. Açılan sayfada **Sensor Data Config** sekmesine tıklayalım.



Sayfayı inceleyelim...

Bu sayfada ilk olarak hangi mod ile çalışacağımızı belirtmemiz gerekir. Devzone, geliştirme ortamı olduğu için varsayılan olarak **DEVELOPMENT** modu kullanılır. Üst kısımda yer alan mavi renkli DEVELOPMENT'ın yanındaki ok işaretine tıklayıp ve açılan menüden **DEMO**'yu seçiyorum. Bunun sebebi, hala Demo akışında kullandığım cihaz ile işlem yapmam. Eğer siz Gateway'inizi Devzone'de lisansladıysanız zaten DEVELOPMENT modunda olacaktır, bu nedenle ayarı değiştirmenize gerek yok.

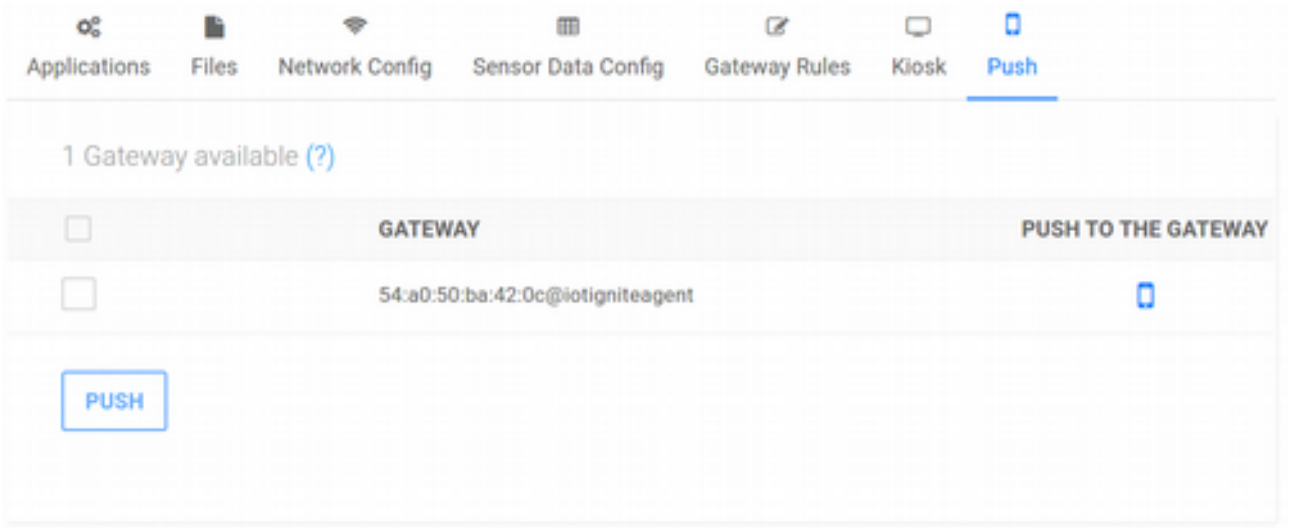
Listede, az önce oluşturduğumuz Humidity yapılandırması yer almaktadır. Switch butonu tıklayıp aktif edin. İsterseniz bu sayfada yapılandırmayı çöp kutusu ikonuna tıklayarak silebilirsiniz de.



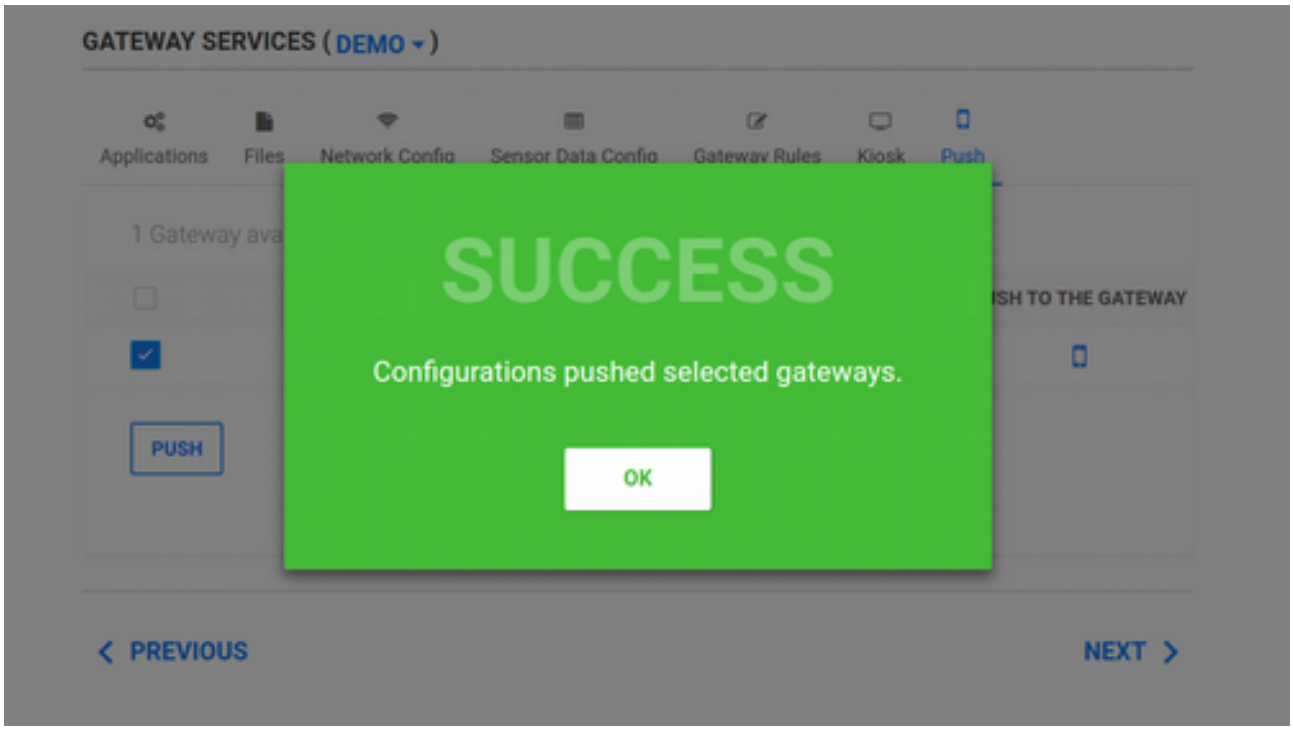
DEMO modunda Humidity için tanımladığımız yapılandırmayı aktif ettik. Üst menüden tekrar DEVELOPMENT veya başka bir moda girerseniz, bu yapılandırmanın diğerlerinde pasif olduğunu göreceksiniz. Bunun anlamı da şudur:

Gateway Services sayfasındayken, birden çok mod ile çalışılır. Yapılandırılan her sensör veri ayarı tüm modlarda kullanılabilir. Listede gördüğümüz Humidity yapılandırması, aslında mod içinde değil, sensör veri yapılandırma havuzunda yer almaktadır. Switch buton ile aktif edilmesi, seçilen mod yapılandırmalarına bu sensör yapılandırma ayarlarını da ekle demektir. Daha sonra biz bu güncellenmiş modu tekrar cihaza gönderdiğimizde, içinde yer alan sensör veri yapılandırması da Gateway'e gitmiş olacaktır.

Gateway’e yeni DEMO modunu push etmek için **Push** sekmesine tıklayın.



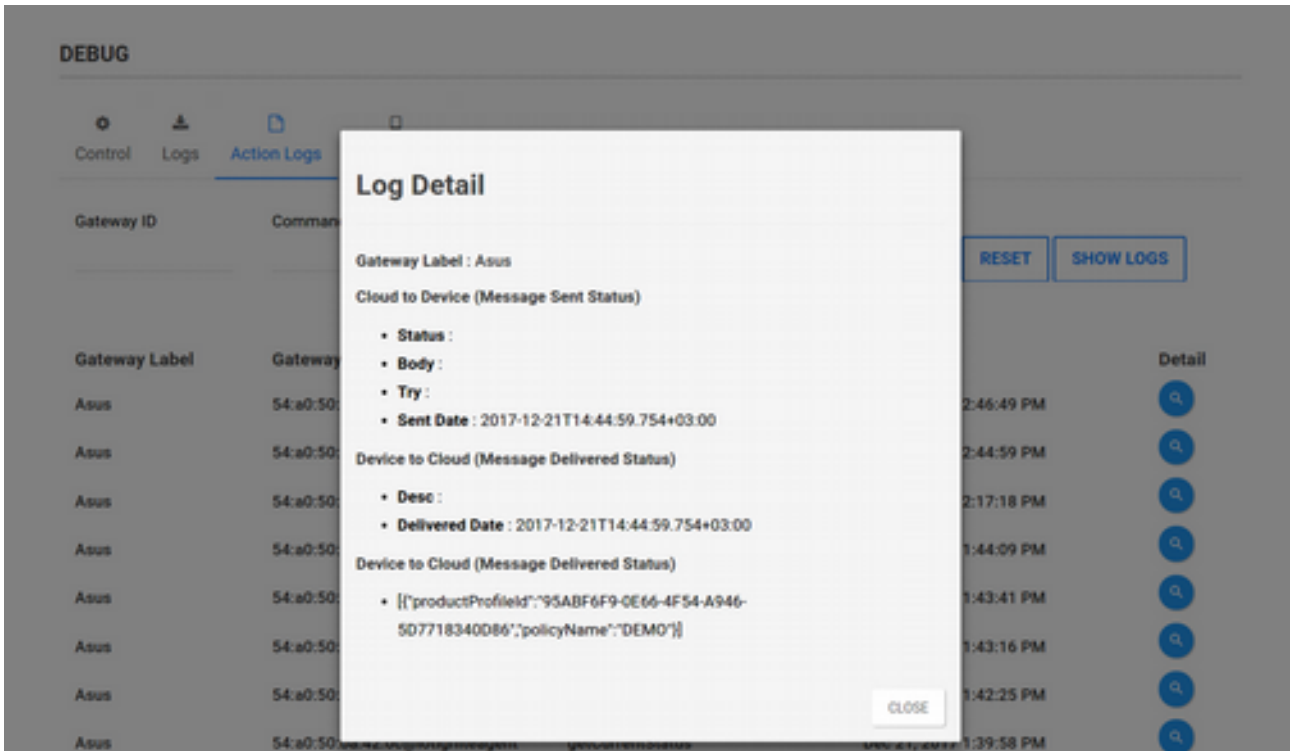
Listeden Gateway’i seçip aktif edelim. Ardından sağ tarafta yer alan Gateway ikonuna tıklayıp push edelim. Eğer birden fazla Gateway olsaydı, alt kısımda yer alan **PUSH** butonu ile de topluca gönderebilirdik (Bir mod üzerinde değişiklik yaptığımızda bunun o moda sahip gateway'lere hemen yansıması için PUSH komutu kullanılır. Ancak bunu göndermesek bile gateway'ler güncellenmiş bir mod olup olmadığını bulut sistemden periyodik olarak sorgularlar. Güncellenmiş mod varsa bunu otomatik olarak indirirler).



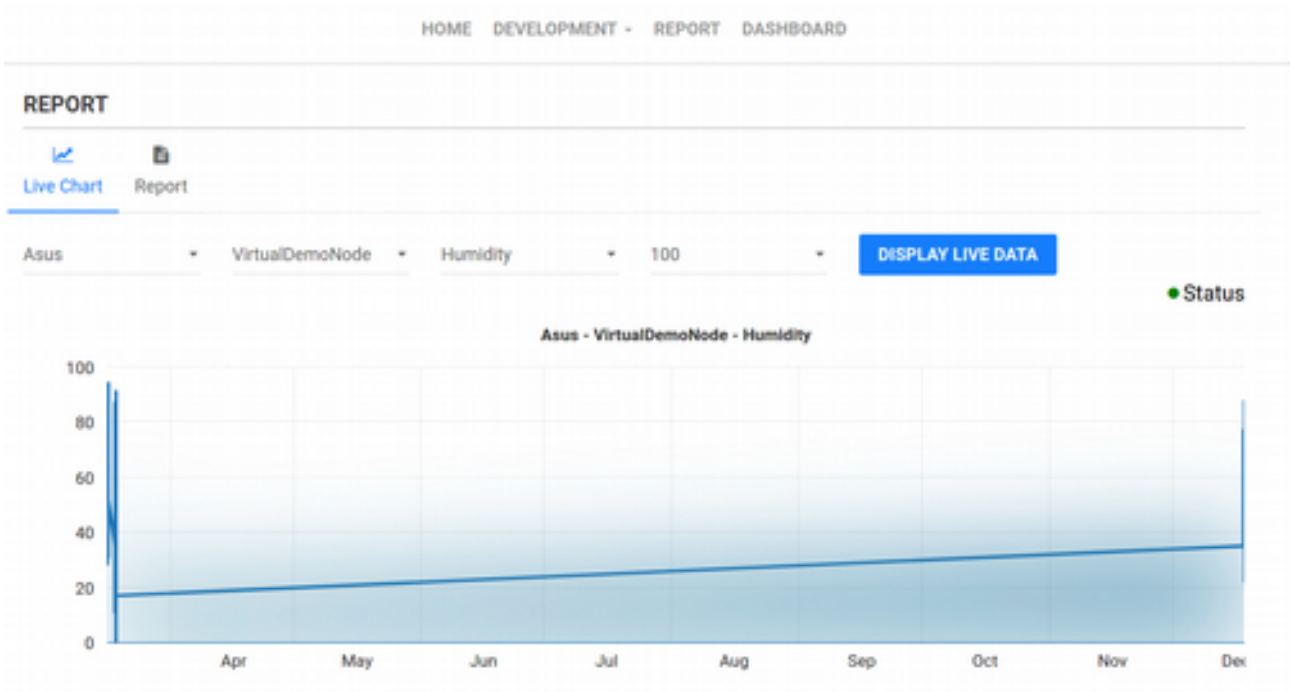
İşlem başarılı olduğunda yukarıdaki gibi bir bilgilendirme mesajı alacaksınız. Gateway, yeni modu aldığında da ekranında “*Updating device policie*s” mesajı çıkacaktır. Bu mesaj çıktığında, artık mod güncellenmiş demektir.

Eğer **log** kayıtlarını görmek isterseniz, üst menüden **Debug** sayfasına girin. Açılan sayfada **Action Logs** (Aksiyon Kayıtları) sekmesine tıklayın. **SHOW LOGS** (Log’ları Göster) butonuna tıklayın ve

hesabınızdaki logları gösterin. **Command** (Komut) türü **sendProductProfile** olan, az önce yaptığımız mod güncelleme işlemidir. **Detail** butonuna tıklayın ve detayları inceleyin.

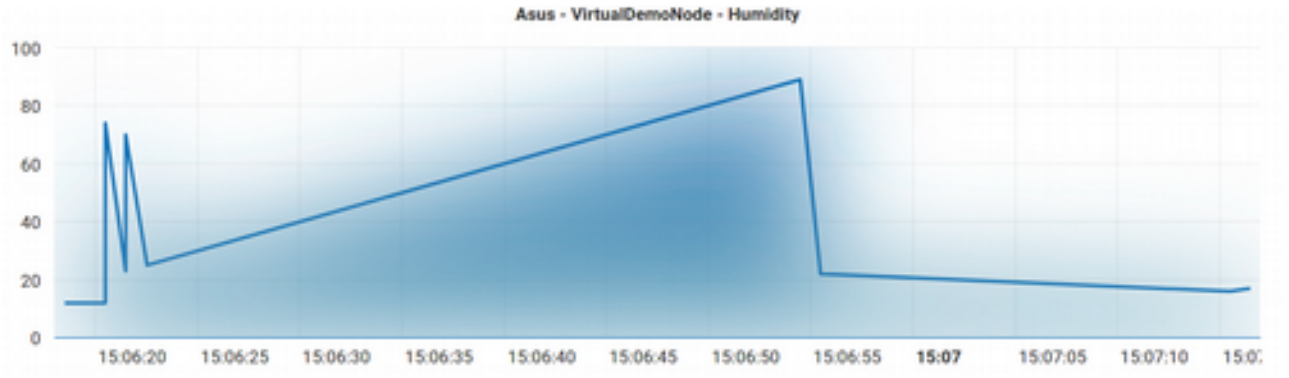


Humidity sensör verilerinin yeni yapılandırma ile nasıl davranış gösterdiğini test etmek için de Devzone'da üst menüden **REPORT** (Rapor)'a tıklayın. **Live Chart** sekmesindeyken Gateway'i, Node'u (VirtualDemoNode), sensörü (Humidity) seçip, veri sayısını da **10** yapın. **DISPLAY LIVE DATA** butonuna tıklayın.

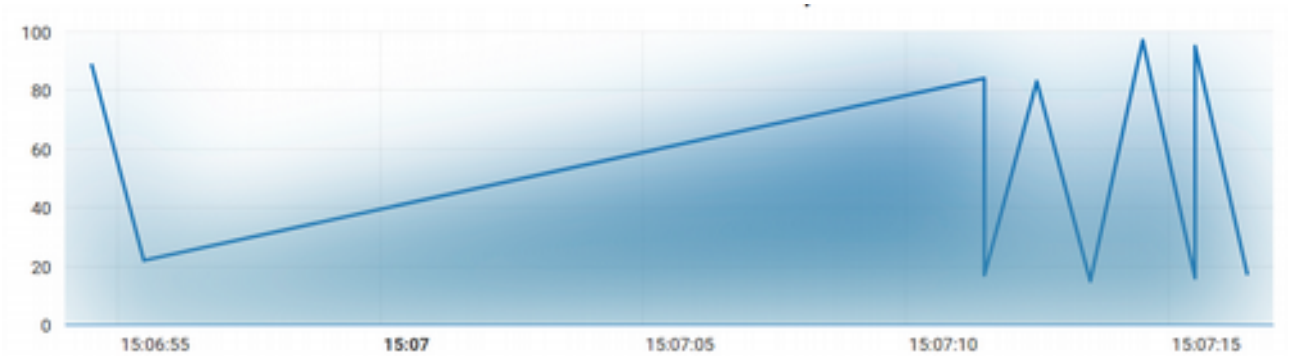


Açılan grafikte, son 10 veri gösterilmektedir. İsterseniz veri sayısını, listeden 10-1000 arası değiştirebilirsiniz.

Android Gateway'den Demo APP'i açın ve Humidity değerini değiştirin. 10 saniye aralıklarla en son veriyi gönderecek ve yukarıdaki grafikte de veri gösterilecektir.



Grafikte, 10 saniyelik zaman diliminde en son üretilen değer **Websocket** üzerinden okunarak gösterilmektedir. Ancak tekrar **DISPLAY LIVE DATA** derseniz, arada kalan verileri de (zikzaklı olan bölge) görebilirsiniz.



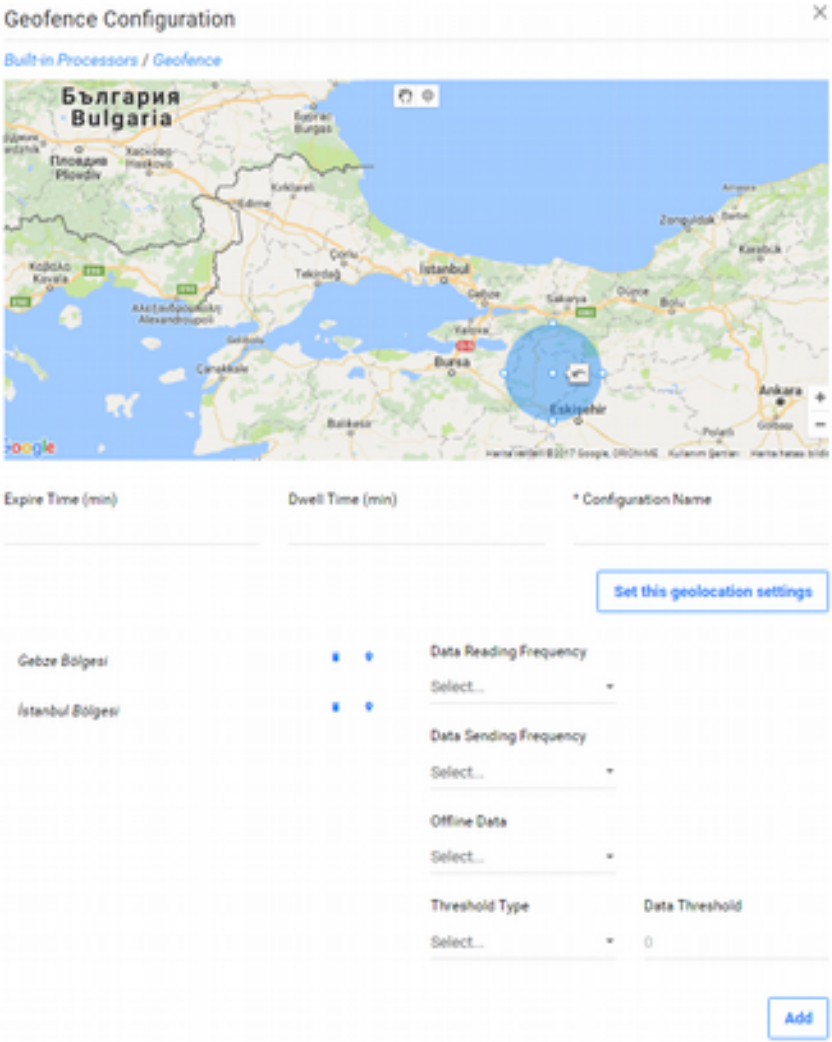
Eğer her bir verinin geçmişini detaylıca görmek isterseniz, **Report** sekmesine girip **GET REPORT** (Rapor Getir) butonuna tıklayarak (isterseniz zaman aralığı seçebilirsiniz) gelen listeden inceleyebilirsiniz.

Geofence ile Gelişmiş GPS Konum Yapılandırması Yapmak

Genel sensör veri yapılandırmalarının dışında özel olarak iki adet daha Built-in Processor (Yerleşik İşleyiciler) bulunmaktadır. Bunlar; **Geofence** ve **Date-Time** sensörleri. Bu sensörler için genel yapılandırma ayarları dışında ekstra ayarlar da bulunmaktadır.

Geofence; Gateway'in haritada belli bir konum (çember) içine girmesi ya da çıkmasıyla tetiklenir.

Aşağıdaki şekli incelediğimizde sağ alt köşede genel yapılandırma ayarlarımız görülmektedir. Fakat dikkat edin, ek olarak Key-Value şeklinde ekleyebileceğiniz özel değerler bulunmamaktadır. Sol tarafta ise "Gebze Bölgesi", "İstanbul Bölgesi" şeklinde maddeler yer alıyor. Bu maddeler, Gateway'in Geofence sensörünün hangi konumlarda tetikleneceğini belirten dairesel koordinatlardır. Harita üzerinden göreceğiniz üzere mavi renkli bir daire alan görülmektedir. Bu alan, büyütüp küçültülebilir, haritanın istenilen bölgesine taşınabilir. Mavi alan içinde kalan bölgeye Gateway giriş yaptığında Geofence sensörü tetiklenir. Listede görüldüğü üzere sadece bir bölge değil, istenilen kadar bölge seçilebilir. Tanımlanan listeleri silmek için, sağ taraflarında yer alan çöp ikonlarına tıklamak yeterlidir. Hangi alanı kapsadığını görmek için de harita ikonuna tıklanır ve üst kısımdaki haritada o bölge görülebilir.



Haritanın alt kısmında da görüldüğü üzere 3 adet daha parametre gerekmektedir. Bunlar;

- **Configuration Name**: Yapılandırma ismi. Zorunludur.
- **Expire Time (min)**: Dakika cinsinden gecikme süresi. Tanımlanan süre sonunda işlem otomatik olarak es geçilecektir. Opsiyoneldir.

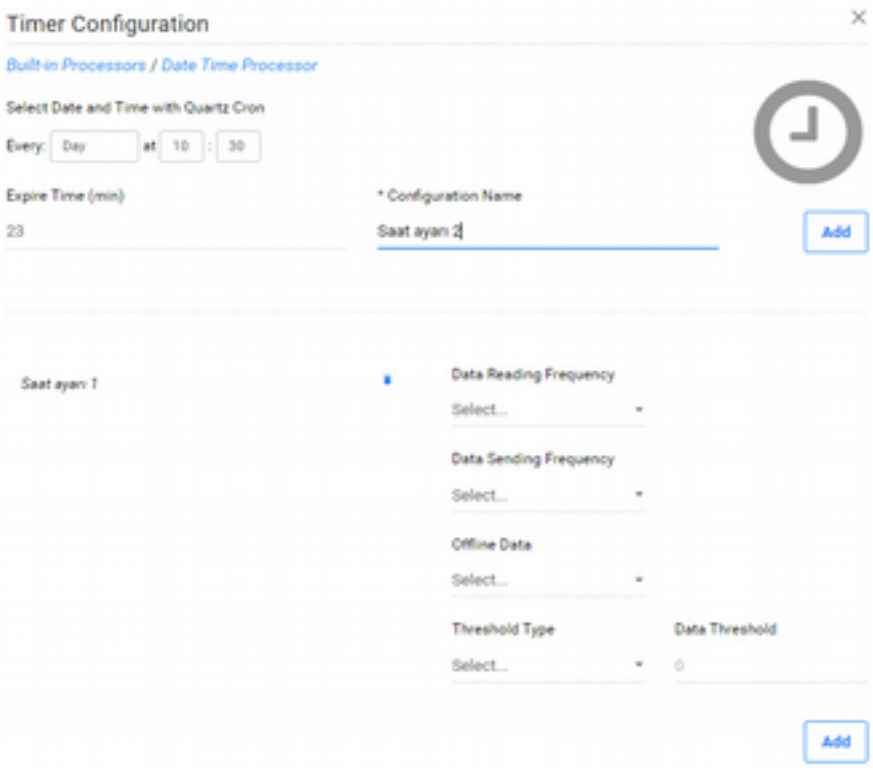
- **Dwell Time (min)**: Dakika cinsinden bekleme süresi. Gateway'in, belirlenen konum çemberine girdikten sonra Geofence sensörünün tetikleneceği bekleme süresi.

Harita üzerinde çember aracı ile bölge belirlendikten sonra **Set This Geolocation Settings** (Bu Geolocation Ayarlarını Yap) butonuna tıklandığında, belirlenen konum, yapılandırma ayarlarına eklenecektir. Diğer varsayılan genel yapılandırma ayarları ile de Geolocation sensörünün veri okuma, gönderme ve kaydetme ayarları yapılandırılmalıdır. Son olarak **Add** butonuna tıklanarak yapılandırma kaydedilir.

Sensör tetiklendiğinde, IoT-Ignite Cloud'a, haritadan seçip tanımlamış olduğunuz lokasyonun ismi, yani **Configuration Name**'i gönderilir. Böylelikle hangi konuma Gateway'in girdiği anlaşılabilir.

Date-Time Processor ile Gelişmiş Zamanlanmış Yapılandırmalar Yapmak

Gateway tarafında belirli bir zamanın periyodunun tekrarlaması ile birlikte bir olay (event) oluşturulması sağlanabilir. Aynı Geofence sensöründe olduğu gibi bir Processor olarak çalışmaktadır. Bir veya birden fazla zaman periyodu tanımlanır ve hangi zaman periyodu o an gerçekleşmişse, IoT-Ignice Cloud'a o zaman periyodunun Configuration Name'i gönderilir.



Yukarıdaki şekli incelediğinizde yine sağ alt köşede genel yapılandırma ayarları yer almaktadır. Sol tarafta ise "Saat ayarı 1" şeklinde, o an tanımlanmış olan bir zaman periyodu görülmektedir.

Üst kısımda yer alan seçim menüsü ile **Quartz Cron** formatı ile her dakika, her saatin belli bir dakikasında, her günün belli bir saatinde, her haftanın belli bir gününün belli bir saatinde, her ayın belli bir gününün belli bir saatinde ve son olarak her yılın belli bir ayının belli bir gününün belli bir saatinde olacak şekilde zaman periyotları belirlenebilir. Aşağıdaki örnekte her yılın Ocak ayının 1. gününde saat 00:00 olarak seçim yapılmıştır.

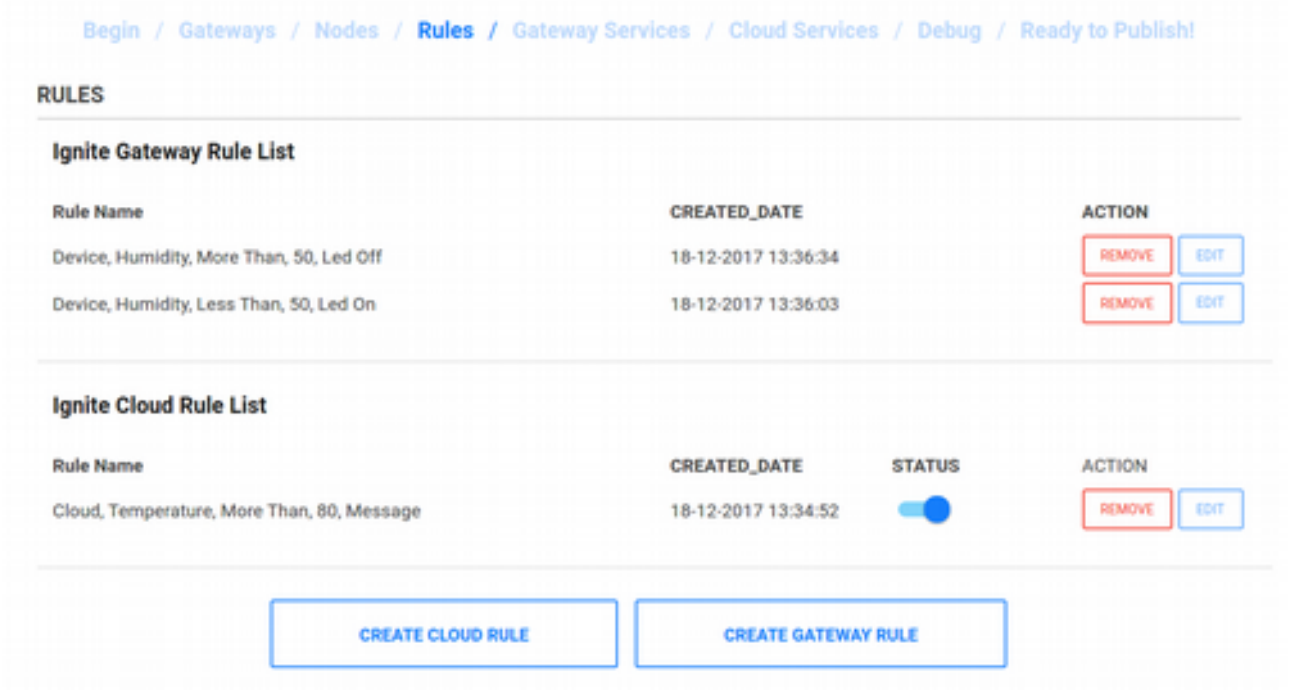
Zaman periyodu seçilmesi zorunlu bir opsiyondur. Ek olarak aşağıdaki iki opsiyonu daha alır.

- **Expire Time (min)**: Dakika cinsinden gecikme süresi. Opsiyoneldir. Örneğin zaman periyodu Minute olarak seçilirse, her dakika başında IoT-Ignite Cloud'a o yapılandırmanın isminin gönderilmesi gerekir. Ancak Expire Time değeri 5 verilirse, her dakika başı gitmesi gereken değer 5 dakika boyunca askıya alınır ve 5., 10., 15... gibi dakikalarda IoT-Ignite Cloud'a yapılandırma ismi gönderilir.
- **Configuration Name**: Yapılandırma ismi. Zorunludur.

Yapılandırmaları Güncellemek

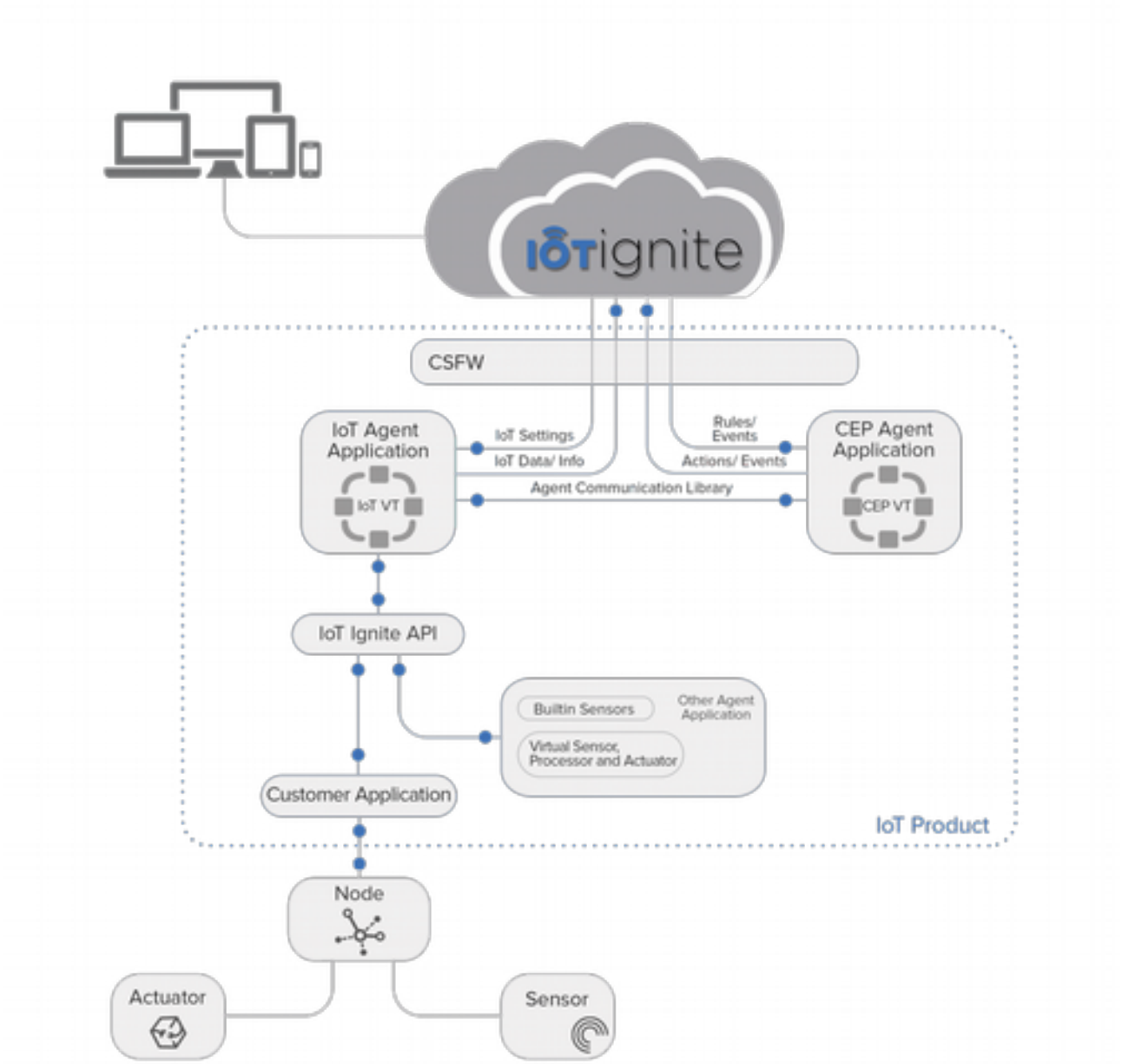
Daha önce tanımladığınız sensör veri yapılandırmalarını güncellemek için tekrar o sensöre tıklayıp açılan yapılandırma arayüzünden parametreleri değiştirip **Add** butonu ile güncelleyebilirsiniz. Eğer yapılandırmayı tamamen silmek istiyorsanız, menüden **Gateway Services** butonuna tıklayıp, çalışacağınız modu seçin. **Sensor Data Config** sekmesinde yer alan listede ilgili sensör yapılandırmasını bulup çöp kutusu ikonuna tıklayarak silebilirsiniz. Ancak unutmayın, bu yapılandırma sadece IoT-Ignite Cloud'ta envanterden silinmiş olur, Gateway'den silinmez. Gateway'de de silmek için güncellenen modu tekrar Gateway'e push etmeniz gerekmektedir.

Kurallar (Rules)



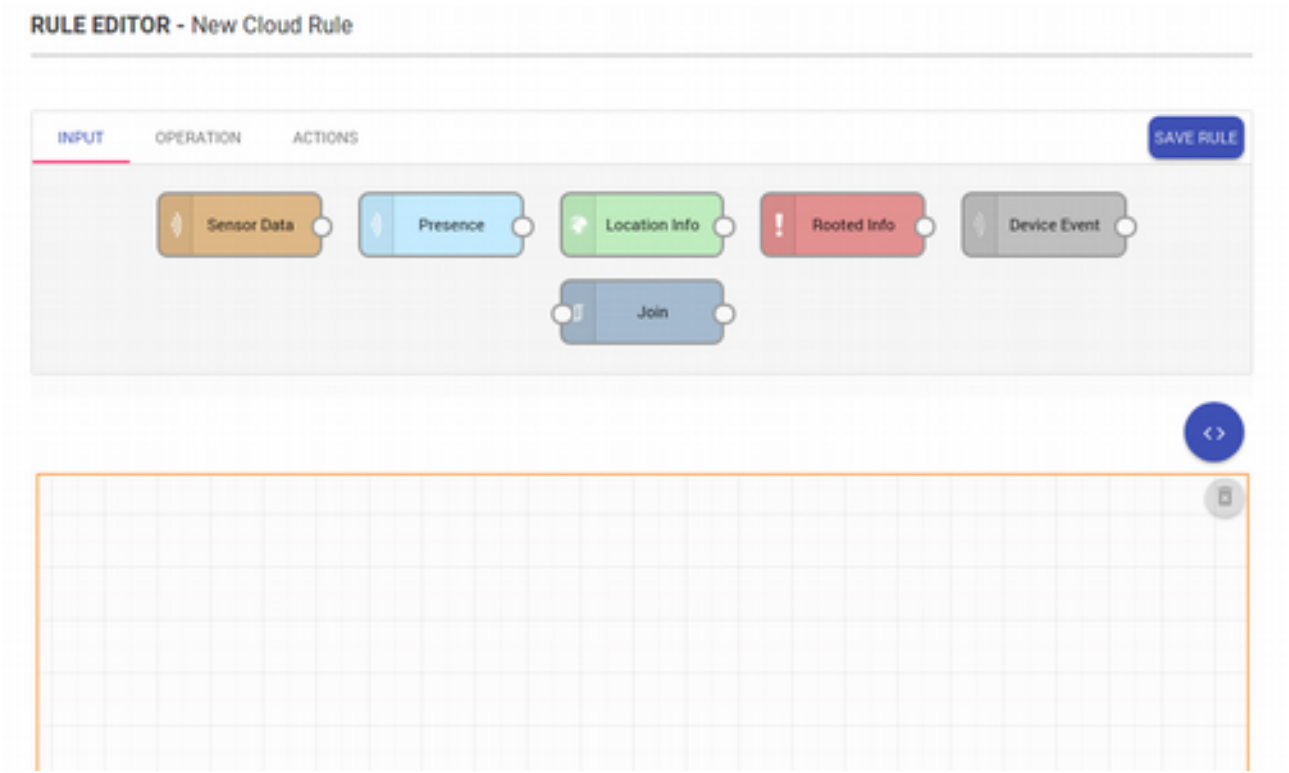
Kuralları Demo akışında görmüştük ve ne işe yaradıklarını Gateway üzerinde test ederek incelemiştik. Ancak Demo akışında basit bir editör ile basit kurallar tanımlamıştık. EHUB'ta ise daha gelişmiş bir Rule Editor (Kural Editörü) kullanarak daha karışık ve kompleks kurallar tanımlamak mümkündür.

Kural oluşturmak için üst menüden Nodes sayfasına girilmelidir. Sayfaya girdiğimizde, Demo akışında tanımladığımız kuralları görebiliriz. Bunları **REMOVE** (Kaldır) butonuna tıklayarak silebiliriz. Bu aşamada silinen kurallar direkt olarak Gateway'e de Mod içerisinde bildirilecektir (Yani mod içinden kural kaldırılır ve güncel mod tekrar gönderilir).



Kural Editörü (Rule Editor)

Yeni bir Gateway veya Cloud Rule oluşturmak için, ekranın alt tarafında yer alan **CREATE CLOUD RULE** veya **CREATE GATEWAY RULE** butonlarına tıklanır. Seçiminize göre sayfanın alt kısmında Rule Editor açılacaktır. Aşağıdaki şekilde Cloud Rule oluşturmak için Rule Editor arayüzü açık olarak görülmektedir.



Kural Editörü, herhangi bir teknik altyapıyı yönetmek zorunda kalmadan, IoT-Ignite Cloud'a bağlı Gateway'ler tarafından global ölçekte üretilen verileri toplayan, analiz eden ve bunlara göre karar veren IoT servisleri oluşturmayı mümkün kılar. Kural Editörü sayesinde **Karmaşık Olay İşleme** (CEP) yapılabilir. Kural veya kurallar zinciri tanımlanırken görsel olarak sürükle-bırak yöntemi ile rahatlıkla CEP tanımlanabilir. Bu kurallar, Cloud veya Gateway'de çalışabilir.

Kural Editörü'nde veri kaynağı (**INPUT**), durum (**OPERATION**) ve aksiyon (**ACTION**) bileşenleri sürükle bırak yöntemi ile editörün boş alanda tasarlanır. Bu bileşenlerin her birine fare ile çift tıklanarak parametreler girilir ve yapılandırılır. Son olarak her bir bileşen birbirine bağlantı noktalarından akış ipleri ile bağlanarak birleştirilir. Nihayetinde istenilen kural oluşturulmuş olur.

Tanımlanacak olan kuralların sayısı ile ilgili herhangi bir kısıtlama yoktur. İstediğiniz zaman herhangi bir kuralı etkinleştirebilir, devre dışı bırakabilir veya tamamen silebilirsiniz.

Kural Oluşturma Akışı

Devzone'da bir kural oluşturabilmek için ilk olarak **Rules** sayfasına girip, sayfanın altında yer alan **CREATE CLOUD RULE** veya **CREATE GATEWAY RULE** butonlarından birini tıklamamız gerekmektedir. Bundan sonra, alt alanda boş bir tuval ile Rule Editor karşımıza çıkacaktır. Şimdi bu editörde neler yapabileceğimizi görelim.

> Rule Editor'ü incelerken Cloud Rule üzerinden gideceğiz. Cloud Rule, Gateway Rule'a göre daha geniş opsiyonlara sahiptir. Bu opsiyonlardan bir kısmı Gateway Rule ile de ortaktır. Bu nedenle Cloud Rule'u öğrendiğinizde zaten Gateway Rule'u da öğrenmiş olacaksınız.

Veri Kaynağı (INPUT)

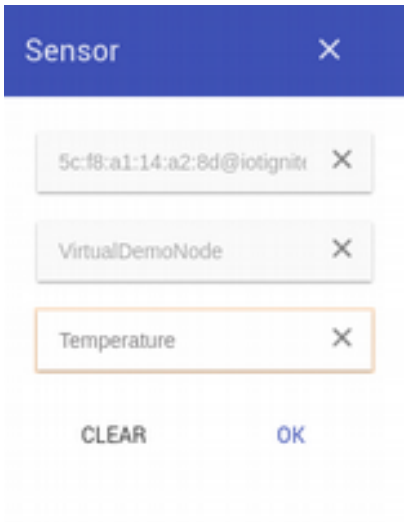
Verinin kaynağını belirten bileşenleri tanımlar. Şimdiye kadar sadece sensör verilerini görmüştük. Şimdi diğer veri kaynaklarına da değineceğiz.

Kural Editörü'nde yer alan bileşenler yukarıdaki şekilde görülmektedir. Toplamda 5 adet girdi ve 1 adet de karmaşık kurallarda kullanacağımız **Join** (Birleştirme) bileşeni var. Bunlardan her birini fare ile tutup alt alanda yer alan boş tuvale çekerek bırakabilirsiniz. Ardında çektiğiniz bileşene fare ile çift tıklayıp parametrelerini tanımlamanız gerekmektedir.

> Rule'ların çalışabilmesi için Sensör Veri Yapılandırmaları'nın yapılması gerektiğini unutmayın!

- **Sensor Data**: Gateway'lerde yer alan sensörlerin seçimi yapılır. Bu sensörlerden gelen veriler büyük-küçük gibi karşılaştırma operatörleri ile kontrol edilerek çeşitli aksiyonları çağrılır.

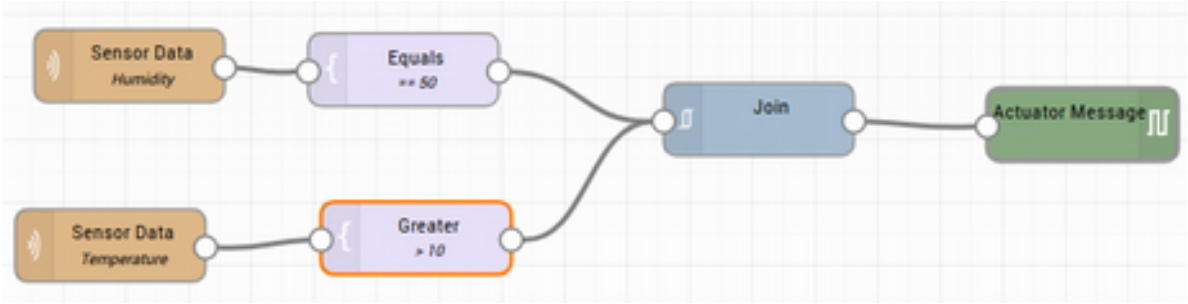
  Seçim işlemi yapılırken yukarıdan aşağıya olacak şekilde ilerlenir. İlk kutudan Gateway seçilir. Ardından ikinci kutuda seçtiğiniz Gateway'e ait Node'lar listelenir. Node seçtiğinizde de yine seçilen o Node altındaki Sensör'ler listelenir. Seçim işleminden sonra OK butonuna tıklanarak işlem tamamlanır.



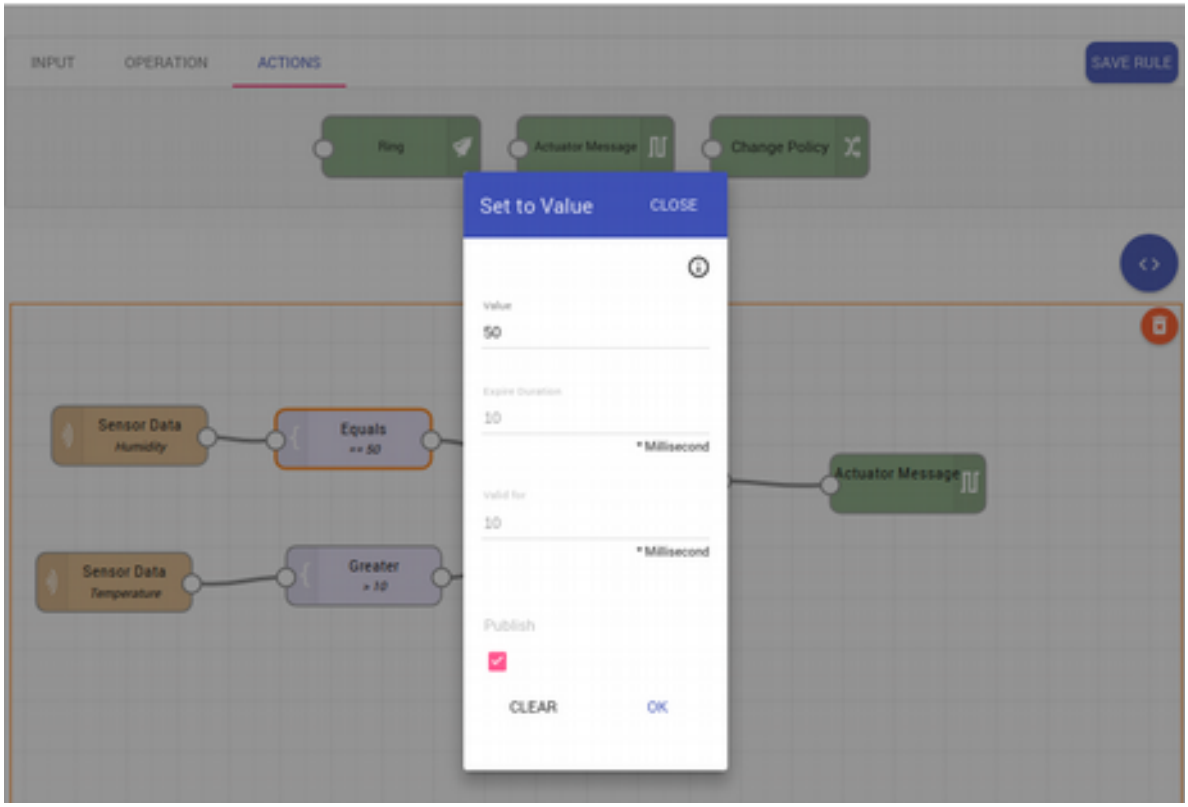
- **Presence:** Gateway Rule ile ortak bir bileşendir. Varlık kontrolü yapar. Direkt olarak sadece Gateway, Node veya Sensör seçimi yapılabilir. Eğer hiçbir seçim yapılmazsa, o hesap üzerindeki tüm Gateway'ler için geçerli olur. Bu girdi bileşeni için sadece **Equal** (Eşittir) operatörü kullanılabilir ve **ONLINE** ve **OFFLINE** string değerleri eşitliği kontrol edilir.
- **Location Info**: Gateway Rule ile ortak bir bileşendir. Gateway, IoT-Ignite Cloud'a birtakım veriler göndermektedir. Bu bilgilerin içinde konum bilgisi de yer almaktadır. Konum bilgisinde enlem, boylam ve servis sağlayıcının ismi bulunmaktadır. Bu bileşende konum bilgisi izlenecek olan Gateway seçilir. Bu girdi bileşeni için sadece Location (Lokasyon) operatörü kullanılabilir ve bu operatörde de harita üzerinden lokasyon seçimi yapılır (Aynı Geofence sensöründeki gibi).

- **Rooted Info**: Gateway Rule ile ortak bir bileşendir. Gateway'in Root edilmiş veya edilmemiş olduğu bilgisidir. Bu girdi bileşeni için sadece Equal (Eşittir) operatörü kullanılabilir ve TRUE ve FALSE boolean değerlerinin eşitliği kontrol edilir.
- **Gateway Event:** Gateway Rule ile ortakt bir bileşendir. Daha önce tanımlanmış olan Gateway Rule'larının Gateway'de gerçekleşmesi ile birlikte tetiklenir. Gateway Rule ve Gateway'in seçilmesi zorunludur. Kuralın hangi Gateway tarafında çalıştığı belirtilmelidir. Bununla birlikte Gateway kuralında eğer karmaşık bir yapı kurularak birden fazla sensör ile ilişkilendirilmişse ve belli bir sensörün değerinin belirtilen değer eşiğine ulaşması ile de Gateway Event oluşturulabilir.

  Örneğin aşağıda yer alan karmaşık kuralda Humidity değeri 50'ye eşitse ve Temperature değeri 10'dan büyükse (bunlar Join ile birleştiriliyor) Actuator Message aksiyonu oluşturularak VirtualDemoNode Node'u altında Lamp Actuator'üne 1 mesajı gönderilerek lamba açılıyor.



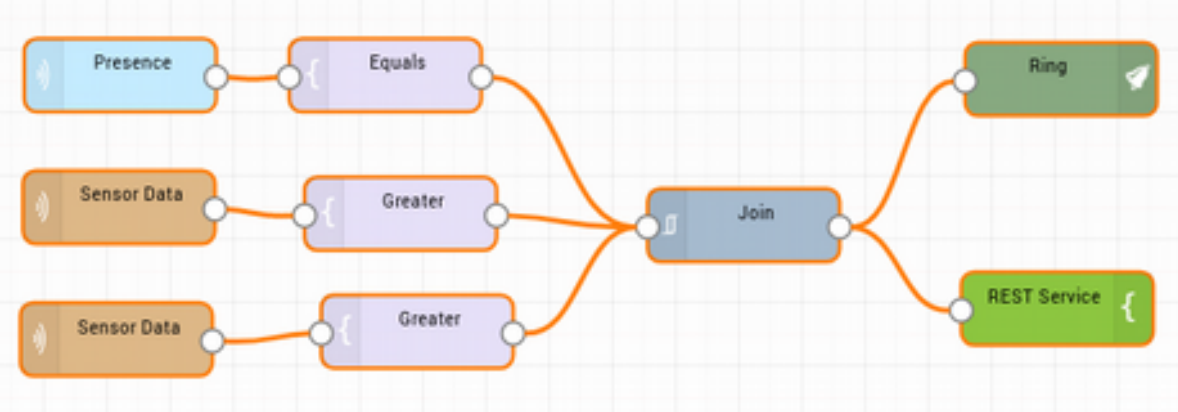
Ancak bu kuralda önemli olan Humidity değerinin 50'ye eşit olması ise, Equals operatöründe **Publish** opsiyonunun aktif edilmesi ile Gateway Event oluşturulabilir. Diğer koşul sağlanmasa ve kural tamamlanamasa da Gateway Event tetiklenecektir.



Bu girdi bileşeni için sadece **Equal** (Eşittir) operatörü kullanılabilir ve **VALID** string değerleri eşitliği kontrol edilir.

- **Join**: Gateway Event örneğimizde gördüğümüz gibi birden fazla kaynaktan gelen verinin birleştirilerek VE mantıksal işlemine tabi tutulmasıdır. Operasyon alanında yer alan karşılaştırma bileşenleri ile bağlanabilmektedirler.

  Aşağıdaki örnekte 2 adet Sensör verisi ve 1 adet de Presence verisi değerleri Join ile birleştirilmiş. Eğer her üçünden de gelen veriler, karşılaştırma bileşenlerinde tanımlanan eşiklerden geçerse Ring ve REST Service aksiyonları çalıştırılacaktır. Birleştirme işlemi yapmak için bileşenlerin sağ ve sol taraflarında yer alan beyaz yuvarlak alanlara fare ile basılı tutup diğer bağlanacak olan beyaz alana bağlantı ipini çekip bırakmak yeterlidir.



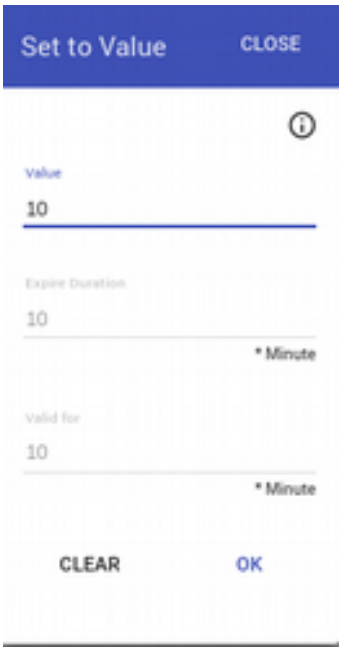
Operasyon (OPERATION)

INPUT grubunda yer alan veri kaynaklarından gelen değerler için bir değer eşiği tanımlanır ve sensörden gelen değer ile bu değer eşiği bileşenin türüne göre **TRUE** sonucunu verirse, **ACTIONS** grubunda yer alan aksiyonlara doğru akış devam eder.

Normal programlama dillerinden de alışık olduğunuz 4 adet karşılaştırma operatörü (eşit, eşit değil, büyük, küçük) ile birlikte 2 adet de özel operatör bulunmaktadır. Bunlar; Location (Sadece Cloud Rule'da kullanılır) ve Geofence'dir.

Operasyon bileşenleri sol tarafından veri kaynaklarına (INPUT) bağlanır, sağ taraftan ise ya Join bileşenine ya da aksiyonlara (ACTIONS) bağlanır.

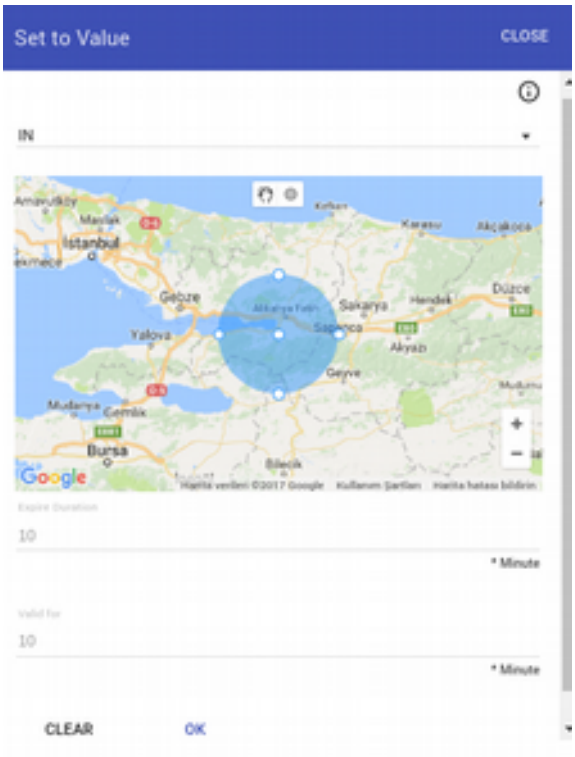
- Equals: Eşitlik operatörüdür. Girilen eşik değeri ile gelen verinin eşitliği kontrol edilir.



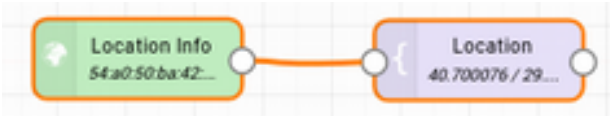
Yukarıdaki şekilde görüldüğü gibi 3 adet parametre değeri girilir. **Value**, zorunludur. Gelen verinin kontrol edileceği veri eşiğidir.

Expire Duration: Koşulun geçerlilik süresi parametresidir. Bu süre dolana kadar ikinci bir aksiyon tetiklemesi gerçekleştirilmez. **Valid for**; geçerlilik süresidir. Belirtilen süre boyunca, gelen verinin belirtilen eşik değerini sağlaması, yani kuralın geçerli olması gerekmektedir. Expire Duration ve Valid for parametreleri **dakika** cinsinden değer alır ve diğer bileşenlerinde ortak kullanılmaktadır.

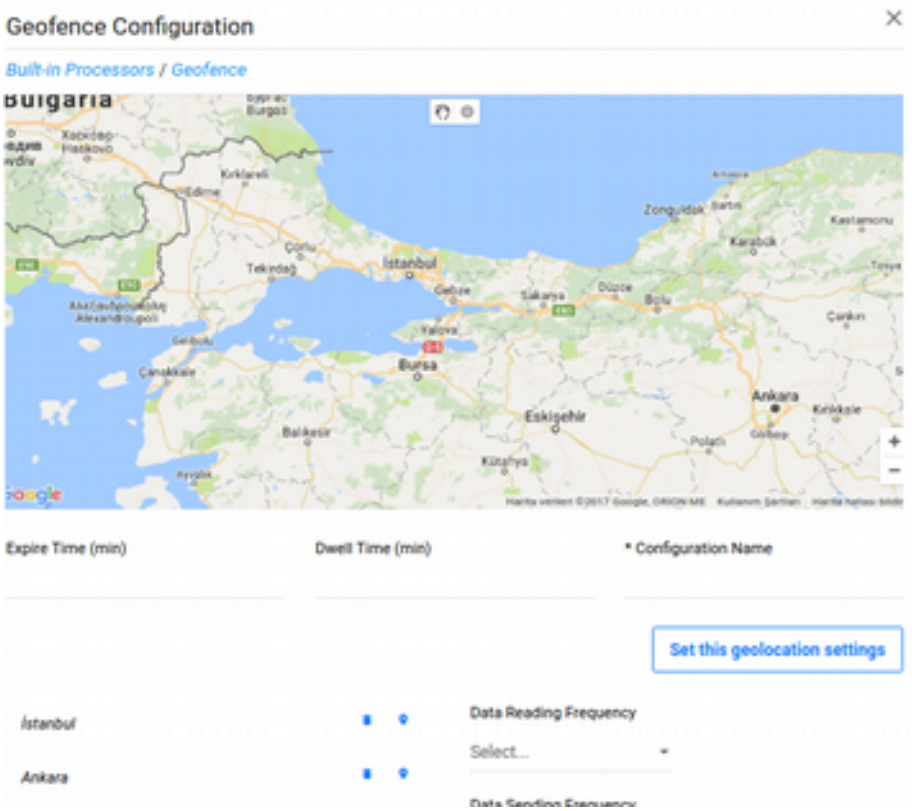
- **Not Equals:** Eşit değil operatörüdür.
- **Greater:** Büyüktür operatörüdür.
- **Lower:** Küçüktür operatörüdür.
- **Location:** Gateway'in, haritadan dairesel olarak seçilen bölge içine girmesi veya o alandan çıkması ile birlikte gerçekleşir. **INPUT** olarak **Location Info** bileşeni ile birlikte çalışır. Location Info bileşeninden Gateway seçilir. Location bileşeninde ise aşağıdaki gibi haritadan bir bölge seçilir. Bölge içine girdiğinde tetiklenmesi için **IN**, bölgeden çıktığında tetiklenmesi için **OUT** parametresi seçilir.



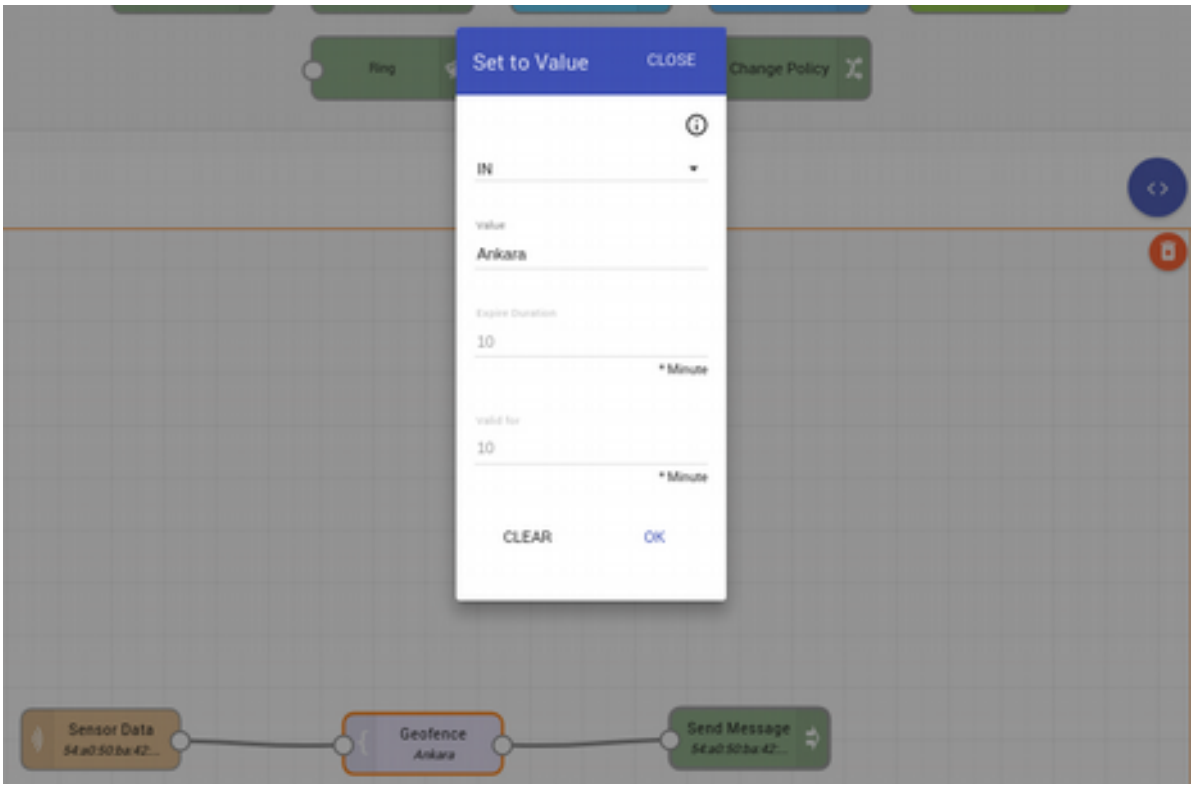
Aşağıda, örnek bir kullanım görülmektedir.



- **Geofence: Sensor Data** bileşeni kullanılarak Gateway'in **Built-in Processor**'lerinden **Geofence** seçilerek kullanılır. Sensör veri yapılandırma ayarlarından Geofence için yapılan harita üzerindeki yapılandırma ile belirlenen dairesel çember bölgeye giriş (**IN**) ve çıkış (**OUT**) tanımlama yapılır. Geofence yapılandırmalarında birden fazla bölge tanımlaması yapıldığı için **Value** alanına, o bölgesel tanımlamanın **Name**'i yazılır. Mesela aşağıdaki gibi bir Geofence yapılandırmamız olduğunu varsayalım.

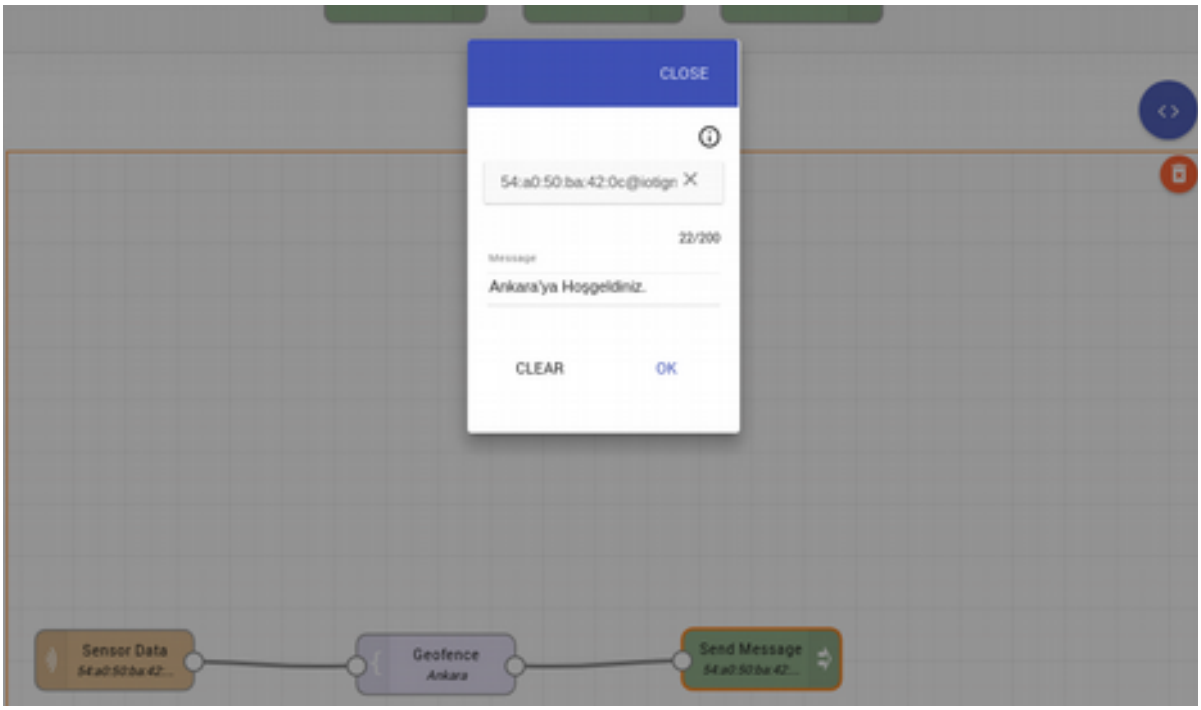


Yapılandırmaya baktığımızda 2 adet bölge eklenmiş. Bunlar; **İstanbul** ve **Ankara**'dır. Şimdi örnek olarak Gateway Ankara'ya girdiğinde Gateway ekranında "Ankara'ya Hoşgeldiniz." mesajı yazdıran bir kural tanımlayalım.



Sensor Data'da Gateway'in **Geofence**'i seçildi. **OPERATION**'da ise Geofence bileşeni yukarıdaki gibi yapılandırılarak **Value** değerine **Ankara** yazıldı. **Send Message** (Mesaj

Gönder) aksiyon bileşeninde de Gateway seçildi ve ekranda gösterilecek olan mesaj da **Message** alanına yazılıp kaydedildi.



> Actions'larda hedef olarak başka Gateway'ler de seçilebilir, yani INPUT'ta kaynak olarak gösterilen Gateway olmak zorunda değildir. Böylece Gateway'leri birbiri ile ilişkilendirebilirsiniz. Eğer birden fazla Gateway'e aksiyon tanımlanacaksa, her biri ayrı aksiyon bileşeni ile tanımlanmalıdır. Eğer ki çok fazla Gateway varsa, bu iş için EHUB'ta Gateway'leri bir grup altında toplayıp etiketlendirebilirsiniz. Aksiyonlarda da hedef olarak bu grup etiketi seçilirse, o grup altındaki tüm Gateway'ler için aksiyon çalıştırılır.

Aksiyon (ACTION)

Kural tanımlamanın son adımında aksiyonlar seçilir. Sensörlerden gelen verilerin mantıksal işlemlerden geçmesi sonucunda eğer kural şartları sağlanıyorsa, aksiyon mesajlarına geçilir ve birazdan göreceğimiz aksiyonlardan bir veya daha fazlası çalıştırılır.

- **Ring:** Gateway ve Cloud Rule'larında ortak bir aksiyon bileşenidir.

  Demo akışında da gördüğümüz bu aksiyon, Gateway'de bir uyarı sesi çaldırır. İsterseniz bu uyarı sesini başka bir ses dosyası ile değiştirebilirsiniz. Ses dosyası, **Mod** içerisinde **Content Store**'da yer almak zorundadır. Mod'u alan Gateway'e ses dosyası da kendisi için tanımlanan **path** (yere) indirilecektir. Ek olarak sesin seviyesi (1-10 arası) ve çalma süresi de saniye cinsinden tanımlanabilir.

  Aşağıdaki örnek yapılandırmada **6 ses şiddeti** ile **1 saniye** ile **sdcard** içindeki **alarm.mp3** çalacaktır (İlerleyen aşamalarda Content Store'a dosya yüklemeyi ve path tanımlamayı göreceksiniz).

- **Actuator Message:** Gateway ve Cloud Rule'larında ortak bir aksiyon bileşenidir.

  Demo akışından hatırlayacağınız üzere Lamp sensörü hem veri gönderen hem de veri alan bir yapıdaydı. Sensöre 1 değerini gönderdiğimizde lamba açılıyor, 0 değerini gönderdiğimizde de lamba kapanıyordu. Actuator Message aksiyon bileşenini kullanarak istediğimiz bir Gateway'deki Actuator'e istediğimiz veriyi 200 karakter sınırı ile (int, string, json, array gibi) push edilebilir.

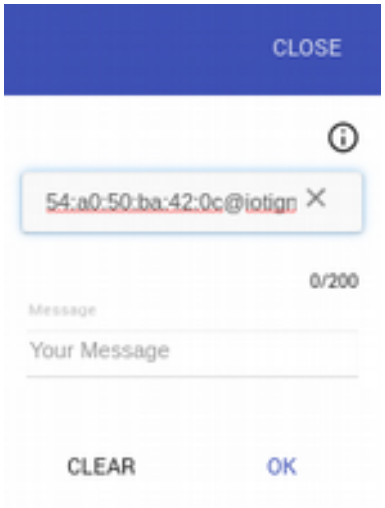
- **Change Policy**: Gateway ve Cloud Rule'larında ortak bir aksiyon bileşenidir.

  İstenilen Gateway'in Policy, yani politikalarının değiştirilmesini sağlar. Politika, mod içinde bulunur. Mod içinde de bir veya birden fazla politika bulunabilir.

Devzone'a hesap açıldığında varsayılan olarak DEMO, DEVELOPMENT, PRODUCTION (Devzone , bir geliştirme ortamı olduğu için görünmez, sadece EHUB'ta görünür) ve ANDROIDTHINGS modları otomatik olarak tanımlanmıştır. Bunlar da yine kendi isimleri ile birer Policy içermektedir. Policy'ler Mode içindedir. Yani DEMO Policy'si DEMO Mode'u içinde yer almaktadır. Birden fazla politika içeren mod Gateway'de yer aldığında, istenilen durumda istenilen politika aktif hale getirilebilir. Yeni bir mod oluşturmak için Begin sayfasından giriş yapabiliyorduk ve Gateway Services sayfasında da istenilen modu yapılandırabiliyoruz (ilerleyen konularda göreceğiz). Modlara uygulamalar, dosyalar eklenebilir, ağ yapılandırmaları yapılabilir (bluetooth kapatılabilir, VPN engellenebilir, USB devre dışı bırakılabilir, dosya indirme engellenebilir vs.), sensör verileri yapılandırılabilir, Gateway kuralları aktif veya pasif yapılabilir, Gateway Kiosk moduna geçirilebilir...

- **Send Message:** Sadece Gateway Rule'da kullanılır.

  Demo akışında da görmüştük. Gateway'e string olarak bir mesaj gönderilebilir. Mesaj, Gateway'in ekranında (eğer varsa) gösterilecektir.



- **REST Service:** Sadece Gateway Rule'da kullanılır.

Kuralın sağlanmasıyla birlikte bir REST servisine çağrı yapılabilir. 3 adet parametre tanımlanmaktadır. Bunlar; URL, Token ve Data'dır.

URL: REST servisinin adresi.

Token: REST ile kimlik doğrulaması yapılabilmesi için gerekli olan anahtar. Bu token, REST çağrısı yapılacağı zaman header'a eklenmektedir.

Data: HTTP Post ile REST çağrısı yaparken gönderilecek olan veridir.

Aşağıdaki projeyi klonlayarak REST kullanımını detaylıca inceleyebilirsiniz.

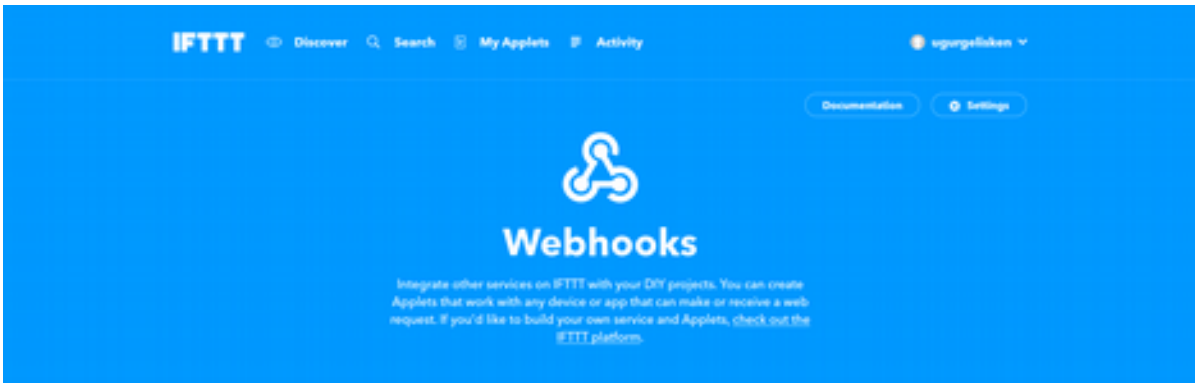
```
git clone https://github.com/IoT-Ignite/sample-rest-service.git
```

IFTTT, Twitter ve Send Mail aksiyon bileşenlerinin mutlaka kimlik doğrulama ve bir takım yapılandırma işlemlerinin EHUB üzerinden yapılması gerekmektedir. Her biri için yapılandırma ayarlarını sırasıyla göreceğiz. Yapılandırmaları yaptıktan sonra bu aksiyon bileşenleri ile ister EHUB'tan ister Devzone'da kural oluşturmaya devam edebilirsiniz. Yapılandırmalar yapılmazsa, her biri için kullanılamaz uyarıları çıkacaktır.

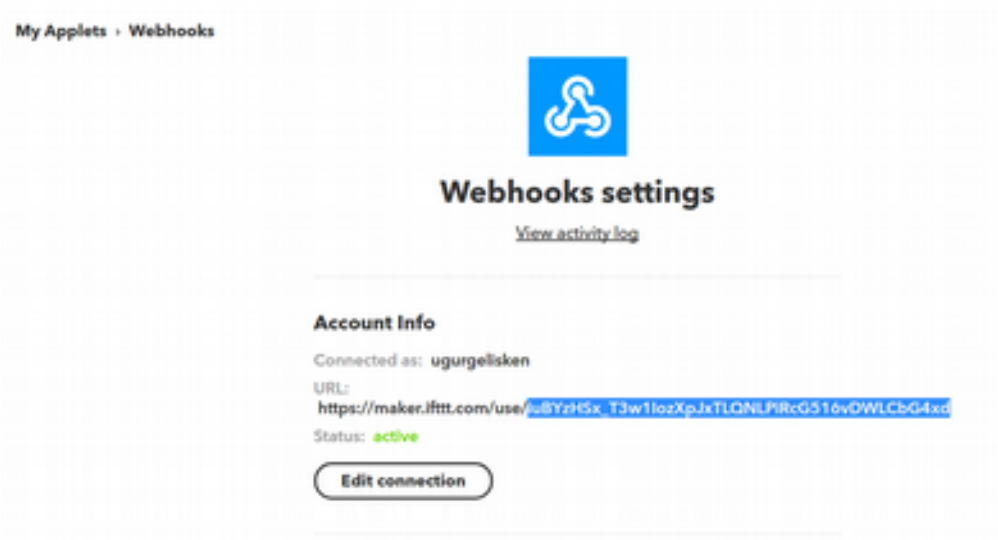
- **IFTTT:** Sadece Gateway Rule için EHUB'ta kullanılmaktadır.

  Açılımı "If This Then That" olan, anlamı ise "Bu Olursa Bunu Yap" demek olan bu platform, kısaca kullandığınız servisleri birbirine bağlamayı ve bu servislerin kullanımını kolaylaştırmayı sağlamaktadır.

  IoT-Ignite ile IFTTT kullanabilmek için bir **IFTTT Key**'i edinmeniz gerekmektedir. https://ifttt.com adresine girip üyelik açarak bir **KEY** edinebilirsiniz. Üyelik işleminden sonra https://ifttt.com/maker_webhooks adresine girip **Connect** butonu ile bağlantı kurun.

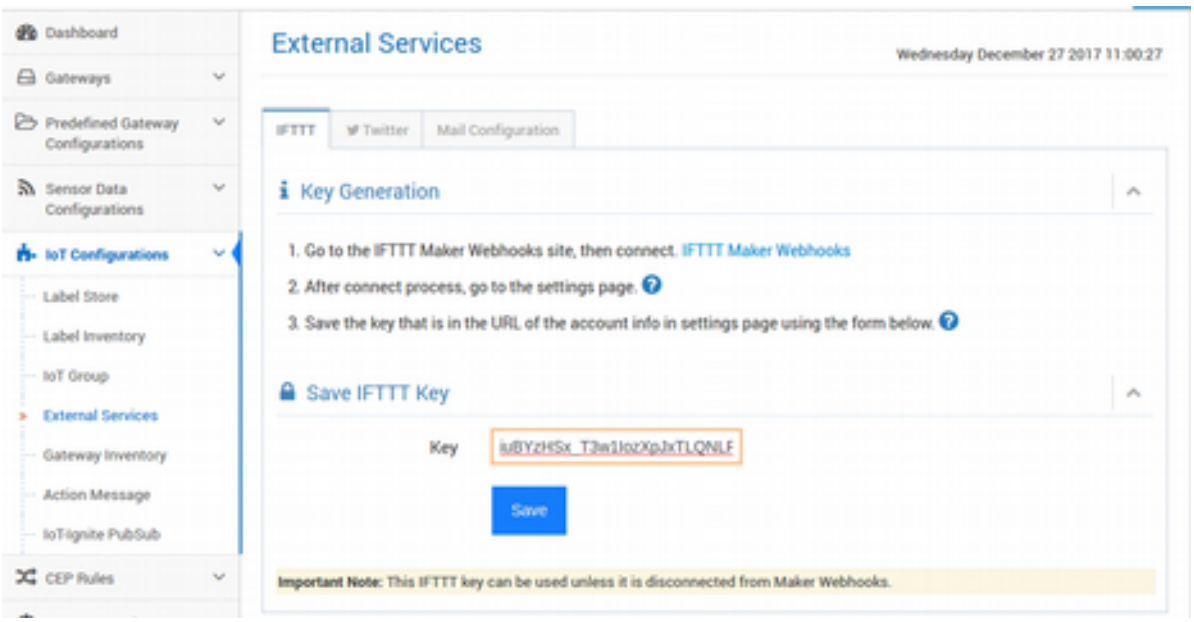


Açılan sayfada sağ taraftaki **Settings** butonuna tıklayın.

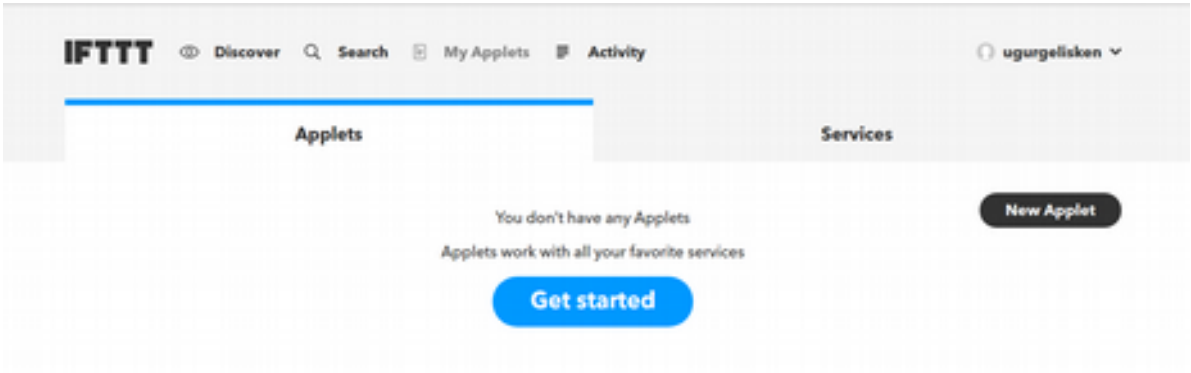


Gelen sayfada, bağın sonunda yer alan alfanümerik seri, sizin IFTTT KEY'inizdir.

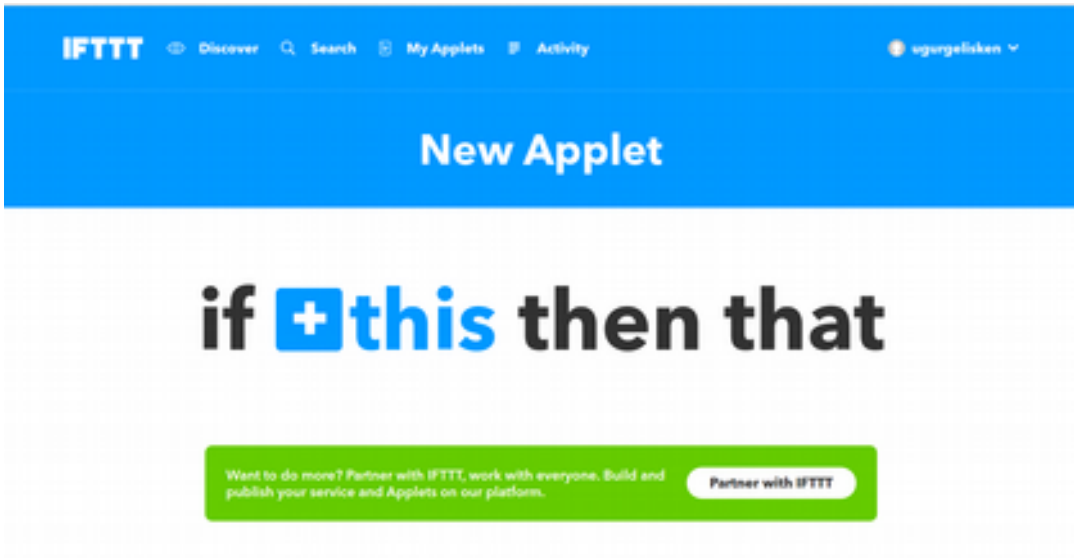
Elde ettiğiniz bu IFTTT KEY'i **EHUB'ta** IoT **Configurations > External Services** (Harici Servisler) sayfasında IFTTT yapılandırma kısmında girilerek **SAVE (Kaydet) b**utonuna tıklanarak kaydedilir. Böylece IFTTT ile bağlantınız kurulmuş olunur.



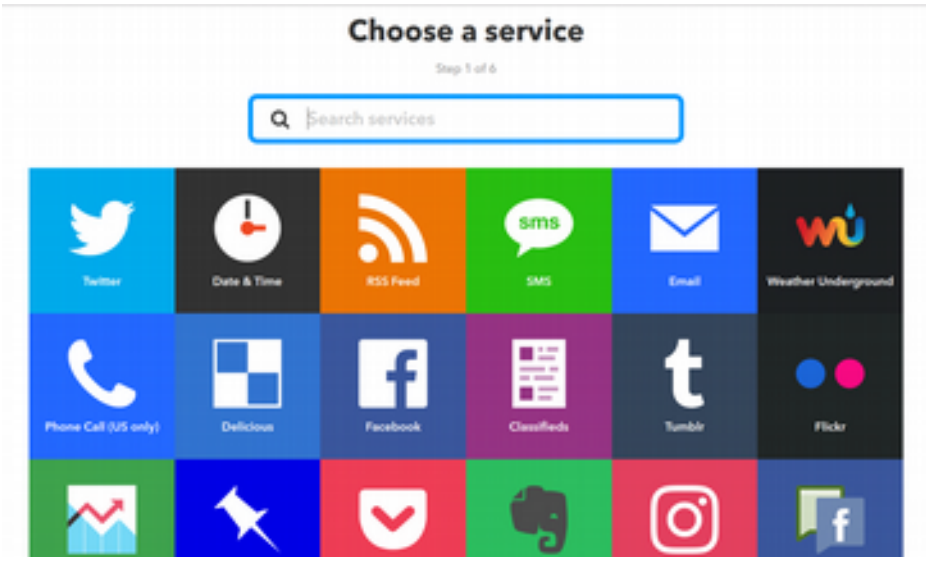
Örnek olması için bir IFTTT servisi yapalım. Üst menüden **My Applets**'a tıklayın. Açılan sayfada sağ tarafta yer alan **New Applet** butonuna tıklayın.



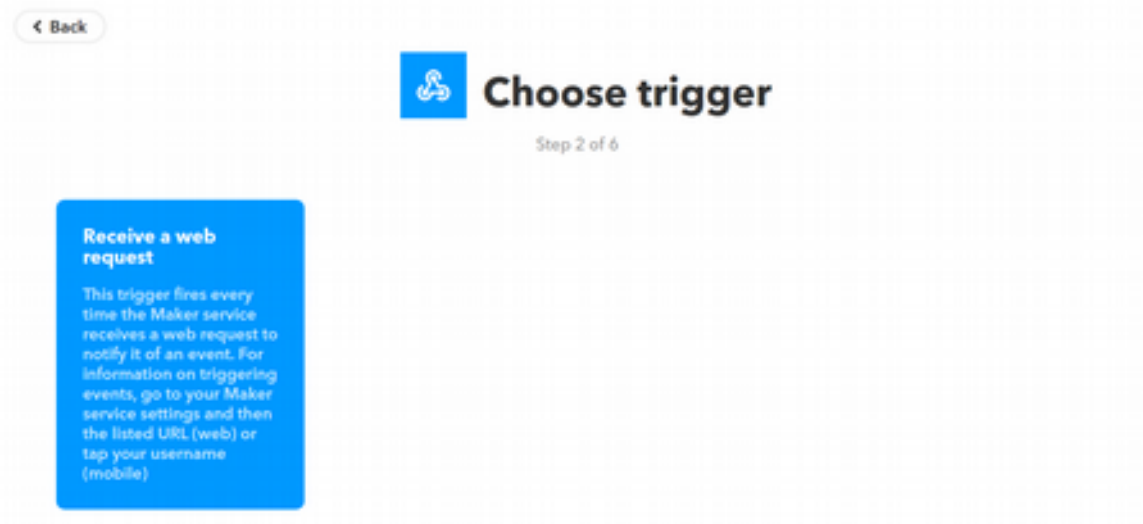
Sonraki sayfada da mavi renkli **+this** butonuna tıklayın.



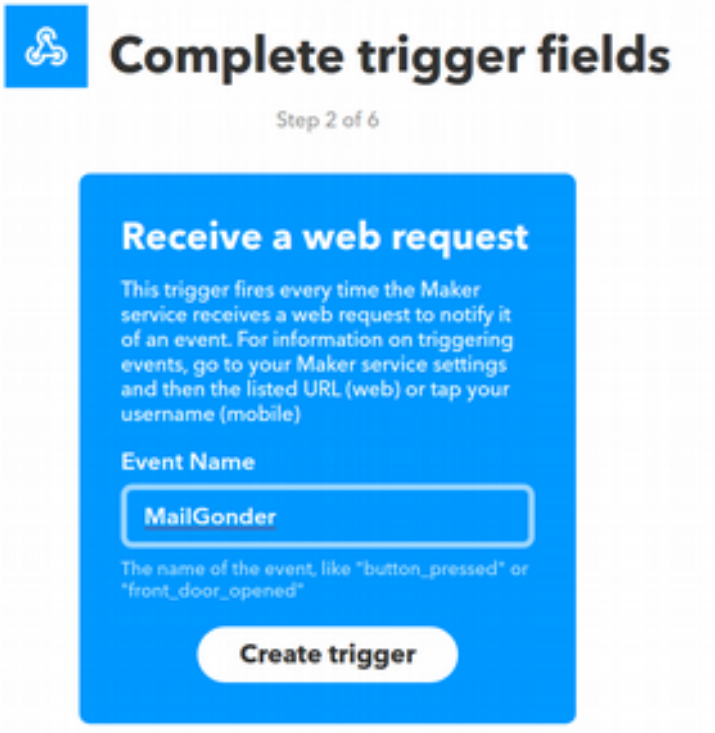
Örneğimizde e-posta gönderme gibi basit bir işlem yaptıracağız. Listeden **Webhooks**'u seçin.



Choose Trigger (Tetikleyici Seç) sayfasında **Receive a web request**'i seçin.

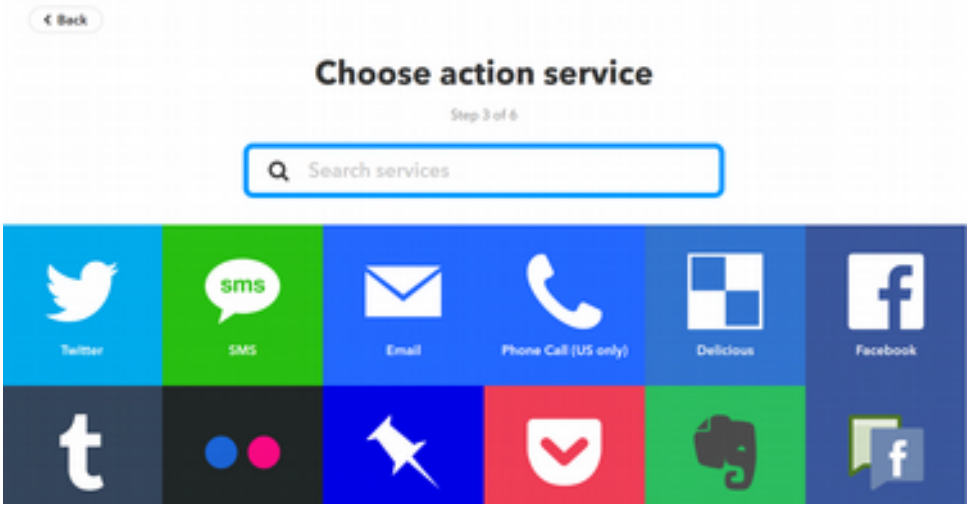


IFTTT servisi için bir isim girin. Örneğimizde **MailGonder** adını girdik. Ardından **Create trigger** butonuna tıklayın.

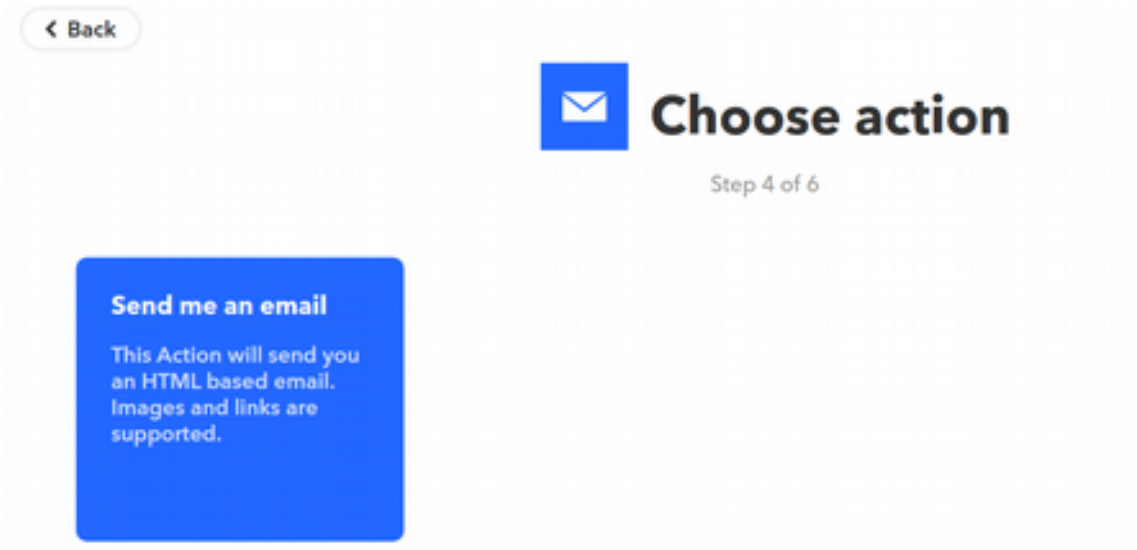


Bir sonraki adımda karşınıza aşağıdaki gibi bir ekran gelecek. +**that** butonuna tıklayın ve ilerleyin.

Açılan **Choose action service** (Aksiyon servisi seç) sayfasında listeden **Email**'i bulun ve seçin.



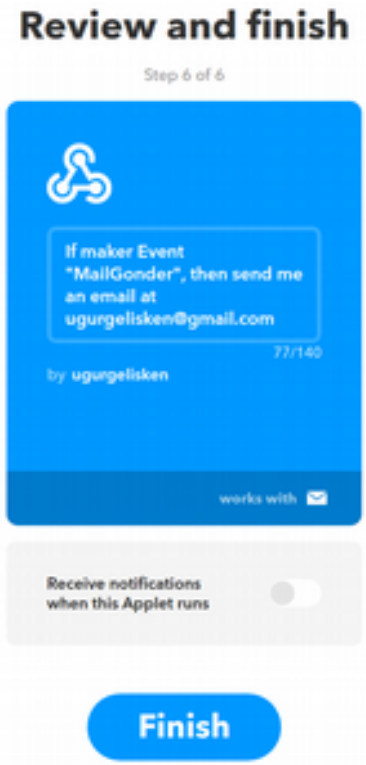
Bir sonraki sayfa olan **Choose action** (Aksiyon seç) sayfasında **Send me an email**'i seçin.



Nihayetinde karşınıza e-mail şablonu çıkacak.

Email şablonunda görüldüğü gibi **EventName**, **OccuredAt**, **Value1**, **Value2** ve **Value3** değerleri parametrik olarak tanımlanmış. Yani bu parametreler beslenerek e-posta gönderme işlemi yapılabilir.

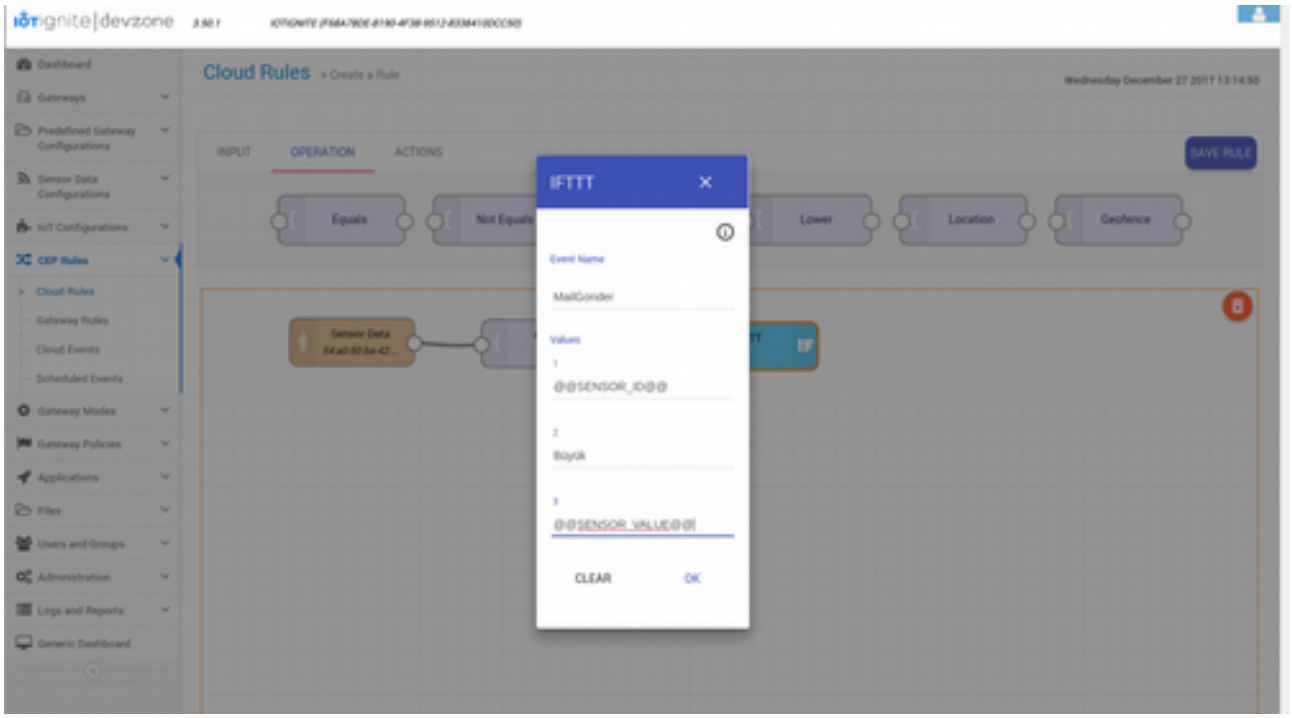
Create action butonuna tıklanarak işlem tamamlanır. **Subject** (Konu) alanında mail'in içeriğini düzenleyebilirsiniz.



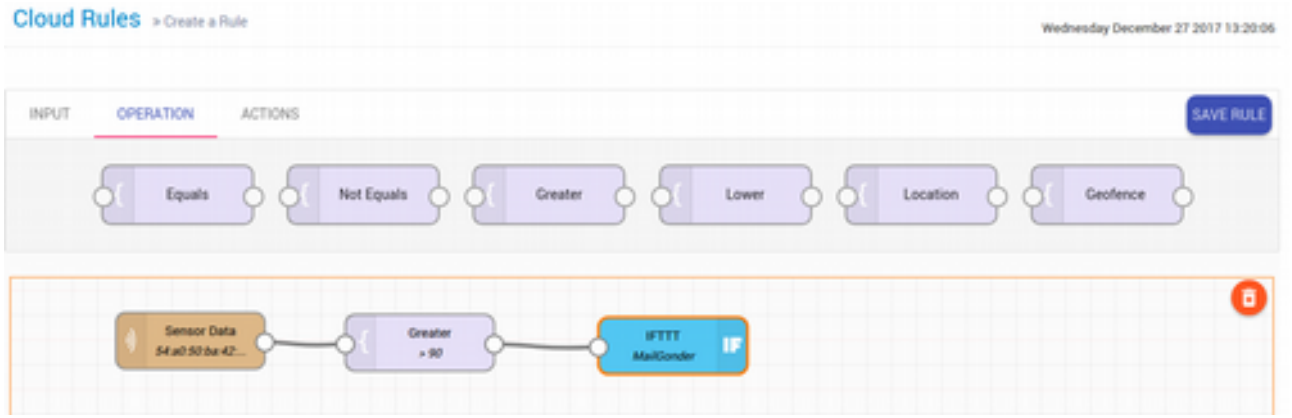
Finish (Son) butonuna tıklanarak işlem tamamlanır. Tekrar **EHUB**'tan **CEP Rule > Cloud Rule** ile **Rule Editor**'ü açın.

Aşağıdaki gibi bir örnek yapalım. Sensor Data olarak Gateway'imizdeki **VirtualDemoNode** altında yer alan **Temperature** sensöründen gelen değer **90'dan büyük** olursa **IFTTT** ile e-mail gönderilmesini sağlayalım.

IFTTT bileşeninde **Event Name** (Olay İsmi)'ne IFTTT'de aksiyon tanımlarken kullandığımız **MailGonder** ismini girdik. Mail içeriğinde de **Value1**'e sensörün adını girmek için **@@SENSOR_ID@@** ismini, **Value2**'ye operasyonun adını (Greater, Büyüktür) ve **Value3**'e de sensörden gelen değeri **@@SENSOR_VALUE@@** ile yazıp **OK** butonu ile kaydediyoruz. Tanımladığımız kural Cloud Rule olduğu için mod Gateway'e push edilmez. **OccuredAt** parametresine bir değer verilmez, kendisi zaten olay gerçekleştiğinde otomatik olarak zaman bilgisi ile doldurulur.



Son olarak **Rule Editor**'ün sağ üst köşesinde yer alan **SAVE RULE** (Kural Kaydet) ile kuralı kaydedin.

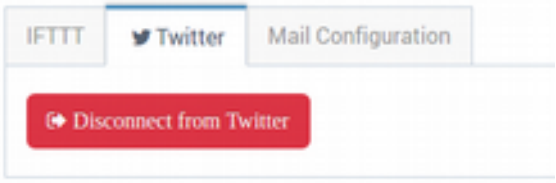


Kural gerçekleştiğinde Devzone hesabınızdaki e-mail hesabınıza mesaj gelecektir.

- **Twitter:** Sadece Gateway Rule için EHUB'ta kullanılmaktadır.

Kural koşullarının sağlanması ile birlikte Twitter hesabınızın duvarına mesaj yazdırılabilir. Bunun için öncelikle **EHUB**'ta **External Services** altında **Twitter** sekmesinden **Connect with Twitter** ile giriş yapmanız gerekmektedir.

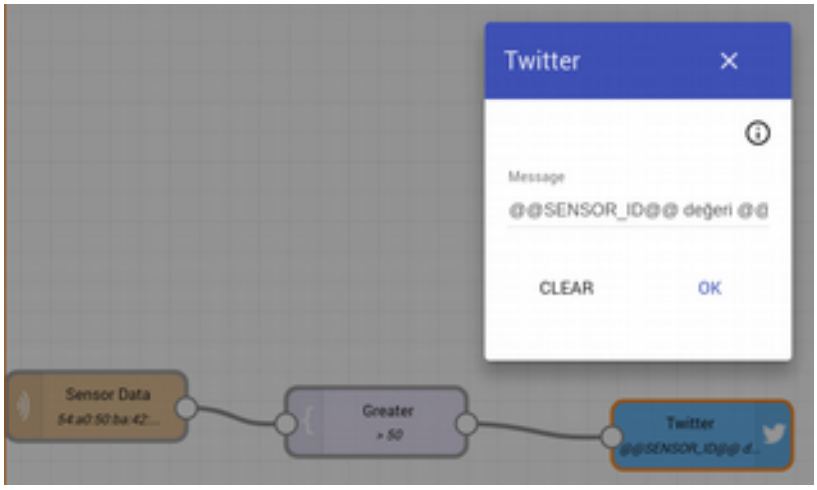
Giriş işleminden sonra **IoT-Ignite** uygulamasına izin vermeniz gerekmektedir.**Uygulamaya izin ver** butonuna tıklayın ve devam edin. Bu işlemden sonra artık Twitter bağlantınız kurulmuş durumdadır.



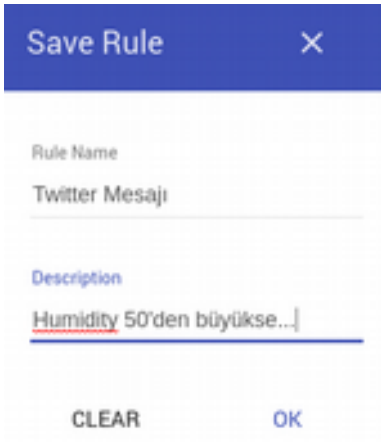
Twitter aksiyonu ile EHUB'ta Cloud Rule oluşturalım. CEP Rules > Cloud Rules sayfasına girin ve New Cloud Rule butonu ile Rule Editor'ü açın.

Kural olarak Humidity değeri 50'den büyükse Twitter mesajı atsın istiyoruz. Twitter aksiyon bileşenine aşağıdaki gibi mesajımızı ekliyoruz.

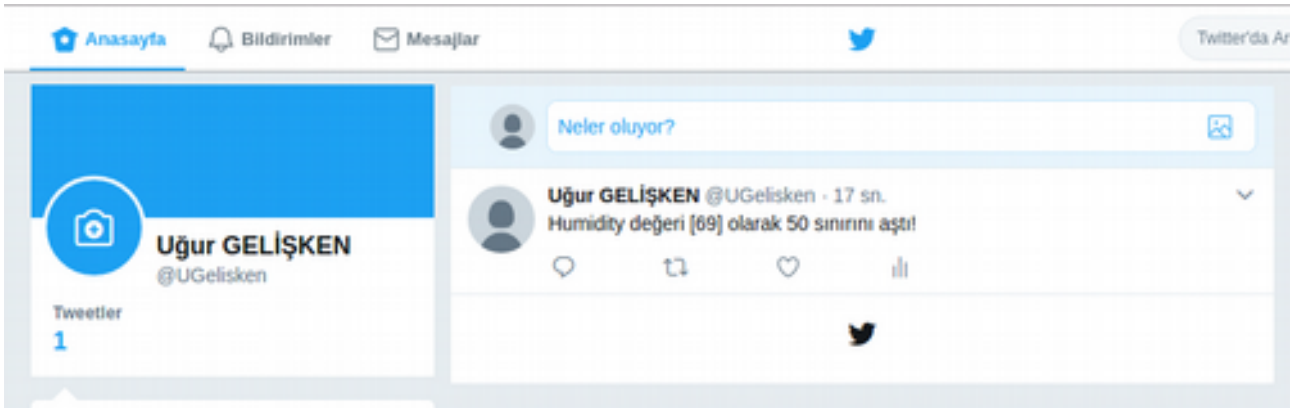
@@SENSOR_ID@@ değeri @@SENSOR_VALUE@@ olarak 50 sınırını aştı!



Daha sonra **SAVE RULE** ile kuralı kaydedin.

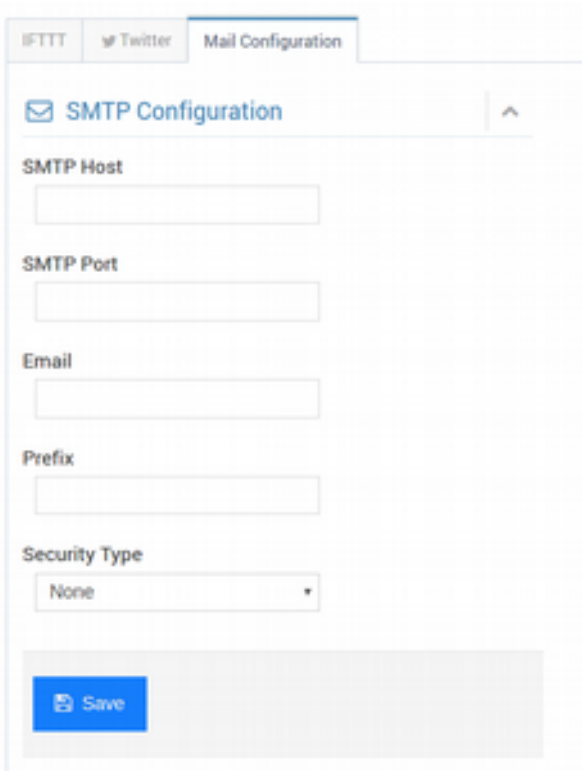


Kural gerçekleştiğinde Twitter hesabınızın duvarında aşağıdaki gibi bir ileti gelecektir.



- **Send Mail:** Sadece Gateway Rule için EHUB'ta kullanılmaktadır.

Kural oluştuğunda, hesabınızda tanımlı olan e-posta adresine mail gönderimi yapabilirsiniz. Bunun için yine EHUB'tan **IoT Configurations > External Services** altında yer alan **Mail Configurations** sekmesinde **SMTP** yapılandırması yapmanız gerekmektedir.



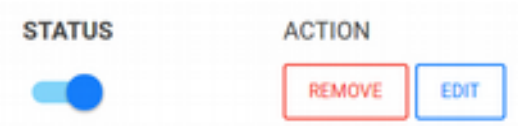
Aksiyon Mesajlarında Anahtar Kelimelerin Kullanımı

Aksiyon bileşenlerinde **message** alanlarında dinamik değerleri eklemek için aşağıdaki gibi bazı anahtar kelimeleri (keyword) kullanabilirsiniz.

Parametre	**Anahtar Kelime (Keyword)**
Sensör Değeri	@@SENSOR_VALUE@@
Gateway ID	@@GATEAWAY_ID@@
Node ID	@@NODE_ID@@
Sensör ID	@@SENSOR_ID@@
Zaman	@@TIME@@

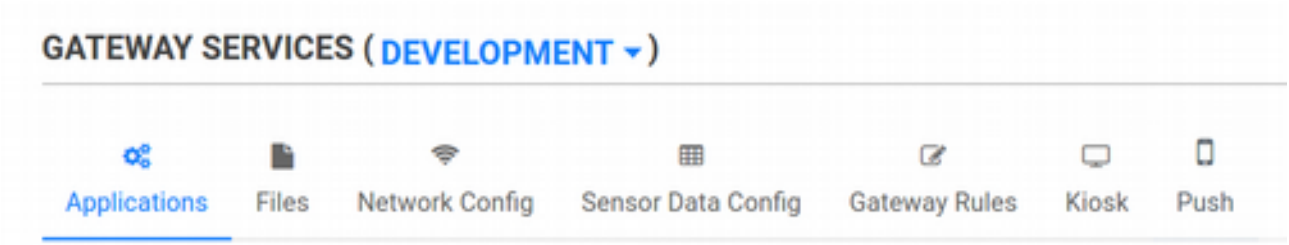
Kuralları Aktif-Pasif Yapmak, Silmek ve Düzenlemek

Kuralları aktif-pasif yapmak için **Status** (Durum) sütununda yer alan switch butonu kullanabilirsiniz. Kuralı silmek için **Remove** (Kaldır) butonuna tıklamak ve onaylamak yeterlidir. Kuralı yeniden düzenlemek için de **Edit** (Düzenle) butonuna tıklanır. Edit'e tıklanması ile birlikte Rule Editor açılacak ve o kurala ait yapılandırma ayarları tuvalde gösterilecektir.



Gateway Servisleri

Üst menüden Gateway Services'e tıklayarak ulaşabileceğimiz Gateway Servisleri sayfası, Mod içinde yer alacak olan öğeleri yapılandırdığımız araçlara sahiptir. Bu sayfa Applications (Uygulamalar), Files (Dosyalar), Network Config (Ağ Ayarları), Sensor Data Config (Sensör Veri Ayarları), Gateway Rules (Gateway Kuralları), Kiosk (Tam Ekran Uygulama) ve Push (Gönder) sekmelerinden oluşmaktadır. Her bir sekme birbirinden bağımsızdır, ancak yapılan yeni yapılandırmanın Gateway'e veya Gateway'lere mutlaka Push sekmesinde yer alan araçlarla gönderilmesi gerektiğini unutmayın.



Yapılandırılacak Olan Servis Modunu Seçmek

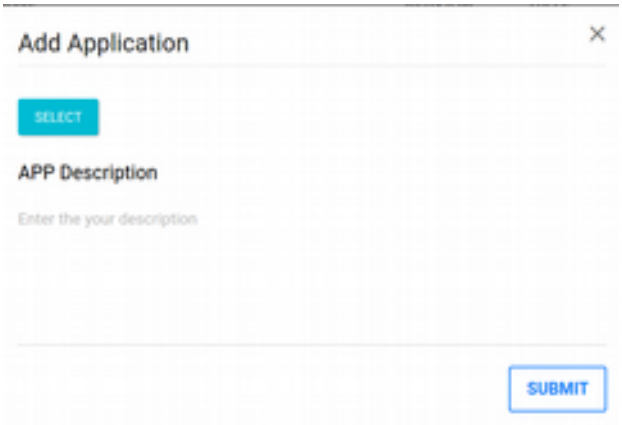
Daha önce de bahsetmiştik, tekrar değinmekte fayda var. Gateway Services sayfasında çalışırken mutlaka mavi renkli alanda yer alan ok butonuna tıklayarak üzerinde çalışacağımız Mod'u seçmemiz gerekmektedir. Varsayılan olarak **DEVELOPMENT** modu aktiftir. Hangi mod ile çalışırsanız, o modun ayarlarını yeniden yapılandırır ve onu Gateway'lere push edersiniz. Dilerseniz push işlemi yapmadan da sırasıyla diğer **DEMO** ve **ANDROIDTHINGS** modlarını da yapılandırmanız mümkündür.

Uygulama (Application)

Mod içine APK dosyaları yükleyebilir, silebilir ve bütün Gateway'lerde o uygulamayı tek tıklama ile kurulumunu başlatabilirsiniz.

- **App Store'a Uygulama Yüklemek**

Mod'a bir APK yüklemek için **ADD APPLICATION** (Uygulama Ekle) butonuna tıklanır. Açılan pencerede **SELECT** (Seç) butonuna tıklanarak yerel sistemdeki .apk dosyası seçilir. Daha sonra **APP Description** (Uygulama Açıklaması) alanına uygulama hakkında kısa bir açıklayıcı bilgi yazılarak **SUBMIT** (Kaydet) butonuna tıklanarak yükleme işlemi başlatılır. Yükleme süresince "Uploading... Please wait." mesajı görüntülenecektir, bu aşamada bekleyiniz. Yükleme tamamlanınca pencere kapanacaktır. Uygulama ismi ve versiyon bilgisi .apk'dan otomatik olarak gelmektedir.



- **Yüklü Uygulamayı Serviste Aktif-Pasif Yapmak, APP Store'dan Silmek**

Yükleme tamamlandıktan sonra **Applications** sekmesinde uygulamamız görüntülenecektir. Dikkat edin, uygulama aslında direkt olarak Mod'a yüklenmez! Yüklenen uygulama, APK Store (APK) altına olarak alınmaktadır. Eğer uygulamayı switch butonu ile aktif ederseniz, o zaman Mod içine eklenmiş olacaktır. Örneğimizde Ludo Star isimli bir oyun APK'sı yükledik ve Mod içine ekledik. Push işlemi ile istediğimiz Gateway'e bu uygulamayı herhangi bir onay gerektirmeden indirebilir ve Gateway kullanıcı onayı ile de yüklenmesini sağlayabilirsiniz.

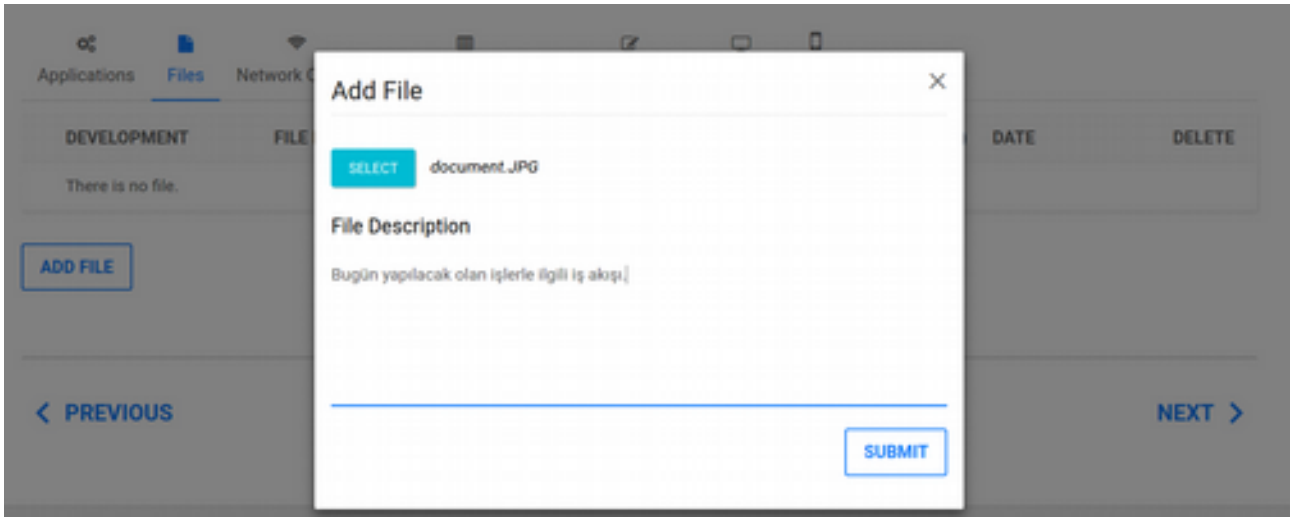
App Store'dan uygulamayı kaldırmak için de **DELETE** sütunu altında yer alan **çöp kutusu** ikonuna tıklamak yeterlidir.

Dosyalar (Files)

Aynı APK dosyalarının yüklenmesi gibi servisimize, (yani mod'a) herhangi bir dosya da yükleyebilmemiz mümkündür. Sonrasında da bu dosyaları istediğimiz Gateway'e veya Gateway'lere push edebiliriz.

- **File Store'a Dosya Yüklemek**

Dosya yüklemek için **Files** sekmesindeyken **ADD FILE** (Dosya Ekle) butonuna tıklanır. Açılan pencerede **SELECT** butonu ile dosya seçilir ve dosya açıklaması yazılır, **SUBMIT** butonu ile de yükleme işlemi başlatılır.



Yükleme tamamlandığında dosyamız **File Store**'a (Dosya Deposu) eklenecektir. Dosyayı mod içine almak için switch butonu aktif etmek yeterlidir. Yalnız burada ek bir yapılandırma daha yapmanız gerekmektedir.

Switch butonu aktif ettiğinizde karşınıza aşağıdaki gibi bir arayüz çıkacak. Bu arayüz, push edilecek olan dosyanın hangi dizine ve hangi klasör altına yükleneceğini belirlemenizi sağlar.

Storage (Depo) opsiyonunda **Internal** (Dahili) ve **SD Card** seçimleri yer almaktadır. Internal ile dahili hafızaya, SD Card ile de harici hafızaya kopyalama yapılır. **Folder** (Klasör) opsiyonuna da klasör ismi verilebilir. Son olarak **SET PATH AND ADD MODE** butonuna tıklanarak dosya için path tanımlanır ve mod'a eklenir.

Push sekmesinden aktif olan modu push ettiğinizde, Gateway'e veya Gateway'lere bu dosya da (birden fazla dosya da olabilir) gönderilmiş olacaktır.

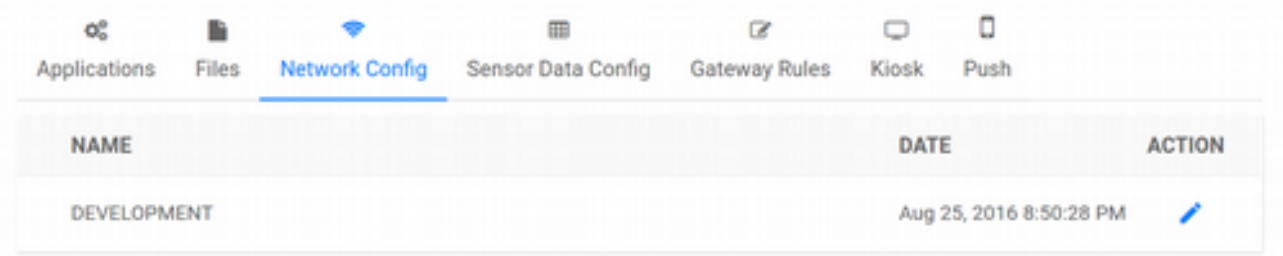
- **Yüklü Dosyayı Serviste Aktif-Pasif Yapmak, File Store'dan Silmek**

Dosyayı mod'dan kaldırmak için switch butonu pasif yapılır. File Store'dan da kaldırmak isterseniz **DELETE** sütunu altında yer alan **çöp kutusu** ikonuna tıklamak yeterlidir.

Ağ (Network) Yapılandırması

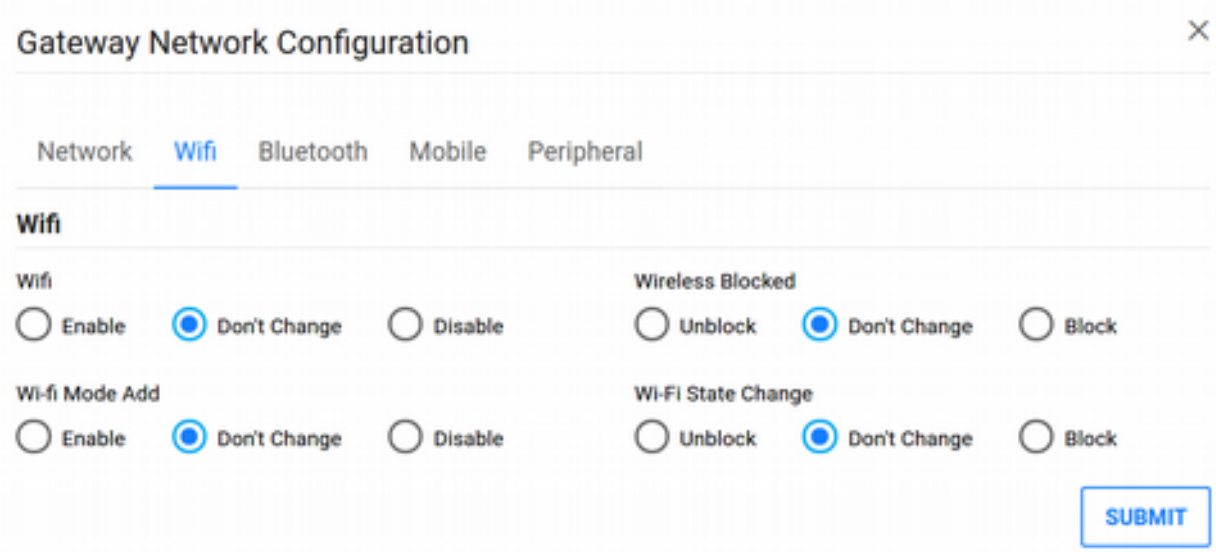
Ağ yapılandırmalarında **Network**, **Wifi**, **Bluetooth**, **Mobile** ve **Peripheral** kategorileri altında yer alan opsiyonlarla yapılandırmalar yapabilmemiz mümkündür.

Network Config sekmesine girdiğinizde, o mod için (seçili olan mod) bir yapılandırma dosyasını listede mod ismi ile göreceksiniz.



Bu yapılandırma dosyası, Devzone üyeliği ile birlikte oluşturulmuş ve varsayılan olarak bütün opsiyonları **Dont Change** olarak sabitlenmiştir. **ACTION** sütununda yer alan **düzenle** ikonuna

tıklayarak **Gateway Network Configuration** (Ağ Geçidi Ağ Yapılandırması) penceresini açabilirsiniz.



Enable aktif yapar, **Disable** ise pasif yapar. **Don’t Change** ise herhangi bir zorlama yapmaz, Gateway’de tanımlanan ayarın kullanılması için izin verir. Yapılandırma ayarları yapıldıktan sonra **SUBMIT** butonu ile kaydedilir. Ardından yine Gateway veya Gateway’lere bu yeni ağ yapılandırma ayarlarının push edilmesi gerekmektedir.

- **Network**

Network grubu altında aşağıdaki opsiyonlar yer almaktadır: **Wifi Tethering**, **USB Tethering**, **Hotspot Config Change** ve **VPN**.

- **Wifi**

Wifi grubu altında aşağıdaki opsiyonlar yer almaktadır: **Wifi**, **Wireless Blocked**, **Wifi Mode Add** ve **Wifi State Change**.

- **Bluetooth**

Bluetooth grubu altında aşağıdaki opsiyonlar yer almaktadır: **Bluetooth** ve **Bluetooth Blocked**.

- **Mobile**

Mobile grubu altında aşağıdaki opsiyonlar yer almaktadır: **Download Over Mobile Blocking**, **Mobile Data Enable**, **Mobile Data Blocking** ve **Preferred APN Blocking**.

- **Peripheral**

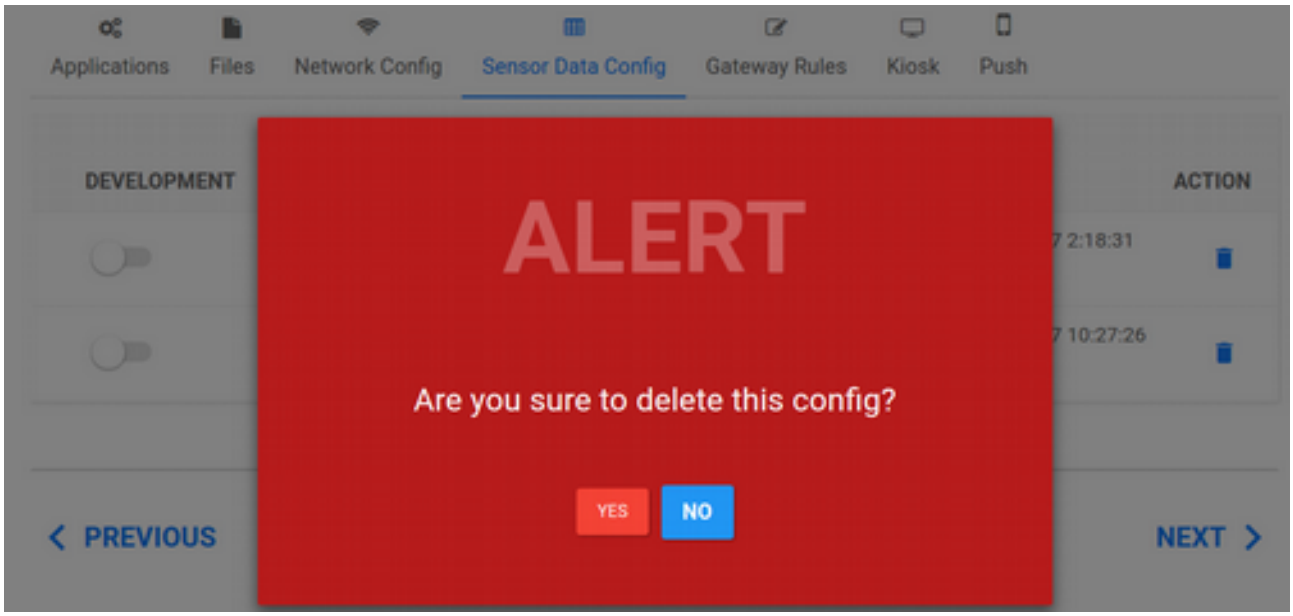
Peripheral grubu altında aşağıdaki opsiyonlar yer almaktadır: **Default ADB Settings**, **ADB Blocked**, **USB Storage Blocking**, **USB Card Blocking**, **Camera Picture**, **Camera Video**, **Camera Blocking** ve **Screenshot Blocking**.

> Ağ yapılandırmalarını yaparken dikkatli olun! Yapacağınız herhangi bir tercih, sizi zor duruma düşürebilir. Örneğin mobil veriyi bloklarsanız, Gateway offline konumuna düşecektir ve bir sonraki yapılandırma ayarlarınızı alamayacak ve sensör verilerini gönderemeyecektir. Veya bluetooth’u kapatırsanız, bluetooth ile bağlantı kurduğu node’ların Gateway ile bağlantısı kopacaktır.

Sensör Veri Yapılandırması (Sensor Data Config)

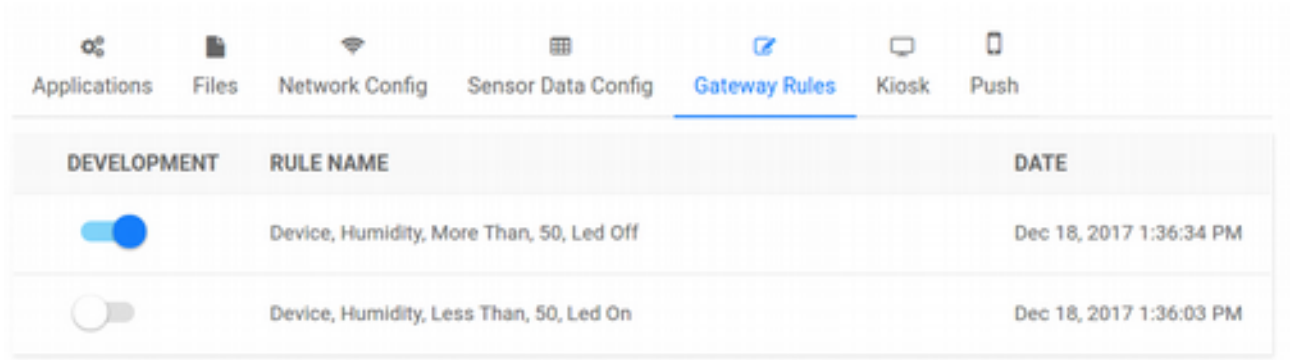
Daha önce de görmüş olduğumuz sensör yapılandırma ayarlarımız **Sensor Data Config** sekmesinde yer almaktadır. Varsayılan olarak pasif konumda Store'da beklemektedir. İstenilen yapılandırma ayarı switch butonu ile aktif edilerek mod içine aktarılır. Ardından **Push** sekmesinden Gateway veya Gateway'lere güncel mod gönderilir.

Modu silmek için de **ACTION** sütunu altında yer alan **çöp kutusu** ikonuna tıklamak ve açılan pencereden onaylamak yeterlidir.



Gateway Kuralları (Rules)

Rules sayfasında hazırladığımız Gateway kuralları bu sekme altında listelenmektedir. Listede yer alan kuralları aktif ve pasif yaparak mod içine aktarabilir, sonrasında da yine Push sekmesinden Gateway veya Gateway'lere güncel mod'u gönderebilirsiniz.

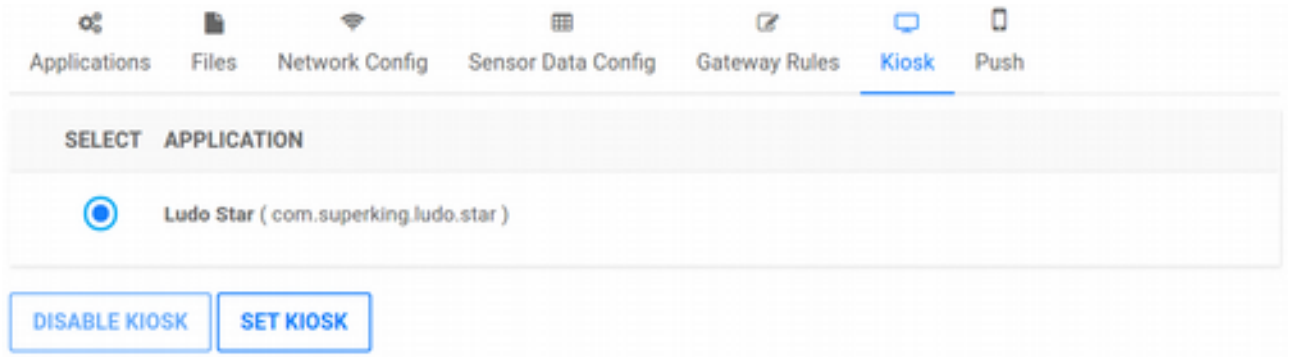


Kiosk

Applications sekmesinden yüklenmiş ve mod içinde Gateway'e push edilmiş olan bir uygulamanın Kiosk modunda açılması sağlanabilir. Kiosk olarak açılan uygulama tam ekran olarak açılır. Iot-Ignite yönetimi üzerinden bir müdahale (izin) yapılmadıkça uygulama kapatılamaz.

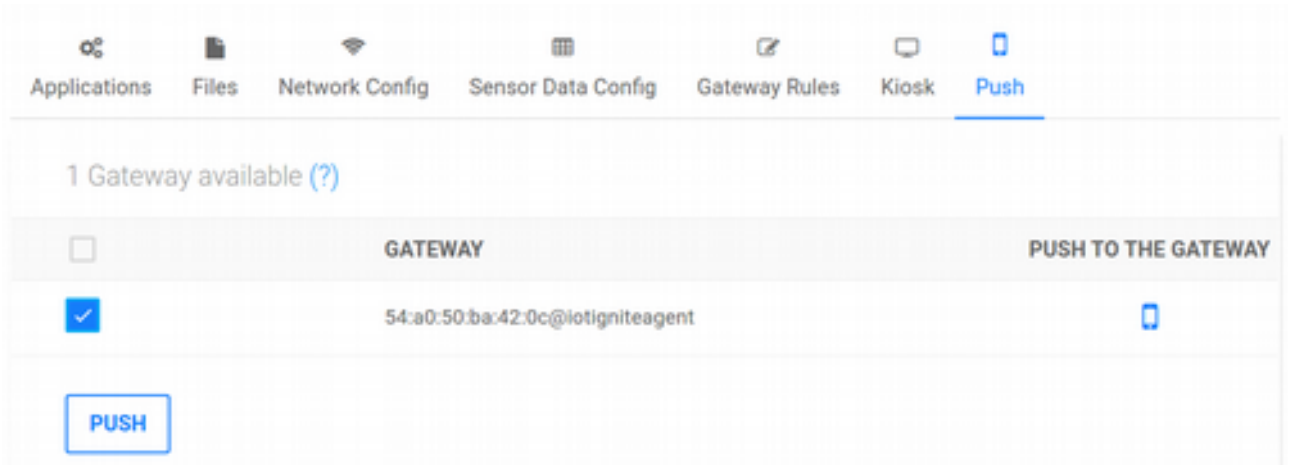
Kiosk olarak çalıştırılacak uygulama listeden seçilir ve **SET KIOSK** (Kiosk Yap) butonuna tıklanır. Ardından yine Push sekmesine geçilerek istenilen Gateway veya Gateway'lerde uygulamanın açılması ve Kiosk olarak gösterilmesi sağlanır. Gateway'leri Kiosk uygulamadan çıkarmak için ise

yine **Kiosk** sekmesindeyken **DISABLE KIOSK** (Kiosk'u Devre Dışı Bırak) butonuna tıklanıp, yine Push işlemi yapılmalıdır.



Yapılandırılmış Olan Servis Modunu Gateway'lere Yüklemek (Push)

Güncellenen mod'un Gateway veya Gateway'lere push edilebilmesi için son sekme olan **Push** sekmesi kullanılır. Bu sekmede, **lisanslı** olan Gateway'ler, **mod türlerine göre** listelenmektedir. Eğer güncel mod tek bir Gateway'e push edilecekse, listede **PUSH TO THE GATEWAY** sütunu altında, ilgili Gateway'e ait push ikonuna tıklanır. Eğer birden fazla Gateway'e push edilecekse, yine aynı işlemi tek tek yapabileceğiniz gibi; sol alt köşede yer alan PUSH butonuna da tıklayarak sırasıyla her bir Gateway'e güncel mod gönderilir.

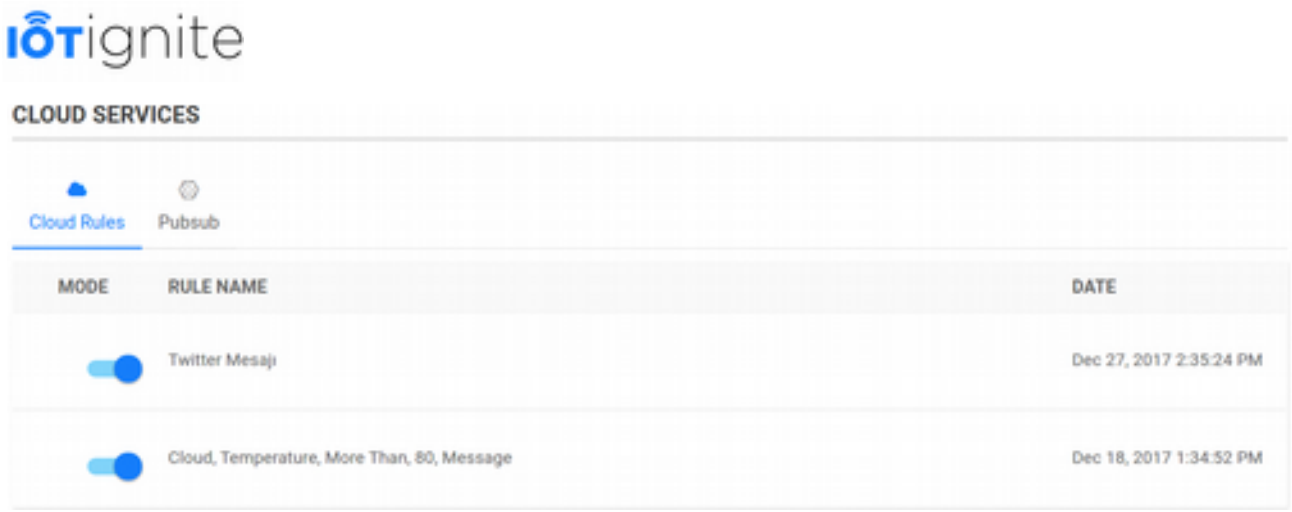


Cloud Servisleri

Üst menüden **Cloud Services** butonuna tıklanarak açılan sayfadır. Cloud kurallarını aktif-pasif yapabileceğiniz ve Pubsub hakkında bilgi alabileceğiniz kısımdır. Cloud servislerinde herhangi bir push işlemi yapılmaz, çünkü direkt olarak IoT-Ignite Cloud üzerinde çalışmaktadır.

Cloud Kuralları (Rules)

Rules sayfasında Rule Editor ile oluşturduğunuz kuralların listelendiği bölümdür. Bu alanda kuralları aktif veya pasif yapabilirsiniz. Aynı işlemi Rules sayfasından da yapabildiğimizi hatırlayalım.

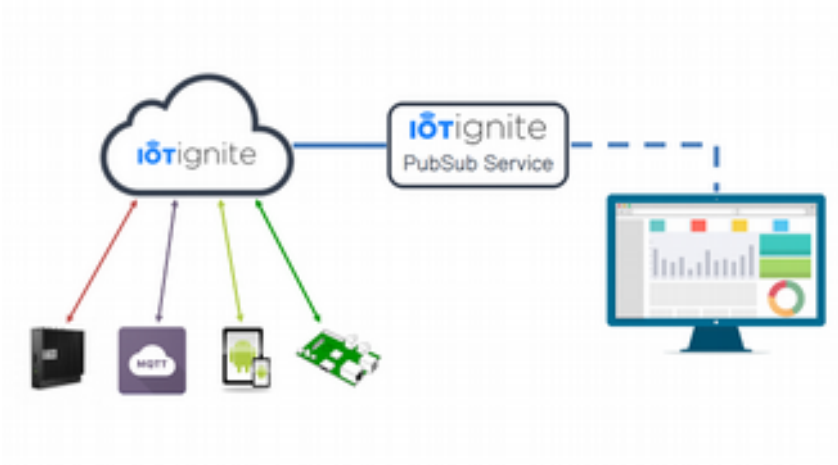


Pubsub

IoT-Ignite, gerçek zamanlı (realtime) işlemler için Gateway mesajlarını dinlemektedir. İstenildiğinde bu mesajlar Pubsub hizmet ile harici dış hizmetlere de güvenli bir şekilde aktarabilmektedir.

IoT-Ignite Pubsub hizmeti, veri hacimleri açısından yüksek ölçeklenebilirlik ve gerçek zamanlı olarak gelen verileri işlemek için gereken birden çok uygulama tarafından eşzamanlı erişim için tasarlanmış olan **Apache Kafka** tarafından desteklenen bir mesajlaşma sistemidir.

Bu hizmet, özel çözümler gerektirdiği için IoT-Ignite ile doğrudan iletişime (https://www.iot-ignite.com/contact/) geçilmelidir. IoT-Ignite yetkilileri gerekli bilgileri size sağlayacaktır.



IoT-Ignite Pubsub Servis Örnek Uygulaması

Aşağıda yer alan Git reposundan örnek projeyi klonlayıp inceleyebilirsiniz.

```
git clone https://github.com/IoT-Ignite/iotignite-kafka-example.git
```

Bu örnek, verileri görselleştirmek için IoT-Ignite Pubsub Servisi'ne nasıl subscribe (abone) olunacağını ve gelen verilerin WebSocket ile nasıl alınacağını göstermektedir.

Bu uygulamama **SpringBoot** tarafından desteklenir ve IoT-Ignite Pubsub Servisi'ne abone olmak için **Kafka** istemcisini kullanır. Aynı zamanda web arayüzüne gelen iletileri akışını sağlamak için de **WebSocket** kullanılır.

Uygulamayı kullanabilmek için IoT-Ignite ile iletişime geçip keytab dosyanızı isteyin. Hesabınız IoT-Ignite Pubsub hizmetini kullanmak için aktif edildikten sonra; **Kerberos Principal keytab** ve **Kerberos yapılandırma dosyası** size gönderilecektir. Bu iki dosyayı, uygulamanın başlangıç

parametreleri olarak kullanın. Ayrıca **/src/main/resources/application.properties** içindeki **tenantDomain** parametresini, Tenant Domain (**IOTIGNITE**, Devzone Dashboard'ta görülmektedir) adınızla değiştirmeniz gerekmektedir. Gerekli değişiklikleri yaptıktan sonra Maven ile uygulama oluşturabilir ve aşağıdaki parametrelerle başlayabilirsiniz.

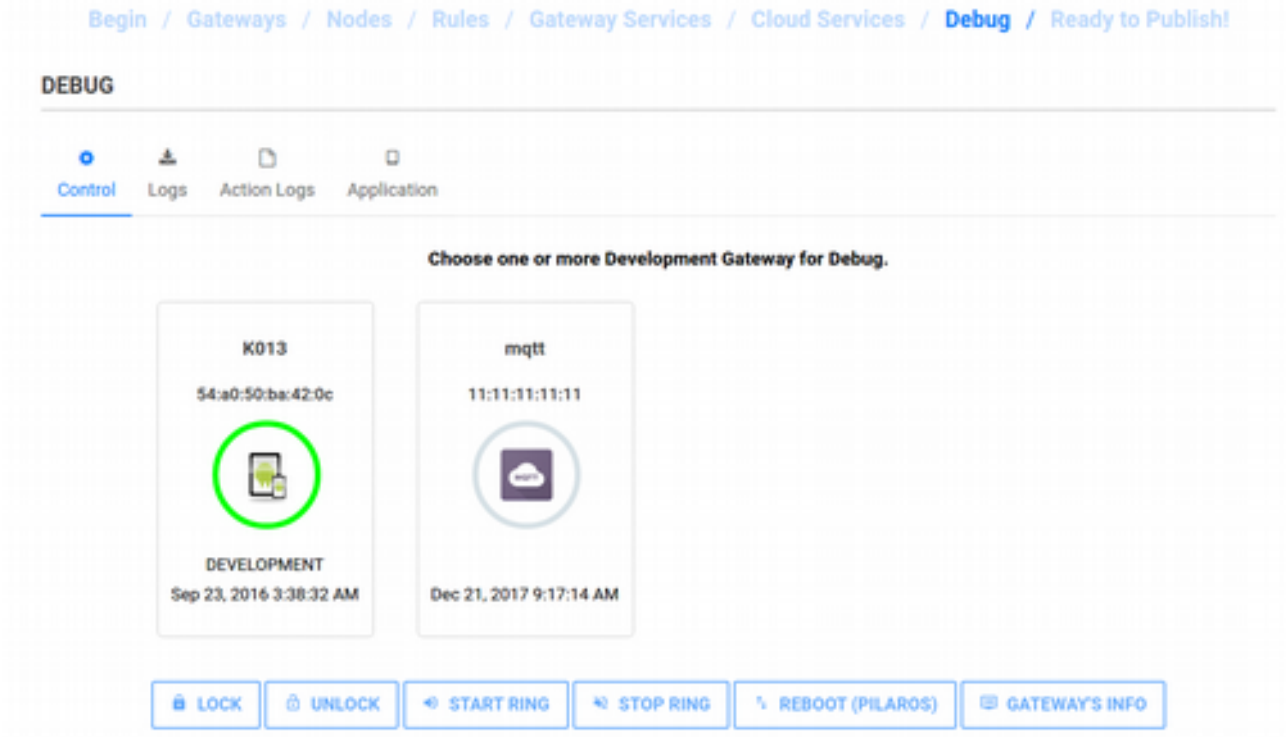
```
$ mvn clean install
$ cd target
$ java -Djava.security.auth.login.config=/path/to/client_jaas.conf
-Djava.security.krb5.conf=/path/to/krb5.conf
-jar iotignite-kafka-example-0.1.0.jar
```

Sunucu oluşturulduktan ve başlatıldıktan sonra web tarayıcınızda http://localhost:8080 adresini açın. Web sayfasında **IoT-Ignite Demo APP**'in **Sıcaklık** ve **Nem** değerlerini kullanan Gateway'lerden herhangi biri görüntülenecektir. Ayrıca tüm Gateway mesajları, web sayfasındaki konsol isimli alanına akış yapacaktır.

Debug ve Log Yönetimi

Debug ve Log yönetimi ile Gateway'ler üzerinde birtakım testler yapabilir, Gateway üzerindeki dönemsel log dosyalarını görebilir ve Gateway'lerde yüklü olan uygulamaları görüntüleyebiliriz.

Üst menüden **Debug** butonuna tıklayarak erişeceğimiz bu sayfa, aşağıdaki gibidir.



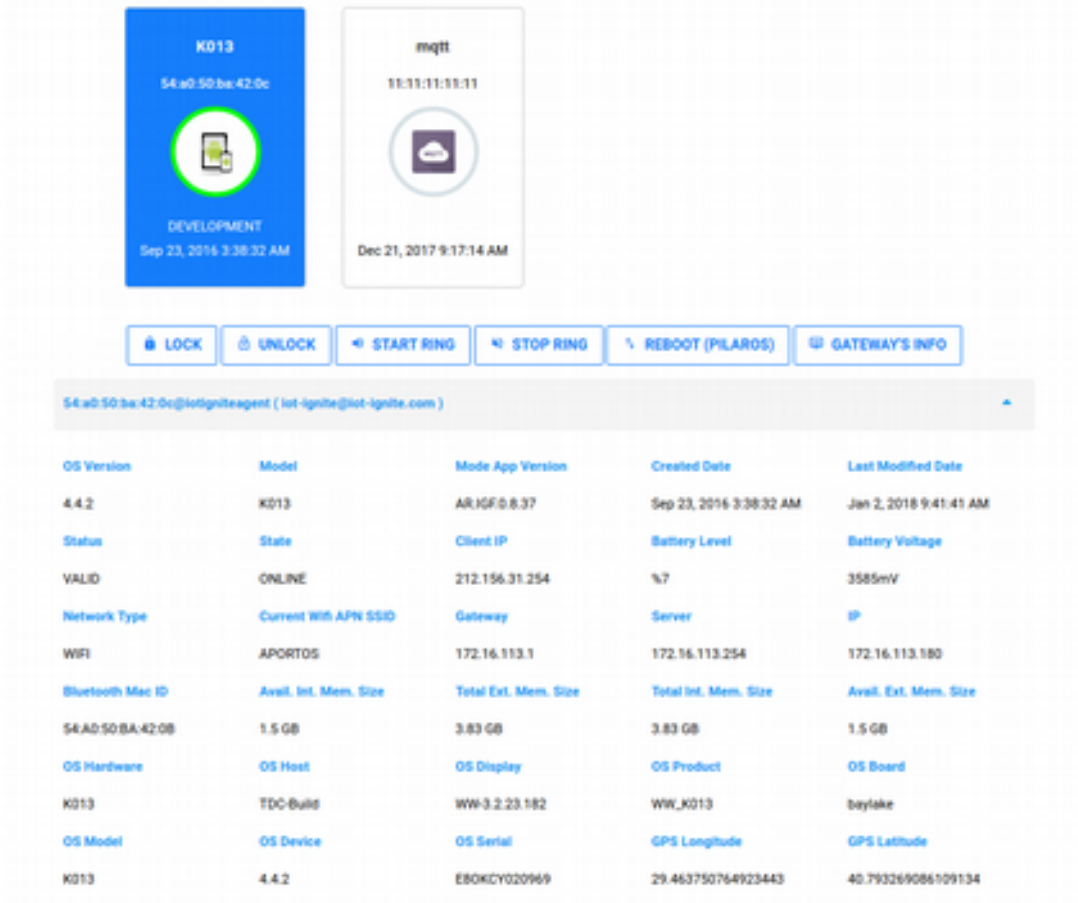
Debug sayfasına girdiğimizde yukarıdaki şekilde görüldüğü üzere **Control**, **Logs**, **Action Logs** ve **Application** sekmelerinden oluşan bir tablı sistem yer almaktadır. Aktif olarak da doğrudan Control sekmesi açık gelmektedir. Şimdi bu sekmeleri ve bu sekmelerde yer alan araçları inceleyelim.

Gateway Kontrolü (Control)

Bu sekme altında, lisanslanmış olan Gateway'ler kutucuklar halinde görüntülenmektedir. Kutu içerisinde, Gateway'in modeli, id'si, Gateway'in türüne göre bir görsel, eğer Gateway online ise görseli çevreleyen yeşil renkli bir çember, Gateway'in modu ve lisanslanma tarihi yer almaktadır.

Listedeki kutuların her birini tıklayıp aktif veya pasif yaparak çoklu seçebilirsiniz. Seçtiğiniz Gateway'leri de alt kısımda yer alan LOCK, UNLOCK, START RING, STOP RING, REBOOT ve GATEWAY'S INFO butonları ile kontrol edebilirsiniz.

- **LOCK**: Gateway'in ekranını kilitler ve ekranda bir kilit resmi gösterir.
- **UNLOCK**: Kilitlenmiş ekranı tekrar serbest bırakır.
- **START RING**: Gateway'de zil sesi çalmaya başlar.
- **STOP RING**: Çalan zil sesini kapatır.
- **REBOOT**: Eğer Gateway'in işletim sistemi PilarOS ise, Gateway'i yeniden başlatır.
- **GATEWAY'S INFO**: Gateway hakkında güncel bilgileri getirir ve tablo olarak ekranda gösterir. Gateway'in bilgileri aşağıdaki gibi görülmektedir.

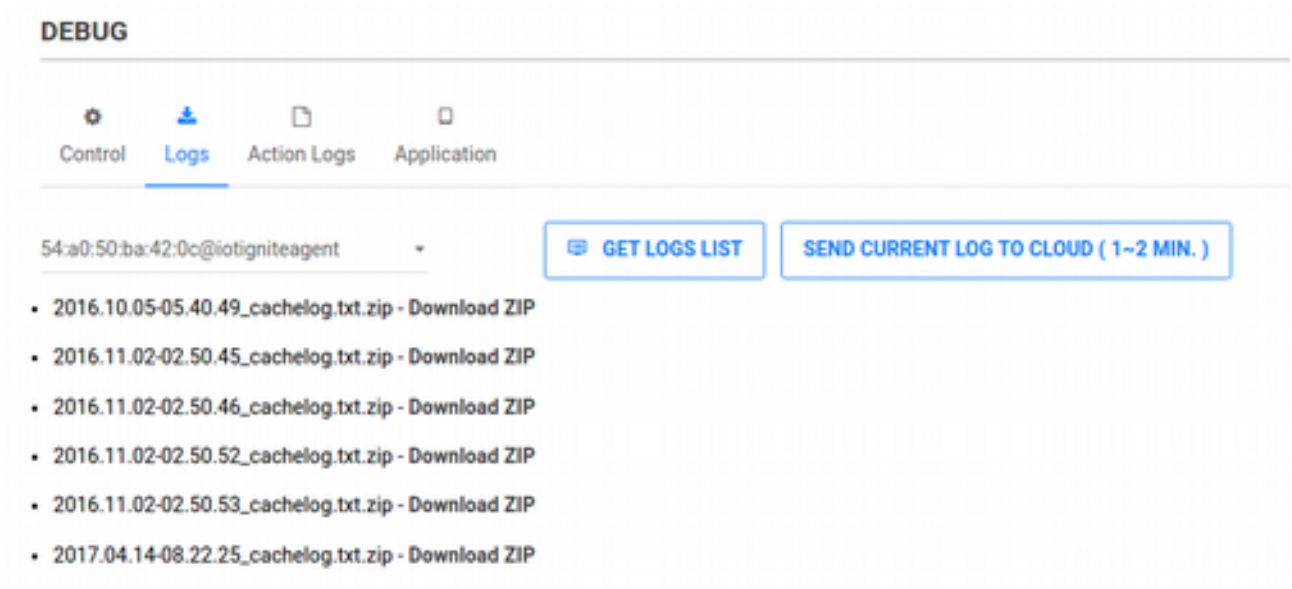


Loglar (Logs)

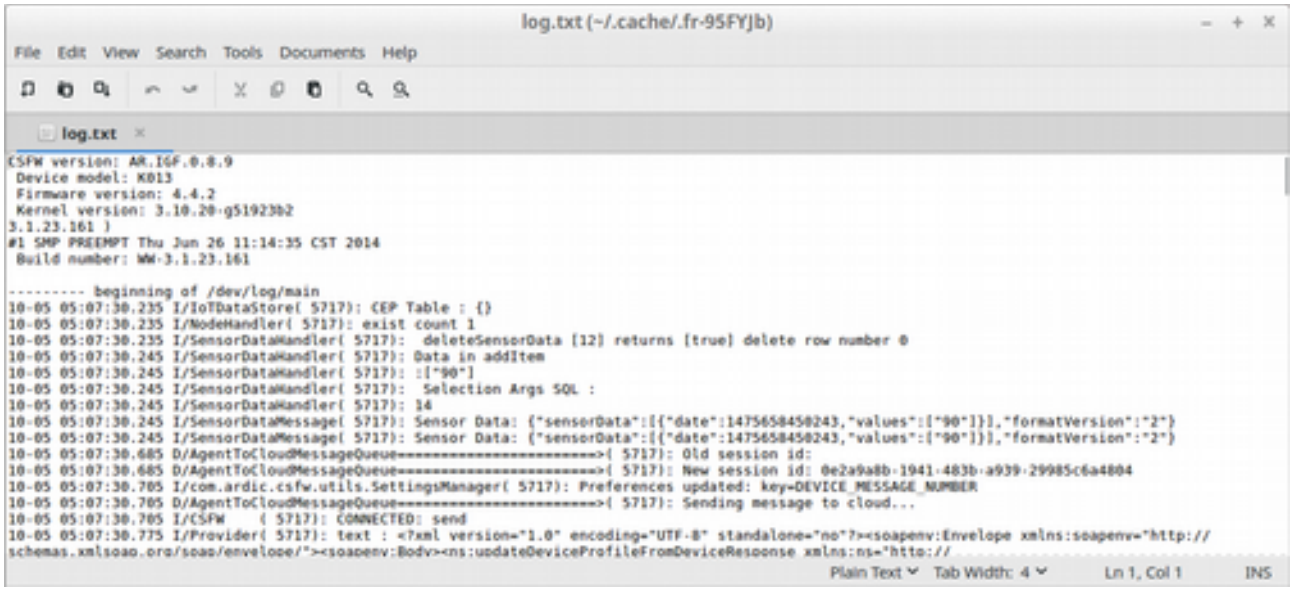
Gateway üzerinde kayıtlı olan dönemsel log dosyalarını **ZIP** formatında indirip inceleyebileceğiniz, anlık olarak da Gateway'e log kaydı yapması için istekte bulunabileceğiniz bir araçtır.

- **Seçilen Gateway'in Log Kayıtlarını Getirmek**

Listeden ilgili Gateway seçilip **GET LOGS LIST** (Log'ların Listesini Getir) butonuna bastığınızda, aşağıdaki gibi dönemsel tarihlerle oluşturulmuş ve **.zip** dosyası olarak kaydedilmiş kayıtları göreceksiniz. **Download ZIP** butonlarına tıkladıkça, ilgili log dosyasını indirip inceleyebilirsiniz.

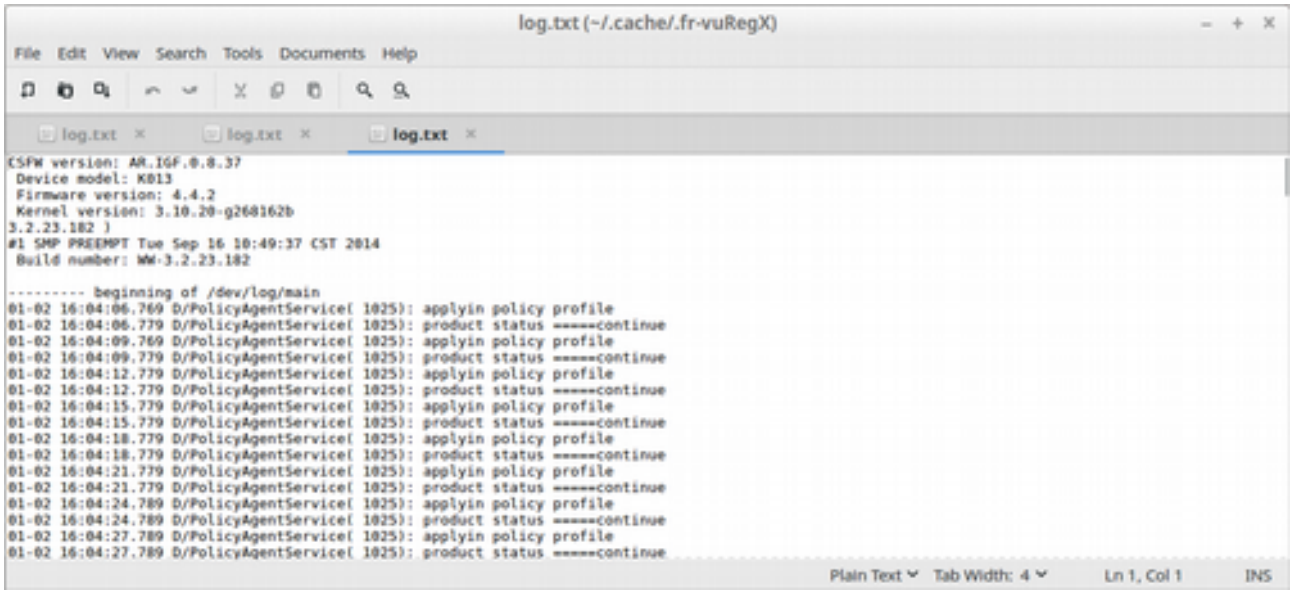


İndirip açtığınız ZIP dosyası içinde **log.txt** dosyası yer almaktadır. İndirilen buu dosyada aşağıdaki gibi detaylı kayıtlar sunacaktır.



- **Seçilen Gateway'den Yeni Log İsteğinde Bulunmak**

Eğer o anki güncel log kaydı almak istiyorsanız, **SEND CURRENT LOG TO CLOUD** (Güncel Log'u Cloud'a Gönder) butonuna tıklayın. Yaklaşık **1-2 dakika** içerisinde Gateway bir log oluşturacaktır. Bu log dosyasını görmek için tekrar **GET LOGS LIST** butonuna tıklayın. Güncel log kaydı aşağıdaki gibidir.

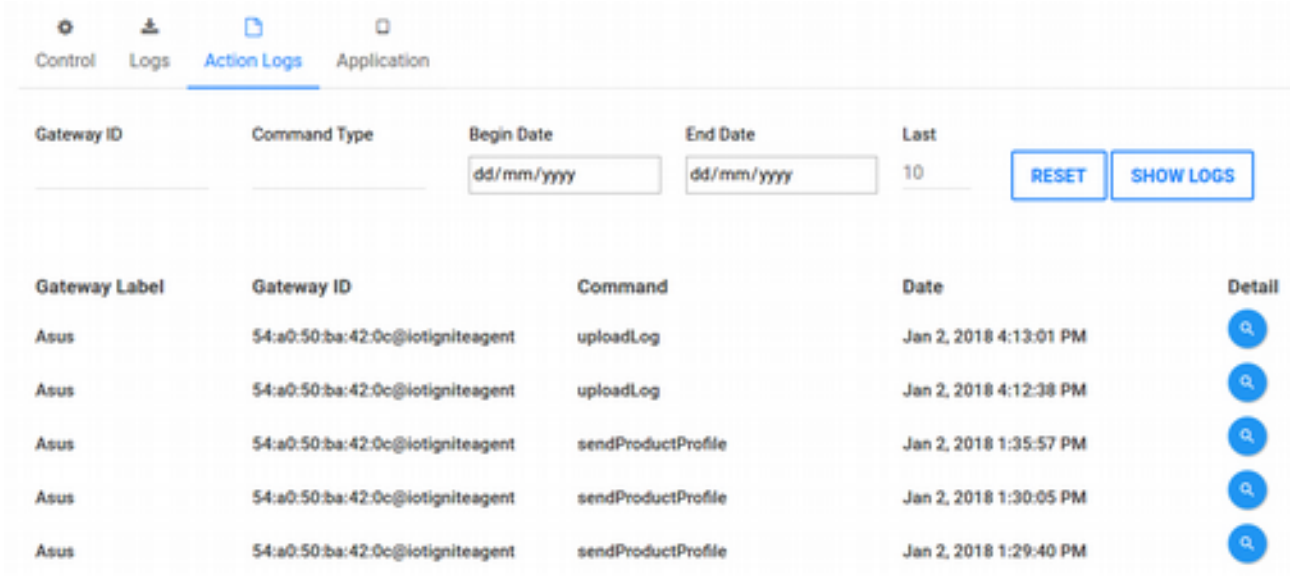


Aksiyon Logları (Action Logs)

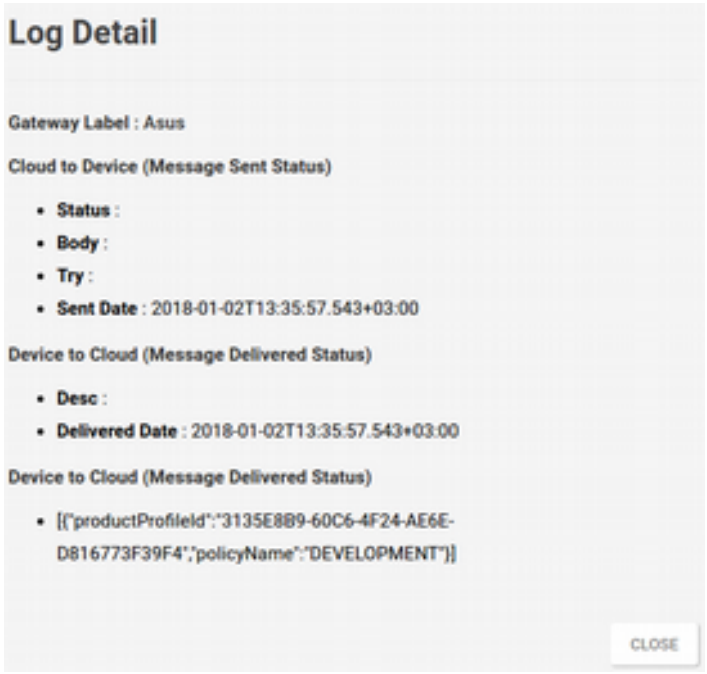
Gateway/Gateway'lerin gerçekleştirmiş olduğu her bir aksiyonlarla ilgili kayıtları listeleyebileceğiniz arayüzdür.

- **Aksiyon Kayıtlarını Listelemek ve İncelemek**

Hesabınızda lisanslanmış olan Gateway'lerin aksiyon kayıtlarını listelemek için **Action Logs** sekmesin altında yer alan **SHOW LOGS** (Kayıtları Göster) butonuna tıklandığında, aşağıdaki şekilde görüldüğü üzere aksiyon kayıtları listelenecektir.



Listeyi incelediğimizde hangi Gateway için hangi **Command** (Komut)'un gerçekleştirildiği ve zamanı görülmektedir. Komutlar, uploadLog, sendProductProfile gibi olabilmektedir. Her bir aksiyonun detay butonuna tıklandığında, aşağıdaki gibi bir detay ekranı açılacaktır.



Detayları incelediğimizde; hangi Gateway'e ait bir aksiyon mesaj kaydının olduğu, Cloud'tan Gateway'e hangi mesajın hangi tarihte gönderildiği, Gateway'den de Cloud'a hangi cevabın döndüğü ile ilgili detaylar yer almaktadır.

- **Aksiyon Kayıtları Filtreleri**

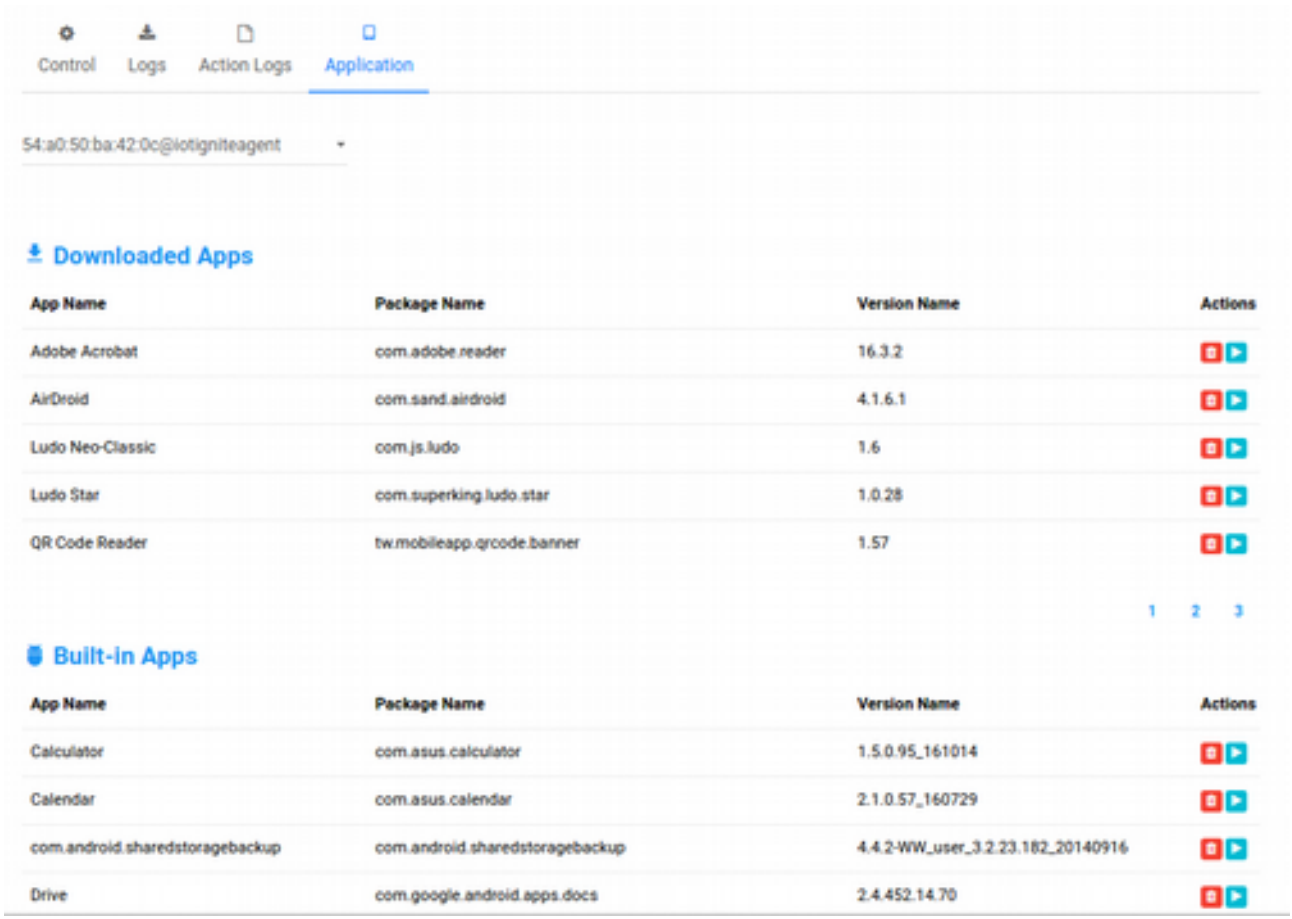
Varsayılan filtre ayarları ile son 10 aksiyon kaydı listelenmektedir. Ancak isterseniz **Begin Date** (Başlangıç Tarihi) ve **End Date** (Bitiş Tarihi) aralıklarını tanımlayarak, belli bir aralığındaki belli bir sayıda kaydı listeleyebilirsiniz. Ayrıca **Gateway ID** ve **Command Type** (Komut Tipi) de belirtilerek belli bir arama kriteri belirleyip istenilen Gateway'in istenilen aksiyon kaydına daha hızlı ulaşılabilir.

Uygulamalar (Application)

Debug sayfasında inceleyeceğimiz son sekme Application'da Gateway'lerde yüklü olan **Built-in** (Yerleşik) ve **Downloaded** (İndirilmiş) olan uygulamaları **listeleyebilir**, bu uygulamaları **silebilir** veya uzaktan **çalıştırabiliriz**.

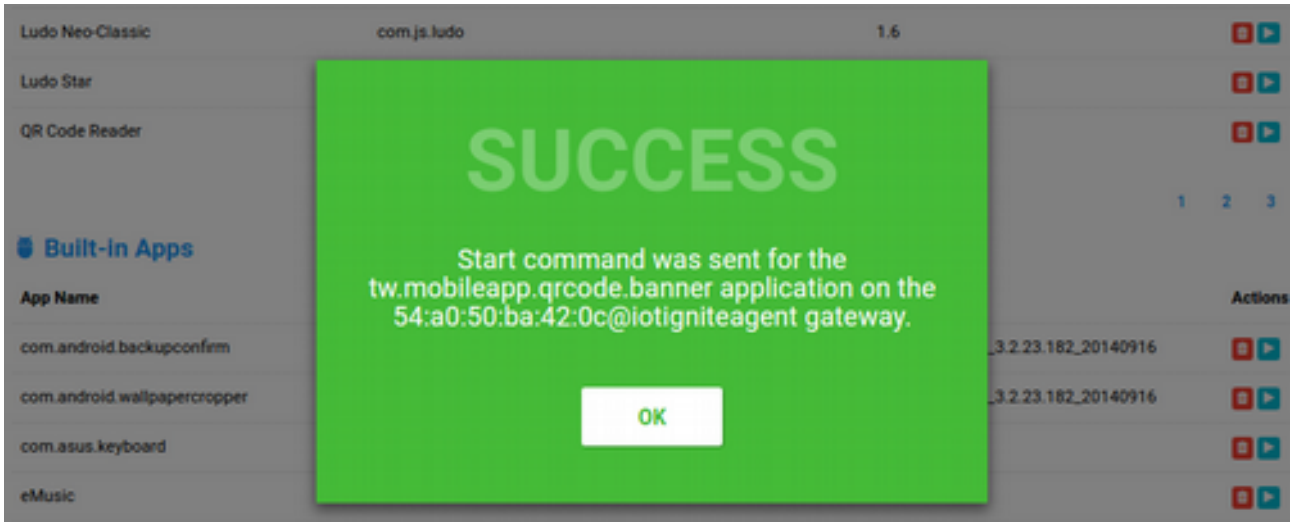
- **Seçilen Gateway'deki Built-in ve Sonradan Yüklü Uygulamaları Listelemek**

Listeden herhangi bir Gateway seçildiğinde, otomatik olarak o Gateway'deki uygulamalar gruplandırılarak listelenecektir. Listede uygulamanın ismi, paket adı ve versiyon bilgileri yer almaktadır. Ek olarak o uygulamayı Gateway'den kaldırabilmeniz veya uzaktan çalıştırabilmeniz için gerekli araçlar da bulunmaktadır.



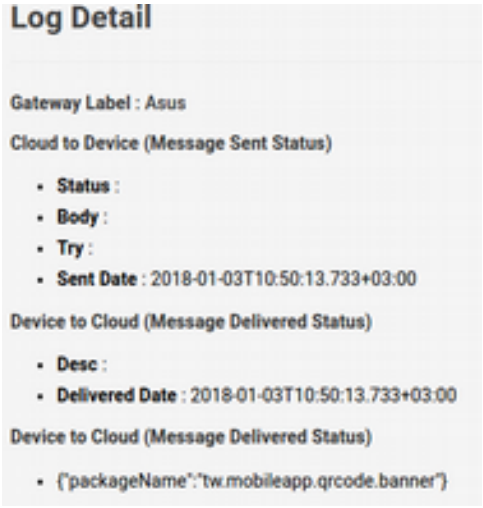
- **Uzaktan Uygulama Çalıştırmak ve Silmek**

Gateway'deki bir uygulamayı uzaktan çalıştırabilmek için Actions (Aksiyonlar) sütunu altında yer alan mavi ikona tıklamak yeterlidir. Bu işlem ile Gateway'e uygulamayı açması için gerekli olan komut gönderilecektir.



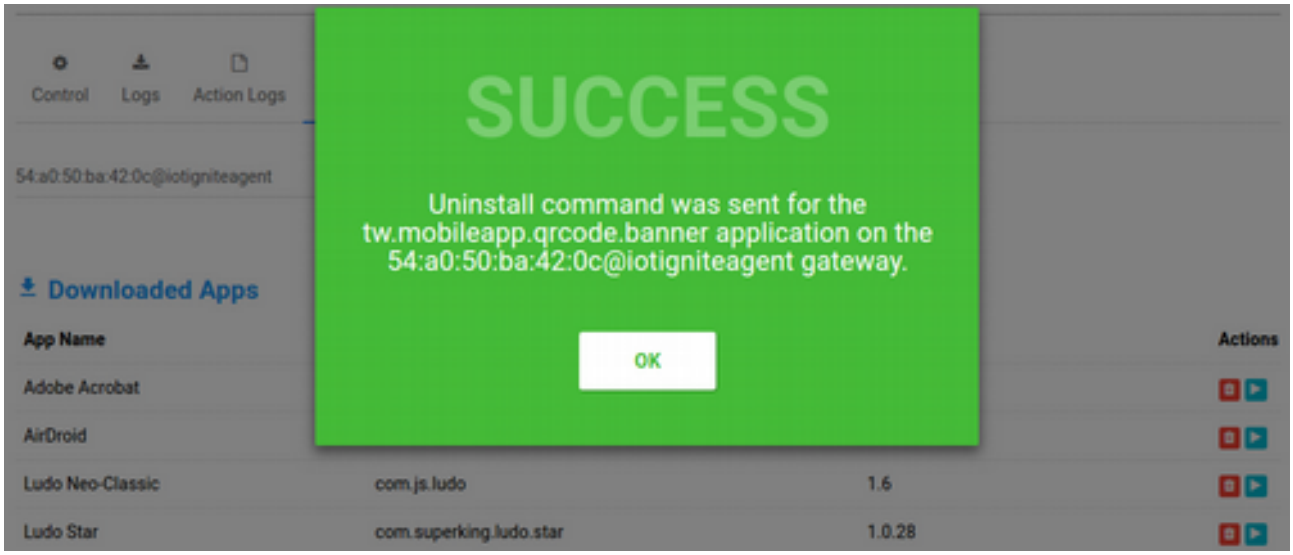
İsterseniz tekrar Action Logs sekmesine geçip, bu komutun Gateway'e ulaşıp ulaşmadığını, ulaştıysa da Gateway'in ne cevap verdiğini görebilirsiniz.

Uygulama başlatma komutu **startApp** komut ismi ile gönderilmiş. Detayına baktığımızda da aşağıdaki gibi görülmektedir.



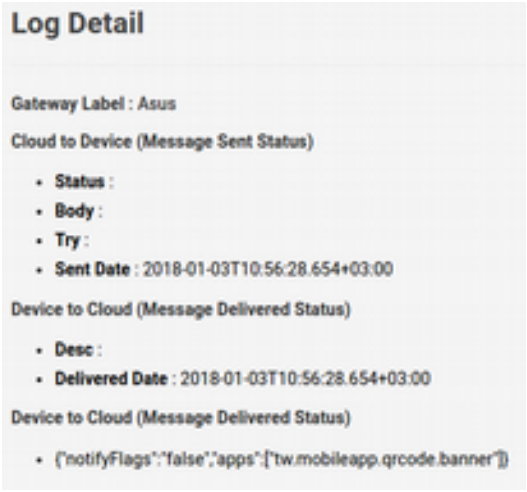
Detay raporuna göre komut Gateway'e gönderilmiş ve Gateway de Cloud'a cevap dönmüş. Gateway'e baktığımızda da **QR Code Reader** programının başlatıldığını görebiliyoruz.

Uygulamayı silmek için ise **çöp** ikonuna tıklayıp sonuca bakalım.



Bu defa da paket ismi ile birlikte, **Uninstall** komutunun Gateway'e gönderildiği mesajı karşımıza çıkıyor. **Tekrar Action Logs**'a bakalım...

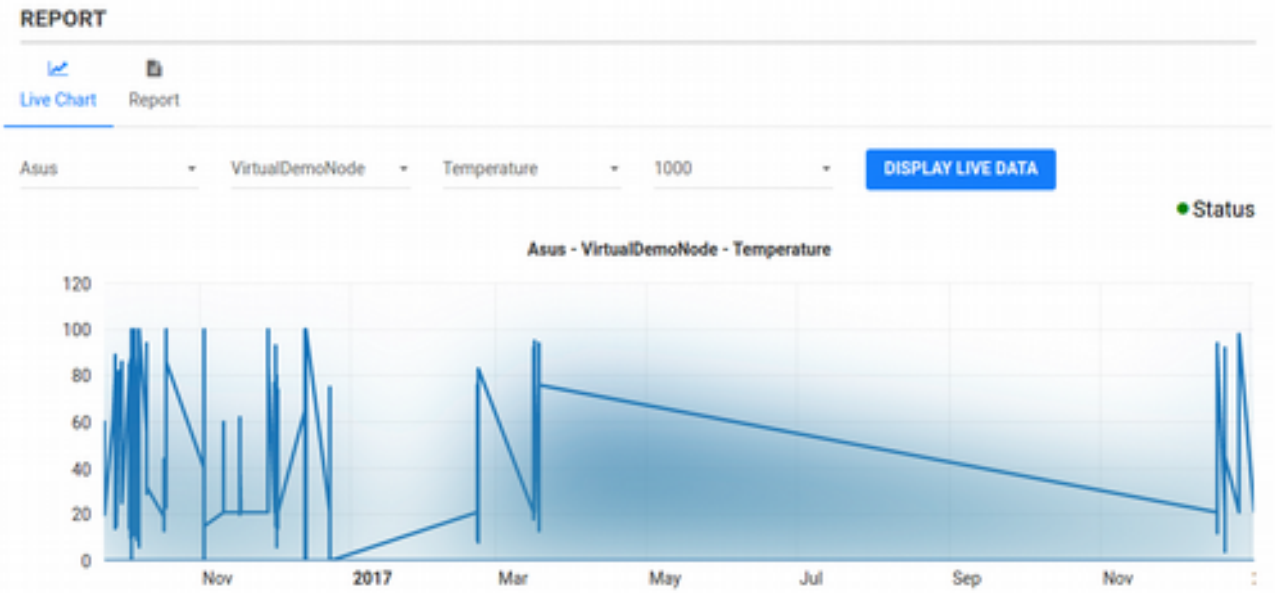
Uygulama silme komutu **deleteApplication** komutu ile Gateway'e gönderilmiş.



Raporlama (Report)

Üst menüden **REPORT** ile gireceğimiz bu sayfada, Gateway'lerde yer alan sensör verilerini anlık veya geçmişe dönük olarak izleyebiliriz..

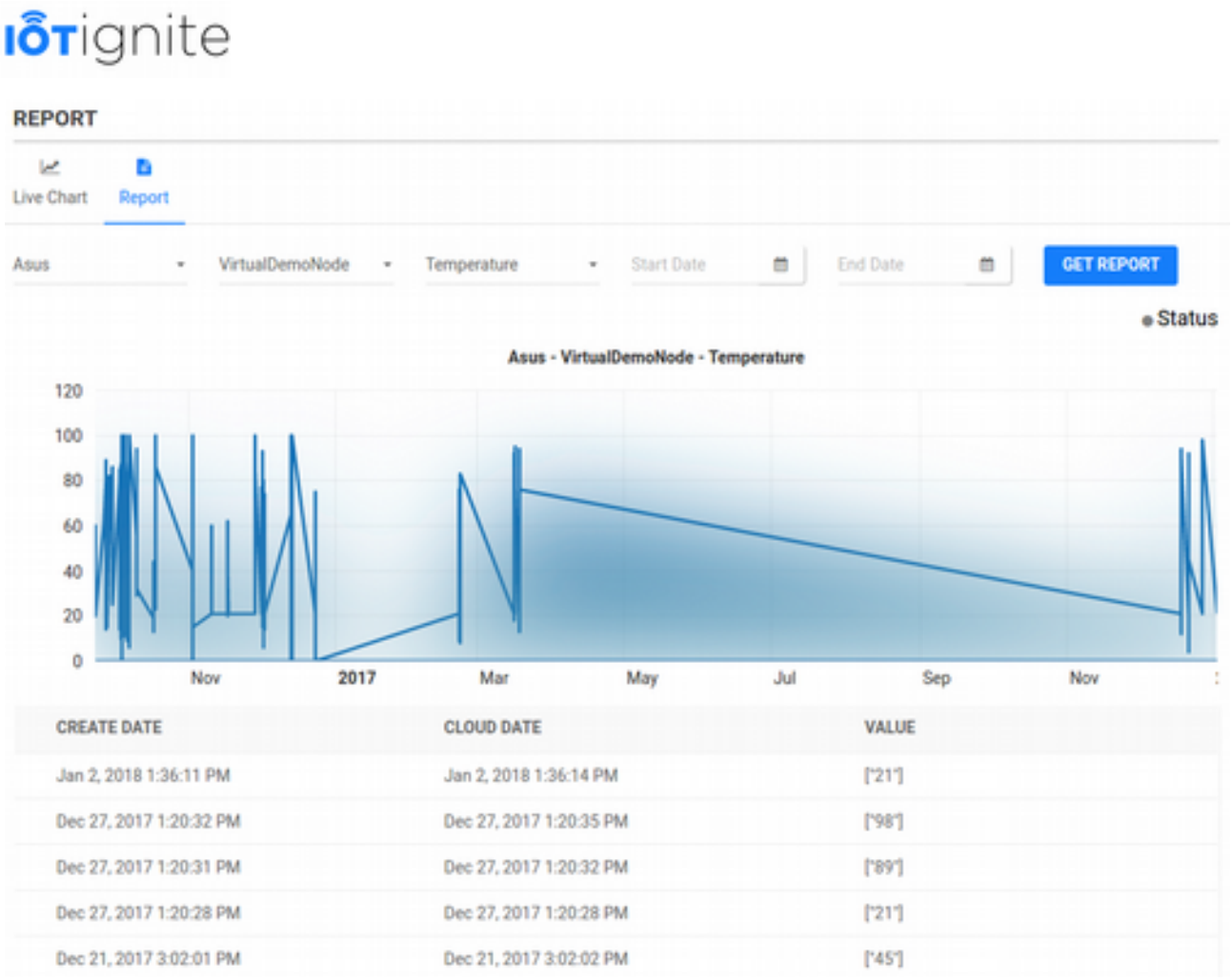
- **Live Chart ie Anlık Veri Akışını Gözlemlemek**



REPORT sayfasının ilk sekmesi olan **Live Chart** (Canlı Grafik)'te geçmişe dönük veriler ile birlikte WebSocket üzerinden gelen yeni veriler de grafiğe eklenerek görüntülenir.

- **Report ile Veri Akışı Kayıtlarını İncelemek**

Report (Rapor) sekmesinde ise, sadece geçmişe dönük veriler, detaylı bir rapor halinde gösterilir. Detay raporunda, Gateway'de sensörün ürettiği değerin üretilme tarihi, Cloud'a kayıt altına alındığı tarih ve sensörün değeri gösterilmektedir.



> Devzone ile ilgili bilmemiz gereken her şeyi öğrendik. Bir sonraki konuda, çoklu Gateway ve sensörler ile çalışırken aynı anda birden fazla veriyi tek bir ekran üzerinden **monitörleme** konusuna değineceğiz.

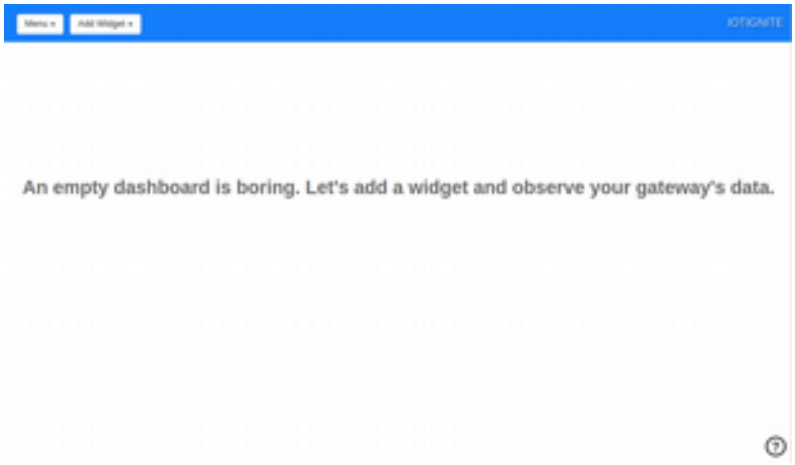
Generic Dashboard

Devzone'da üst menüde yer alan **DASHBOARD** (Gösterge Paneli) butonuna tıklandığınızda, ayrı bir sayfaya (ayrı bir IoT-Ignite ürünü) yönleneceksiniz. Açılan sayfadaki ürün, Generic Dashboard (Genel Gösterge Paneli) isimli bir IoT-Ignite ürünüdür ve kullanımı ücretsizdir.

Generic Dashboard, bir veya birden fazla Gateway'i destekleyip, çeşitli grafik widget'leri ile sensör verilerini bir arada görebileceğiniz bir monitörleme aracıdır.

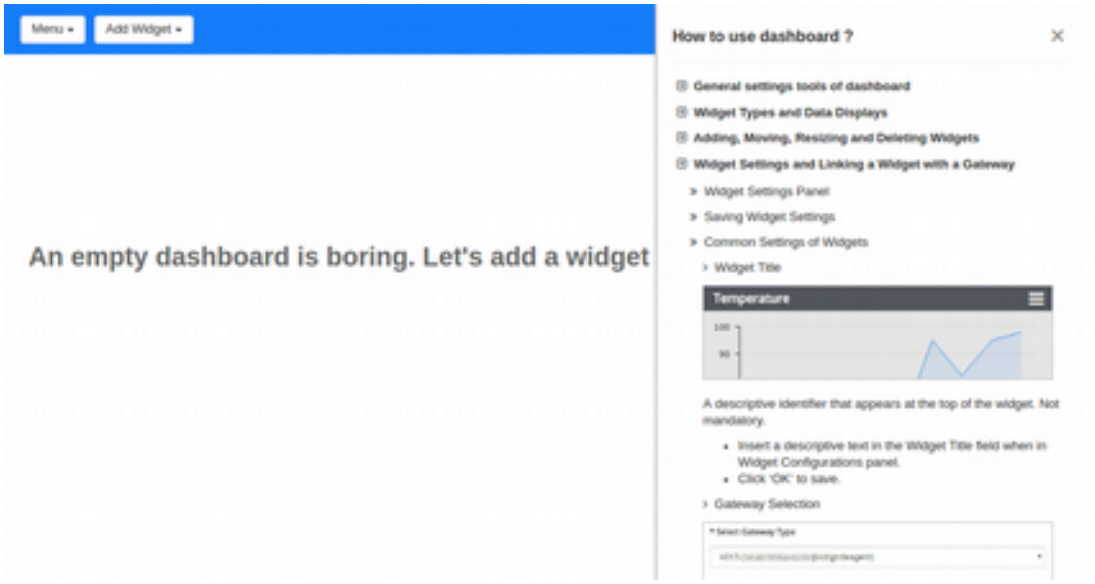
Generic Dashboard, birden fazla çalışma alanını destekler ve istenildiğinde hızlıca çalışma alanları arasında hızlıca geçiş yapabilmenizi sağlar. Her bir çalışma alanında kendi düzeninizi widget'ler ile tasarlayabilir, kaydedebilir ve daha sonra tekrar açabilirsiniz. Ayrıca tema özelliği ile görünümünü kişiselleştirebilmenize de olanak tanır.

Generic Dashboard'a ilk girdiğinizde, aşağıdaki gibi boş bir ekran ile karşılaşacaksınız.



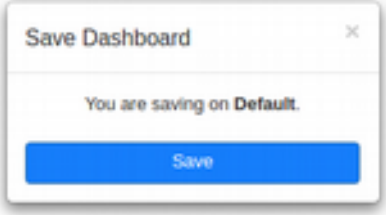
Generic Dashboard Genel Hatları

Generic Dashboard'un genel yapısını incelediğimizde, üst kısımda bir mavi şerit menü görüyoruz. Menünün sağ tarafında **Tenant İsmi** (Brand Name) yer almaktadır. Sol üst köşede ise **Menü** ve widget'leri ekleyebilmeniz için **Add Widget** (Widget Ekle) açılır menüleri yer almaktadır. Sağ alt köşede de Dashboard'un kullanımını öğrenebileceğiniz bir yardım butonu yer almaktadır. **Yardım** butonuna tıkladığınızda, aşağıdaki şekilde görüldüğü gibi sağ tarafta yardım sayfası açılarak gelecektir.

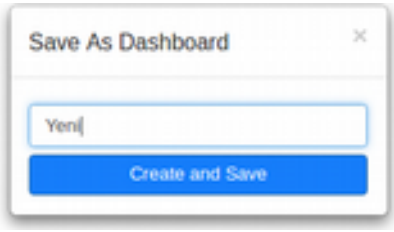


Yeni Dashboard Alanı Oluşturmak ve Silmek

Generic Dashboard'a giriş yaptığınızda, boş bir çalışma alanı sizi beklemektedir. Bu alan üzerinde widget'ları oluşturup (veya hiç değişiklik yapmadan) kaydedebilirsiniz. Menü'den **Save** (Kaydet) butonuna tıkladığınızda varsayılan olarak Default ismi ile çalışma alanınız kaydedilecektir. Kayıt işlemi için açılır menüden **Save** butonuna tıklayarak onay vermeniz gerekmektedir.

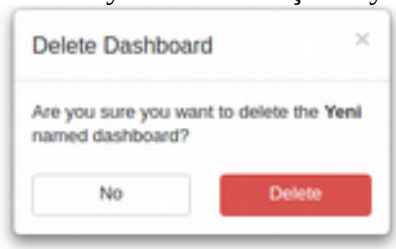


Eğer farklı bir isimde veya üzerinde çalıştığınız çalışma alanını yeni bir isimle farklı bir alan olarak kaydetmek isterseniz, Menü'den **Save As** (Farklı Kaydet) butonuna tıklayın. Yine açılan pencereden çalışma alanı için yeni bir isim verip **Create and Save** (Oluştur ve Kaydet) butonuna tıklayın.



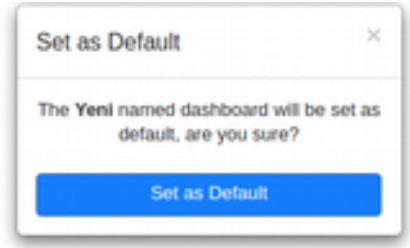
Örneğimizde **Yeni** ismini verip kaydediyoruz. Bu işlemin sonunda çalışma alanı kaydedilecek ve o alan üzerinde çalışmaya devam edilecek. Üst menünün orta alanında da o an üzerinde çalıştığınız çalışma alanının ismi görünecektir.

Kaydettiğiniz herhangi bir çalışma alanını da silmek için, o çalışma alanı açıkken Menü'den **Delete** (Sil) butonuna tıklamanız ve onaylamanız yeterlidir. Bu işlemi yaparken dikkatli olmayı unutmayın!



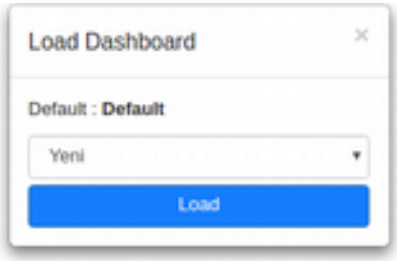
Varsayılan Dashboard Alanını Belirtmek

Generic Dashboard'ta çoklu çalışma alanları ile çalışabildiğimizi biliyoruz. Ancak Generic Dashboard'a giriş yapıldığında otomatik olarak bu çoklu çalışma alanlarından birinin yüklenmesi gerekmektedir. Varsayılan olarak **Default** ile veya ilk oluşturduğunuz çalışma alanı (veya sadece bir tane varsa o çalışma alanı yüklenir) varsayılan çalışma alanı olmaktadır. Ancak isterseniz farklı bir çalışma alanını da, o an çalışma alanı açıkken Menü'den **Set as Default** (Varsayılan Olarak Tanımla) butonuna tıklamak ve onaylamak yeterlidir.



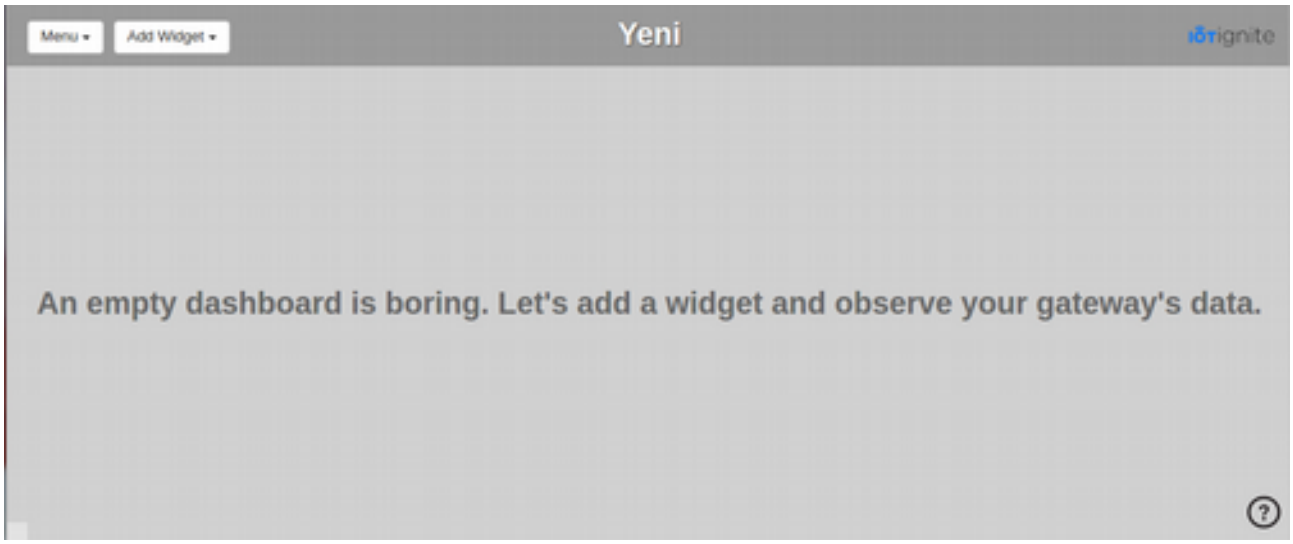
Çalışma Alanları Arası Geçiş Yapmak

Birden fazla çalışma alanınız varsa, Menü'den **Load** (Yükle) butonun tıklayıp, açılan arayüzden de yüklemek istediğiniz çalışma alanını seçip **Load** butonunu tıklamanız yeterlidir.



Tema Seçimi

Varsayılan olarak Generic Dashboard, mavi renk tonları ve açık gri zemin üzerine bir renk kombinasyonu ile karşınıza gelmektedir. Ancak isterseniz her bir çalışma alanı için farklı renk tonlarında temalar da uygulayabilirsiniz. Bunun için Menü'den **Personalize** (Kişiselleştirme) butonuna tıklayıp, gelen arayüzden **Style** (Stil) listesinden **Default** (varsayılan tema), **Dark**, **Light** ve **Soft** isimleri ile farklı farklı tonları da deneyebilirsiniz. Ayrıca üst menüde sağ üst köşede yer alan servis ismi yerine bir logo kullanmak isterseniz, logonun **URL**'ini **Logo** alanına yazıp **Save** ile kaydedebilirsiniz. Örneğin **Soft** stili ile **IoT-Ignite** logosunu da kullanıp sonuca bakalım.



> Menü'den yapılan her işlem yapıldığında, işlemin sonucunu belirtmek için ekranın sağ alt köşesinden bilgilendirme mesajları çıkmaktadır.

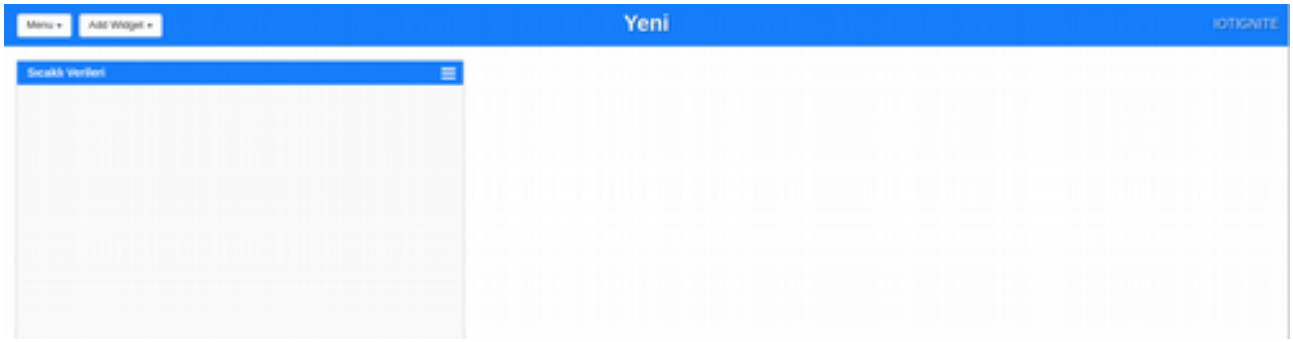
Widget Nedir

Widget, sürükle-bırak ile çalışma alanında taşınabilen, yeniden boyutlandırılabilen ve içinde çeşitli veri gösterip metotlarının bulunduğu kutucuklardır. Bu kutucuklardan sınırsız sayıda oluşturulabilir, her biri için ayrı Gateway ve sensör'den gelen veriler çeşitli metotlar ile gösterilebilir. Her bir widget, kendisinde tanımlı olan Gateway'in sensöründen gelen verileri anlık olarak WebSocket (her bir widget için ayrı WebSocket açılır) üzerinden dinler ve gerçek zamanlı olarak gösterir. Bazı widget'lar da Gateway'lerde yer alan Actuator sensörlere (veri alabilen) parametre gönderilmesini sağlar.
Menü'den Add Widget (Widget Ekle) butonuna tıkladığımızda açılır bir liste çıkacaktır. Listeden istediğimiz widget'ı tıkladığımızda, çalışma alanına sol üst köşeye bloklanacak şekilde boş alanlara widget'lar eklenecektir. Her yeni eklenen widget, yapılandırılmak için hazır olarak beklemektedir.

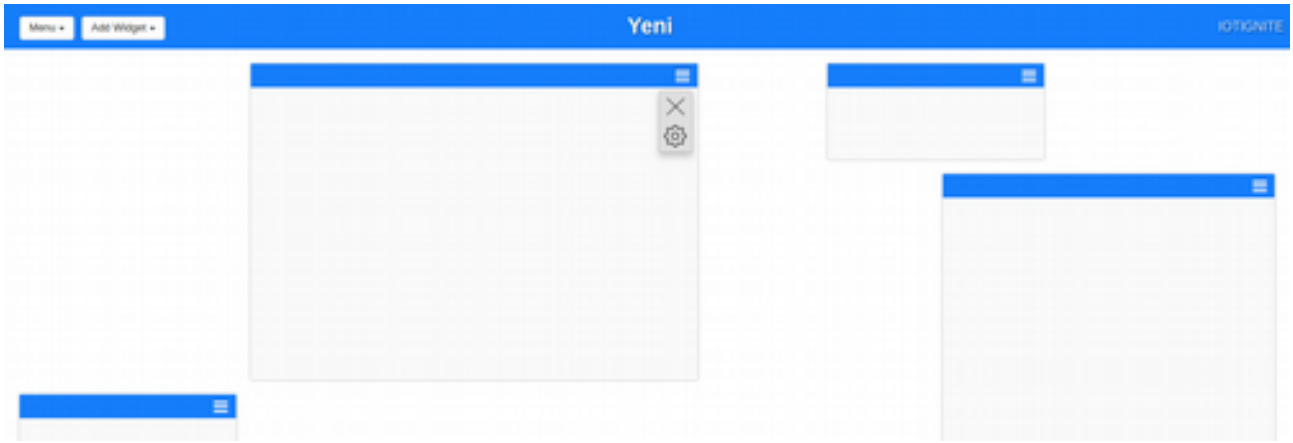
Dashboard’ta Widget Oluşturmak, Konumlandırmak ve Silmek

Menü’den Add Widget’i tıklayıp açılan listeden **Time Series**’i (Zaman Serisi) seçelim.

Görüldüğü üzere sol üst köşeye Time Series widget’i yerleşti.

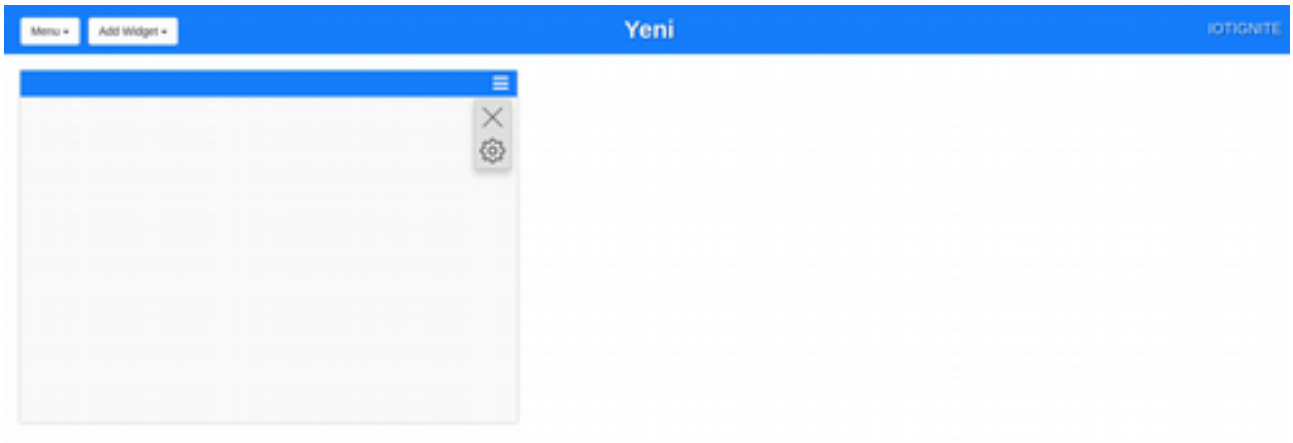


Widget’i başka bir konuma taşımak için mavi renkli widget başlık alanından tutup, çalışma alanının başka bir noktasına taşıyıp bırakmanız yeterlidir. Widget’i kaldırmak, yani silmek için ise başlık alanının sağ tarafında yer alan menü ikonuna tıklayıp, açılan listeden X (sil) butonuna tıklamak yeterlidir. Aşağıdaki şekilde 4 adet farklı farklı widget’in değişik alanlarda yerleşimi görülmektedir.



Widget Yapılandırma Ayarları ve Sensör Verilerini Göstermek

Widget’in sağ üst köşesinde yer alan **menü** ikonuna da tıklayıp **dişli** butonuna tıklayarak, o widget’a ait yapılandırma ekranını açalım.



Açılan yapılandırma ayarları penceresinde bazı özellikler diğer widget türleri ile de ortaktır. Ortak olanlar:

- **Widget Title**; Widget için bir başlık metnidir. Mesela; “Sıcaklık Verileri” gibi...
- **Select Gateway Type**; Gateway’in seçilmesi istenir.
- **Select Sensor;** Gateway seçildikten sonra, o Gateway’e ait sensörleri listelenir ve listeden bir sensör seçilmesi beklenir.
- **Widget Notification Mode** (Widget Bildirim Modu) altında da **Pulse** ve **Alarm** olarak iki adet seçim bulunur. Eğer WebSocket’ten **real-time** olarak sensör verisi gelirse ve bu veri de sizin için önemli ise; Pulse opsiyonunu seçmeniz ile birlikte widget zoomin-zoomout efekti yapacak ve dikkat çekecektir. Eğer beep şeklinde bir uyarı da duymak istiyorsanız Alarm opsiyonunu da seçebilirsiniz.

Bu ayarlar dışında, her bir widget’ın **Data Parameters Settings** (Veri Parameterleri Ayarları) ve **Data Parameters Selection** (Veri Parametreleri Seçimi) olmak üzere bir takım parametrik ayarları da bulunmaktadır.

Aşağıdaki şekilde Time Series widget’ına ait ayarların bulunduğu arayüz görülmektedir.

Data Parameters Selection’da 5 farklı veri formatından birisinin seçilmesi gerekmektedir. Buradaki amaç; o sensörden gelecek olan verinin türünün seçilmesidir.

- **Single Data**: Gelen veri **tekildir**. Örneğin sensörden gelen veri **45** gibi bir değer olabilir. Veya **ONLINE** şeklinde bir string. Eğer veri bir **integer** ise genelde **Time Series**, **Gauge**, **On/Off**, **Slider** widget'larında gösterilir. Eğer bir **string** olarak gösterilecekse **Data List** ve **Text** widget'larında da kullanılabilir.
- **Multi Data (Plain Text)**: Gelen veri bir **Array** olabilir. Bu opsiyon seçilince düz metin olarak gelen Array gösterilir. Genelde gelen tüm Array verilerinin incelenmesi için **Data List** ve **Text** widget'larında kullanılır.
- **Multi Data Selected Index**: Gelen veri bir **Array**'dir ve bu Array'in belli bir **index**'inde yer alan verinin kullanılması için seçilen bir opsiyondur. Bu seçildiğinde, dizin numarasının da girilmesi gerekecektir. Varsayılan olarak seçili dizin **0**'dır, yani ilk elemandır.
- **JSON (Plain Text)**: Aynı Multi Data (Plain Text)'teki mantık bu opsiyonda da geçerlidir. Aradaki tek fark; gelen verinin birinde Array, diğerinde JSON olmasıdır.
- **JSON Selected Key**: Aynı Multi Data Selected Index'teki mantık bu opsiyonda da geçerlidir. Aradaki tek fark; birinde Array'de dizin seçilirken, diğerinde key seçilmelidir. Varsayılan olarak **key** isminde bir anahtar seçilmektedir.

Data Paramters Settings ise her bir widget'a has özellikler barındırmaktadır. Bu ayarlara da widget türlerini incelerken değineceğiz.

Widget Türleri

Generic Dashboard'ta toplamda 9 adet widget türü bulunmaktadır. Bunları sırasıyla Gateway'imizdeki sensörler ile ilişkilendirerek inceleyelim...

> Widget'ları kullanabilmeniz için bilmeniz gereken en önemli kriterlerden biri de sensörden gelen verinin hangi formatta olduğudur. Gelen veri formatını biliyorsanız, o veriyi gösterecek en uygun widget'i seçebilirsiniz. Örneğin 3 Axis Acccelerometer sensörü size X,Y,Z konumlarını bir Array olarak gönderir. Bu 3 değerden hangisini ve nasıl göstereceğinizi belirlemeniz gerekir. Veya sıcaklık sensörü tek bir sayısal değer gönderir; bu veriyi göstermek için de Gauge, Time Series veya Text widget'ları uygun olmaktadır. Veya sadece string veri gönderen bir sensörünüz varsa, onu da Text widget'inde göstermek daha uygun olacaktır. Ayrıca eğer Actuator sensör verisini gösterecek ve bu sensöre de veri push edecekseniz (Demo akışındaki Lamb Actuator'ünü hatırlayın), Slider veya On/Off widget'ini kullanmak gerekecektir.

Time Series

Sensörden gelen tekil sayısal değerlerin bir grafik halinde gösterilmesini sağlar.

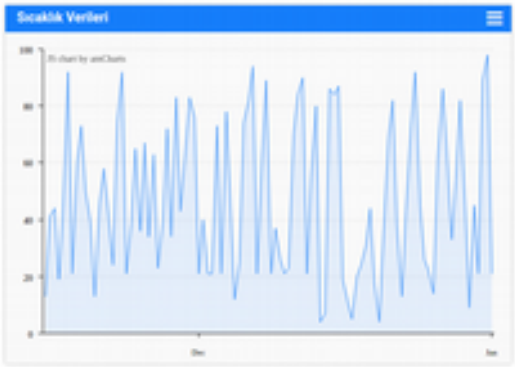
Widget Parameters Settings'te yer alan opsiyonlar şunlardır:

- **Selected time period**: **Now** (Şimdi), **Last One Minute** (Son Bir Dakika), **Last One Hour** (Son Bir Saat) ve **Last One Day** (Son Bir Gün) olmak üzere 4 farklı zaman periyodu vardır. Bu periyotları kapsayan geçmiş veriler çekilerek zaman grafiği oluşturulur ve sonrasında gerçek zamanlı olarak WebSocket'ten gelen veriler sırasıyla bu grafiğin sonuna eklenir.
- **Select time period format**: **Isochronal** (Eş zaman) ve **Relative** (Bağıl) olmak üzere iki opsiyon vardır. Gelen veriler grafikte gösterilirken her bir veri aralığının eşit mi olacağı veya gerçek zamanına göre mi gösterileceği seçilir.

- **Number of data to be displayed**: Selected time period'ta seçilen zaman periyodunda, son gelen sensör verilerinden kaç tanesinin gösterileceği seçilir. Varsayılan olarak 10 değer gösterilir. **10**, **20**, **30**, **50**, **100**, **150** ve **200** olarak seçilebilir.

Örnek olması amacı ile Android Gateway'in Temperature sensörünün Single Data opsiyonu ile son bir günde son 100 data'yı eş zamanlı olarak gösterelim.

> Widget ayarlarını yaptıktan sonra tekrar düzenlemek isterseniz, ayarlar panelini açıp güncelleyip OK butonuna tıklayarak yeni verilerle widget'ı yeniden yeni ayarları ile başlatabilirsiniz.



Gauge

Sensörden **en son gelen** gelen tekil sayısal değerlerin kalibre (ibre gösterge) halinde gösterilmesini sağlar.

Widget Parameters Settings'te yer alan opsiyonlar şunlardır:

- **Definition of value:** String olarak gelen verinin tanımı yapılır. İbrenin alt tarafında görülür. Örneğin sıcaklık sensöründen gelen bir veri ise bu alana **°C** yazılabilir. Zorunlu değildir.
- **Definition of MIN value**: Göstergenin minimum değeri sayısal olarak tanımlanır. Varsayılan değeri **0**'dır.
- **Definition of MAX value**: Göstergenin maximum değeri sayısal olarak tnaımlanır. Varsayılan değeri **100**'dür.

Örnek olması amacı ile Android Gateway'in Temperature sensörünün Single Data opsiyonu ile MIN: 20 ve MAX: 50 olarak gösterelim.

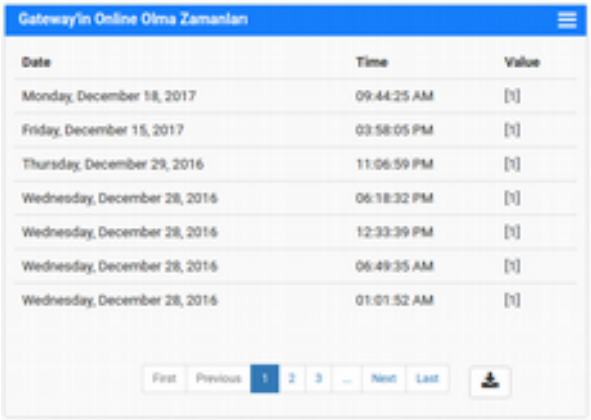
Data List

Verileri bir liste halinde gösterir. Listede tarih, saat ve değer olmak üzere 3 sütun yer almaktadır. WebSocket'ten gelen son veri, listenin en başına eklenir. Veriler **.csv** formatı ile indirilebilir.

Widget Parameters Settings'te yer alan opsiyonlar şunlardır:

- **Number of data to be displayed**: Selected time period'ta seçilen zaman periyodunda, son gelen sensör verilerinden kaç tanesinin gösterileceği seçilir. Varsayılan olarak 10 değer gösterilir. **10**, **20**, **30**, **50**, **100**, **150** ve **200** olarak seçilebilir.

Örnek olması amacı ile Android Gateway'in Multi Data (Plain text) opsiyonu **Built-in Processors** Nodu'larına bağlı **Agent Connection** (Bağlantı) sensöründen gelen online olma zamanlarını son 50 verisini gösterelim.



Text

Sensörden gelen veriler **string** olarak gösterilir.

Widget Parameters Settings'te yer alan opsiyonlar şunlardır:

- **Definition of value:** String olarak gelen verinin tanımı yapılır.

Örnek olması amacı ile Android Gateway'in **Multi Data Selected Index** opsiyonu **Built-in Processors** Node'larına bağlı 3-axis Accelerometer sensöründen X,Y,Z verilerini 3 farkı Text widget'ında **Selected Index** sırasıyla **0 (X)**, **1 (Y)** ve **2 (Z)** ile yan yana gösterelim.

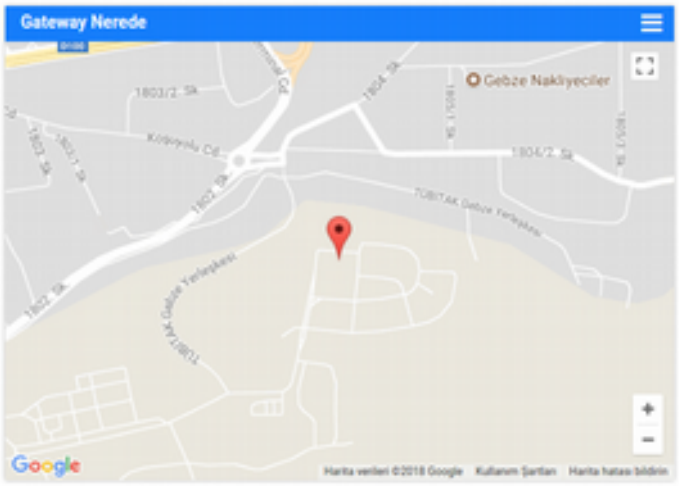
> 3-axis Accelerometer sensörü normalde hiç veri göndermeyecektir, bu duruma şaşırmayın! Çünkü bu sensör için veri yapılandırması yapılmadı. İlk olarak Devzone'da **Nodes** sayfasından ilgili Gateway'i tıklayıp açılan listede, **Built-in Processors** altında yer alan **3-axis Accelerometer**'i seçip, **Data Reading Frequency**'i **Read when occur**, **Data Sending Frequency**'i de **Send when read** olarak yapılandırıp kaydetmeniz gerekmektedir. Sonrasında da bu yeni yapılandırma ayarlarını **DEVELOPMENT** modundayken (veya hangi modda çalışıyorsanız), **Gateway Services** sayfasından **Sensor Data Config** sekmesi altında aktif edip, sonra da **Push** sekmesi altından güncellenmiş modu Gateway'e göndermemiz (push) gerekmektedir. Sonrasında artık sensör anlık olarak verileri

> gönderecektir.

Map

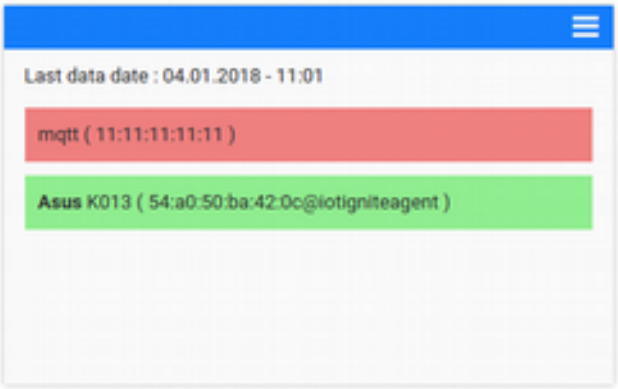
Gateway'in harita üzerinde nerede olduğunu gösteren (eğer Gateway'de GPS bulunuyor ve aktifse) bir widget türüdür. Konum verisinin güncellenmesi için **Check Peroid (Minute)** opsiyonu ile dakika cinsinden **1**, **5**, **10**, **30** ve **60** dakikalarını seçebilirsiniz.

Harita olarak **Google Maps API**'leri kullanılmaktadır. Haritanın **zoom** özelliğini kullanarak Gateway'e yakınlaşıp uzaklaşabilirsiniz. En son zoom yaptığınız ayar, kayıtlı olarak kalacaktır.



Gateway List

Hesabınızda lisanslı olan Gateway'leri listeleyen ve **online** (çevrimiçi) olanları yeşil, **offline** (çevrimdışı) olanları da kırmızı renkle gösteren bir widget türüdür. Listenin üst kısmında güncelleme tarihi gösterilmektedir. Verilerin güncellenmesi için **Check Perriod (Minute)** opsiyonu ile **1**, **5**, **10**, **30** ve **60** dakika süreleri seçilebilir.



On / Off

İkili sistemde veri gönderen **Sensor** ve **Actuator**'larda kullanılan bir widget türüdür. Veriyi gösterebildiği gibi tanımlanan sensör bir Actuator ise, ona veri push edilebilmesini de sağlar. Widget içinde bir **switch button** bulunmaktadır. Tanımlanacak olan ikili sistem verilerine göre

switch button on veya off konumlarına geçiş yapar. Aynı şekilde tam tersi durumda kullanıcı tarafından switch butonda değişiklik yapılırsa, yapılan değişiklik Gateway'in Actuator'üne gönderilir.

Widget Parameters Settings'te yer alan opsiyonlar şunlardır:

- **Definition of value:** String olarak gelen verinin tanımı yapılır.
- **True state value**: Switch butonun **on** (açık) durumuna geçmesi için Actuator'den gelecek olan veri tanımlanır. Yani bu tanımlanan veri geldiğinde switch buton on durumuna geçer. Varsayılan değer 1'dir.
- **False state value**: Switch butonun **off** (kapalı) durumuna geçmesi için Actuator'den gelecek olan veri tanımlanır. Varsayılan değer 0'dır.
- **True push value**: Switch butonun kullanıcı tarafından **on** durumuna geçirilmesi hali ile Actuator'e push edilecek olan veri tanımlanır. Varsayılan değeri boştur.
- **False push value**: Switch butonun kullanıcı tarafından **off** durumuna geçirilmesi hali ile Actuator'e push edilecek olan veri tanımlanır. Varsayılan değeri boştur.

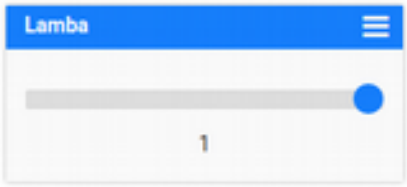
Örnek olması amacı ile Android Gateway'in VirtualDemoNode node'ları altında yer alan Lamp sensörünün açık/kapalı olması durumunu switch butonun on/off durumları ile eşleştirelim. Aynı şekilde switch butonun durumunu da değiştirdikçe Lamp'a veri push edip açıp kapatalım. Bunun için **Single Data** ile opsiyonu True state value: **1**, True push value: **1**, False state value: **0** ve False push value: **0** olarak tanımlanmalıdır.

> Actuator'un hangi verilerle çalıştığından emin olmak için Devzone'da REPORT sayfasına girip Report sekmesinden ilgili Gateway altından Actuator'ü seçip GET REPORT butonuna tıklayın. Gelen raporda VALUE sütununda veri yapısını ve veri değerlerini göreceksiniz.

Slider

Mantık olarak On / Off widget'ı ile aynı mantıkta çalışır. Ancak bu widget da switch buton yerine bir **slider** bileşeni kullanılmaktadır. Amaç; 1 ve 0, yani on ve off durumlarının yetmediği durumlarda daha farklı skalada çalışabilmek içindir. Mesela bir klima için tanımlanmışsa, klimanın sıcaklık değerini bu slider'da görebilir, isterseniz de slider'ı kaydırarak klimanın sıcaklığını değiştirebilirsiniz. Slider'ın **MIN**, **MAX** ve **STEP** parametrelerinin tanımlanması gerekmektedir. Slider her kaydırıldığında, bu tanımlanan parametrelere göre skala değeri üretilecek ve Actuator'e gönderilecektir. Actuator'den de veri geldiğinde, yine bu slider'da skalasına göre değerine yerleşecektir. Eğer değer aralığı dışında ise MIN (eğer skaladan düşük veri gelirse) veya MAX (eğer skaladan büyük veri gelirse) değerinde gösterilecektir. Ek olarak Slider'ın altında da o anki değer string olarak gösterilmektedir.

Örnek olması amacı ile yine Lamp ile eşleştirelim. **Single Data** ile MIN:**0**, MAX:**1** ve STEP: **1** olacak şekilde parametreleri düzenleyelim.

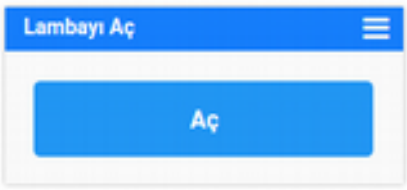


Command Send Widget'ı ile Actuator'e Karmaşık Veri Göndermek

Daha önce görmüş olduğumuz On/ Off ve Slider widget'ları Actuator'lere basit seviyede veri gönderimi sağladık. Ancak istersek gelen ve giden veri türlerini daha karmaşık yapılarla da yapabiliriz. Mesela bir JSON oluşturup birden fazla parametre değeri gönderebilir ve alınabilir. Şu an Android Gateway'de Demo APP üzerinden çalıştığımız için sadece Lamp Actuator'unde 1 ve 0 değerleri ile çalışabildik. Fakat ilerleyen bölümlerde de göreceğiniz üzere kendi Gateway'inizde Node ve Sensor/Actuator oluşturup bunları kaydedebileceksiniz. Kendi tanımlayacağınız Actuator'da da karmaşık veriler kullanabilirsiniz.

Generic Dashboad'ta, Gateway'de özel olarak oluşturduğunuz Actuator'lara komut gönderebilmeniz için Command Send (Komut Gönder) widget'i daha bulunmaktadır. Widget içinde bir adet buton olur ve butona bastıkça tanımladığınız veriler Actuator'a push edilir. Bu widget, sadece Actuator'a veri push eder. Bu widget'in iki farklı parametresi bulunmaktadır. Bunlar;

- **Send Button Text**: Butonda yazacak etiket.
- **Gateway Command:** Actuator'a push edilecek olan veri. Sayı, metin, dizi, obje gibi verileri serbestçe girebilirsiniz. Yine örnek olması amacı ile **Lamp** Actuator'une **1** verisini gönderelim.



Artık bu **Aç** butonuna bastıkça lamba açılacak.

Yeri gelmişken Actuator ve Sensor'un Gateway tarafını da kısaca inceleyelim (İlerleyen bölümlerde detaylıca Gateway'de Sensor ve Actuator oluşturmayı göreceksiniz)....

İlk olarak aşağıdaki linkten **Demo APP**'i Git'ten klonlayın.

```
git clone https://github.com/IoT-Ignite/android-example-IoTIgniteDemoApp.git
```

android-example-IoTIgniteDemoApp/app/src/main/java/com/ardic/android/iotignitedemoapp/

VirtualDemoNodeHandler.java

323..329 satır aralığına bakıldığında Sensor ve Actuator'lerin tanımlanması görülmektedir.

```
        if (myNode.isRegistered()) {
            mTemperatureThing = myNode.createThing(Constants.TEMP_THING, mTempThingType, ThingCategory.EXTERNAL, false, tempThingListener,
            mHumidityThing = myNode.createThing(Constants.HUM_THING, mHumThingType, ThingCategory.EXTERNAL, false, humThingListener, null);
            mLampThing = myNode.createThing(Constants.LAMP_THING, mLampThingType, ThingCategory.EXTERNAL, true, lampThingListener, null);
            registerThingIfNotRegistered(mTemperatureThing);
            registerThingIfNotRegistered(mHumidityThing);
            registerThingIfNotRegistered(mLampThing);
        }
```

```
mTemperatureThing = myNode.createThing(Constants.TEMP_THING, mTempThingType,
ThingCategory.EXTERNAL, false, tempThingListener, null);

mHumidityThing = myNode.createThing(Constants.HUM_THING, mHumThingType,
ThingCategory.EXTERNAL, false, humThingListener, null);

mLampThing = myNode.createThing(Constants.LAMP_THING, mLampThingType,
ThingCategory.EXTERNAL, true, lampThingListener, null);

registerThingIfNotRegistered(mTemperatureThing);
registerThingIfNotRegistered(mHumidityThing);
registerThingIfNotRegistered(mLampThing);
```

Dikkat edeceğiniz üzere Temperature ve Humidity için **createThing()** metodu ile tanımlama yapılırken 4. parametreleri **false**, Lamp için ise **true** olarak tanımlanmıştır. Böylece Temperature ve Humidity birer Sensör, Lamp ise Actuator olarak tanımlanmaktadır. Aslında bu parametre, **Cloud Rule** oluştururken **ACTIONS**'ta **Actuator Message**'da sensörün görünüp görünmemesini belirlemektedir. Bu alanda görüntüleniyorsa, o sensöre veri gönderilebilir demektir.

Actuator olarak tanımlanan sensörde de **onActionRecevied()** metodu ile push edilen mesajlar okunabilmektedir (113. satır).

> Gateway'lerdeki Sensör ve Actuator'ların nasıl tanımlanacağını, bunlardan nasıl ve hangi tiplerde veri gönderileceğini ilerleyen bölümlerde uygulamalı olarak göreceksiniz.

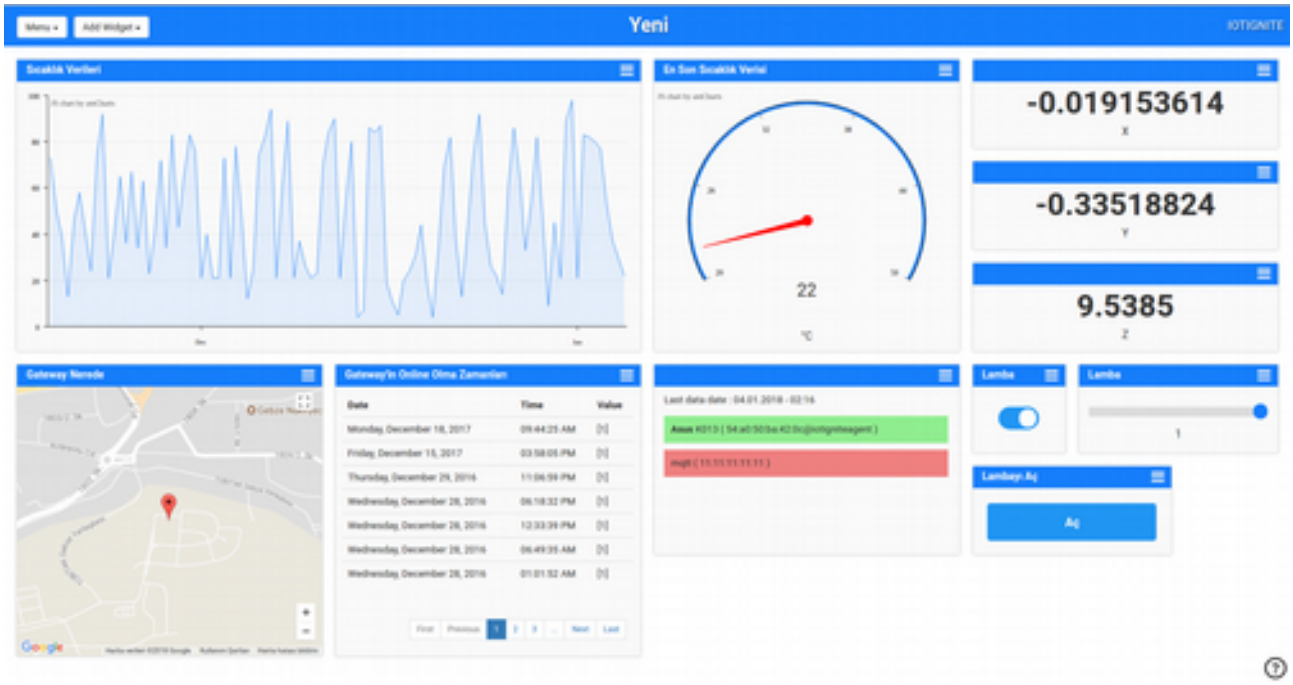
```
        @Override
        public void onActionReceived(String nodeId, String sensorId, final ThingActionData thingActionData) {
            Log.i(TAG, "Action arrived for " + nodeId + " " + sensorId);
            // Toggle lamp in sensor activity
            if(sensorsActivity != null) {
                sensorsActivity.runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        ToggleButton toggleLamp = (ToggleButton) sensorsActivity.findViewById(R.id.toggleLamp);
                        Log.i(TAG, thingActionData.getMessage());
                        int message = 0;
                        try {
                            String s = thingActionData.getMessage();
                            if (s != null) {
                                message = Integer.parseInt(s.replace("\"", ""));
                            }
                        } catch (NumberFormatException e) {
                            Log.i(TAG, "Message Invalid");
                        }
                        toggleLamp.setChecked(message == 1);
                    }
                });
            }
        }
```

```
@Override

    public void onActionReceived(String nodeId, String sensorId,
final ThingActionData thingActionData) {
        Log.i(TAG, "Action arrived for " + nodeId + " " + sensorId);
        // Toggle lamp in sensor activity
        if(sensorsActivity != null) {
            sensorsActivity.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    ToggleButton toggleLamp = (ToggleButton)
sensorsActivity.findViewById(R.id.toggleLamp);
                    Log.i(TAG, thingActionData.getMessage());
                    int message = 0;
                    try {
                        String s = thingActionData.getMessage();
                        if (s != null) {
                            message = Integer.parseInt(s.replace("\"", ""));
                        }
                    } catch (NumberFormatException e) {
                        Log.i(TAG, "Message Invalid");
                    }
                    toggleLamp.setChecked(message == 1);
                }
            });
        }
    }

    @Override
    public void onThingUnregistered(String nodeId, String sensorId) {
        Log.i(TAG, "Lamp thing unregistered!");
        stopReadDataTask(nodeId, sensorId);
    }

};
```

Yazılan kodları incelediğimizde **onActionRecevied()** metodunda 3. parametrede Cloud'tan gelen veri **ThingActionData** parametresine aktarılıyor. **ThingActionData.getMessage()** ile de gelen veri değeri okunabilmektedir. **Try-Catch** bloğunda da gelen veri önce **parse** ediliyor ve **Integer**'a çevriliyor. Sonrasında da gelen veri **toggleLamp** component'inin **setChecked()** metoduna **Int** olarak (**1** veya **0** gelmektedir), atanarak switch button **on/off** konumuna getiriliyor.

Lamp örneğinde basitçe 1 ve 0 Integer değerleri kullanılmaktadır. Ancak gelişmiş bir Actuator hazırlamak istediğinizde gelen veriyi Integer'a değil de JSON'a parse edip, çok daha fazla veri ile çalışabilirsiniz.

En son haliyle Generic Dashboard aşağıdaki gibi görülecektir (Widget'ların yerleşim yerleri farklı olabilir).

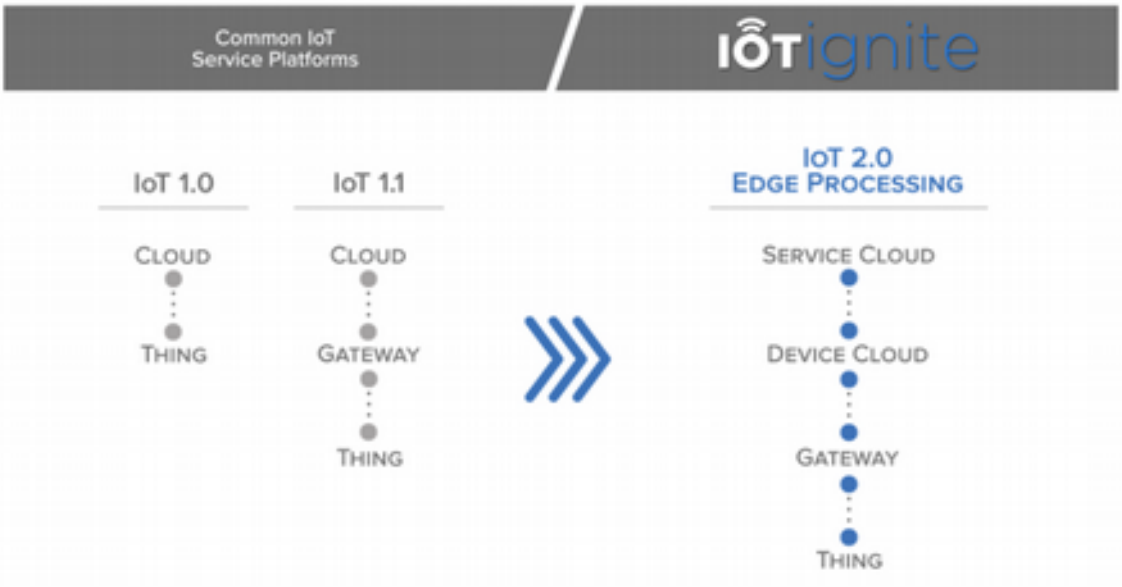

Menu
Add Widget
Yeni
IOTIGNITE
Sıcaklık Verileri
En Son Sıcaklık Verisi
-0.019153614
X
-0.33518824
Y
9.5385
Z
22
°C
Gateway Nerede
Gateway'in Online Olma Zamanları
Date
Time
Value
Lamba
Lamba
1
Lambayı Aç
Aç

BÖLÜM 4

Edge (Fog) Computing

IoT’den IoT 2.0’a Evrim

IoT’den IoT 2.0’a evrimleşmeyi kitabımızın birinci bölümünde ele almıştık. Burada bu süreci Gateway odaklı olarak farklı bir bakış açısıyla ele alacağız. Öncelikle aşağıdaki şemayı inceleyelim.



Yukarıdaki şekil bu süreci net olarak göstermesi bakımından oldukça önemlidir. Öncelikle IoT 1.0’dan başlayarak IoT 2.0’a kadar olan gelişimi adım adım inceleyelim.

IoT 1.0

Günümüzün en yaygın kullanılan IoT dağıtım modelidir. Bu modelde veriler Sensör’den Bulut’a aktarılır. Teknik açıdan bakıldığı zaman, bu modeli geniş coğrafi alana yayılan sistemlerde kullanamayız. Bunun temel nedenleri de aşağıdaki gibidir:

- Dünyada şu ana için 4 milyardan fazla akıllı telefon bulunmaktadır. Ayrıca Cisco’nun tahminlerine göre bu sayı 2020 yılında 50 milyarı geçecek. ARDIC firmasının tahminide buna yakın olmakla birlikte 50 ile 200 milyar arasında bir sayı vermektedir. Özetle IoT 1.0 ile böyle bir ağı kontrol etmek çok zordur.
- IoT 1.0 modelinde veriler doğrudan buluta gönderildiği için bulut sistemlerinin bu kadar veriyle çalışması oldukça zorlaşacaktır. Dolayısıyla büyük verilerle çalışmak bulut sistemlerini hızlı bir şekilde tüketecektir.
- Veri miktarındaki artışla birlikte gerçek zamanlı eylemlerde bulunmak imkansız bir hal alacaktır. Çünkü veri miktarındaki artış veri analizi işleminin daha fazla zamana yayılmasına ve IoT sistemlerinin tıkanmasına neden olacaktır.

Bu problemleri aşabilmek için IoT 1.0 modelinin Gateway adı verilen yeni bir katmana ihtiyacı olacaktır. Gateway, ağın kenarında (edge of the network) bulunan ve doğrudan buluta bağlı olan bir katman olarak tanımlanır.

IoT 1.1

Gateway IoT 1.0’a eklendiği zaman karşımıza IoT 1.1 modeli çıkmaktadır. Bu dağıtım modelinde Gateway katmanı bulunmaktadır. Ancak bu katmanın beraberinde getirdiği bazı sorunlar bulunmaktadır. Şimdi bunlara göz atalım:

- Gateway için hangi işletim sistemi kullanılacak?
- Gateway nasıl bir yönetime sahip olacaktır? (İşletim sistemi, işletim sistemi yapılandırmaları, güvenlik, ağ bağlantısı, sistem üzerinde çalışan servis ve uygulamaların yaşam döngüsü, uzaktan hata ayıklama kabiliyeti vb.)
- Hangi uygulama geliştirme platformları kullanılacak? Java, C/C++, Java Script, Python vb.)

Yukarıdaki soruların hepsine fiili olarak cevap verebilecek herhangi bir standart bulunmamaktadır. Bu durum daha sonra değineceğimiz üzere Edge'nin en büyük dezavantajı olarak karşımıza çıkmaktadır. **IoT-Ignite'nin temel faaliyet noktası; bu gibi sorulara cevap bularak IoT 2.0 standardını kullanmayı sağlamaktır.**

IoT 2.0

Yukarıda anlattıklarımızın sonucunda elimizde şu an var olan IoT dağıtım modeli IoT 2.0'dır. IoT 2.0'da sensör gibi cihazlardan alınan veriler işleme tabi tutulmakta ve gerekirse ileri analiz işlemleri için bulut'a gönderilmektedir. Ayrıca Gateway katmanı acil olan eylemleri gerçekleştirmekte ve gerçek zamanlı işlemleri anında yapabilmektedir. Böyle bir modelde bulutu yormadan ve gereksiz verilerle boğmadan anlık analiz işlemleri ve acil eylemler rahatlıkla yapılabilir. Gateway'in kullanımının en önemli amacı; gerçek zamanlı analizi gerçekleştirmek ve bulut sistemini tüketen verilerin önünde bir baraj gibi set oluşturmaktır.

Gateway Nedir ve Fog Computing'te Yönetilebilir Gateway'in Kritik Önemi

Gateway kavramına bir önceki başlıkta değinmiştik. Gateway, "IoT bir sistemde sensor ve actuator gibi çevre birimlerden alınan verileri anlık olarak analiz edebilen ve gerçek zamanlı eylemlerde bulunarak Bulut sistemlerinin büyük veri yığınları altında tükenmesini engelleyen bir katman." olarak tanımlanabilir.

Edge Computing hakkında bir sonraki başlıktan itibaren ayrıntılı bilgi vereceğiz. Fog Computing ise Cisco firmasının patentini aldığı Edge Mimarisine benzeyen bir IoT uygulama geliştirme katmanıdır. Gateway, Edge Computing ve Fog Computing katmanlarının temel amacı IoT uygulamalarında gerçek zamanlı eylemleri gerçekleştirmek ve Bulut sistemlerini veri yığınlarından korumaktır.

Fog Computing, Cisco tarafından ticari amaçlı olarak geliştirilen ve bir standardı olan kapalı bir sistemdir. OpenFog adı altında geliştirilen bu sistemin geliştirilme sebepleri aşağıdaki gibidir:

- IoT'nin ürettiği büyük verilerle çalışabilmek,
- Bulut sistemlerine gönderilen verilerin anlık işlemleri yapmak için yetersiz kalması,
- Büyük verilerin taşınması mobil şebekeler ile yapılabilmesine rağmen ücretlendirmelerin hala yüksek olması gibi nedenlerden dolayı, Fog Computing katmanı geliştirilmiştir.

Fog'un amaçlarına dikkat edildiği zaman Gateway ile aynı amaçları hedeflediği görülebilmektedir. Yönetilebilir Gateway'in kritik önemini maddelerle özetleyebiliriz:

- IoT-Ignite'de Things olarak ifade edilen sensör verilerinin öncelikle Gateway'de işlenmesi,
- İhtiyaç halinde buluta veri göndermeyi sağlaması,
- Ağ trafiğini azaltması,
- WAN (Wide Area Network) altyapı gereksinimini azaltması,

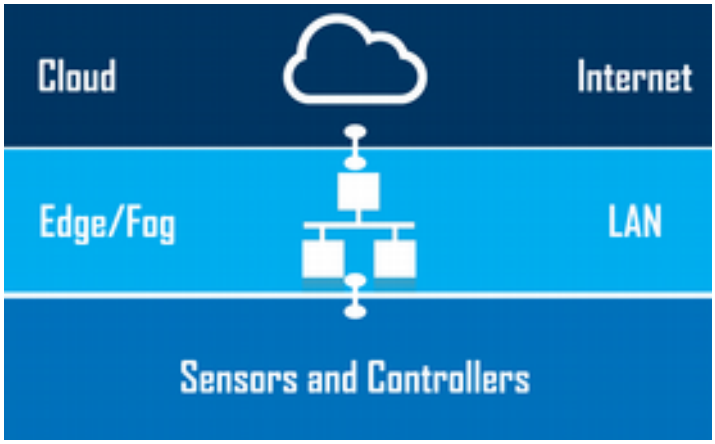
- IoT uygulamalarını gerçekçi ve uygulanabilir bir hale getirmesi,
- Temel işlemleri yapan yeni bir bilgi işlem platformu olması (Gateway'ın IoT uygulamalarındaki anahtar rolüdür),
- Karmaşık olayları işlemesi, edge analytics (kenar analitiği) ve bağlantı yönetimini sağlaması.

Edge Computing Mimarisi

İnternete bağlı cihaz sayısı arttıkça Edge Computing teknolojisine ihtiyacımız bulunmaktadır. Cisco, bu kavramı biraz düzenleyerek Fog Computing adı altından lisansladı.

Bulut mimarisi uzakta bulunan cihazlar üzerinde işlem yapmayı sağladığı için bu sistem etkisini internet ortamında göstermektedir. Ancak Edge ise LAN (Local Area Network), yani yerel ağlarda etkisini gösteren bir kavramdır. Şu an için ara sunucu kavramı da Edge'e dahil edilmektedir. Bu sunucular bulut sunucularından farklı olarak veri kaynağına coğrafi olarak yakın olan alanlara kurulurlar. Ara sunucuların hem Edge hem de bulut sistemlerine çok büyük artısı bulunmaktadır.

Bulut ve Edge arasındaki temel ilişkiyi aşağıdaki şemada görebiliriz.



Edge veya Fog denilen bu yapı, kaynağa yakın durarak veri toplama ve bu veriler üzerinde buluttan bağımsız analitik işlemlerini yapmayı sağlar. Buradaki bağımsızlık bir Edge ile bulut arasında bir bağlantının olmadığı anlamına da gelmez. Bulutu gereksiz bilgilerle yormadan veya meşgul etmeden veri analizi işlemlerini yapabildiğini ifade etmek için kullanmaktayız. Eğer veri, bulutu da ilgilendiriyorsa, şekilde de görüleceği üzere Edge seviyesinde bulunan cihazlar bu verileri buluta gönderebilmektedir.

Verdiğimiz bu kısa bilgilerle birlikte bu başlık altında:

- Edge Computing kavramının ne olduğu,
- Edge Mimarisi,
- Ne zaman kullanılması gerektiği,
- Mobile edge kavramı,
- Edge Computing'in avantaj ve dezavantajları,

... konuları hakkında bilgi sahibi olacaksınız.

Edge Computing Nedir?

Edge Computing, IoT sistemlerinde veri kaynaklarına yakın olan bilgi işlem alt yapısı olarak tanımlanabilir. Ayrıca veri kaynağının yakınında yer alan ve veri işlemeyi gerçekleştirerek bulut bilgi işlem sistemlerini optimize etmeye sağlayan bir kavram şeklinde de tanımlanabilir. Edge'nin temel amacı; bulut sunucularını yormadan veri işlemeyi sağlayarak gerçek zamanda (realtime) eylemlerde bulunmayı sağlamaktır.

Bulut sistemlerinin yaygınlaşması, internete bağlanan cihaz sayısının ve bu cihazlardan elde edilen veri miktarındaki artış Edge'i zorunlu kılmıştır. Uygulamaya bağlı olarak Edge mimarisinde zaman duyarlı olan veriler, kaynak noktasında akıllı bir cihaz ile işlenebilir veya kaynağa coğrafi olarak yakın bir konumda olan bir aracı sunucuya gönderilebilir. Ara sunucular bulut sistemlerine dahil olmayan ama buluta hizmet eden sunuculardır.

Burada verdiğimiz bilgilerden yola çıkarak Edge Computing için kullanabileceğimiz cihazlar şunlardır:

- Akıllı cihazlar (Akıllı telefonlar, Geliştirme kartları vb.) Bunlara kısaca Gateway diyebiliriz.
- Ara Sunucular.

Edge Computing'in IoT uygulamalarındaki temel rolü ise; bulut sistemlerine veri girmek, depolamak, filtrelemek ve göndermek gibi işlemleri yapmaktır. Edge kavramı yeni bir kavram olmamakla birlikte, bu kavramın gündem de olmasını sağlayan temel faktörleri aşağıdaki gibi sıralayabiliriz:

- Sensör maliyetlerinin düşmesi,
- Mikroişlemcilerdeki gelişme. Özellikle büyük verileri işleyebilme kapasitesinin artması,
- IoT sistemlerinde bulunan cihaz sayısındaki artışla doğru orantılı olarak veri miktarının hızla artması,
- Modern makine öğrenimi ve veri analizi yönetimlerindeki kayda değer ilerleme.

Bu faktörlerin gelişmesi ile birlikte şirketler büyük miktarda veriyi en yakın noktada anlayışlı ve akıllı eylemler haline dönüştürmeye çalışmaktadırlar.

Edge Mimarisine Genel Bir Bakış ve Mantığını Kavramak

Edge Computing mimarisini anlamak, bu gibi sistemleri geliştirmek için en önemli adımdır. Edge Computing mimarisinde iki önemli cihaz bulunmaktadır:

- Gateway (IoT-Ignite platformunda, IoT odaklı servisler içeren, özellikle Android işletim sistemi çalıştıran cihazlar)
- Intermediary Server (Ara Sunucular)

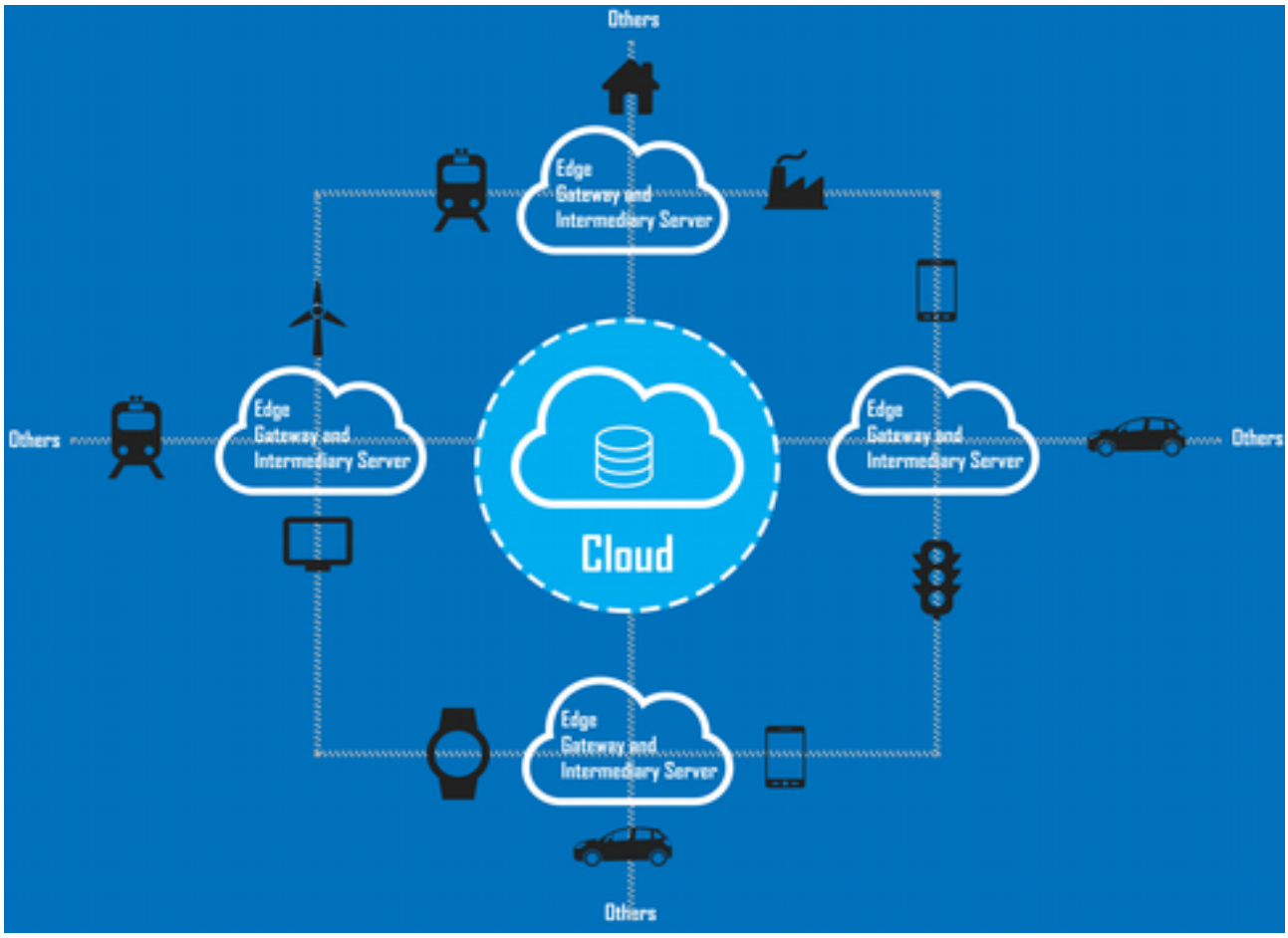
Bu iki tür cihazın amacı IoT ekosisteminde bulunan sensörlerden verileri okuyarak yerel seviyede sistemin kontrolünü ve çalışmasını sağlamaktır. Ara sunucular, Gateway cihazlar gibi davranmakta

ve sadece ihtiyaç duyulan durumlarda kullanılmaktadır. Özellikle aynı sistemde bulunan, ama birbirinden coğrafi olarak uzakta bulunan sistemlerin haberleşmesinde ara sunucular kullanılmaktadır.

Edge Computing mimarisinde çok farklı araç ve cihazlardan gelen veriler Gateway veya ara sunuculara akmaktadır. Burada temel düzeyde analiz işlemi yapılmakta ve ihtiyaç duyulan durumlarda veriler bulut sistemine gönderilmektedir. Böyle bir mimarinin en büyük avantajı; analiz işlemini yapabilecek cihazlar vasıtasıyla gerçek zamanlı eylemde bulunmayı sağlamaktır.

Edge mimarisinde Gateway ve ara sunucular olmakla birlikte bu cihazların tamamı bulut sistemleri ile etkileşim halindedir. Yani burada bulut sistemlerinden bir kopma yok, sadece bulutun geciktiği yerlerde eylemlerin gerçek zamanlı olarak icra edilmesini sağlamak vardır. Bundan dolayı Edge, Cloud ile organizeli şekilde çalışan bir sistemdir.

Edge mimarisini anlamak için aşağıdaki şekli inceleyelim.



Yukarıda görüleceği üzere Edge ve bulut birlikte çalışmaktadır. Ancak Edge'nin bulut ve IoT sistemde bulunan diğer cihazlar arasında kaldığı ve bu cihazlardan gelen verileri öncelikli olarak kontrol ettiği görülmektedir. Burada verilen şema IoT sisteminin en basit halidir. Bu şekilde milyonlarca cihazın aynı anda internete bağlandığını ve sürekli veri gönderdiğini hayal ettiğinizde, bulut sistemlerin yaşama şansı sıfıra düşmektedir. Edge Computing ile gelen cihaz ve sistemlerin temel amacı; cihazlardan gelen veri önünde baraj gibi bir set oluşturmak ve ihtiyaç duyulduğu zaman verileri buluta göndermek vardır.

Edge cihazlarında işlemci olduğu için analiz işlemlerinin bir kısmı da burada yapılmaktadır. Ancak kritik öneme sahip veriler geldiği zaman bu verilerin kesinlikle buluta gönderilmesi gerekmektedir. Eğer bir Edge sistemi başka bir Edge sistemini ilgilendirecek bir veri aldıysa, bu gibi durumlarda bu

verinin buluta gönderilmesi gerekiyor. Aksi durumda yani lokalde ve gerçek zamanlı eylemde bulunmak için, tüm analiz ve kontrol işlemlerini mümkün olduğunca Edge seviyesinde yapmalıyız.

Edge Computing Ne Şartlarda Kullanıbilir?

Edge Computing'i IoT uygulamalarında her zaman için kullanabiliriz. Bu sistemin zorunlu olarak kullanılması gereken şartlardan bahsedelim...

Edge Computing'i aşağıdaki şartlardan biri veya birkaçı ortaya çıktığı zaman kullanabilirsiniz:

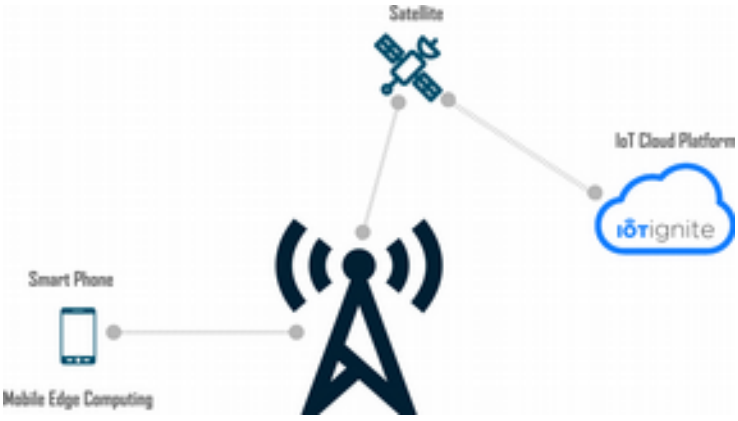
- IoT uygulaması büyük bir coğrafyaya yayıldığında,
- IoT uygulamalarında araçlar, akıllı saatler, akıllı telefonlar, demir yolları, fabrikalar, yani çok çeşitli ve birbirinden farklı cihazları birbirine bağladığınızda. Çünkü çok çeşitli cihazlardan elde edilen veri miktarı hem büyük hem de heterojen bir yapıdadır. Bu gibi durumlarda Edge'i kullanmanız gerekir.
- Verileri saniyeden daha düşük bir zaman diliminde analiz edip anlık olarak eylemde bulunmak istenen durumlarda, kesinlikle Edge'i kullanmanız gerekir.

Bu son madde Edge'in geliştirilmesinde en büyük etkene sahiptir. Çünkü bulut sistemleri, veri analizini anlık olarak gerçekleştiremez. Bundan dolayı gerçek zamanlı işlemleri yapmak mümkün değildir. Aslında bulut sistemlerinde bu zaman meselesinin çok değişken olduğunu söylemek daha doğru olacaktır. Edge ile saniyenin altına yapılan bir işlem bulut ile dakikalar, saatler hatta internet bağlantısında bir sorun varsa haftalar alabilir.

Mobile Edge Kavramı

Mobile Edge kavramı, bir ağ mimari konseptidir. Temel amacı; hücresel ağları kullanarak (2G, 3G, LTE vb.) bulut bilgi işlem yeteneği oluşturmayı sağlar. Mobile Edge'in en büyük avantajı; ağ tıkanıklığını azaltması ve uygulamaların daha iyi performans göstermesini sağlamasıdır. Nihayetinde 4G gibi hücresel ağ teknolojileri daha hızlı veri iletimini gerçekleştirerek, kimi zaman kablolu bağlantılardan daha hızlı işlem yapabilmektedirler.

Mobil teknolojilerin ağırlıklı olduğu bu kavramın tek dezavantajı ise; veri ücretlendirmelerinin yüksek olmasından dolayı, büyük verilerin iletiminin pahalıya mal olmasıdır. Ancak gerçek zamanlı işlem yapma ve küçük boyutlu verilerin iletiminde bu teknolojilerin kullanımı, daha hızlı ve performanslı IoT uygulamalarının geliştirilmesini sağlamaktadır.



Mobile Edge için yapılan en yenilikçi çalışma 21 Şubat 2017 tarihinde Intel ve Ericsson başta olmak üzere birçok firmanın ortaklaşa başlattığı 5G projesidir. Bu teknolojinin temelleri 2008 yılına kadar gitmektedir. Ancak bizi ilgilendiren asıl çalışma Ericssson'un Şubat 2017 tarihinde oluşturduğu 5G platformudur. Bu çalışma end-to-end, yani uçtan uca desteği ile gelen ilk 5G

çalışmasıdır. Dolayısıyla bu 5G çalışması doğrudan IoT-Ignite platformunun çalışmasına istemeden de olsa katkıda bulunan bir çalışmadır. **IoT-Ignite platformunun 2012 yılında yayınlandığını düşünürsek ARDIC firmasının geleceği ne kadar yakından analiz ettiği görülmektedir.**

5G teknolojisi, 2020 yılından itibaren kullanılacak ve ilk kullanım alanı Industrial Internet of Things (IIoT) olacaktır. Ancak IIoT ile birlikte Internet of Things için de çözümler sunacaktır. 5G, 4G teknolojisinden daha hızlı olup aşağıdaki hızlara sahip olacaktır:

- 450Mbps (single-stream)
- 900Mbps (dual-stream)
- 1.3Gbps (three-stream)

IoT uygulamaları dünya genelinde hızla yayılırken, 5G teknolojisinin gelmesi bu süreci daha da hızlandıracaktır. Özellikle yukarıdaki hızları incelediğimizde gerçek zamanlı veri işlemenin ne kadar hızlanacağını kestirmek zor olmasa gerek.

IoT-Ignite platformu; Android cihazları Gateway olarak kullandığı için, bu platformun IoT uygulamaları için ne kadar da önemli olduğu kendini bir kez daha hissettirmektedir.

Mobile Edge'nin faydalarını aşağıdaki gibi sıralayabiliriz:

- Ağ tıkanıklığı sorunlarını aşması ve veri trafiğini hızlandırması. Bu sayede yerel hizmetlerin verimli bir şekilde sunulması sağlanacaktır.
- Güvenlik gerektiren M2M sistemlerinin geliştirilmesini sağlaması.
- Büyük veri ve analiz işlemlerini hızlı bir şekilde gerçekleştirmesi.
- Gecikmeyi en aza indirmesi. Bu durum mükemmel bir kullanıcı deneyimi sunmayı sağlamaktadır.
- IoT sistemde bulunan tüm kablosuz cihazların yer ve konumunu daha hızlı tespit etmeyi sağlaması.

Edge Computing Neden Zorunludur, Avantaj ve Dezavantajları Nelerdir?

Cloud Computing teknolojisinde verilerin ağ üzerinden uzakta bulunan sunuculara gönderilmesi gerekiyor. Şu an için bu sistem IoT uygulamalarının taleplerini karşılamaktadır. Ancak 2020'de 50 milyar cihazın birbirine bağlandığını düşündüğümüzde, Cloud sistemleri tek başına bu kadar veriyi işlemekte yetersiz kalacak. Verilerin bulutta analiz edilmesi ve depolanması bir yana, bu verilerin buluta ağ üzerinden gönderilmesi bile tam bir muamma. Diyelim ki bu problemleri aştık, bizi bekleyen bir diğer sorun gerçek zamana yakın işlem yapmak. Büyük verilerin iletimi ve bunların analiz edilerek başka sistemlerin çalıştırılması ve kontrolü sadece bulut ile gerçek zamanda yapılamaz. Tüm bu sebeplerden dolayı Edge Computing'e ihtiyacımız var.

Burada anlattıklarımızdan yola çıkarak Edge'i zorunlu kılan temel nedeni şöyle özetleyebiliriz; **Büyük miktarda veri iletiminin pahalı olması ve ağ sistemlerinin kaldıramayacağı bir trafik oluşturmasından dolayı Edge Computing'e ihtiyacımız bulunmaktadır.**

Edge Computing, veriyi kaynağa yakın bir yerde işlemeyi ve yalnızca ilgili verileri ağ üzerinden göndermeyi sağlar. Bu sistem büyük verilerin tamamının ağa yönelmesini engelleyen bir baraj olarak düşünülebilir. Sadece ihtiyaç duyulan zamanlarda ağa veri gönderilir.

Örnek vermek gerekirse, ortam sıcaklığını sürekli ölçen bir sistemde elde edilen sıcaklık verisini ağa sürekli göndermek zorunda değiliz. Bunun yerine sıcaklık belirli bir değerin üstüne çıktığı veya altına düştüğü zamanlarda ağa veri göndermek daha stabil uygulamalar geliştirmeyi sağlar. Ya da IoT sistemine bağlı bir kameradan sürekli veri alıp ağa göndermek yerine, bazı hareketler meydana geldiği zaman bunu ağa göndermek gibi uygulamalar Edge Computing'e örnek olarak verilebilir.

Edge Computing teknolojisinin avantajlarını aşağıdaki gibi sıralayabiliriz:

- Anlık verilerin yerel bir işlemciyle işlenmesi sağlanarak gerçek zamanlı işlemler yapılması sağlanır.
- Bulut sisteminde bulunan sunucuların daha az işlem yapmasını sağlar. Edge ile bulut sunucuları sadece depolaması gereken veriler ile çalışır.
- Taşınması gereken verilerin hacmini ve verilerin gitmesi gereken mesafeyi önemli ölçüde azaltarak iletim maliyetlerini düşürmeyi sağlar.
- Gerçek zamanlı yapılması gereken işlemlerin zamanında yapılmasını sağlayarak gecikmeyi azaltır.
- Veri tıkanıklığı veya çakışmasını engelleyerek potansiyel başarısızlığı ortadan kaldırır.
- Güvenlik duvarları ve güvenlik noktalarını arttırarak daha güçlü bir denetleme sağlar.

Edge Computing'in yukarıdaki avantajlarıyla birlikte, teknoloji platformlarının seçiminde bir dezavantajı bulunmaktadır. Özellikle web uygulamaları ve servislerle ilgili tercihlerde... Ancak bu, ilerleyen zamanlarda ihtiyaç duyulan uygulamalar geliştirildikçe bu durum kolay bir şekilde aşılacaktır. **IoT-Ignite platformunun önemi kendisini burada göstermektedir. Çünkü Android cihazları kullanarak Gateway düzeyde Edge Computing uygulamalarını rahatlıkla geliştirebilirsiniz.** Bunun için ihtiyacımız olan yazılım desteğini ve web servislerini bize sağlamaktadır.

Uçtan Uca (End to End) Edge Computing ile IoT Mimarisi Analizi

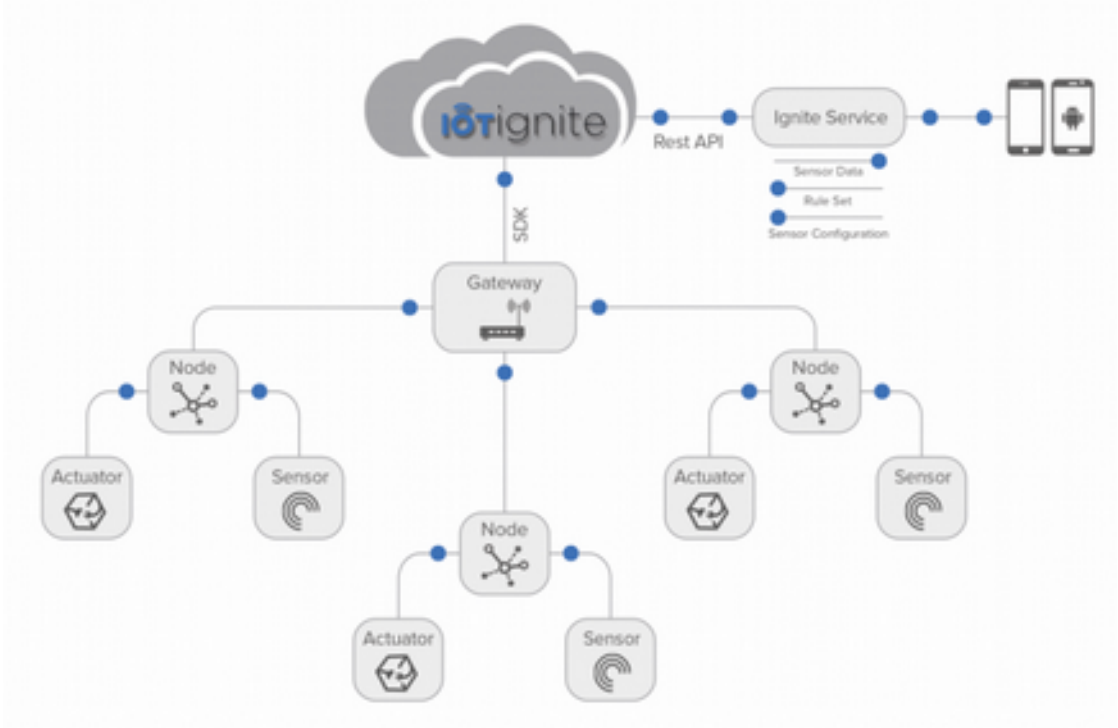
Edge Computing katmanı hakkında bilmemiz gerekenleri paylaştıktan sonra incelememiz gereken bir diğer kavram da End to End, yani Uçtan Uca Edge Computing kavramıdır. Bu yaklaşımda IoT'nin dar alanda odaklanması aşılarak, daha geniş bir platformu uçtan uca yönetme yaklaşımı vardır.

Bu sistemin en iyi örneğini IoT-Ignite platformunda rahatlıkla görmekteyiz. IoT dünyasında yeni yaklaşımlar sergileyen bu platform ile akıllı telefonunuzu kullanarak sistemde bulunan sensörlerden gelen verileri grafiksel bir arayüz ile kontrol edebilirsiniz.

Uçtan uca platformu, Edge Computing'te olduğu gibi Gateway cihazlara sahip olmakla birlikte, ayrıca Node diye tabir edilen bir katmana daha sahiptir. Node katmanında bulunan cihazlar sensörlerden verileri alarak Gateway cihazlara göndermekte, Gateway ise Edge'de olduğu gibi ihtiyaç halinde veri analizi yapmak, gerçek zamanlı eylemde bulunmak ve kritik verilerin buluta taşınması gibi işlemleri yapmaktadır.

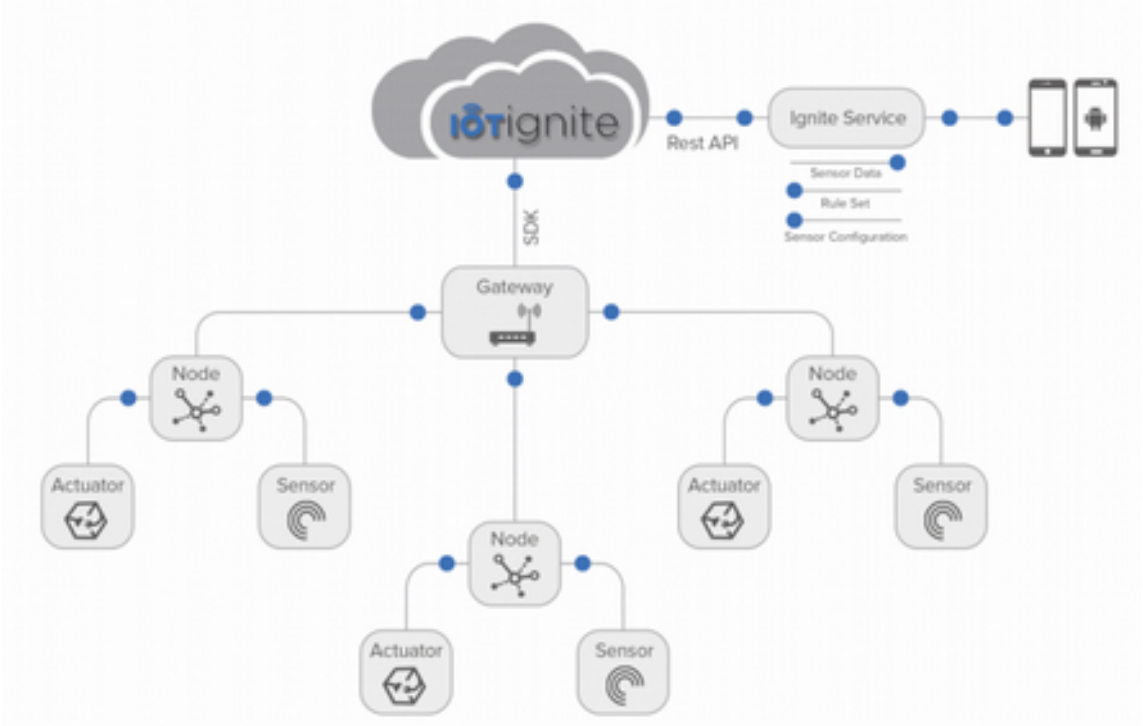
Aşağıdaki şekilde, uçtan uca Edge Computing ile geliştirilen IoT-Ignite platformunun IoT Mimarisi gösterilmektedir. Bu mimariye böyle bir isim verilmesinin temelinde şüphesiz, IoT sistemlerde bulunan iki uç noktayı birleştirmesi yatmaktadır. Bu gibi sistemlerde uç noktalar sensörler ve son

kullanıcılardır. IoT-Ignite platformu sensörlerden son kullanıcıya kadar olan süreci birleştirmeyi başarmıştır.



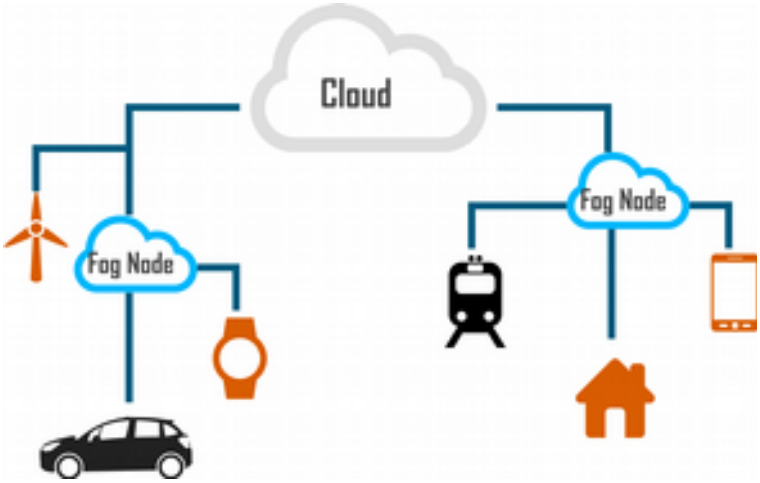
IoT-Ignite'ta Fog Computing Yaklaşımı ve Üstün Özellikleri

IoT-Ignite'de Fog yaklaşımı ve üstün özelliklerine değinmeden önce aşağıda yer alan şekli inceleyelim.



Bu şemada IoT-Ignite platformunun genel olarak çalışma mimarisi gösterilmektedir. Burada sensörlerin öncelikle Node cihazlara oradan da Gateway'e bağlandığı görülmektedir. IoT-Ignite'da veri analizi hem Node hem de Gateway'de yapılabilir. Bir diğer dikkat çeken nokta ise Cloud seviyesinde yapılan işlemlerin Android yüklü akıllı bir cihazda takip edilebilmesidir.

Aşağıda verilen şemada ise Fog Computing platformunun çalışma mimarisini göstermektedir.



Fog Computing'in çalışma mantığı ve mimarisi yukarıdaki gibidir. Burada Gateway yerine Fog Node kavramının olduğunu görmekteyiz. Fog'un içerisine yeterli sayıda işlemciye sahip herhangi bir router, switch veya gateway gibi ağ ile iletişimi sağlayan cihazlar dahil edilebilir. Fog bir sistemde veri analizi yerel ağda gerçekleşir. Ayrıca ağda bulunan ve ağa veri sağlayan tüm cihazların Fog Node'a bağlanması gerekiyor. Bunun temel amacı yerel ağ seviyesinde veri bütünlüğünü sağlamaktır. LAN'da bulunan tüm cihazların tek bir Fog'a bağlanmasını ve bu şekilde sisteme bir bütün olarak bakmayı hedeflemektedir.

IoT-Ignite genel olarak Fog yaklaşımını benimsemekle beraber, Fog Computing'e oranla daha esnek bir kullanıma sahiptir. Bu platformun Fog'a oranla üstünlüklerini aşağıdaki şekilde sıralayabiliriz.

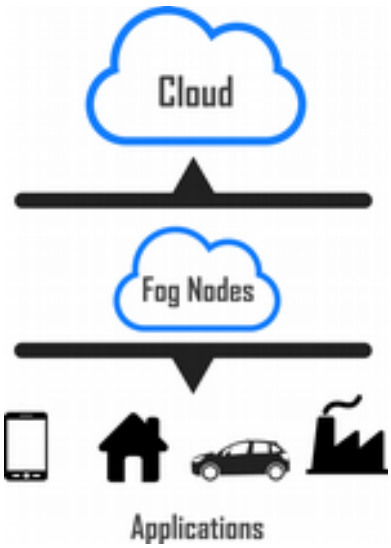
- Özellikle Gateway olarak çeşitli cihazlarla (Raspberry Pi, Android telefon ve tablet, PC vb.) çalışmayı desteklemektedir.
- Yerel ağ seviyesinde veri analizini yapmakla birlikte tüm veri analizlerinin tek bir Gateway'da yapılması zorunlu değildir. Bundan dolayı Node kavramı geliştirilmiştir. IoT ekosisteminde bulunan sensörler önce Node sonra da Node vasıtasıyla Gateway'e bağlanmaktadır. Yani bazı analiz işlemleri Gateway'i meşgul etmeden doğrudan Node ile yapılabilmektedir. Fog'ta ise tüm işlemlerin tek bir Gateway'de yapılması gerekiyor.
- Fog Computing'ten en önemli üstünlüğü **Android** yüklü cihazları Gateway olarak kullanmayı desteklemesidir. Bunun için **Ignite Agent** isimli bir mobile uygulama ile birlikte bazı uygulamalar daha geliştirilmiştir. Fog sisteminde böyle bir destek bulunmamaktadır.
- IoT-Ignite ile gelen bulut platformunun Android yüklü cihazlarla uyumlu olarak çalışması ve sensör verilerinin gerçek zamanlı olarak Android bir uygulama üzerinden takip edilebilmesi. Fog sisteminde böyle bir destek bulunmamaktadır.
- Fog Computing'in öncelikli hedefinin ticari olması, IoT-Ignite'ın ise hem ticari hem de kişisel olarak IoT sistemleri geliştirmeyi desteklemesidir. IoT-Ignite'ın son kullanıcılara yönelik olduğunun en büyük göstergesi, şüphesiz Android cihazları Gateway olarak kullanmayı desteklemesidir.

Tüm bu üstünlüklerle, IoT-Ignite Fog'tan yaklaşık 3 yıl önce yayınlandığı için hem donanım hem de yazılım olarak geniş bir desteğe sahiptir.

Klasik IoT Cloud ve Fog Computing Cloud Karşılaştırması

Klasik bulut sistemleri ve Fog Computing sistemleri birbirinden bağımsız bileşenler değildir. Her ikisi bir araya gelerek daha kaliteli ve gerçek zamanlı işlem yapabilen IoT uygulamaları geliştirmeyi sağlamak için tasarlanmıştır. Her şeyden önce şunu bilmekte fayda var; Fog Computing bulut, IoT uygulamalarında klasik bulut sistemlerinin yetersiz kaldığı durumlarda kullanılmak üzere tasarlanmıştır. Bundan dolayı her iki sistemi birbirinden bağımsız olarak düşünemeyiz. Ancak ikisi de bulut tabanlı olduğu için arada bir karşılaştırma yapılabilir.

Fog ile bulut arasındaki ilişki aşağıdaki gibidir.



Klasik bulut ve Fog Computing karşılaştırmasına değinmeden önce, bu sistemler hakkında kısaca bilgi verelim.

Klasik bulut sistemlerinde sensör gibi çevre birimlerden alınan veriler doğrudan buluta taşınmaktadır. Şu an için internete bağlı cihaz sayısı az olduğu için bulut sistemleri bu talepleri karşılamaktadır. Ancak 2020 yılı bu sistemlerin yetersiz kalmaya başladığı bir yıl olacaktır. Bulut sistemleri bu tarihten itibaren yine kullanılacak, ama bulutun gerçek zamanlı olarak işlem yapamadığı durumlarda Edge Computing ve Fog Computing gibi katmanlar gerçek zamanlı işlem yapmak için kullanılacaktır. Yine bulut sistemleri milyonlarca cihazdan gelen veriler ile çalışamayacağı için Fog sistemleri büyük veriler ile bulut arasında bir set görevi görecek ve kritik olmayan verilerin buluta aktarılmasına engel olacaktır.

Fog Computing, Cisco tarafından geliştirilen veri, hesaplama, depolama ve uygulamaların veri kaynağı ile bulut arasındaki en mantıklı ve verimli yerde dağıtıldığı merkezi olmayan bir bilgi işlem altyapısıdır. Fog Computing, esas olarak bulut bilgi işlemini ve servisleri ağın kenarına kadar genişletir ve bulutun avantajlarını ve gücünü, verilerin oluşturulduğu ve uygulandığı yere yaklaştırır. Fog'un amacı; verimliliği artırmak ve işleme, analiz ve depolama için buluta taşınan verilerin miktarını azaltmaktır.

Cisco, Fog Computing'i Cloud hizmetlerini ağın kenarına kadar uzatan bir kavram olarak tanımlamaktadır. Cloud sistemlerinde olduğu gibi son kullanıcıların ihtiyacı olan veri depolama, analiz ve uygulama hizmetleri vermeyi sağlar. Fog sistemi bu özellikleriyle son kullanıcılara yakın ve geniş coğrafi alana yayılan IoT uygulamaları geliştirmeyi sağlamaktadır.

Kenar aygıtları ve sensörler, verilerin üretildiği ve toplandığı yerde bulunsa da, gelişmiş analitik ve makine öğrenme görevlerini yerine getirecek hesaplama ve depolama kaynaklarına sahip değildirler. Bulut sunucularının bunları yapabilecek gücü olsa da, genellikle verileri işlemek ve

zamanında yanıt vermek için çok uzaktırlar. Fog Computing de ise işlemler, akıllı bir cihazda veya akıllı yönlendiricide gerçekleşir ve böylece buluta gönderilen veri miktarını azalır.

Bu kavramlardan dolayı Edge ve Fog genellikle birbirinin yerine kullanılmaktadır. Çünkü her ikisi de, veri analizini, verinin üretildiği yerde yapmayı sağlamaktadır. Ancak her ikisi arasındaki temel fark; hesaplama işleminin yerleştirildiği konumdur. Fog sistemlerde veri analizi yerel ağda gerçekleşir. Veriler, bitiş noktalarından bir ağ geçidine aktarılır ve burada işleme alınır daha sonra kaynaklara iletilir. Edge'de ise bu işlemler Gateway cihazlarda yapılmaktadır.

Edge bir sistemde bulunan Gateway cihazların birbirinden bağımsız olarak çalıştığı için hangi verilerin analiz edileceğine, hangi verilerin yerel olarak depolanacağına ve hangi verilerin buluta gönderileceğine karar vermeyi sağladığından bir kesim Edge'yi desteklerken, diğer bir kesim de

ağın daha iyi bir veri bütünlüğü sağladığı için Fog'u desteklemektedir.

Fog Computing hakkında bu bilgileri verdikten sonra bu sistemin avantajlarından bahsedip sonrasında klasik bulut ile Fog Computing karşılaştırmasını bir tabloda görelim.

- Ağ üzerindeki veri trafiğini azaltması,
- Sistem maliyetlerini azaltması,
- Gecikmeyi azaltarak gerçek zamanlı eylemde bulunmayı sağlaması,
- Şifrelenmiş verilerin güvenlik açısından son kullanıcıya yakın olması ve saldırılara karşı daha güvenli ortam sağlanması,
- Başarısızlık ihtimalini azalması,
- Veri işlemede yüksek seviyede güvenirlik ve hata toleransı sağlaması,
- Ağ trafiğinde daha aza bant genişliği tüketmeyi sağlaması.

Klasik Cloud ve Fog Computing arasındaki farklılıkları daha iyi görmek adına aşağıdaki tabloyu inceleyebiliriz.

Kriter	**Klasik Bulut**	**Fog**
Gecikme	Yüksek	Çok Düşük
IoT Sistemdeki Konumu	Internet Ortamında	Yerel Ağda
Müşteri Yakınlığı	Uzak	Yakın
Güvenlik	Zayıf	Güçlü
Veriye Saldırı İhtimali	Yüksek	Düşük
Coğrafi Dağılım	Merkezi	Dağınık
Mobile Destek	Sınırlı	Var
Gerçek Zamanlı Etkileşim	Var	Var
Gerçek Zamanlı Eylem	Belirli Değil	Saniyenin Altında

Bu tablodan görüleceği üzere gerçek zamanlı eylemlerde bulunmak için bulut sistemlerin ne kadar yetersiz olduğu görülmektedir. Bu farklılıklara rağmen bulut ve Fog Computing beraber çalışan

birer model olarak karşımıza çıkmaktadır. Tabi Fog'un geliştirilme amacı, bulut sistemlerin eksikliklerini gidermektir.

IoT-Ignite ve Fog Computing

IoT-Ignite ve Fog sistemlerini daha önce karşılaştırdık. Burada özetle şuna değineceğiz; Fog Computing'in önünde en büyük problemlerden biri olan platform belirsizliği, IoT-Ignite'de bulunmamaktadır. Bu durum Fog mimarisinin yeni bir çalışma olmasından kaynaklanıyor. IoT-Ignite ise Fog'a oranla daha tecrübeli ve birçok platform desteğine sahip olarak geliştirilmiştir.

IoT-Ignite ile Fog arasındaki en büyük ayırım ise Android cihazları Gateway olarak kullanmasıdır. Özellikle son kullanıcı tarafında en yaygın kullanıma sahip olan bu cihazların, IoT-Ignite'ta kullanılması uçtan uca uygulamalarını geliştirirken son kullanıcıların işlemlerini oldukça pratik yapmasını sağlamaktadır.

IoT-Ignite ile Desteklenen Platformlar

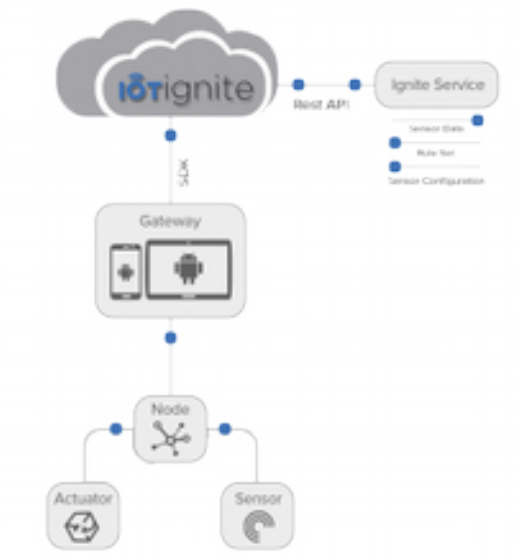
IoT-Ignite ile aşağıda verilen platformları kullanarak IoT uygulamaları geliştirebiliriz:

- Android
- PilarOS
- MQTT ile Genel Kullanım (Windows, Linux ...)

Şimdi bunlar hakkında kısaca bilgi verelim...

Android

IoT-Ignite ile Fog Computing arasındaki en önemli ayırım şüphesiz Android işletim sistemi yüklü telefon ve tabletlerin Gateway olarak kullanılabilmesidir. Çünkü Android ile gelen geniş yelpazedeki sensör desteği ile birçok Android cihazı sıcaklık, nem, manyetometre, ivmeölçer ve daha birçok sensöre dahili (built-in) olarak sahip olmaktadır. Böyle bir gelişme beraberinde Android cihazların Gateway olarak kullanılabilmesini sağlamaktadır.



IoT-Ignite platformunun geniş bir kullanıcı yelpazesi için geliştirilen IoT-Ignite aracılığı sayesinde herhangi bir Android telefon veya Android tablet bir ağ geçidine dönüştürülebilir. Bunun için **IoT-**

Ignite Agent yazılımını **Play Store**'dan indirmek yeterli olacaktır. İndirme işleminden sonra okutacağınız bir **QR** kodu ile IoT-Ignite Cloud'a rahatlıkla bağlanabilirsiniz.

IoT-Ignite Cloud ile iletişim kuran Android bir cihazla aşağıda verile işlemleri yapabiliriz;

- 2000'den fazla Ignite Cloud API'si kullanılabilir,
- Android cihazda veya bulutta yeni kurallar oluşturabilir ve bunları rahatlıkla yönetebilirsiniz.
- IoT servisleri oluşturmak için cihaza harici bir Node bağlantısı yapılabilir.

Android yüklü telefon ve tabletleri Gateway olarak kullanabilmek için cihazınızın aşağıda verilen iki gereksinimi karşılaması yeterlidir:

- Android 4 veya daha üstü bir sürüme,
- ve IoT-Ignite Agent yazılımına sahip olması gerekmektedir.

Pilaros

PilarOS, ARDIC'ın 1800'den fazla ek API ile geliştirilmiş endüstriyel Android sürümüdür. Android cihazları Gateway olarak kullanmayı sağlayan IoT-Ignite özellikle Android işletim sistemine büyük önem vermektedir. PilarOS'ta Android tabanlı olduğu için bu işletim sistemini Raspberry Pi 3 geliştirme kartına yükleyerek Gateway olarak kullanabiliriz.

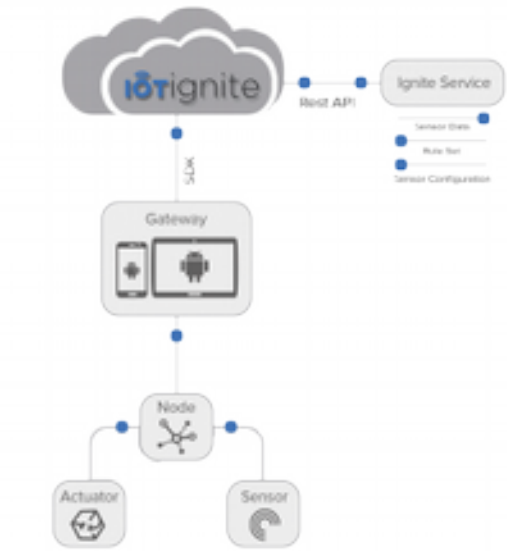
PilarOS işletim sistemini hem Linux hem de Windows ortamında imaj dosyasını indirerek Raspberry Pi 3 kartına yükleyebiliriz. Bunu kitabımızın 7. bölümünde ele alacağımız için sonraya bırakıyoruz.

MQTT ile Genel Kullanım

Linux'ta olduğu gibi IoT-Ignite platformu Windows işletim sistemini de desteklemektedir. Özellikle Windows platformunda IoT uygulamaları geliştirmek için MQTT protokolünü kullanabilirsiniz.

Neden Android Gateway ve Android Gateway'in Fog Computing'teki Kritik Önemi

İnternete bağlı cihaz sayısının çok fazla olduğu bir ekosistemde Android cihazların Gateway olarak kullanılması, Fog Computing için özellikle önem arz etmektedir. Çünkü Fog Computing platformunda Android telefon veya cihazların Gateway olarak kullanılması söz konusu değil. Bunun için farklı Gateway aygıtların ağa dahil edilerek IoT sistemini yönetmesi beklenir. Ancak bu sistem, ilerleyen zamanlarda internete bağlı cihaz sayısını daha da arttıracaktır. IoT-Ignite ise olaya daha farklı bir açıyla yaklaşmakta ve internete yeni bir cihaz eklemekten ziyade ağa bağlı olan cihazları Gateway olarak kullanarak cihaz artışını gerekmedikçe arttırmamaya gayret göstermektedir.



Android cihazların ucuz olması ve yaygın bir şekilde kullanılmasından dolayı internete bağlı cihaz sayısı en fazla olan aygıtlar şüphesiz Android yüklü cihazlardır. IoT-Ignite ağda mevcut olan bu Android cihazları Gateway olarak kullanarak bu cihazların ağ ve bulut sistemleri için bir engel değilde çözüm olarak kullanılmasını sağlamaktadır. Bu durum Fog Computing içinde çok önemlidir. Zira Gateway olarak ağa birden fazla farklı cihaz eklemek yerine mevcut olan cihazları ağ geçidi olarak kullanmak her zaman için çözüm odaklı bir yaklaşımdır.

Android'i Gateway olarak kullanmanın diğer faydalarını aşağıdaki gibi sıralayabiliriz:

- Dünya genelinde yaygın olması,
- Android Things ile IoT platformlara özel bir destek vermesi,
- Akıllı telefon, tablet, TV, otomobil gibi çok çeşitli cihazlarda çalışabilmesi,
- Linux tabanlı olmasından dolayı her platforma uygun olacak şekilde yeniden dizayn edilebilmesi,
- Android cihazların özellikle telefonların dahili sensörler ile gelmesi,
- Güçlü işlemci altyapısına sahip olması.

ARDIC IoT Ignite® Gateway

Internet of Things (IoT)

Nesnelerin İnterneti (IoT); genel olarak, sağlık, eğitim, perakende gibi farklı alanlara yönelik çözümlere ve hizmetlere erişim olanağı tanıyan, ortak bir ağ çatısı altında birlikte çalışan sensör, aktüatör, mobil cihaz ve diğer uç cihazları barındıran sistemler için kullanılır.

IoT Gateway

Sorunsuz bir IoT çözümü; farklı donanım ve işletim sistemleri, ağ iletişimleri ile iş akışlarını desteklerken kolay dağıtım, birlikte çalışabilirlik, uzaktan bakım için sahada çalışacak bilgisayarları barındırmalıdır. Sahada kullanılan bu bilgisayarlara "IoT Gateway" (IoT Ağ Geçidi) diyoruz. Doğru özellikleri ve yetenekleri olan bir IoT ağ geçidi, IoT servisleri için saha bilgisayarı olarak istenen tüm gereksinimleri karşılayabilir. Böyle bir ağ geçidinin özelliklerini aşağıdaki gibi özetleyebiliriz:

- **Ağ Bağlantı Desteği**: ZigBee, düşük enerjili Bluetooth, Ethernet, Wi-Fi gibi geniş yelpazedeki saha ağ protokollerini destekleyerek sahadaki tüm uç cihazlara İnternet bağlantısının sağlanması.
- **Yönetilebilirlik:** Merkezi bir erişim ve yönetim noktası sağlayarak uzaktan yönetimin sağlanması.
- **Güvenlik**: İstemciler ve servis sağlayıcılar arasında güvenli iletişim kanallarının ve veri depolama alanlarının sağlanması.
- **Donanım Envanter Takibi**: Sensör ve aktüatörlerin çoğu, çalıştıkları donanımı biricik olarak belirleyecek yetenekten yoksundur. Ağ geçidi böyle uç cihazlar için benzersiz bir donanım kimliği sağlayarak envanter takibinin ve sahada daha kolay donanım bakımının yapılabilmesini sağlar.
- **Kullanıcı Tanımlama**: Cihazları kullanıcılara bağlayarak bu özelliği yerine getirir.
- **Uygulama Çalışma Ortamı**: Çözüme özel uygulamalara IoT işlemlerine yönelik hazırlanmış bir çalışma ortamı sağlar.
- **IoT Desteği:** Bir ağ geçidinde GPS, sıcaklık, ivme sensörleri, ses kaydedici veya kamera gibi donanımlar bulunabilir. Bu donanımların Gateway'de olması sayesinde daha hızlı, güvenilir ve daha az maliyetle IoT çözümleri oluşturulabilir.

IoT Gateway için İşletim Sistemi Seçimi

IoT ağ geçidinin işletim sistemi seçiminde önemli faktörler şunlardır:

- Geliştirme, bakım ve işletme maliyeti.
- Yerel ve uzaktan yönetim.
- Ağ ve çevresel birim desteği.
- Güvenlik.
- Üçüncü parti uygulamalar için geliştirme, test ve çalıştırma ortamları desteği.
- Gömülü yazılımın kararlılığı.
- Uzak ve yerel yönetim konsolları için destek.

Akıllı telefonlarda ve tabletlerde en çok tercih edilen ve kullanıcı sayısı en yüksek mobil işletim sistemi olarak Android kendisini kanıtlamıştır. Kullanım alanı mobil cihazlarla kısıtlı olmayıp; otomobil, televizyon ve medya kutularında da pazar payı büyüktür. PC'lerde de Android çalıştırılabilir.

Android'e IoT Gateway işletim sistemi olarak avantaj getiren özellikler şöyle sıralanabilir:

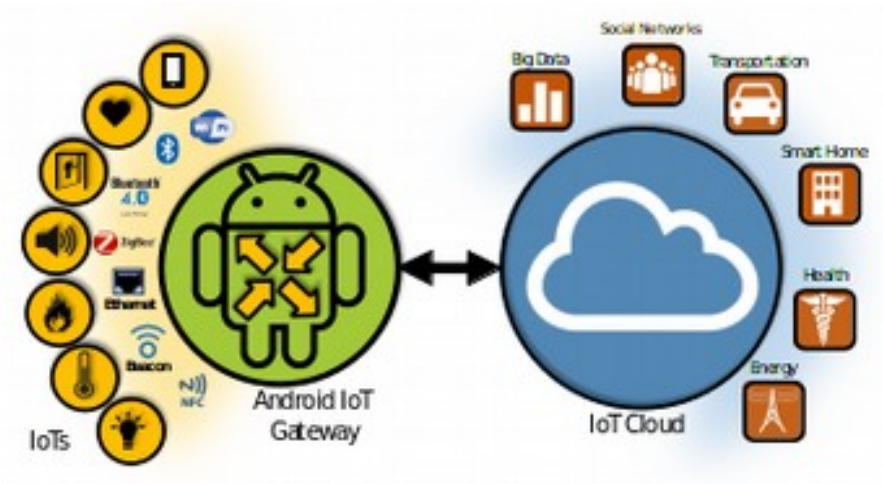
- Android farklı donanımlarda çalışabilir. Hem ARM hem Intel mimarileri üzerinde Android'i destekleyen mobil SoC'lar mevcuttur.
- İşletim sistemini etkilemeden üçüncü taraf uygulamaları çalıştırabilir.
- Açık kaynak koda sahip olduğundan çözüme yönelik değişikler kolayca yapılabilir.
- Mobil özellikler, kullanıcı deneyimi, güç yönetimi gibi özellikler işletim sistemiyle birlikte gelir.
- Kablosuz ve mobil bağlantıları destekleyen temel ağ geçidi işlevlerine sahiptir.

- NFS, beacon, BTLE (Bluetooth Low Energy) gibi en yeni teknolojileri de kapsayan ağ bağlantılarını destekler.
- Çeşitli ekranları ve giriş cihazlarını destekleyen pratik bir kullanıcı arayüzüne sahiptir.
- Kamera, ses kaydı, ivmeölçerler, GPS gibi donanım desteklerini içerir.
- İşletim sistemi, uygulama geliştirme ve desteği için birçok grup bulunmaktadır. Hatırı sayılır bir uygulama geliştirici topluluğu vardır.
- Linux çekirdeği üzerine kurulu olduğu için mevcut ve gelecekteki Linux uygulamaları için kolay entegrasyon sağlar.

Özelleştirme Desteği

Android, yeni USB adaptörlerini, kablosuz IoT ağlarını, yeni sensörler veya aktüatörler tanımak için özelleştirilebilir.

Özelleştirilmiş ana kullanıcı uygulaması (Launcher) ile kolay yerinde kurulum ve yapılandırma sunar.



Özelleştirilmiş IoT ajanı, IoT ve ağ geçidi iletişimini sağlar. Bu yazılım, bulut tabanlı IoT hizmetleri için güvenli bir iletişim kanalı oluşturur.

ARDIC’ın Uçtan Uca IoT Çözümü

ARDIC’in IoT Ignite® çözümü, IoT-Ignite Bulutu ile uyumlu çalışan özel Android cihazlarıyla oluşturulmuştur. IoT-Ignite Gateway içinde gömülü olarak bulunan IoT-Ignite ajanı, bulut tabanlı IoT hizmetleri için güvenli bir iletişim kanalı kurar ve mevcut IoT Ignite® servislerini Gateway’lere bağlanan uç noktalara kadar ulaştırır.

IoT-Ignite servisleriyle birlikte Gateway’in ana avantajlarını şöyle sıralayabiliriz:

- Uç cihazlar ve üçüncü parti uygulamalar ile kusursuz entegrasyon sağlar.
- Bulutta yer alan uygulama ve içerik mağazaları aracılığıyla farklı çözümlere özel uygulama ve içerik dağıtabilir.
- Uç cihazlar için kablosuz, Bluetooth, BLE ve TCP / IP iletişim desteği sağlar.
- Güvenli iletişim kanallarını kullanır.
- Uç cihazlardan veri toplama, depolama ve işleme temel işlevleri arasındadır.
- Karmaşık olay işleme ve işleme desteği vardır.
- Kolay yazılım ve konfigürasyon dağıtımı, bakım (uygulama ve servis güncellemesi, OTA (Over the Air Update) güncellemesi, log toplama), yönetimi yapılabilir.
- Verilerin toplanması, hesaplanması, analizi, güvenli olarak bulut sistemlere aktarır.
- Cihazları izleme ve yönetimi için uzaktan yönetim konsolları mevcuttur.
- İşletme sahipleri, hizmet sağlayıcılar ve uygulama geliştiricileri için entegre edilmiş bir kullanıcı platformu mevcuttur.

Pilaros ve AFEX

Pilaros ARDIC’ın lisanslı ürünüdür. Pilaros içinde AFEX ve IoT-Ignite bulut ajan yazılımı bulunmaktadır. IoT ve kurumsal çözümlerin kullanılması için, uzaktan cihaz provizyonu ve yönetilebilirliği, uygulama ve veri güvenliği ile üçüncü parti uygulamalarla entegrasyon düzeyi arttırılmış bir Android sürümü olan Pilaros işletim sistemi kullanılmaktadır.

AFEX (Android Framework Extensions), işletim sisteminin güvenliğini ve yönetilirliğini arttıran çok sayıda API'yi (Application Programming Interface) ve servisi içinde barındırmaktadır.

IoT-Ignite ajanı, üçüncü parti uygulamalara IoT veri modelleme, iletme, saklama desteği ile geniş yelpazede ağ (TCP/IP, UDP, Bluetooth vb.) ve iletişim (MQTT, Native Android vb.) kanalı sunmaktadır.

ARDIC'ın ECP (Edge Computing Platform) Çözümleri

ARDIC'ın ECP'ye yönelik çözümler için Pilaros yüklü ARDIC gateway'leri kullanılmaktadır.

ECP'nin kullanıldığı Gateway çeşitleri şöyle listelenebilir:

- **IoT-Ignite Gateway**: IoT çözümleri için iç ve dış mekan ile fabrika ortamlarında kullanılacak Gateway'ler.
- **Android-Pilaros Box**: Digital Signage ya da medya oynatıcı kutu olarak Gateway'lerin kullanılması sağlanmaktadır.
- **İnce Sunucu (Thin Client)**: Web tarayıcı, doküman işleme, uzak terminal bağlantıları, medya oynatma için PC yerine kullanılabilen bilgisayar, klavye, fare ve monitörden oluşan gateway çözümüdür.

Marketteki Tipik IoT Platformları ve Edge Computing ile IoT-Ignite'ın Karşılaştırılması

Markette mevcut IoT platformları oldukça çeşitlilik göstermektedir. Bu platformlar bulut tabanlı IoT olarak tanımlanmaktadır.

Genel olarak piyasada bulunan IoT platformları üç grup altında incelenebilir:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

Şimdi bunları kısaca açıklayıp marketteki uygulamalarını inceleyelim...

Software as a Service (SaaS)

SaaS IoT platformu ile geliştirilen bulut sistemlerinde yazılım veya uygulamalar birer hizmet olarak geliştirilir ve kullanıcıların kullanımına sunulur. En büyük avantajı; farklı yerlerden farklı

cihazlardan kullanıcının verilere erişmesini sağlaması ve bulutta bulunan yazılımları kullanarak dosyalarını düzenlemesi veya yeni veriler oluşturmasıdır.

Bu kategoriye verilecek örnek platformlar şunlardır:

- Microsoft Office 365
- Google Apps
- Amazon Web Services
- Dropbox

Platform as a Service (PaaS)

PaaS modelinde firma veya şirket tarafından geliştirilen bulut platformu, basit bulut uygulamalarından kurumsal bulut uygulamalarında kadar ihtiyacınız olan tüm kaynakların yer aldığı, geliştirme ve dağıtıma yönelik eksiksiz bir bulu ortamı sağlar. Kuruluşların geneli PaaS modelini geliştirme çerçevesi, analiz veya iş zekası, iş akışı, dizin, güvenlik ve zamanlama gibi senaryoları gerçekleştirmek için kullanır.

Buraya verilecek en iyi örnek şüphesiz IoT-Ignite platformudur. Bu platform bir PaaS girişimi olarak geliştirilmiş olup farklı yazılım donanım desteği ile IoT uygulamalar geliştirmemiz sağlamaktadır.

IoT-Ignite, bir PaaS girişimi olarak aşağıdaki avantajlara sahiptir:

- Geliştirme ve kodlama süresini düşürmesi,
- Geliştirme özelliklerini arttırması,
- Mobil platform için daha kolay geliştirme yapmayı sağlaması,
- Uygun maliyetler ile gelişmiş IoT uygulamaları geliştirmeyi sağlaması,
- Uygulama yaşam döngüsünü etkin şekilde yönetebilmesi gibi.

IoT-Ignite dışında diğer PaaS tabanlı platformlar ise aşağıdaki gibidir:

- Windows Azure
- IBM Smart Cloud
- OpenShift
- Cloud Foundry
- Google App Engine
- Cloud Bees

Infrastructure as a Service (IaaS)

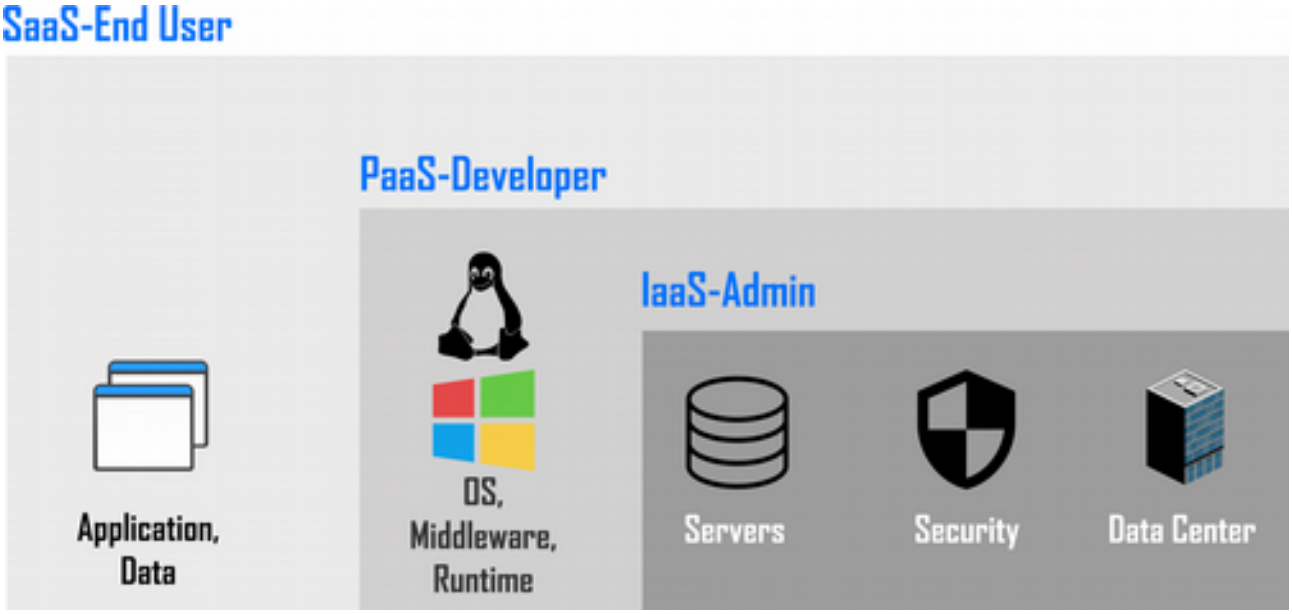
IaaS, IoT platformu gerçek zamanlı bilgi işlem altyapısı olarak tanımlanabilir. İnternet üzerinden sağlanıp yine buradan yönetilmektedir. Kendi fiziksel sunucularınızı ve diğer veri merkezi alt yapısını satın almanın ve yönetmenin getirdiği karmaşa ve maliyetten kurtulmak için IaaS platformunu kullanabilirsiniz. Bu sistemlerde her kaynak ayrı bir hizmet olarak sunulur, böylece yalnızca ihtiyacınız olan hizmeti ihtiyacınız kadar kiralayıp maliyetleri düşürebilirsiniz.

IoT-Ignite platformu, IaaS platformunun tüm özelliklerini üzerinde barındırmaktadır. Çünkü IaaS ile verilen hizmetler PaaS platformu ile de sağlanabilmektedir. Örneğin, IoT-Ignite platformu ile 5 Gateway'e cihazı ücretsiz olarak kayıt edebilirsiniz. Daha fazlası için bu sayıdan sonra ücret ödersiniz.

Piyasada bulunan IaaS uygulamaları şunlardır:

- IoT-Ignite
- Amazon Web Service
- Microsoft Azure
- Cisco Metapod

Şimdi bu platformlar arasındaki ilişkiyi bir şema üzerinde gösterelim.



Şekli incelediğimiz zaman IaaS'ın sistemi kuran firma veya şirkete yakın olduğunu ve burada bulunan hizmetlerin firma tarafından geliştirilip son kullanıcılar ve geliştiriciler tarafından kullanıldığı görülmektedir.

PaaS platformu ise, özellikle geliştiricilere hitap etmektedir. Bu modelde nihayetinde son kullancılara hizmet edecek şekilde ürün geliştirme yapmaktadır. Bunlar uygulama veya veri sağlama şeklinde olmaktadır.

SaaS platformu ise son kullanıcılara hizmet veren ve desteğini PaaS ve IaaS platformlarından olan bir IoT platformudur.

IoT platformları hakkında bilmemiz gerekenleri sizlerle paylaştıktan sonra, son olarak Edge Computing ve IoT-Ignite karşılaştırmasına geçebiliriz.

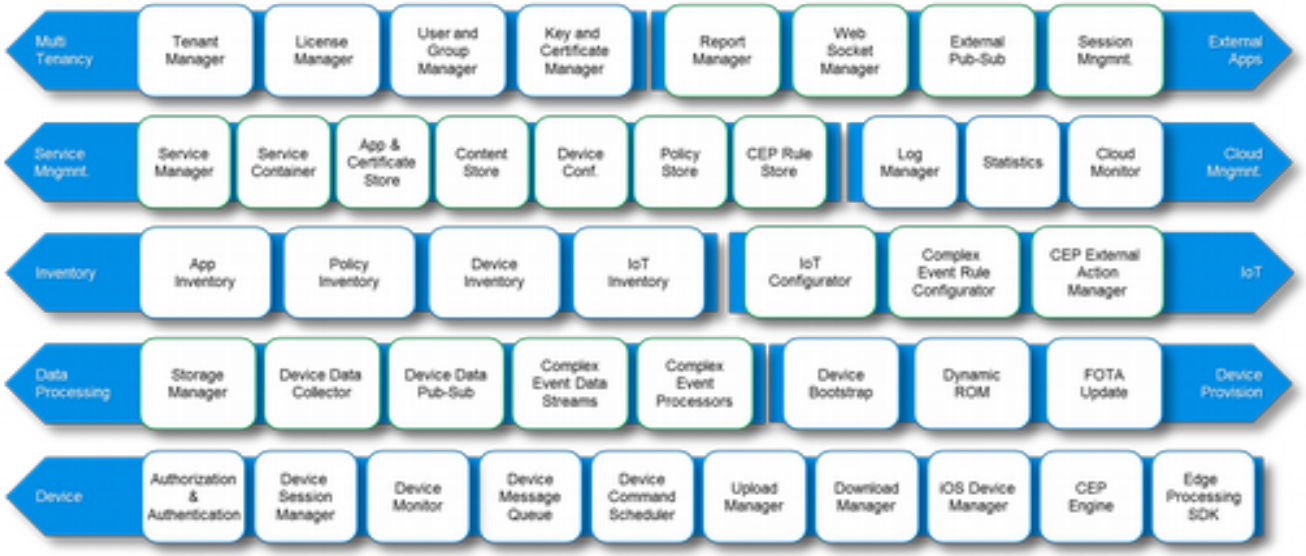
Her şeyden önce IoT-Ignite, Edge Computing'i benimsemiş bir platformdur. **Hatta Edge'in uygulamaya geçmiş hali IoT-Ignite'tır diyebiliriz.** Her iki platformun amacı bulut sistemlerinin büyük verilerle karşı karşıya kalarak kilitlenmesini engellemek ve işlemleri gerçek zamanlı olarak gerçekleştirmektir. Ayrıca Gateway kavramı her ikisinde de bulunmaktadır. Ancak IoT-Ignite Gateway olarak Android cihazlara ayrı bir önem vererek Edge'den biraz sıyrılmaktadır. Edge'de Android'e doğrudan bir destek bulunmamakla birlikte, genel olarak Gateway terimi kullanılmaktadır.

BÖLÜM 5

IoT-Ignite Cloud Platformu ve EHUB Enterprise ile Fonksiyonel Bileşenlerin Yönetimi

IoT-Ignite Cloud Servis Bileşenleri

IoT-Ignite Cloud katmanını daha önceki bölümlerimizde incelemiş, IoT-Ignite API'ları ile nasıl çalışabileceğimizi de görmüştük. Artık IoT-Ignite'a aşina olduğumuza göre Cloud katmanında genel olarak hangi bileşenlerle hizmetlerin verildiğini inceleyebiliriz.



Yukarıdaki şekli incelediğinizde, bileşenler gruplandırılarak gösterilmiştir. Şimdi bu bileşenleri alt katmandan üst katmana olacak şekilde inceleyelim... Detaylarını yine ilerleyen bölümlerde yer yer göreceğiz.

Gateway (Ağ Geçidi)

En alt katmanda bulunan Device, Gateway olarak tanımlanmaktadır. Kullanmış olduğunuz Gateway'de aşağıdaki bileşenler yer almaktadır.

- **Authorization & Authentication (Yetkilendirme ve Kimlik Doğrulaması):** Servis hesabının oturumu başlatmasıdır. Her bir Gateway, bir son kullanıcıya bağlı olmaktadır ve Gateway'lerin Ignite Cloud ile etkileşime geçebilmesi için oturumun başlatılmış olması gerekir.
- **Device Session Manager (Cihaz Oturum Yönetimi)**: Başlatılmış olan oturumun aktif olarak açık kalmasıdır.
- **Device Monitor (Cihaz İzleme)**: Gateway'lerin gerçek zamanlı olarak durumlarının izlenebilmesidir. Mesela; pil, konum, hafıza gibi...
- **Device Message Queue (Cihaz İleti Sırası):** Gateway'e bağlı olan Node ve sensörlerden gelen verilerin Fog Computing sürecinden geçmesinden sonra, yığın verinin Ignite Cloud'a veri kaybı yaşanmadan iletilmesidir.
- **Device Command Scheduler (Cihaz Komut Zamanlayıcısı)**: Gateway'de bazı komutların anlık, bazı komutların da belirlenen tarihlerde zamanlanmış olarak çalıştırılmasıdır.
- **Upload Manager (Yükleme Yöneticisi):** Gateway'lerde kullanılması için uygulama veya dosyaların yüklenip bir bulut sisteminde (Store) tutulması, uygulamaların Policy'lere eklenmesidir.
- **Download Manager (İndirme Yöneticisi):** Store'da bulunan uygulama veya dosyaların Gateway'lere indirilebilmesidir.

- **iOS Device Manager (iOS Cihaz Yöneticisi):** MDM (Mobile Device Management) tarafında iOS cihazlara da verilen destek.
- **CEP Engine (Karmaşık Olay İşleme Motoru)**: Kural editörleri ile karmaşık olayları basitçe şematize edilip kuralların Gateway'de çalıştırılmasıdır.
- **Edge Processing SDK (Edge Gateway İşleme SDK'sı)**: Gateway'leri tam anlamıyla kontrol edebilmek ve yönetebilmek için kullanılan SDK.

Veri İşleme (Data Processing)

Fog Computing'in odak noktası Gateway'de veri işleme sürecidir. IoT-Ignite servislerinin de temel mimarisini oluşturan Fog Computing, Gateway'e bağlı olan Node ve Sensor'lerden gelen verileri doğrudan buluta iletmez. Daha önce de görmüş olduğumuz Gateway Kuralları ile Gateway tarafında çevrimdışı olarak birtakım işlemlerin gerçekleştirilmesi için veri işlemenin Gateway tarafında yapılması gerekmektedir. Ayrıca her okunan veri anlamlı olmayabilir, parazit olabilir. Bu gibi durumlar için verilerin Gateway'de filtrelenmesi, düzenlenmesi ve anlamlı bir şekilde buluta iletilmesi gerekmektedir. Böylece ağ trafiği daha performanslı kullanılırken, aynı zamanda veri işleme yükü tamamen buluta değil, Gateway'lere dağıtık bir şekildedir. Bu da büyük ölçüde performans, kritik anlarda da Gateway'in anlık kararlar verebilmesini sağlamaktadır.

- **Storage Manager (Depolama Yönetimi)**: Gateway'de depolama yönetimi ile her bir veri saklanabilir ve Ignite Cloud'tan her bir veri istenildiğinde Gateway'den alınabilir.
- **Device Data Collector (Gateway Veri Toplayıcı):** Gateway'de toplanan veriler istenilen sürede depolanabilir, istenilen zamanlarda yığın veri halinde buluta gönderilebilir veya veriler gerçek zamanlı olarak buluta yönlendirilebilir. Gateway'de meydana gelen her olay için log kayıtları tutulmaktadır.
- **Device Data Pub-Sub (Gateway Verisi Yayınlama/Toplama)**: Gateway'den gelen veriler doğrudan Ignite Cloud'a gider. Ancak PubSub ile bu veriler 3. parti servislere de yönlendirilebilir.

Gateway Hakimiyeti (Device Provision)

- **Device Bootstrap** (Cihaz Önyükleme): PilarOS yüklü Gateway'ler (ürünler), ilk başlatıldığı andan itibaren IoT-Ignite ile etkileşime geçerek lisans almak için hazır beklemektedir.
- **Dynamic ROM** (DROM): ID'leri bilinen ve içinde IoT-Ignite Agent yüklü Gateway'lerin, internete bağlandıkları anda otomatik olarak lisans alması, bunun neticesinde de Mode ve Policy ayarlarının Gateway'lere gönderilmesi sağlanır.
- **FOTA (Firmware Over The Air) Update** (Havadan Yazılım Güncelleme): Lisanslı olan Gateway'lerin tekli veya çoklu olarak gömülü yazılımlarının güncellenmesi, yeniden kurulması IoT-Ignite ile çok basittir. Gateway'lere istenilen işletim sistemi versiyonu indirilir ve kurulum otomatik olarak gerçekleştirilir.

Envanter (Inventory)

- **App Inventory** (Uygulama Envanteri): Serviste Gateway'lerde kullanılmak üzere APK formatlı uygulamalar bir uygulama mağazasına yüklenebilir, daha sonra güncellenebilir veya silinebilir. İstenildiğinde Gateway'lere Mode ile gönderilebilir.
- **Policy Inventory** (Politika Envanteri): Mode'lar, bir veya birden fazla Policy barındırabilir. Policy'ler, oluşturulduğunda envanterde tutulur. Sonrasında her bir Policy yeniden düzenlenebilir, silinebilir veya bir Mode içine alınıp Gateway'lere gönderilebilir.

- **Device Inventory** (Gateway Envanteri): Lisanslı her bir Gateway envanterde gösterilir ve her biriyle veya gruplar halinde yönetilebilmesi mümkündür.
- **IoT Inventory** (IoT Envanteri): IoT envanteri; servisinizde yer alan bütün yapılandırma ayarlarını tutmaktadır. Tek bir servis ile çalışabileceğiniz gibi farklı Gateway'ler için farklı Mode'lar ile farklı IoT yapılandırmaları uygulayabilirsiniz.

Servis Yönetimi (Service Management)

- **Service Manager** (Servis Yönetimi): IoT-Ignite'ın Cloud ve Gateway yönetimi için güçlü ve kullanımı kolay servis yönetim araçları bulunmaktadır. Bu araçlar ile servisinizi uçtan uca yönetebilmeniz sağlanmaktadır.
- **App & Certificate Store** (Uygulama & Sertifika Deposu): Servis yönetiminde Gateway'lerde kullanılacak olan uygulamalar ve bu uygulamalara ait sertifikaların yönetimi için güçlü bir uygulama mağazası ve sertifika yönetim aracı yer almaktadır.
- **Content Store** (İçerik Deposu): Mode kullanımı ile birlikte, Mode içinde çeşitli dosya formatları ile içerikleri Gateway'lere aktarabilmeniz mümkündür. Bu içeriklerin yönetiminde ise bir içerik mağazası IoT-Ignite servisinde yer almaktadır.
- **Device Configurations** (Aygıt Yapılandırmaları): Gateway'lerin yapılandırılması IoT-Ignite yönetim panelleri ile rahatlıkla yapılabilmekte, güncellenebilmektedir.
- **Policy Store** (Politika Deposu): Gateway'in kendi kendisini kontrol edebilmesi için bir takım politikaların kullanılması için bir veya birden fazla politikanın yönetilebilmesi gerekmektedir. Policy Store, çoklu politika ile çalışılabilmesine ve Gateway'lerin de birden fazla politika alabilmesine, istenilen şartlarda istenilen politikaya geçiş yapılabilmesinde rol oynar.
- **CEP Rule Store** (Karmaşık Olay İşlemcisi Kural Deposu): Policy içinde yer alan CEP Rule (Complex Event Processing, Karmaşık Olay Yönetimi Kuralı) ile sensörlerden gelen verilere göre Gateway'in hangi davranışı sergileyeceği veya hangi akışları kontrol edeceği belirlenebilir. Bir veya birden fazla, Cloud (Online) veya Gateway'de (Offline) çalışacak kurallar bir editör ile oluşturulabilir. CEP Rule Store'da bulunan bu kurallar gerektiği hallerde ilgili Mode'ların içine aktarılabilir, aktif veya pasif yapılabilir.

Bulut Yönetimi (Cloud Management)

- **Log Manager** (Log Yönetimi): Gateway ve Cloud tarafında gerçekleşen her işlem kayıt altına alınmaktadır ve yönetilebilmektedir.
- **Statistics** (İstatistikler): Gateway / sensörlere ait bütün veriler kayıpsız olarak saklanmaktadır. Bu veriler çeşitli grafik araçları ile yorumlanabilmekte ve analiz edilebilmektedir.
- **Cloud Monitor** (Bulut İzleme): Çeşitli Dashboard'lar ile kişiselleştirilebilir bir yapı ile Gateway'ler ve sensör verileri çeşitli grafik araçları ile görüntülenebilmekte, anlık olarak takip edilebilmektedir.

Çoklu Müşteri Hesabı (Multi Tenancy)

- **Tenant Manager** (Müşteri Yönetimi): IoT-Ignite, çoklu müşteri hesaplarının yönetimini destekler, müşteri hesaplarının birbirinden izole olarak yönetimini sağlar.

- **License Manager** (Lisans Yönetimi): Her servisin lisans sayısı özelleştirilebilir ve hesapta yer alan lisansların hangi Gateway'ler ile eşleştiği, ne kadarının kullanıldığı, ne kadar kaldığı gibi yönetimler rahatlıkla yapılabilir.
- **User And Group Manager** (Kullanıcı ve Grup Yönetimi): Gateway lisanslarının eşleştiği End User (diğer ismi ile Gateway User)'ların yönetimi ve bu End User'ların gruplanmasıyla da toplu olarak yönetimler rahatlıkla sağlanabilmektedir.
- **Key and Certificate Manager** (Anahtar ve Sertifika Yönetimi): Sertifika yönetimi ile AFEX, Root sertifikası, kullanıcı uygulamaları sertifikaları veya uygulama imzaları serviste tutulabilir.

Harici Uygulamalar (External Apps)

- **Report Manager** (Rapor Yönetimi): Ignite Cloud'a servis bilgileri veya sensörlerden gönderilen her bir veri kayıt altına alınır. İstenildiği anda detaylı rapor üretilebilir, analiz yapılabilir.
- **WebSocket Manager** (WebSocket Yönetimi): WebSocket yönetimi sayesinde her bir Gateway'in her bir Sensör verisi anlık olarak dinlenebilir.
- **External PubSub** (Harici Publish/Subscribe): IoT-Ignite Cloud, 3. parti servisler veya diğer Cloud sistemleri ile senkronize bir şekilde çalışabilmekte, Pub/Sub sistemi ile de kendi geliştireceğiniz sistemlere entegrasyonu sağlanabilmektedir.
- **Session Management** (Oturumu Yönetimi): IoT-Ignite Cloud, aynı anda yüzbinlerce Gateway'in yetkilendirilmesini sağlayabilir, oturumunu açabilir ve oturum yönetimini kesintisiz olarak aktif tutabilir.

IoT-Ignite Cloud'un Diğer Güçlü Özellikleri

Yukarıda saymış olduğumuz IoT-Ignite'ın fonksiyonel bileşenlerinin dışında, mimarisinde genel olarak aşağıdaki özellikleri de bulunmaktadır.

- **Micro Servis Mimarisi:** IoT-Ignite'ta her bir servis dağıtık bir şekilde mimari edilmiştir. Böylece servislerin kullanımında az kaynak tüketimi sağlanmakta, daha performanslı olmaktadır.
- **Çoklu Hesap Yönetimi:** IoT-Ignite, çoklu müşteri hesaplarını yönetmeyi destekler ve her bir müşteri izoledir. Yani satın alınan bir servis altında, servis sahibi tarafından birden fazla alt müşteriye hizmet verilebilir. Hiçbir müşteri hesabı diğerinde yer alan verileri göremez, erişemez.
- **Kesintisiz Servis:** IoT-Ignite servisleri kesintisiz olarak çalışmaktadır.
- **Yüksek Ölçeklenebilirlik**: IoT-Ignite Cloud, değişen ihtiyaçlara göre ölçeklenebilir ve genişletilebilir bir yapıdadır.
- **Analitik & Büyük Veri:** Gateway'lerden IoT-Ignite'a gelen her veri saklanmaktadır. Veriler belirli periyotlara ve filtrelemelere göre grafiklere dökülerek analiz edilebilmektedir. Ayrıca bu veriler çeşitli Büyük Veri işlemlerine de entegre edilebilir ve daha gelişmiş analiz ve yorumlamalar için kullanılabilir.

EHUB Enterprise

EHUB, IoT-Ignite platformunda servisinizi uçtan uca yönetebileceğiniz bir arayüz sağlar. Kullanımı kolay, kullanıcı dostu ve güçlü yapısı ile servisinizi yönetebilmek adına bütün işlemleri rahatlıkla yapabilirsiniz. Örneğin;

- Birden fazla müşteriye, onlarca, yüzlerce, binlerce alt hesap ile hizmet verebilirsiniz.
- Gateway'leri gruplandırarak çoklu cihaz yönetimleri sağlayabilirsiniz.
- Gateway'leri lisanslayabilir veya lisansları silebilirsiniz. Lisanslanma detaylarını görebilirsiniz.
- Android Gateway'ler dışında MQTT Gateway'ler de oluşturup, bunlara Node ve Sensor bağlayabilirsiniz.
- Varsayılan Gateway yapılandırmaları (APN, VPN, Wifi, Hotspot, Email) yapabilirsiniz.
- Çoklu modlar oluşturup her bir modu dilediğiniz gibi yapılandırabilirsiniz (default policy, policies, apps, files, services, app certificates, sensor specific data configurations, gateway rules, configurations).
- Yeni politikalar (app, system, firmware, wifi, peripheral, browser, trust flow, cellular data, black permissions) oluşturup yapılandırabilirsiniz.
- Sensör veri yapılandırmaları oluşturabilir, bunları saklayabilir, daha sonra da istediğiniz sensörde kullanabilirsiniz.
- Sensör türü tanımlayabilirsiniz.
- Gateway ve Cloud kuralları oluşturabilirsiniz.
- Zamanlanmış görevler oluşturabilirsiniz.
- App Store kategorileri oluşturup, her bir kategoriye uygulamalar yükleyebilirsiniz.
- App'ler için güvenlik sertifikaları yükleyebilirsiniz.
- File Store kategorileri oluşturup, her bir kategoriye dosyalar yükleyebilirsiniz.
- Kullanıcı gruplarını (System User, Gateway User, Admin Areas, Working Group) yönetebilirsiniz.
- Log kayıtlarını ve raporları inceleyebilirsiniz.
- DROM yapılandırmaları yapabilirsiniz. Böylece Gateway'leri uzaktan lisanslayabilirsiniz.
- Zamanlanmış komutlar ve görevler oluşturabilirsiniz.
- App Key ve EPR yapılandırmalarınızı düzenleyebilirsiniz.

Bunlar, EHUB ile yapabileceklerinizin sadece başlıcaları... Şimdi bunları kısaca inceleyelim. Detaylı kullanımlarını da ilerleyen bölümlerde uygulama geliştirirken göreceksiniz.

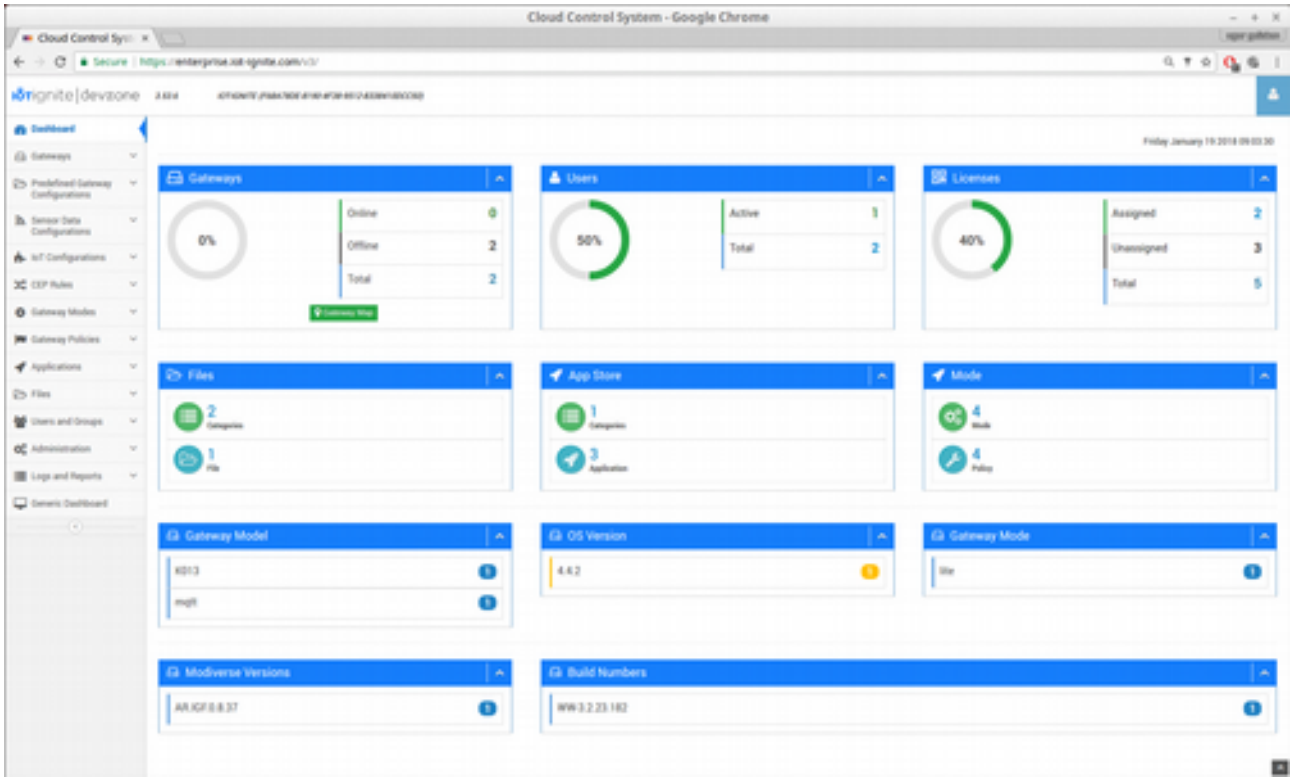
> Daha önceki bölümlerimizde yer yer EHUB Enterprise'a geçiş yapıp incelemelerde bulunarak bazı işlemlerimizi gerçekleştirmiştik. Konu tekrarı olmaması amacıyla, daha önce gördüğümüz kısımlar üzerinde fazla durulmayacaktır. Yine bazı işlemler de ilerleyen konularla entegreli olduğu için, konu içerisinde daha detaylıca incelenecektir.

Dashboard’un İncelenmesi ve Yorumlanması

EHUB Enterprise’a aşağıdaki adresten giriş yapabilirsiniz.

https://enterprise.iot-ignite.com/v3/

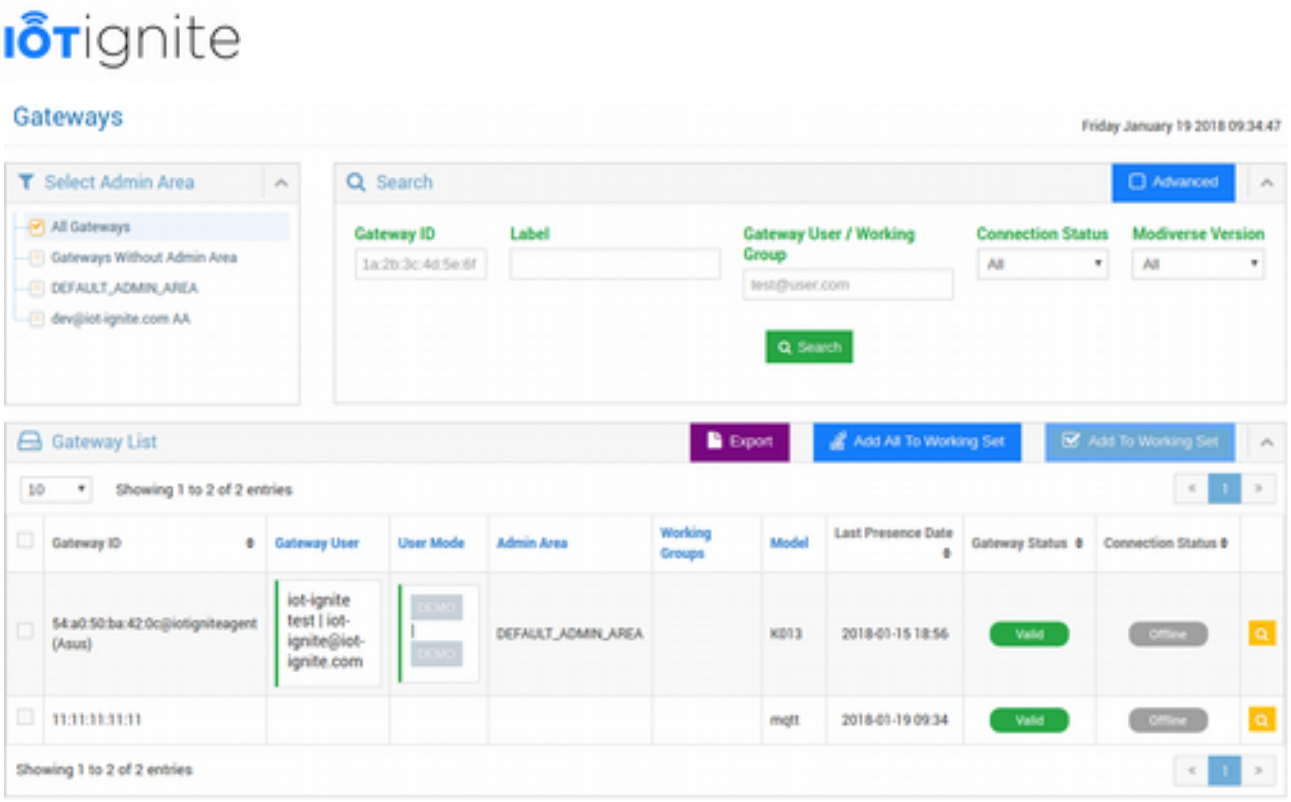
Giriş yaptığınızda, aşağıdaki gibi bir Dashboard ile karşılaşacaksınız.



Dashboard’ta sol tarafta menü yer almaktadır. Menüden istediğiniz araca kolaylıkla erişebilirsiniz. Ekranda yer alan kutucuklarda ise servisinize ait bir takım veriler özet olarak gösterilmektedir. Bunları sırasıyla inceleyelim...

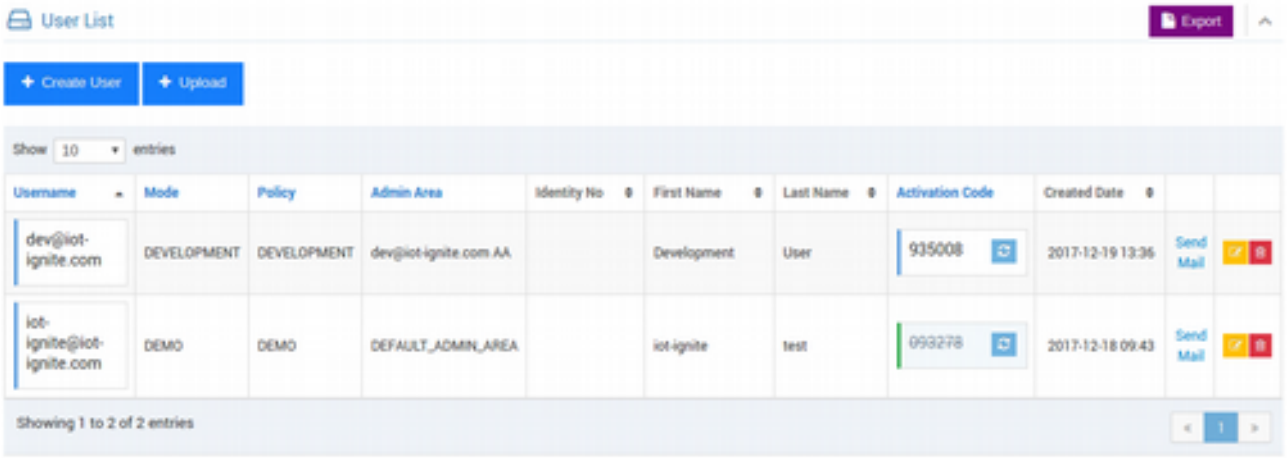
Gateways

Lisanslı olan Gateway’lerin toplam sayısı, online olanları sayısı ve offline olanların sayısını gösterir. Burada yer alan sayılara tıklandığında filtreleme yapılarak ilgili Gateway’lerin listelendiği sayfaya yönlendirilir.



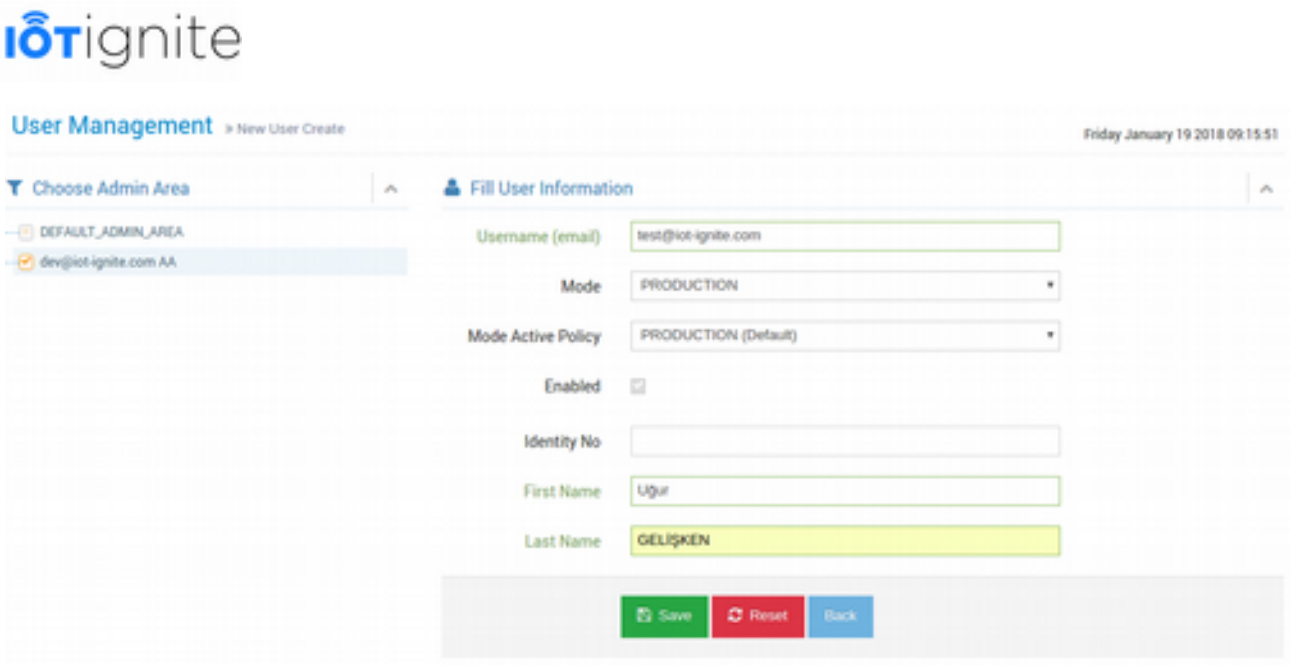
Users

Serviste tanımlı olan **End User**'ları listeler. **Users And Groups** altında yer alır. Devzone'dan üye olunduğunda 2 adet End User oluşturulur. Bunlar **dev@{email}.com** ve Devzone üyeliği açarken kullandığınız **email** ile oluşturulmuş End User hesaplarıdır. Bu hesaplardan dev@{email}.com olana **DEVELOPMENT** yaparken kullandığınız Gateway'ler lisanslanmaktadır. Üyelik hesabınız ile oluşturulan End User'a da DEMO **senaryosunda** lisansladığınız Android Gateway lisanslanır.

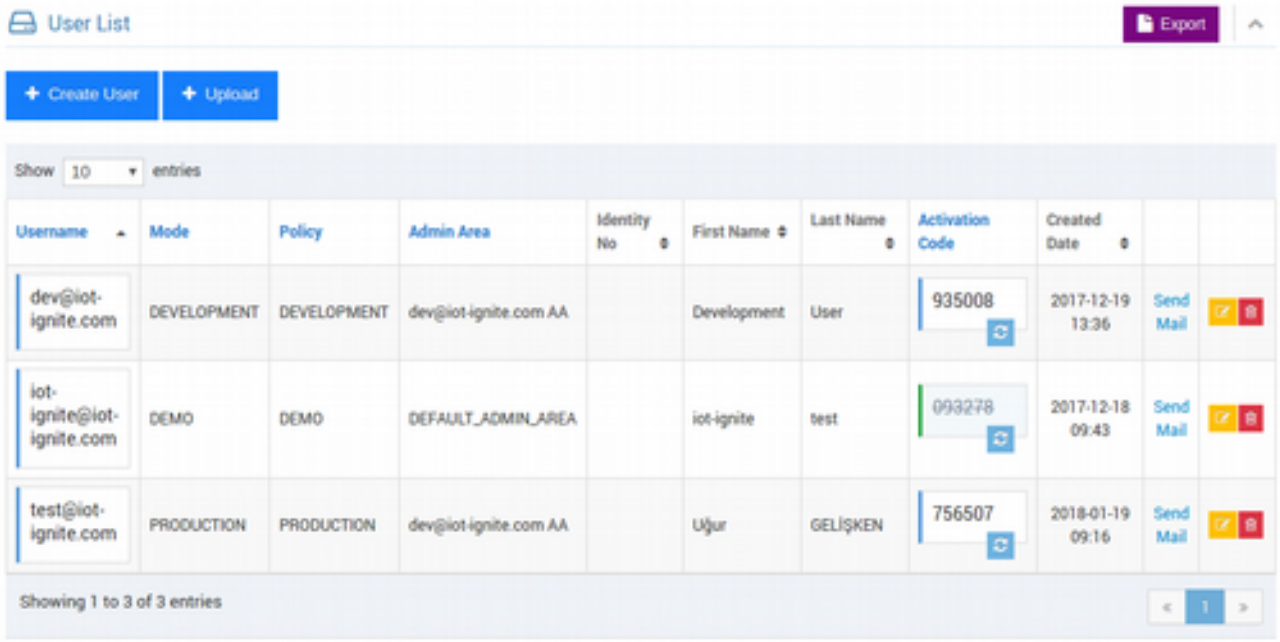


Eğer isterseniz bu sayfada **Create User** (Kullanıcı Oluştur) butonuna tıklayarak yeni bir End User oluşturabilirsiniz. User oluştururken bu End User'ın **Mode**'unu ve **Aktif Policy**'sini de seçebilirsiniz. Örnek olması amacı ile bir End User oluşturalım.

Aşağıdaki gibi formu doldurun ve sol tarafta yer alan **Admin Area** (Yönetici Bölgesi) alanından da bu End User'ın hangi Admin Area'ya bağlı olacağını seçin.



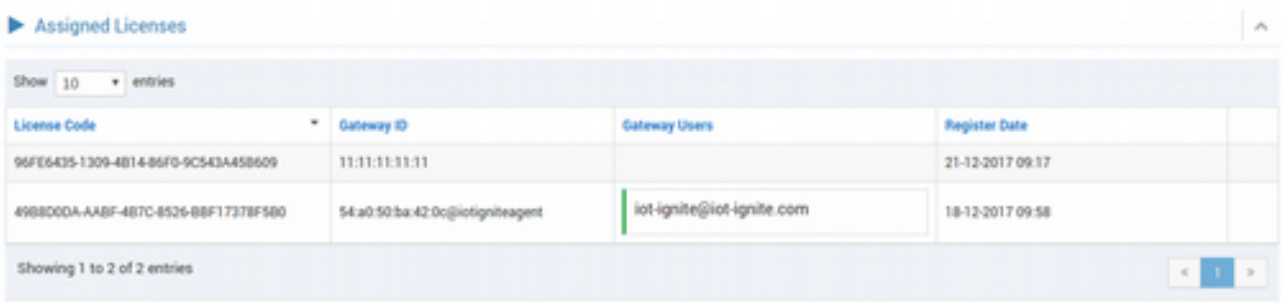
İşlem sonunda yeni bir End User oluşturmuş olacaksınız. Menüden **Users and Groups** (Kullanıcılar ve Gruplar) altında **Gateway Users** (Gateway Kullanıcıları) butonunu tıkladığınızda End User'ları göreceksiniz (End User ile Gateway User aynıdır).



Tanımladığınız bu yeni End User'a herhangi bir Gateway lisanslamak için de menüden **Administration** (Yönetim) > **Register Gateway** (Gateway Lisansla) altından **Generic Register** (Genel Lisanslama) seçip lisanslama yapabilirsiniz. Açılan sayfadaki listeden Gateway User listesinden bu oluşturduğumuz yeni End User'ı seçtiğimizde, hemen altında bir **QR Code** çıkacaktır. Bu QR Code ile daha önce Devzone'da gördüğümüz şekilde lisanslama yapabiliriz.

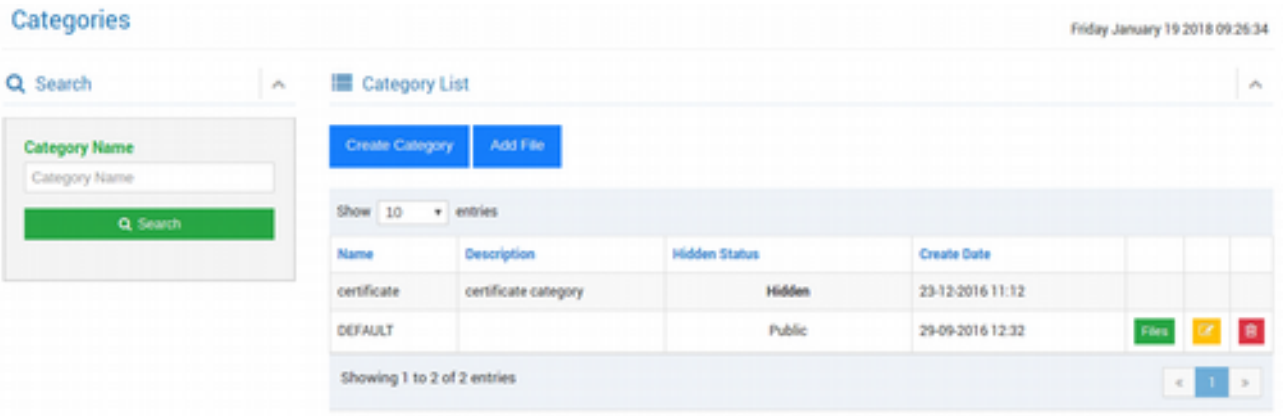
Licenses

Servisinize tanımlı olan toplam lisans sayısını, kullanılmış ve kullanılmamış lisansların sayısını görebilirsiniz. **Assigned** (Atanan) sayısına tıkladığınızda, kullanılmış olan lisanslarla ilgili detayları göreceğiniz sayfaya yönleneceksiniz.

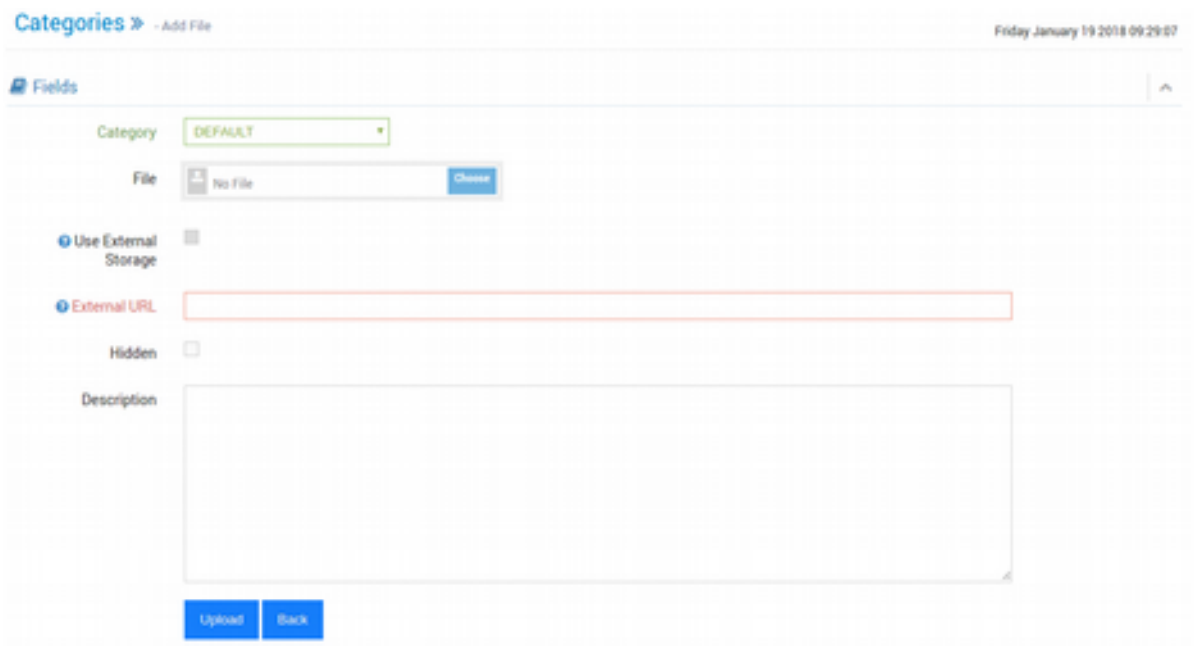


Files

File Store (Dosya Mağazası)'nda yer alan dosyaların kategori sayısını ve toplamda da ne kadar dosya olduğunun sayısını görebilirsiniz. Kategori, varsayılan olarak 2 tanedir. Bunlardan biri **certificate** (sertifika dosyalarının olduğu kategori) ve diğeri de genel kullanıma açık (public) olan **DEFAULT** kategorisidir.



Kategori listesinden **Create Category** (Kategori Oluştur) butonu ile yeni bir kategori oluşturabilir, **Add File** (Dosya Ekle) ile de certificate kategorisi dışındaki diğer kategorilere dosyalar yükleyebilirsiniz.

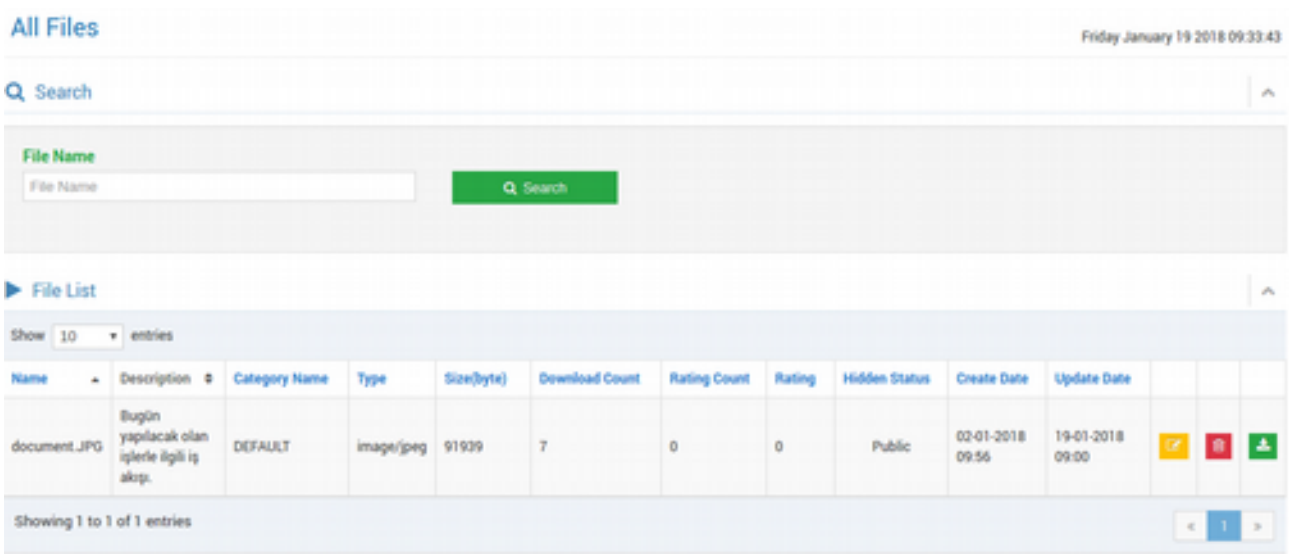


Dosya yükleme formunda ilk olarak dosyanın hangi kategori altına yükleneceği seçilir. Sonrasında da dosya seçimi yapılır.

Use External Storage (Dış Depolama Alanını Kullan) opsiyonu seçilirse, bu dosya API Konfigurasyon Ayarları'nda belirtilen URL'de depolanacaktır. Use External Storage yazısının solunda yer alan ? İşareti üzerine fare ile geldiğinizde ayarlara erişebileceğiniz bağlantı açılacaktır (Eğer kendi sunucunuzdan uygulamanın indirilmesini istiyorsanız; açılan ayarlar sayfasında Dış Depolama Alanı seçeneği seçip URL alanında da yolu tanımlayabilirsiniz. Bu depolama alanı dosya eklerken isteğe göre kullanılabilir, zorunlu değildir.). Eğer herhangi bir depolama işlemi yapmadan dosya indirilmesini istiyorsanız, **External URL** (Dış URL) alanında dosyanın bulunduğu depolama alanındaki tam adresini belirtebilirsiniz.

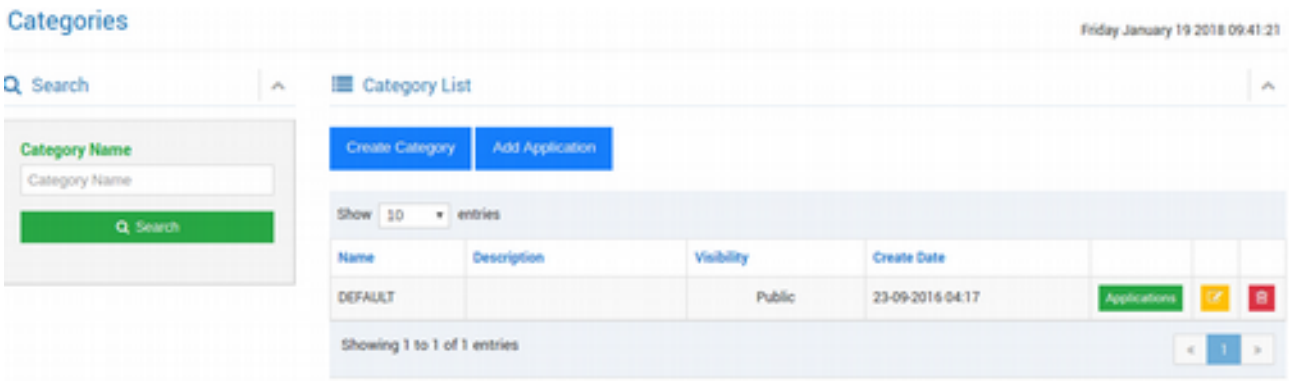
Dosyanın gizli olması için **Hidden** opsiyonu seçilebilir. Dosya hakkında tanımlayıcı bilgi belirtmek için de **Description** (Özet) alanına özet girilebilir.

Dosya kategorilerini ve var olan dosyaları görmek için menüden **Files** (Dosyalar) altında **Categories** (Kategorileri)'i veya **All Files** (Bütün Dosyalar)'ı tıklayıp dosyaları listeleyebilir, inceleyebilir, düzenleyebilir veya silebilirsiniz.

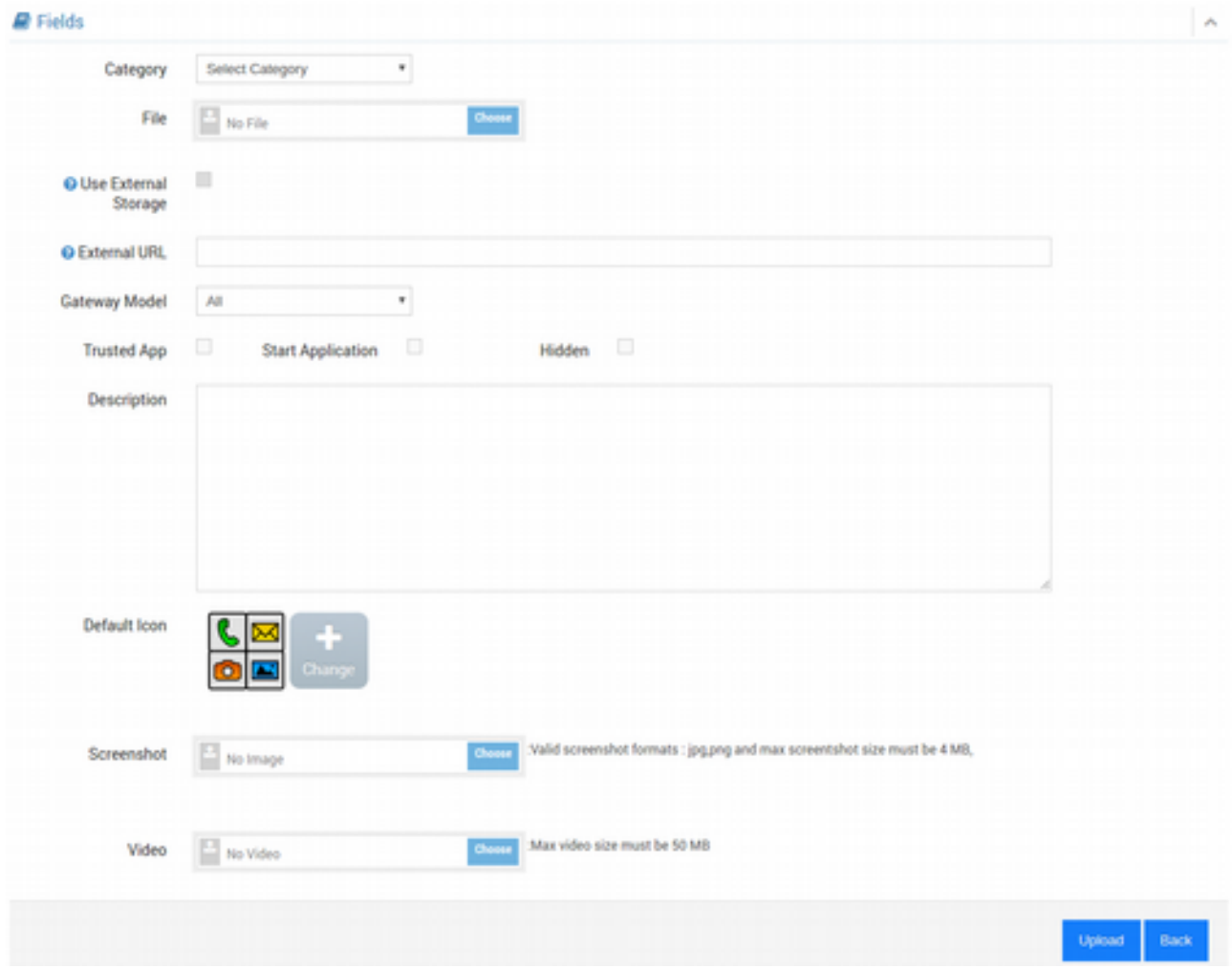


App Store

Servisinizde yer alan uygulamaların kategorisi ve uygulamaların toplam sayısını görebilirsiniz. Menüden **Applications** (Uygulamalar) altında yer almaktadır. Varsayılan olarak 1 adet **DEFAULT** ismi ile kategori bulunmaktadır.



Create Category (Kategori Oluştur) butonuna tıklayarak kategori için bir isim ve açıklama yazarak kolayca yeni bir uygulama kategorisi oluşturabilirsiniz. **Add Application** (Uygulama Ekle) butonuna tıklayarak da açılan formu doldurup servisinize bir uygulama ekleyebilirsiniz. Daha sonra bu uygulamaları Mod içine ekleyip Gateway'lere yükleyebilirsiniz.

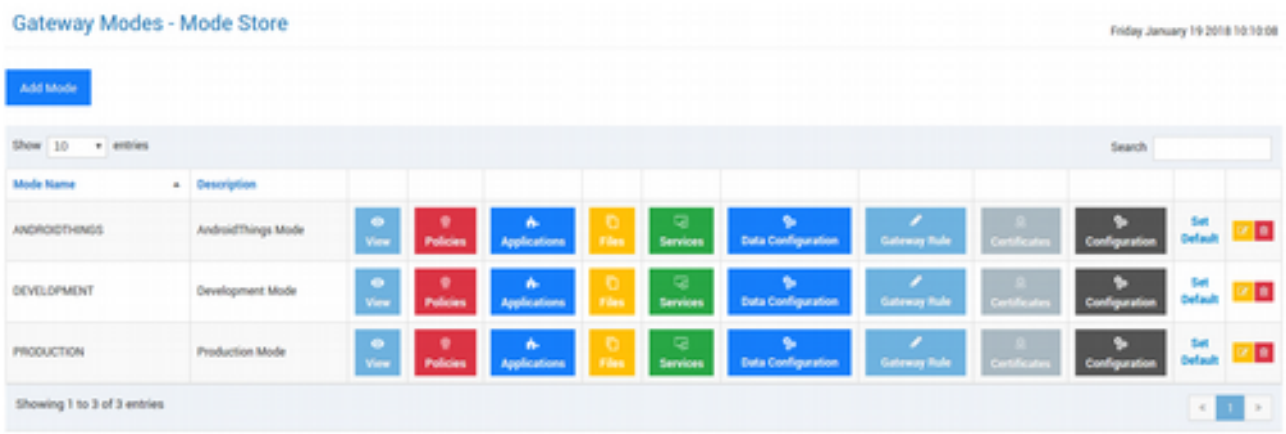


Açılan formda ilk olarak uygulamanın hangi kategori altına yükleneceği seçilmelidir. Sonrasında dosya seçimi yapılır ve diğer opsiyonları da doldurulur.

- **Gateway Model**: Uygulamanın hangi Gateway modellerinde yükleneceğini belirtir.
- **Trusted App**: Sertifika gerektiren uygulamalar için .apk içindeki sertifikanın otomatik olarak servise eklenmesidir. Sertifikanın da mutlaka App ile birlikte Mode'a eklenmesi gerekmektedir. Trusted App olarak işaretlenmiş bir uygulama, gateway üzerinde IoT-Ignite'ın sunduğu API'leri kullanabilir.
- **Start Application**: Uygulamanın yüklendikten sonra otomatik olarak başlatılmasıdır.
- **Default Icon**: Uygulama listesinde görüntülenecek icon.
- **Screenshot**: Uygulama için bir JGP veya PNG formatlarında ekran görüntüsü eklenebilir.
- **Video**: Uygulama için tanıtıcı bir video eklenebilir.

Mode ve Policy

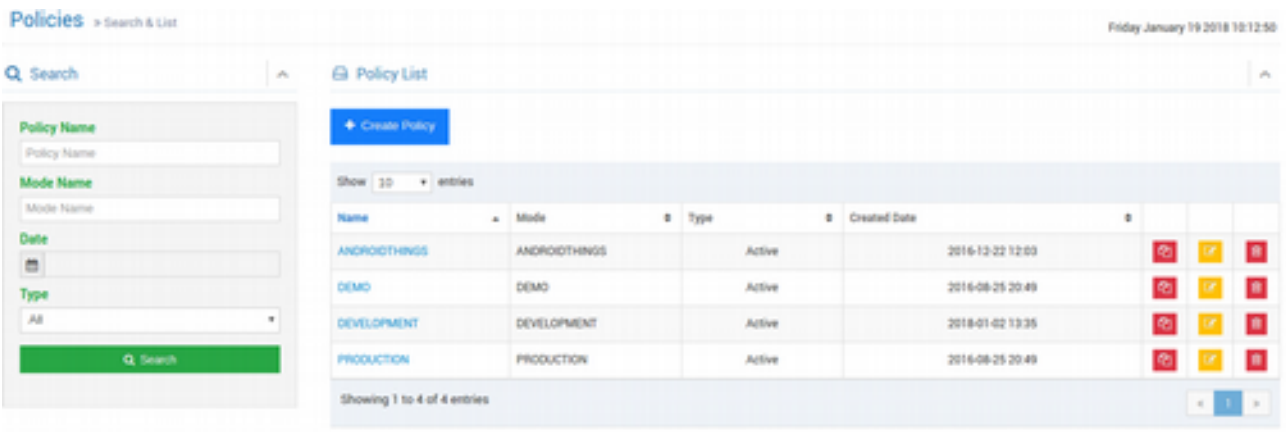
Serviste tanımlı olan Mode ve Policy'leri listeler. Policy'lere menüden **Gateway Policies**, Mode'lara da **Gateway Modes** altından erişebilirsiniz. Varsayılan olarak 4'er adet bulunmaktadır. Aşağıda, Mode'lar yer almaktadır.



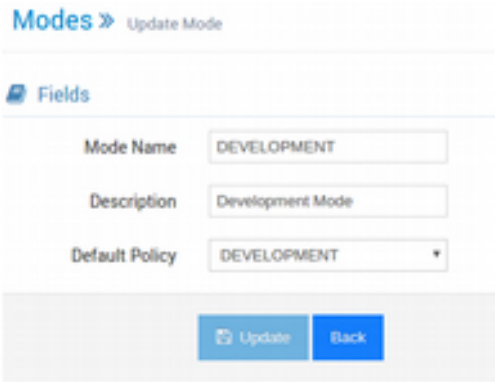
Add Mode (Mod Ekle) butonuna tıklanarak yeni bir Mode oluşturulabilir. Bir Mode oluşturabilmek için de mutlaka o Mode içinde bir Policy olmak zorundadır.

Dashboard'tan Policy sayısına tıklayıp **Policy Store** (Politika Deposu)'a erişebilirsiniz.

Menüden **Gateway Policies** (Gateway Politikaları) > **Policy Store**'a tıklayarak da erişebilirsiniz.

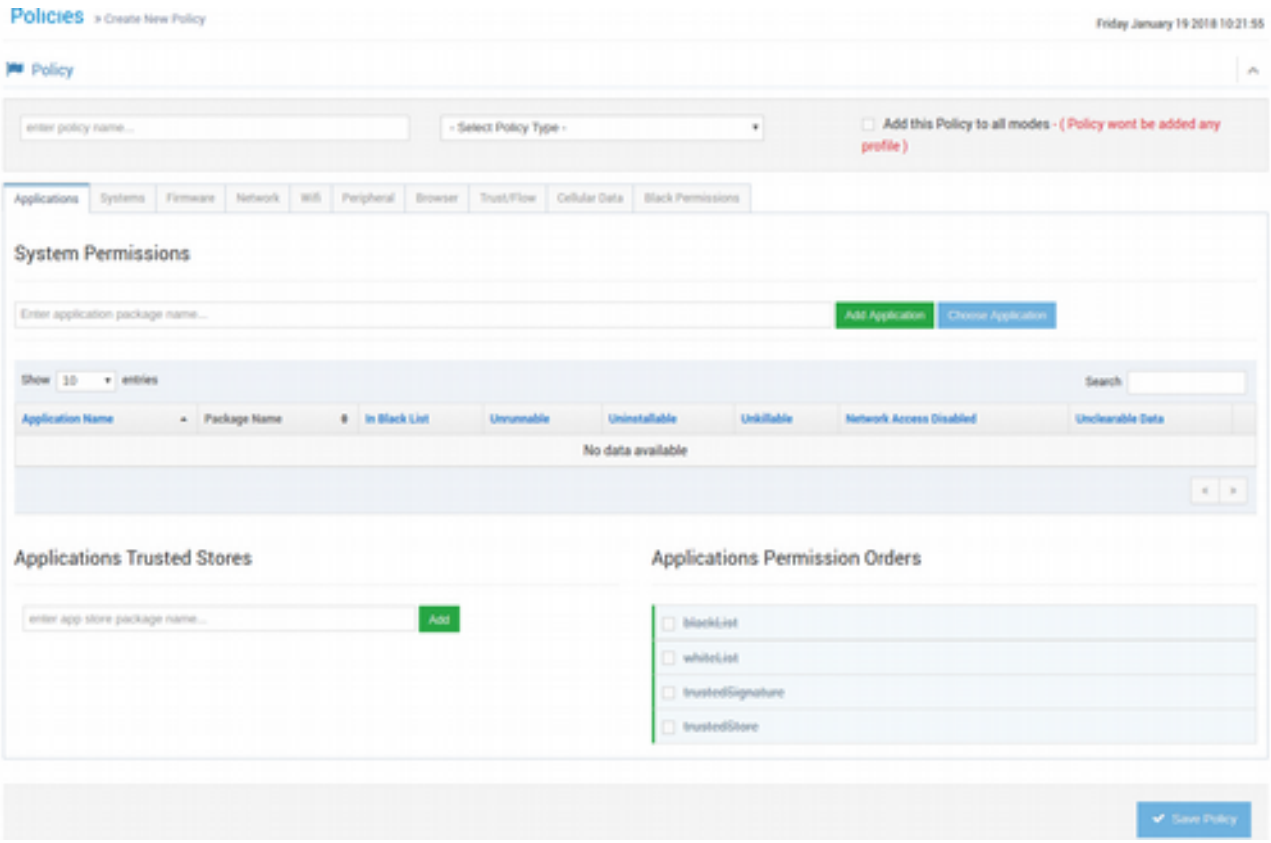


Bu listede var olan Policy'leri görebilirsiniz. İsterseniz detaylarını inceleyebilir, ister düzenleyebilir isterseniz de silebilirsiniz. Ancak bir Policy'i silmeniz, ona bağlı olan Mode'ları da etkileyecekir. Bu nedenle eğer bir Policy bir Mode'un default'u ise, o Policy silinemez. Mesela DEVELOPMENT Mode'unun Policy'si DEVELOPMENT Policy'dir. Bunu da menüden **Gateway Modes** (Gateway Modları) > **Mode Store** (Mod Deposu) altında yer alan Mode'ların herhangi birinin detayına baktığımızda görebiliriz. **Detay** (listede sağ taraftaki sarı ikon) ikonuna basıp DEVELOPMENT Mode'unun detayına bakalım...



Açılan ayarlarda DEVELOPMENT Mode'nun **Default Policy** (Varsayılan Politika) ayarının DEVELOPMENT Policy'si olduğu görülmektedir.

Policy Store'da **Create Policy** (Politika Oluştur) butonuna tıklayarak yeni bir Policy oluşturabilirsiniz. Yukarıdaki örnekte aynı isim ile Mode ve Policy görülmektedir. IoT-Ignite'a yeni bir üyelik başlatıldığında varsayılan olarak bir Mode ve aynı isimle bir Policy oluşturulur, Mode'a eklenir. Ancak isterseniz bu Policy'nin ismini sonradan değiştirebilirsiniz. Veya yeni bir Policy oluşturulduktan sonra Mode'a ekleyip, yukarıdaki şekilde gösterilen Default Policy seçimi ile Mode'un varsayılanı olarak ayarlayabilirsiniz.



Görüldüğü üzere Policy oluştururken oldukça fazla yapılandırma ayarı yapılabilmektedir. İsterseniz daha önce oluşturulmuş olan Policy'lerin detaylarını inceleyip, nasıl yapılandırma yapabileceğiniz hakkında fikir edinebilirsiniz.

Yeni bir Policy oluştururken yapabileceğiniz ayarlamaları kısaca inceleyelim...

Enter policy name...: Policy için bir isim belirtilir. Örneğimizde isim olarak POLICY_1 adında bir Policy oluşturalım.

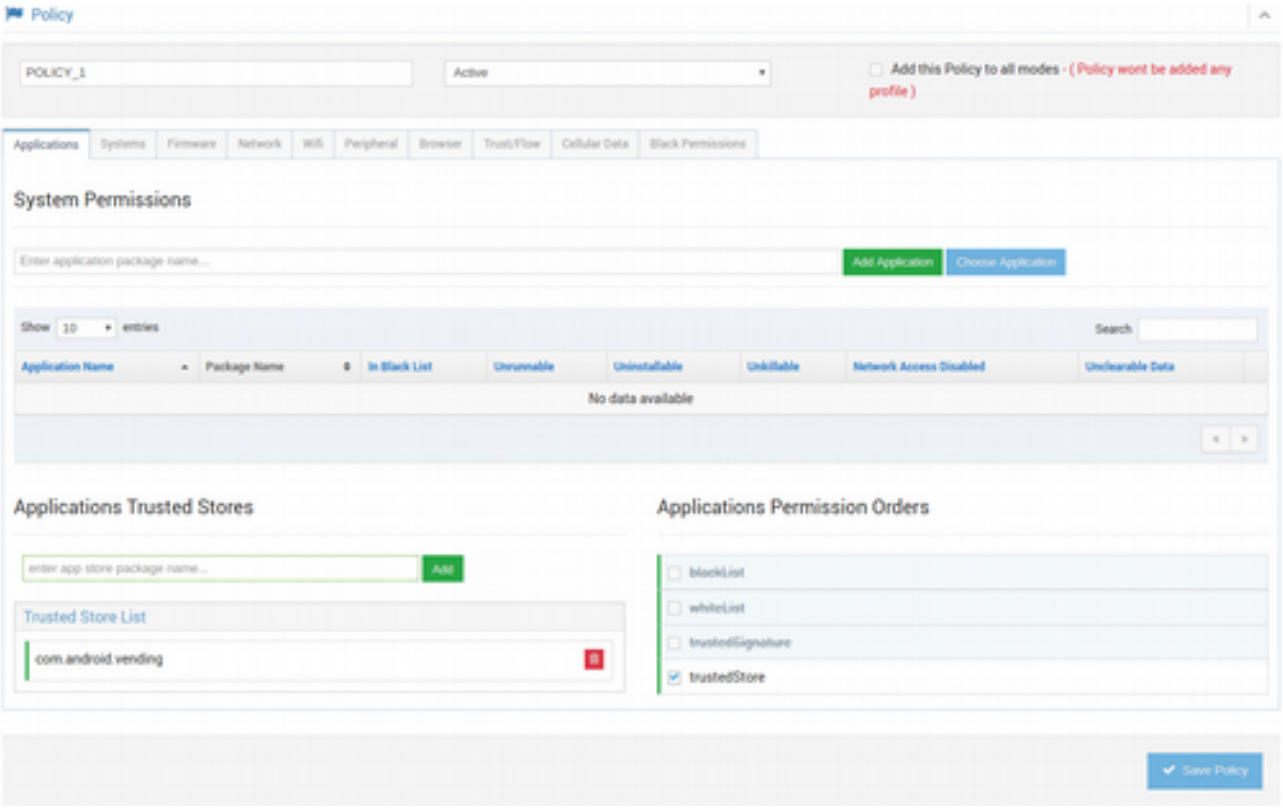
Select Policy Type: Hazırlayacağınız Policy için bir tür belirtmeniz gerekir. Eğer hemen aktif olarak kullanmak istiyorsanız **Active** (Aktif), eğer geçici olarak pasif şablon halinde kalmasını istiyorsanız (sonradan tekrar aktif yapılabilir), **Temp** (Geçici) türü seçilir. Biz Active'i seçiyoruz.

Add this Policy to all modes: Policy'i oluşturduğunuzda, bu Policy'nin bütün Mode'ların içine eklenmesini istiyorsanız bu opsiyonu seçebilirsiniz.

Application Trusted Stores: Uygulamaların indirilebilmesi için kullanılan App Store uygulamalarının (Mesela Google Play) hangisinin güvenilir olmasını istiyorsanız, daha doğrusu App'lerin hangi kaynaklardan indirilmesine izin veriyorsanız bu alanda o App Store uygulamasının paket ismi girilir. Mesela sadece Google Play'den uygulama indirilsin isteniyorsa;

com.android.vending paket ismi girilir ve Add butonu ile eklenir. Bu alanda yapılan işlemler zorunlu değildir.

Applications Permission Orders: Uygulama izin sırası anlamına gelir. Blacklist, Whitelist, Trusted Signature ve Trusted Store opsiyonları vardır. Her birini tutup sürükleyerek kendi aralarında sıralayabilirsiniz. Böylece izin sıralaması yapabilirsiniz (Eğer biri diğerini eziyorsa, sıralama önemlidir). Az önce Google Play'in paket ismini eklemiştik, bu bir Trusted Store olmaktadır. Bu nedenle yapılan işlemin aktif olması için Trusted Store check'lenerek aktif edilmelidir. Bu alanda yapılan işlemler zorunlu değildir.



Policy oluştururken diğer detaylı ayarlamalar da sekmelerde yer alan **Applications** (Uygulamalar), **Systems** (Sistemler), **Firmware**, **Network** (Ağ), **Wifi, Peripheral** (Çevresel Birim), **Browser** (Tarayıcı), **Trust/Flow**, **Cellular Data** (Hücresel Veri) ve **Black Permissions** (Engellenen Yetkiler)'tır. Bunların bir kısmını Devzone'da **GATEWAY SERVICES** sayfasında **Network Config** yapılandırmalarında yapmıştık, hatırlayın. Burada ise daha detaylı ve daha gelişmiş opsiyonlarla ayarlamalar yapılabilmektedir.

- **Applications**

Mode'u alan Gateway'de bir takım uygulamaların bazı özelliklerini kısıtlamak isteyebilirsiniz. İster **Choose Application** (Uygulama Seç) butonuna tıklayıp listeden uygulama seçip, isterseniz de **Enter application package name** (Uygulama paket ismini gir) alanına o uygulamanın paket ismini girerek uygulamayı tanıtın. Örneğimizde Youtube'u ekledik. Uygulama eklediğimizde, hemen listeye o uygulama alınacak ve bu uygulama için de bir takım opsiyonlar belirecektir. Varsayılan olarak hepsi **False** olarak tanımlıdır. Bu opsiyonlar şunlardır:

In Black List: Eğer True yapılırsa, uygulamayı kara listeye alır. Uygulamayı göstermez. Bu opsiyonun çalışabilmesi için Applications Permission Orders'ta Black List'in aktif olması gerekir.

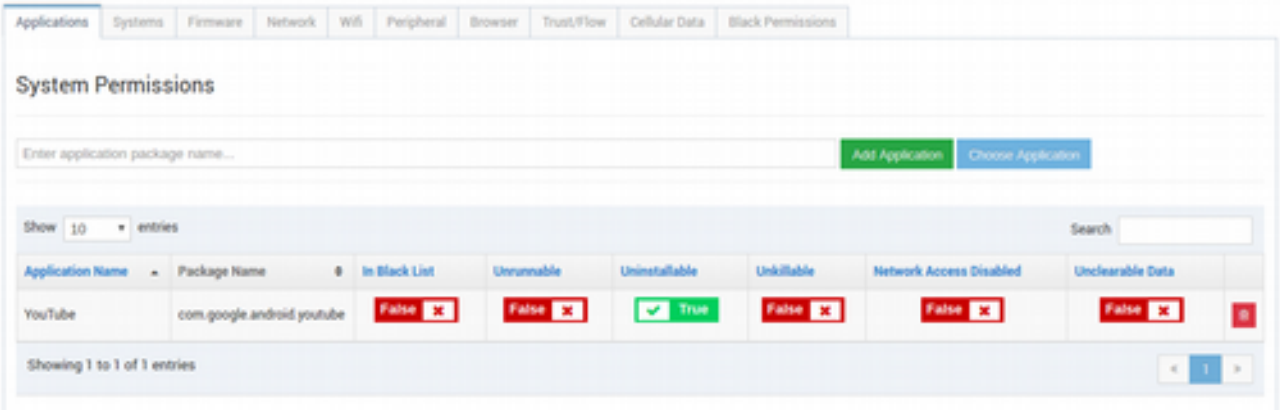
Unrunnable: Eğer True yapılırsa, uygulamayı gösterir ancak çalıştırılmasına izin vermez.

Uninstallable: Eğer True yapılırsa, bu paket ismi ile uygulamanın kaldırılmasına izin vermez.

Unkillable: Eğer True yapılırsa, uygulamanın kapatılabilmesine izin vermez.

Network Access Disabled: Eğer True yapılırsa, ağ erişimi kapatılır.

Unclearable Data: Eğer True yapılırsa, uygulama verilerinin silinebilmesine izin vermez.

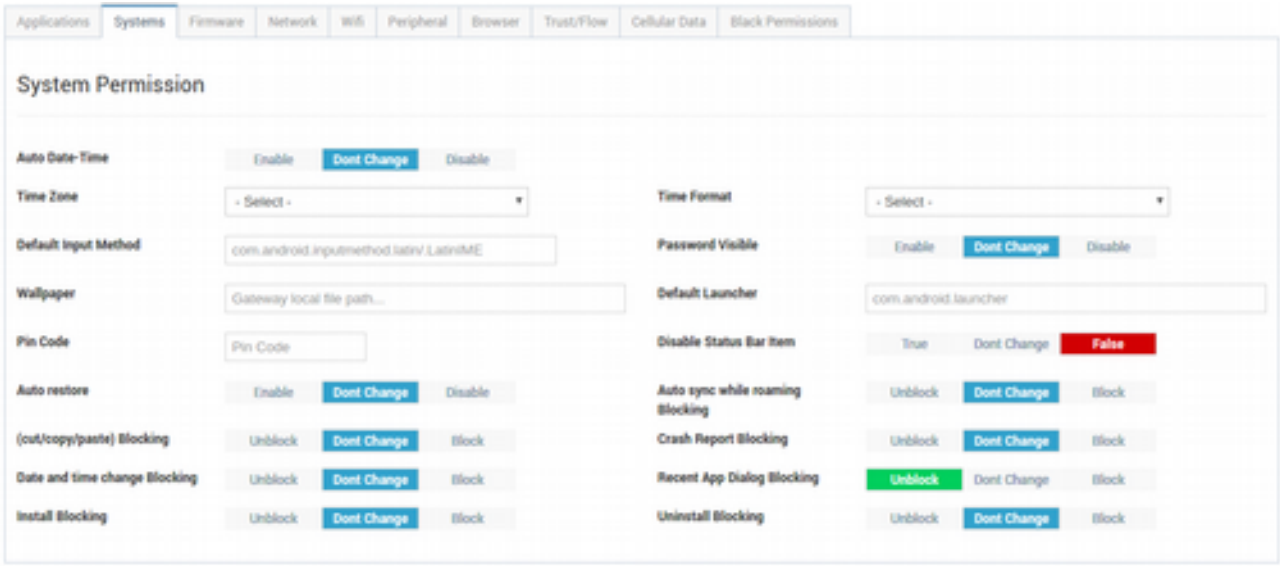


Örneğimizde Youtube uygulamasının kaldırılamaması için bir zorlama yaptık.

Ayrıca bu listede yer alan uygulamalar Gateway tarafında görüntülenecek, listede olmayanlar görüntülenmeyecektir.

- **Systems**

Bir takım sistem ayarlarını yapabileceğiniz opsiyonlar bulunmaktadır. Mesela cihazın tarihi otomatik güncelleyip güncellemeyeceği, zaman dilimi, arkaplan resmi, PIN kodu, uygulama yükleme izinleri, Status Bar'ın gizlenip gösterilmesi gibi ayarlamalar yapılabilmektedir.

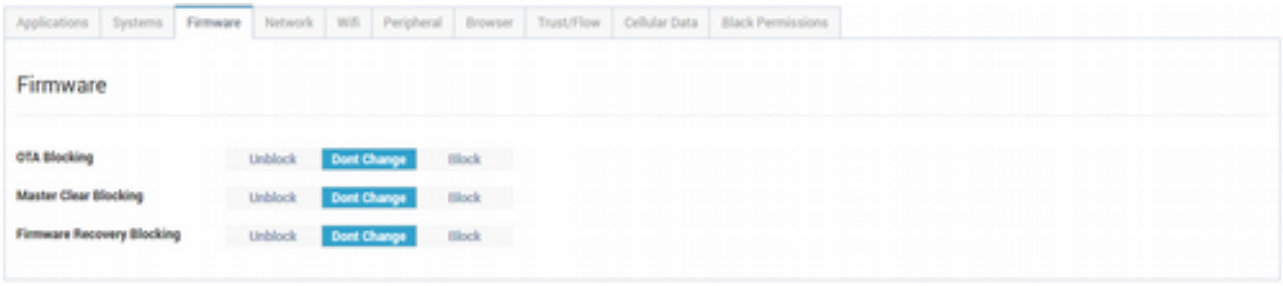


> **Enable**: Opsiyonun aktif olacağını,
>
> **Disable**: Opsiyonun pasif olacağını,
>
> **Unblock**: Opsiyonun kilidinin kaldırılacağını,
>
> **Block**: Opsiyonun kilitleneceğini,

> **True**: İzin verilmesini,
>
> **False**: İzin verilmemesini,
>
> **Don’t Change**: Opsiyon için herhangi bir seçim yapılmaması, Gateway’de ne karar verilmişse onun uygulanması gerektiğini belirtir.

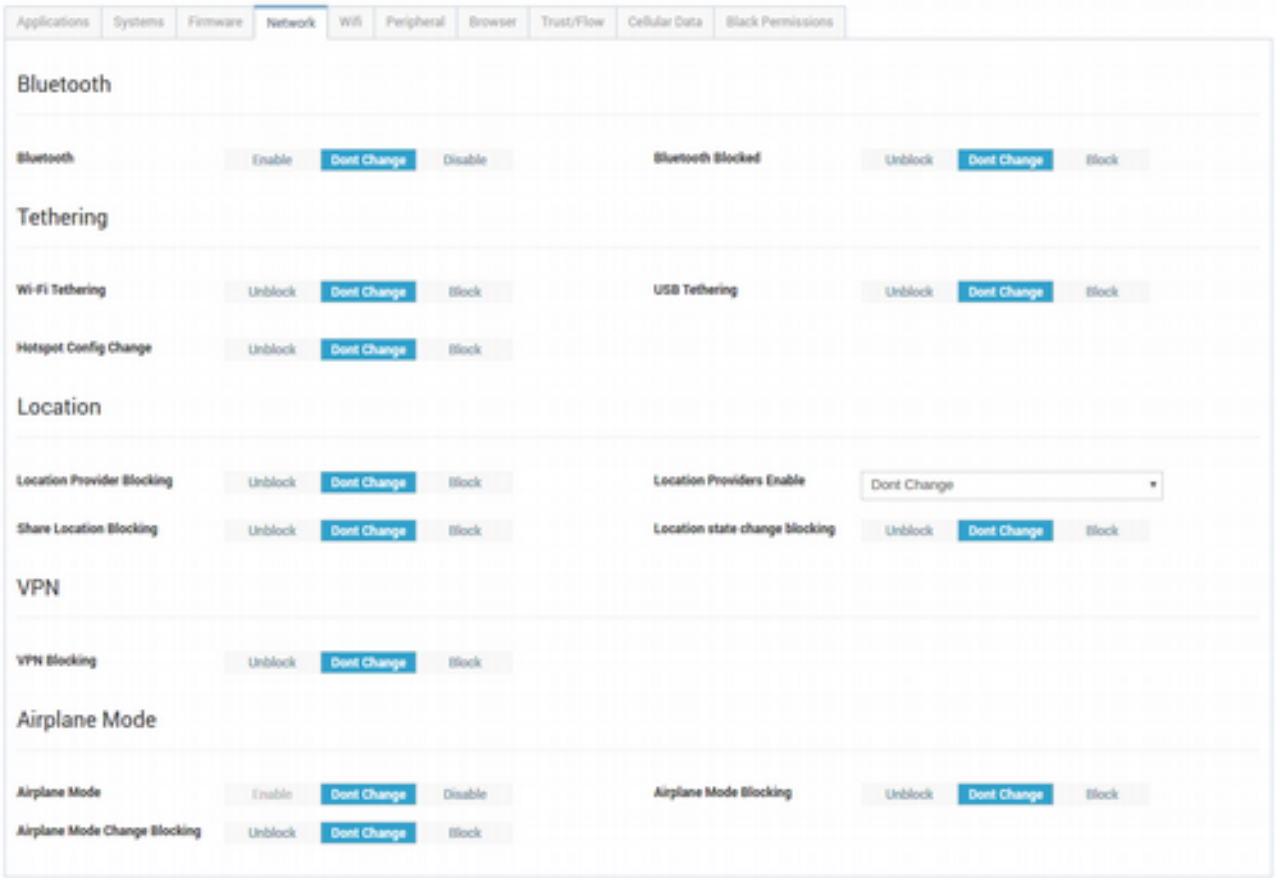
- **Firmware**

Gateway’in yazılımı ile ilgili ayarlamaların yapıldığı alandır. OTA (Over The AIR; operatörlerin veya üreticilerin güncelleme verilerini cihazlara kablosuz bağlantı ile gönderme teknolojisidir) engelleme, Fabrika ayarlarına dönmeyi engelleme, Firmware yüklemeyi engelleme gibi opsiyonlar yapılabilir.



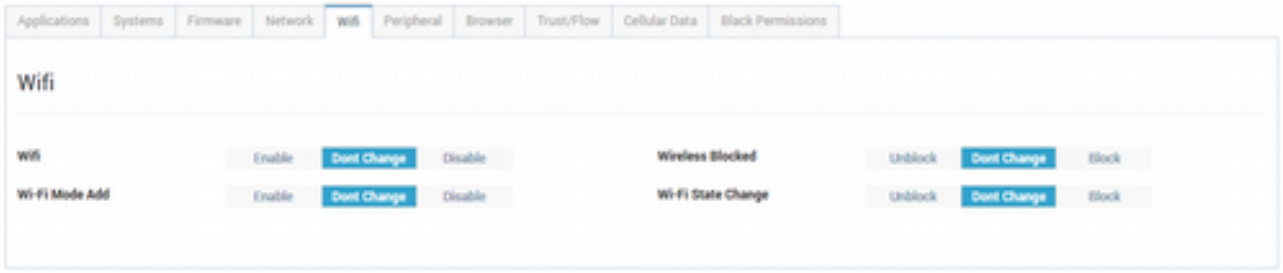
- **Network**

Ağ ile ilgili izinlerin düzenlendiği alandır.



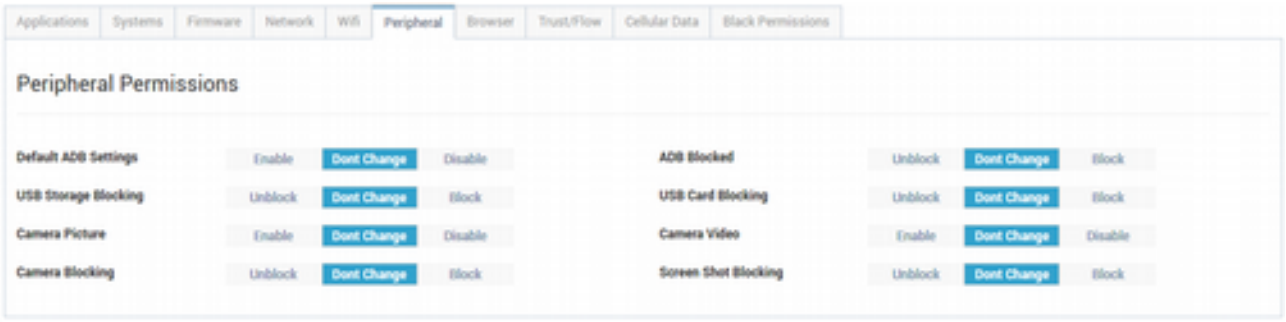
- **Wifi**

Wifi ile ilgili izinlerin düzenlendiği alandır.



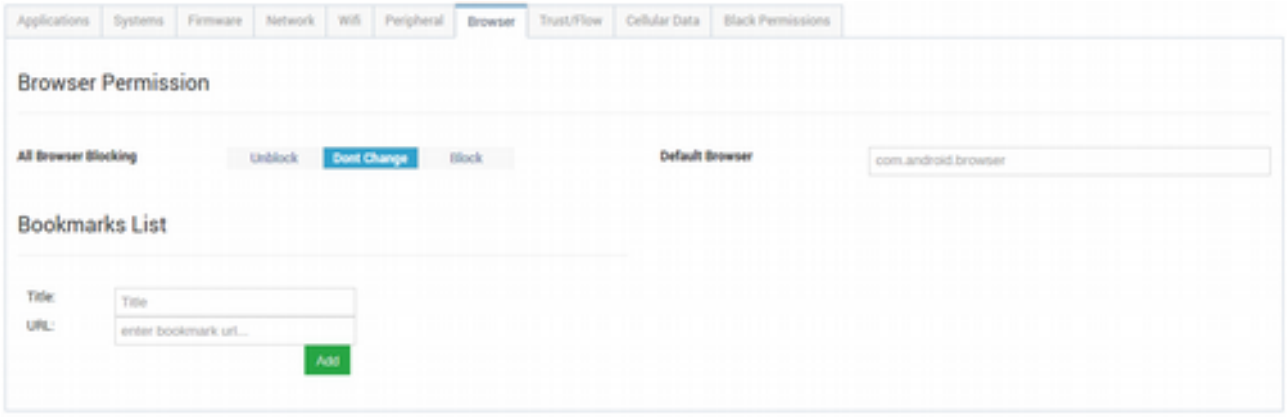
- **Peripheral**

Çevresel birimlerle ilgili izinlerin düzenlendiği alandır.



- **Browser**

Tarayıcı ile ilgili izinlerin ve ayarlamaların yapıldığı alandır.



Eğer Gateway'de birden fazla Browser varsa, varsayılan olarak kullanılacak olan Browser'ın paket ismi **Default Browser** alanında belirtilir. Ayrıca Browser'a **Book Mark** eklemek için de **Title** (Başlık) ve **URL** bilgileri girilip Add butonu ile ekleme yapılabilir.

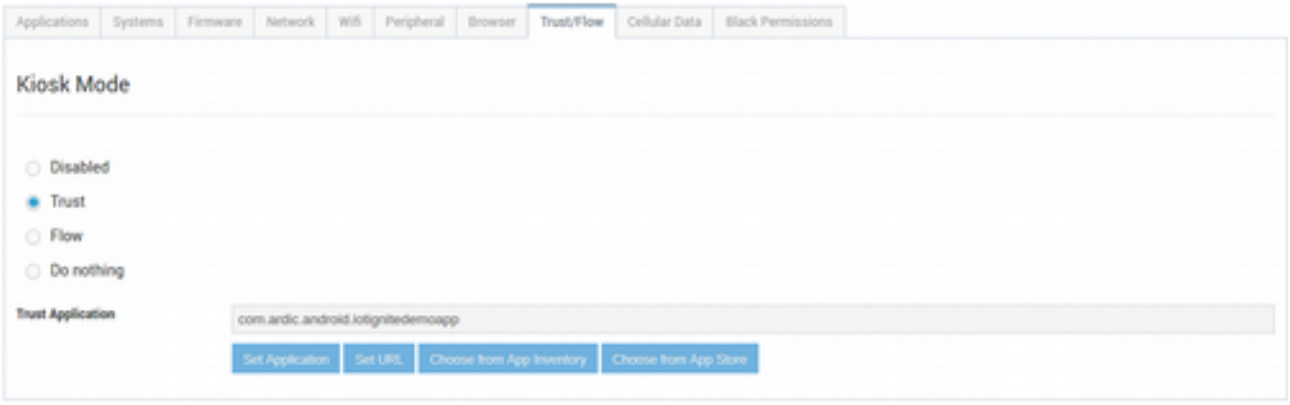
- **Trust/Flow**

Gateway'in bir Kiosk (tek bir uygulamanın aktif olarak tam ekranda çalışır olması) cihazı gibi çalışabilmesini sağlar. 4 adet opsiyon ve her birine ait bir takım ayarlamalar bulunur. Bunlar;

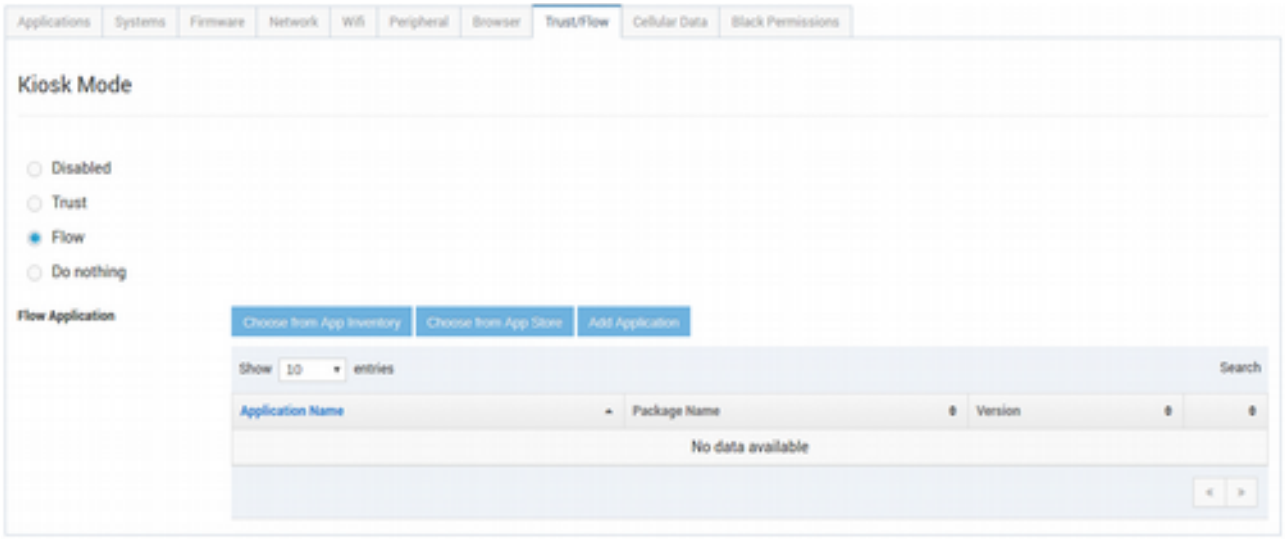
Disabled: Kiosk modunu kapatır.

Trust (Güven): Herhangi bir uygulamanın Kiosk olarak çalışması için **Set Application** (Uygulamayı Yap) butonuna tıklanarak uygulama seçilir. Uygulamayı seçmek için **Choose from**

App Inventory (Uygulama Envanterinden Seç) veya **Choose from App Store** (Uygulama Mağazasından Seç) butonlarına tıklanarak seçim yapılır. Veya doğrudan paket ismi de girilebilir. Eğer bir uygulama değil de bir URL'i browser'da açtırıp sabitlemek isterseniz, **Set URL** (URL'i Yap) butonuna tıklayıp bir URL girebilirsiniz.



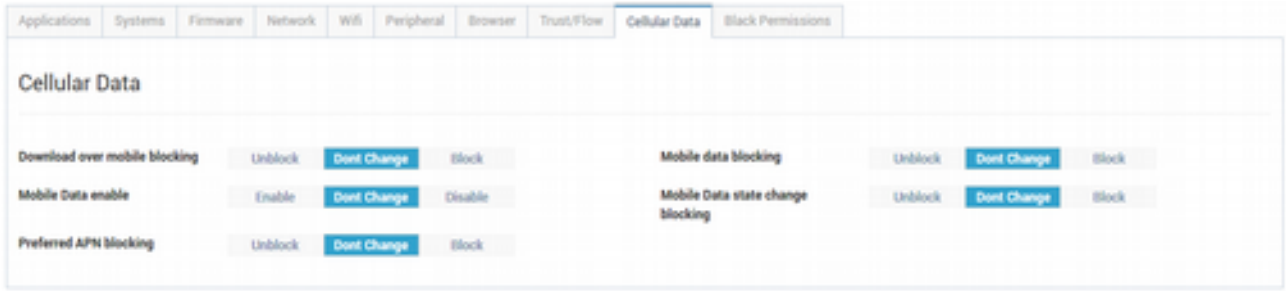
Flow (Akış): Gateway'de bazı uygulamaların gizlenmesi, bazılarının da görüntülenmesi sağlanabilir. Böylece kullanıcıların sadece belli programlara erişebilmesi sağlanır.



Do nothing (Bir şey yapma): Bu opsiyon ile Gateway tarafında ne karar verilmişse o uygulanır.

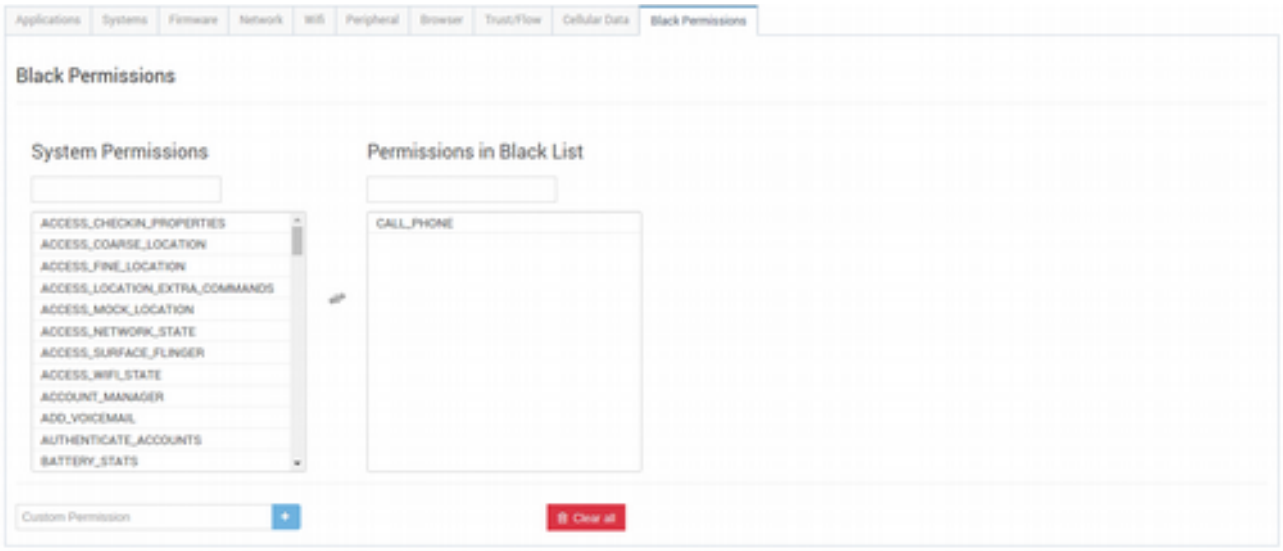
- **Cellular Data**

Hücresel veriler ile ilgili izinlerin düzenlendiği alandır.



- **Black Permissions**

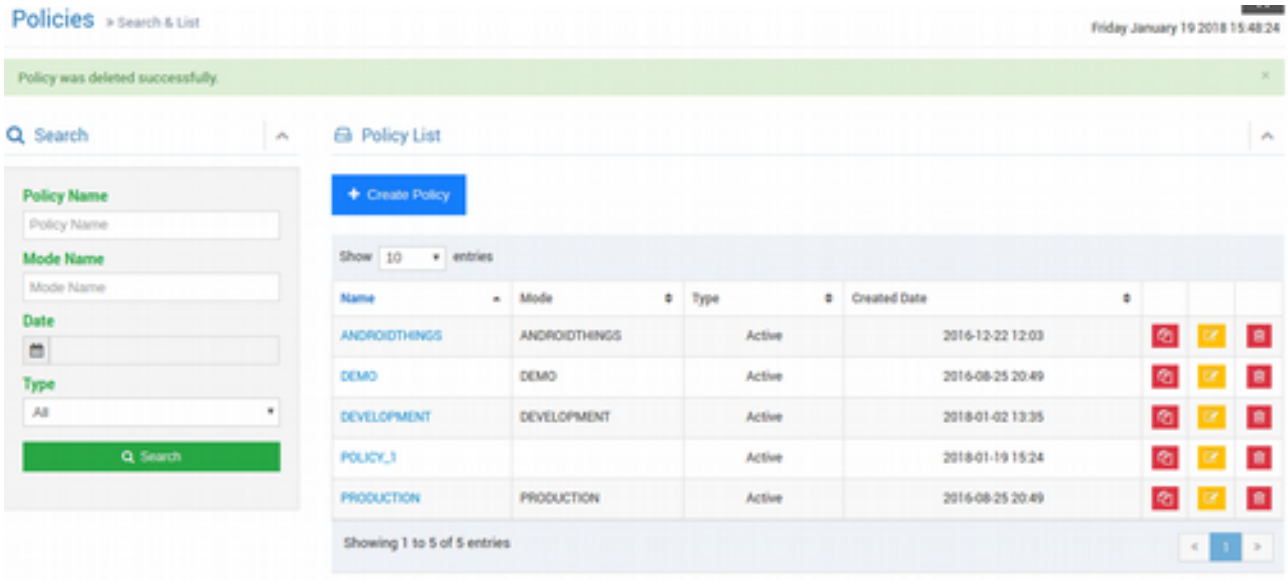
Listede yer alan sistem izinlerinden engellenecek olanlar çift tıklanarak sağ tarafa (Permissions in Black List) taşınabilir. Böylece o izinleri kullanan uygulamalar çalıştırılamaz. Mesela CALL_PHONE izni kara listeye alındı. Böylece bu Policy'i kullanan Android Gateway'lerde telefon araması yapılamaz.



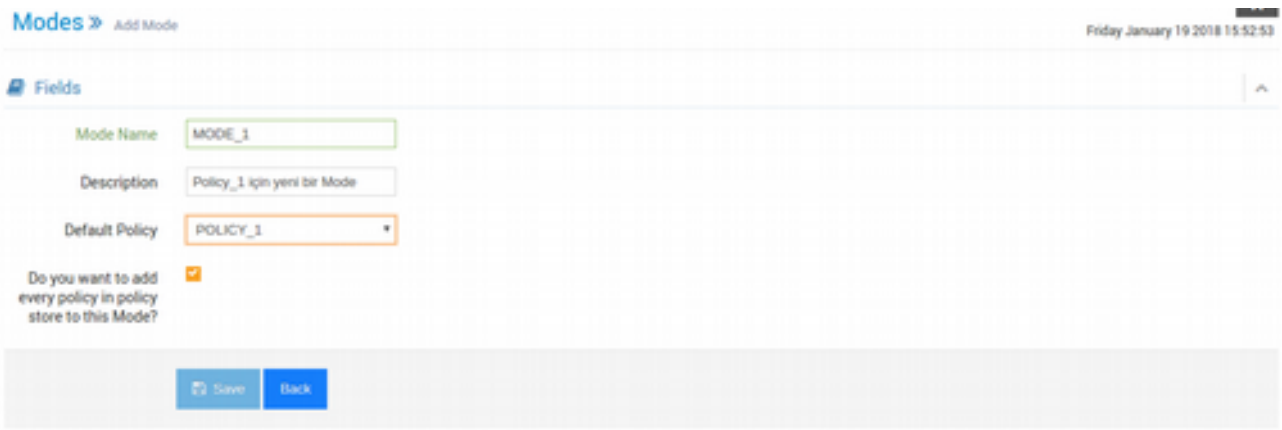
Yeni Oluşturulan Policy'i Mode İçine Almak

POLICY_1 adında yeni bir policy oluşturduk, ancak bu policy sadece **Policy Store**'da hazır olarak bekleyecektir.

Menüden **Gateway Policies** (Gateway Politikaları) > **Policy Store** (Politika Deposu)'a girdiğimizde var olan policy'leri görebilir, düzenleyebilir ve silebiliriz.



Menüden **Gateway Modes** (Gateway Mod'ları) > **Mode Store** (Mod Deposu)'na girin. Burada isterseniz herhangi bir Mode'u değiştirebilir, isterseniz de **Add Mode** (Mod Ekle) butonuna tıklayıp yeni bir Mode oluşturarak **POLICY_1** politikasını bu yeni Mode içine alabilirsiniz. Yeni bir Mode oluşturalım...



Mode Name alanına **MODE_1** ismini girdik.

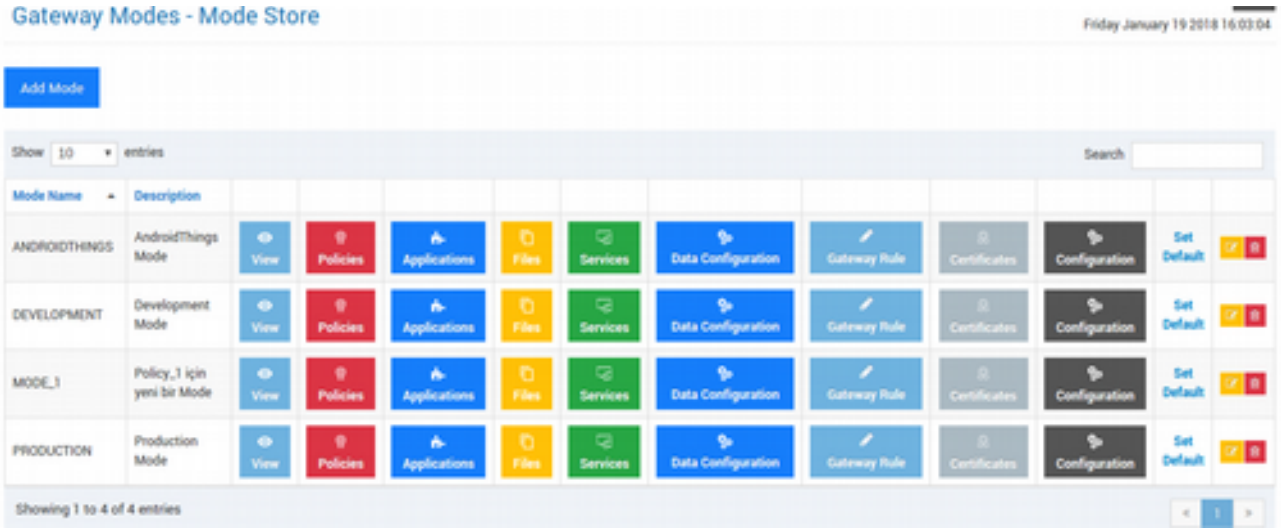
Description alanına basit bir tanımlayıcı içerik girdik.

Default Mode alanında **MODE_1** için de listeden **POLICY_1**'i seçtik. Böylece MODE_1 Mode'unun varsayılan Policy'si POLICY_1 olacaktır.

Alt kısımda yer alan opsiyonu seçtiğimizde, listede yer alan bütün Policy'ler Mode'a eklenecektir. Eğer bu opsiyon seçili olmazsa, sadece Default Mode listesinde seçilen Policy eklenecektir. Opsiyon seçili iken **Save** (Kaydet) butonuna tıklayalım.

> Bir Mode içinde birden fazla Policy olabilir. Bunlardan da biri aksi belirtilmedikçe aktif olsun diye Default olarak tanımlanır. Bazı durumlarda Mode'un içinde çoklu Policy'lerin olması gerekebilir. Örneğin günün belli bir saatinde bir Policy, akşam üzeri de başka bir Policy'nin aktif olması istenebilir ve otomatikleştirilebilir de. Veya bu iş EHUB'tan kullanıcı aracılığıyla sıklıkla yapılıyorsa, her defasında Gateway'lere Mode'u gödermek (push) yerine, Rule Editor'den Change Policy aksiyonu kullanılarak Date Time sensörü ile zaman bilgisine bağlı olarak Mode değişikliği otomatik olarak yapılabilir.

Tekrar Mode Store'a dönelim...



Görüldüğü üzere MODE_1 Mode'umuz listede görünecektir.

Lisanslanan herhangi bir Gateway'in de varsayılan olarak hangi Mode'u ilk olarak almasını istiyorsanız, **Set Default** (Varsayılan Yap) butonuna tıklayıp Mode'u varsayılan yapabilirsiniz.

Gateway Model

Servisinizde lisanslı olan Gateway'lerin modelleri ve bu modellerden kaçar adet olduğu listelenmektedir. Herhangi bir model ismine tıkladığınızda All Gateways (Bütün Gateway'ler) sayfasına yönlenir ve sadece tıkladığınız model isimlerine göre sıralanmış Gateway'leri görürsünüz.

OS Version

Gateway'lerdeki işletim sistemlerinin versiyon numaraları listelenir. Herhangi bir işletim sistemi versiyonuna tıkladığınızda All Gateways (Bütün Gateway'ler) sayfasına yönlenir ve sadece tıkladığınız işletim sistemi versiyonuna göre sıralanmış Gateway'leri görürsünüz.

Gateway Mode

Gateway'lerin üzerindeki IoT-Ignite ajanlarının hangi yetki düzeyinde çalıştığını göstermektedir. Herhangi bir isme tıkladığınızda All Gateways (Bütün Gateway'ler) sayfasına yönlenir ve sadece tıkladığınız isme göre sıralanmış Gateway'leri görürsünüz.

Bunlar;

- **Afex**: PilarOS işletim sistemli Gateway'ler.
- **Lite**: Afex yüklü olmayan diğer Gateway'ler.
- **Safex**: İçerisinde Knox'ın sertifikası olan Samsung Android'li cihazlar.
- **Light Gateway**: MQTT.

Afex (Android Framework Extensions); Android cihazlarının basit ve kolay entegrasyon ile "IoT Gateway Yönetimi" aracılığıyla kontrol edilmesi, yapılandırılması, hazırlanması ve yönetilmesini sağlar. Android API'lerinin yanı sıra, 1000'den fazla API barındırır. Afex, bir ARDIC, yani IoT-Ignite ürünüdür. Bu ek API'ların kullanılması durumunda hizmet sağlayıcıları için bir lisans oluşturulur. Herhangi bir IoT servis sağlayıcısı, sistemi kolayca entegre edebilir ve cihazı, ağı, yöneticiyi, sertifikaları, uygulamaları, tarayıcıları, e-posta konfigurasyonunu vb. yönetebilir.

Afex Referansı: https://devzone.iot-ignite.com/device-api/afex/reference/packages.html

Modiverse Version

Gateway'de yer alan ajan uygulamasının sürüm numarasıdır.

Modiverse; uzaktan cihaz yönetimi yapan iş dünyasına yönelik bir yazılım çözümüdür. Sistem yöneticisi cihaz kayıt, konfigürasyon, yönetim ve güncelleme gibi en kolaydan en karmaşığa tüm MDM (Mobil Cihaz Yönetimi) işlevlerini uygulayabilir; cihazlarda izin verilen uygulamaları, cihaz özellik ve izinlerini uzaktan belirleyebilir ve değiştirebilir. Sisteme kayıtlı Android cihazların web tabanlı bir yönetim arayüzü ile güvenli bir kuruluş ortamından hizmet alabilmesini ve yönetilebilmesini sağlar.

> Google Play'den ücretsiz olarak indirilebilir ve kullanılabilir.

Build Numbers

İşletim sisteminin derleme numarasıdır.

Working Set (Çalışma Seti) ile Gateway'ler Üzerinde Toplu Kontrol Sağlamak

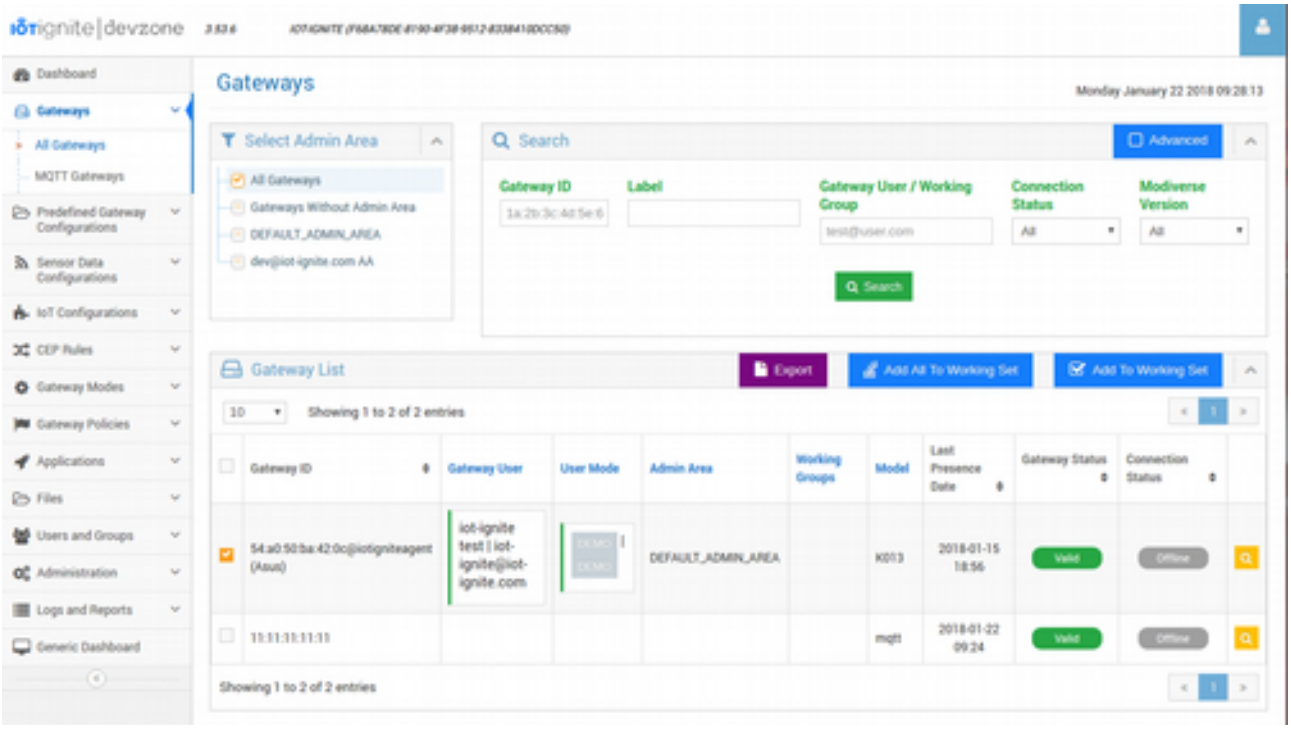
Çalışma Seti; EHUB'ta bir veya birden fazla Gateway'yi yönetmek için istenilen Gateway veya Gateway'leri bir listeye alması, sonrasında da çeşitli işlemler yaparken bu listeyi referans alması anlamına gelmektedir.

EHUB'ta **Gateways > All Gateways** (Bütün Gateway'ler) sayfasına girdiğimizde, servisimizde lisanslı olan Gateway'lerin listelendiğini görmüştük. Listede yer alan Gateway'ler ile işlem yapabilmek için istenilen Gateway'i veya Gateway'leri **Working Set** (Çalışma Seti)'e almamız gerekir. Listenin sağ üst köşesinde yer alan;

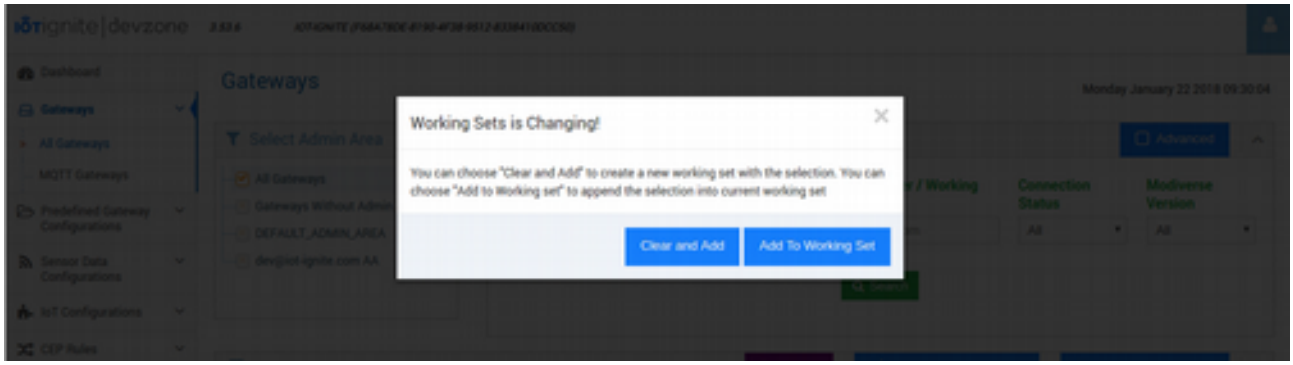
- **Add All to Working Set** ile lisanslı olan bütün Gateway'ler Working Set'e alınır.
- **Add To Working** Set ile de, listede seçili olan Gateway'ler Working Set'e alınır.

Bir veya Birden Fazla Gateway'in Working Set'e Alınması

Örnek olması amacı ile Android Gateway'imizi seçip, **Add To Working Set** (Çalışma Set'ine Ekle) butonuna tıklayalım.



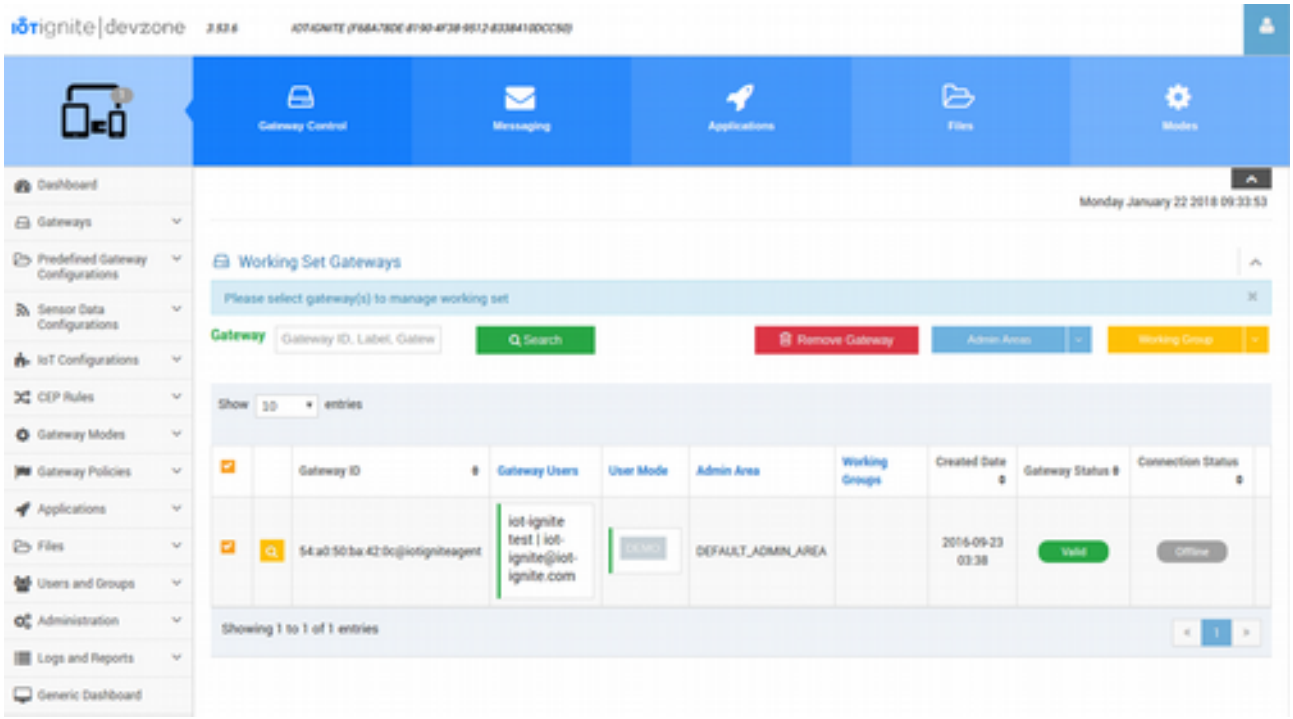
Tıklama işlemi ile birlikte karşınıza gelecek olan seçimden **Clear and Add** butonu ile Working Set'i temizleyip yeniden ekleme yapabilirsiniz. Working Set'te daha önceden Gateway'ler varsa ve onların üzerine ekleme yapmak istiyorsanız **Add To Working Set** butonuna tıklayın. Şu an Working Set boş olduğu için hangisini seçerseniz seçin, fark etmez.



Bu işlem sonucunda da EHUB arayüzünün üst kısmında aşağıdaki şekilde görüldüğü gibi mavi renkli bir panel belirecektir. Bu panel, Working Set üzerinde yapacağınız işlemleri içerir.

Working Set'teki Gateway'leri Listelemek

Sol tarafta yer alan cihazların olduğu kısımda **1** sayısı görülmektedir, bu sayı o an Working Set'te yer alan Gateway'lerin sayısıdır. Bu ikona tıkladığınızda Working Set'in içine girilecek ve ekli olan Gateway'ler listelenecektir.



Working Set'in Boşaltılması

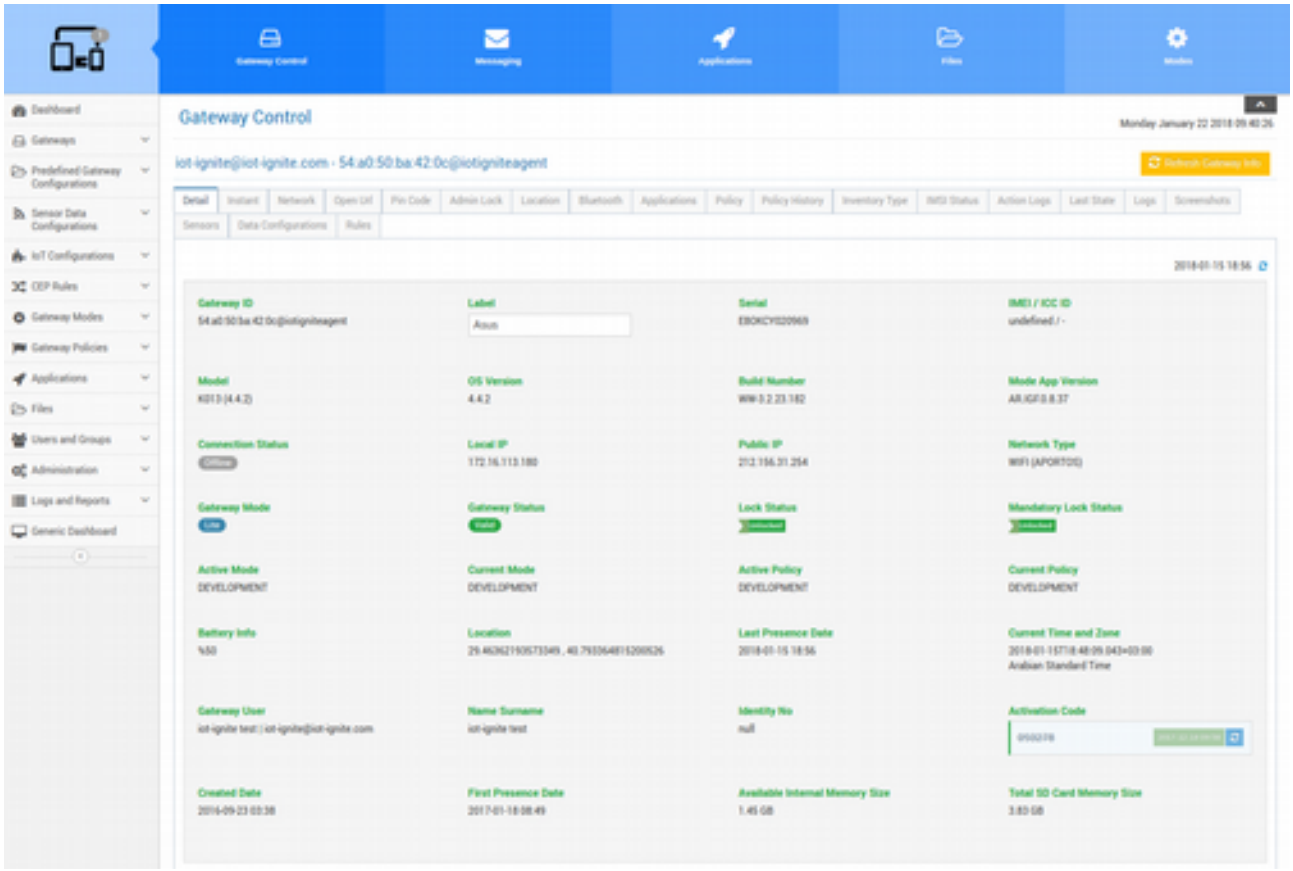
Working Set'e girdiğinizde, isterseniz bu alandan ekli olan Gateway'leri geri çıkarabilirsiniz. Veya EHUB'tan çıkış yaptığınızda, otomatik olarak Working Set de boşaltılacaktır.

Working Set’teki Gateway’lerin Kontrolleri

Mavi renkli Working Set alanında yer alan Gateway Control butonuna tıkladığınızda, Working Set’te yer alan Gateway’lerin bir takım yönetim işlemleri yapılabilmektedir. Bunlardan bazılarını inceleyelim...

Tek Gateway’in Kontrolü

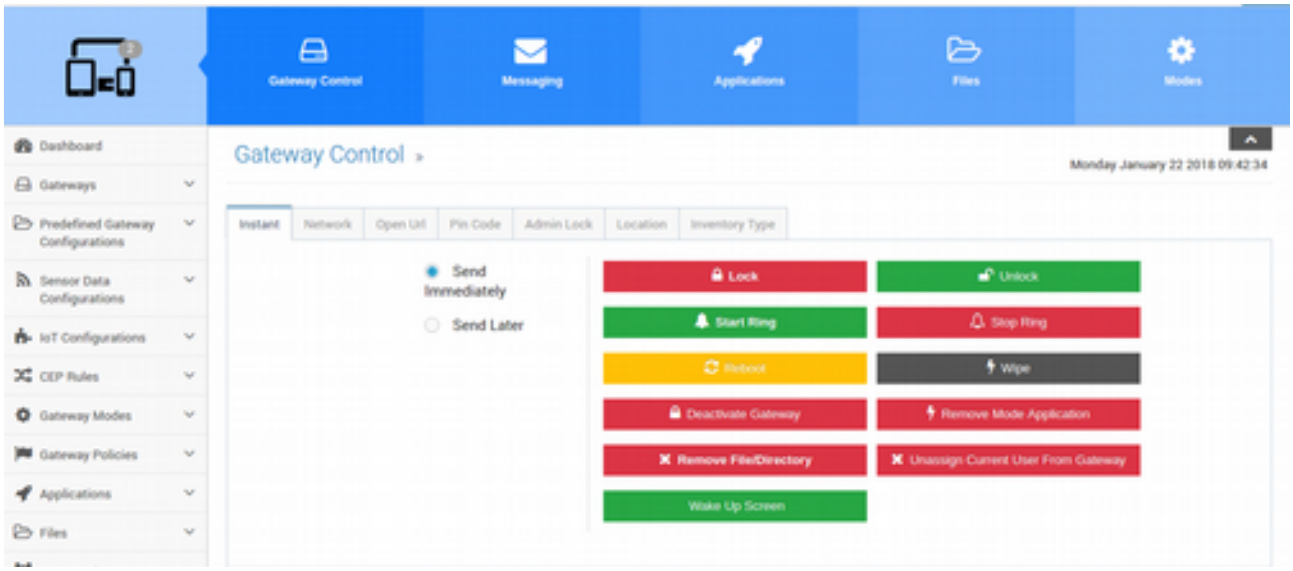
Eğer Working Set’e sadece 1 adet Gateway eklerseniz, aşağıdaki şekilde görüldüğü gibi bir kontrol arayüzü ile karşılaşacaksınız.



Working Set’te sadece tek bir cihaz olduğu için ilk olarak **Detail** (Detay) ekranında o Gateway’e ait bir takım bilgiler listelenmektedir. Bilgilerin sağ üst köşesinde o bilgilerin Gateway’den alınma tarihi gösterilmektedir. **Refresh Gateway Info** butonuna tıklayarak Gateway’den son bilgileri tekrar isteyebilirsiniz. Güncel bilgiler Gateway’den geldiğinde, listede yeni bilgiler görünecektir.

Çoklu Gateway’in Kontrolü

Eğer Working Set’te 1’den fazla Gateway olursa, hepsi için ortak olarak kullanılabilecek kontroller gelmektedir.

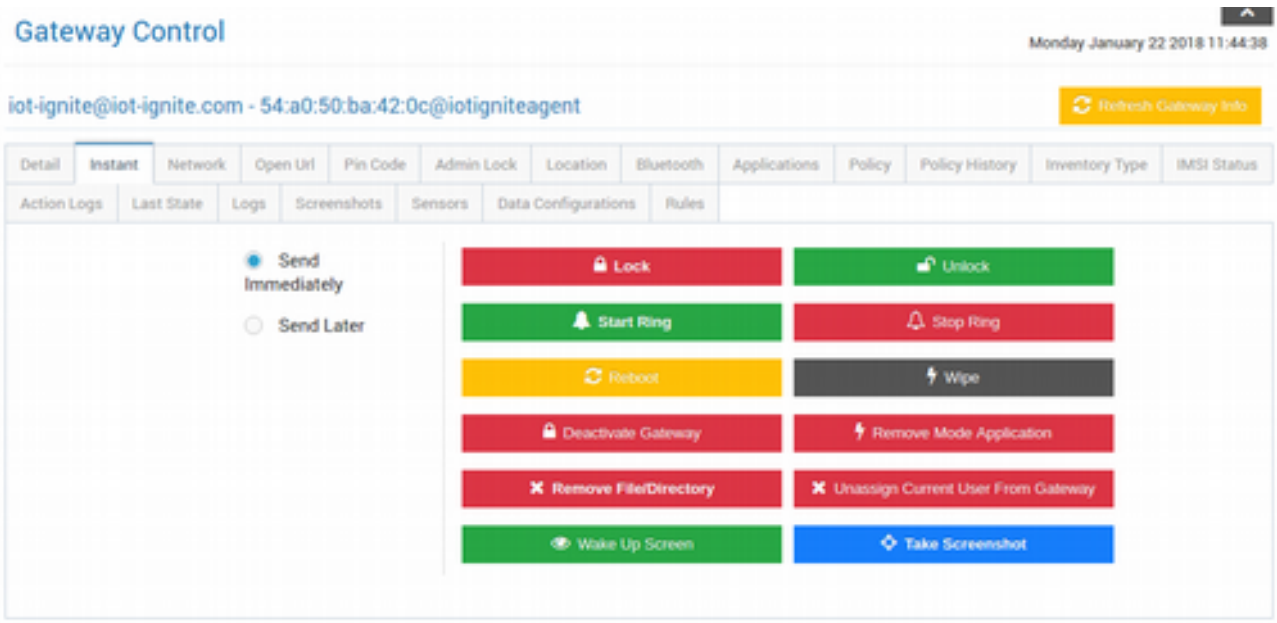


Şimdi, Gateway'ler ile ne gibi kontroller yapılacağını tek bir Android Gateway olduğu durum için inceleyelim...

Gateway Kontrolleri

Working Set'te yer alan Gateway'lerle "kullanıcı yetkilendirmeleri dahilinde" bir takım kontroller gerçekleştirebilirsiniz. **Gateway Control** grubu altında yer alan bu kontrolleri sırası ile kısaca inceleyelim...

- **Instant (Komutlar)**



Bu alanda çalışırken oldukça **dikkatli** olmak gerekmektedir! Bu alanda yapılacak işlemler aşağıdaki gibidir (Bazılarını Devzone'da görmüştük):

Lock: Gateway'lerin ekranını kilitler ve kullanıcı hiçbir işlem yapamaz. Ekranda, kilit ikonu belirir. Kilit ekranında da görüntülenecek olan mesajı, butona tıkladığınızda açılacak olan metin kutusuna girebilirsiniz.

Unlock: Kilitlenmiş olan Gateway'lerin ekran kilidini açar. Kullanıcı artık işlem yapabilir.

Start Ring: Gateway'lerde zil sesi çaldırır.

Stop Ring: Gateway'lerde çaldırılan zili susturur.

Reboot: Gateway'leri yeniden başlatır.

Wipe: Gateway'leri formatlar.

Deactive Gateway: Gateway'lerin lisansını kaldırır.

Remove Mode Application: Gateway'lerden IoT-Ignite ajan uygulamasını kaldırır.

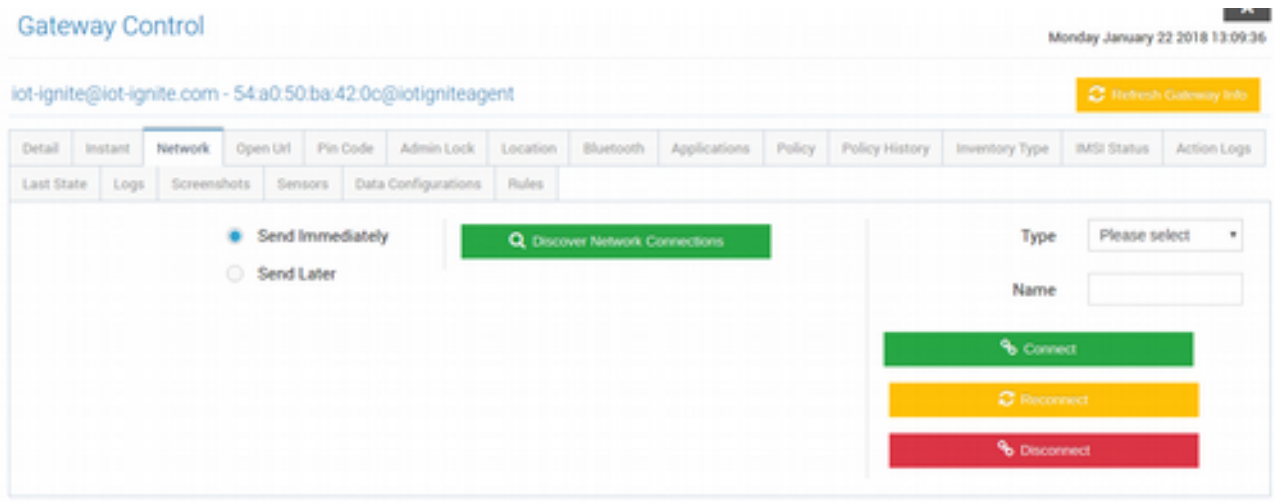
Remove File/Directory: Gateway'lerden bir klasör veya dosyayı kaldırmanızı sağlar. Butona tıkladığınızda klasör yolunu veya dosyayı tanımlamanız için bir arayüz karşınıza çıkar.

Wake Up Screeen: Ekranı uyandırır.

Take Screenshot: Ekran görüntüsü alır (Ekran görüntüsü alabilmek için Gateway AFEX olmalı). Butona tıkladığınızda karşınıza bir takım ayarların yapıldığı arayüz çıkar. Bu arayüzde ekran görüntüsünün nereye kaydedileceği, buluta yüklenip günlenmeyeceği, ekran görüntüsünün dosya adı tanımlanır.

Yukarıda yer alan komutların hemen gerçekleştirilmesi için **Send Immediately** (Hemen Gönder), eğer belli bir tarihte yapılması isteniyorsa **Send Later** (Sonra Gönder) opsiyonu seçilip bir tarih belirtilir.

- **Network (Ağ)**



Gateway'lerin ağ bağlantı durumunu yönetir. **Type** (Tür) alanından ağ durumu **APN**, **Ethernet**, **VPN** ve **Wifi** olarak seçilebilmektedir. Tür seçimi yapıldıktan sonra hangi ağa bağlanacağı da **Name** (İsim) alanına girilir. Tanımlanan ağa bağlantı kurulabilmesi için daha önce o ağa ait parola bilgisinin Gateway'de kayıtlı olması gerekmektedir. Eğer kayıtlı değilse, sağ menüde yer alan **Prefefined Gateway Configurations** (Öntanımlı Gateway Yapılandırmaları) sayfasından bir ağ kaydedebilir ve bu ağ yapılandırmasını da Gateway'e push edebilirsiniz.

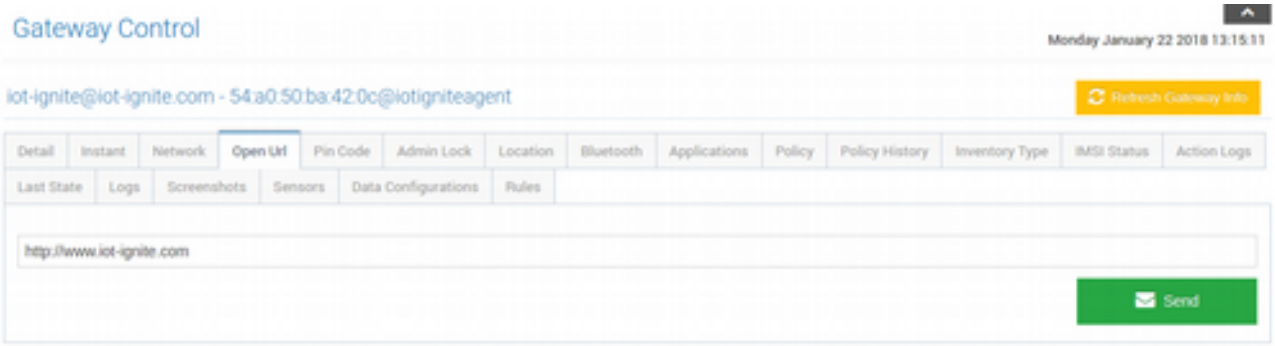
Connect: Belirlediğiniz ağa bağlantı kurar.

Reconnect: Belirlediğiniz ağa yeniden bağlantı kurar.

Disconnect: Ağ bağlantısını koparır.

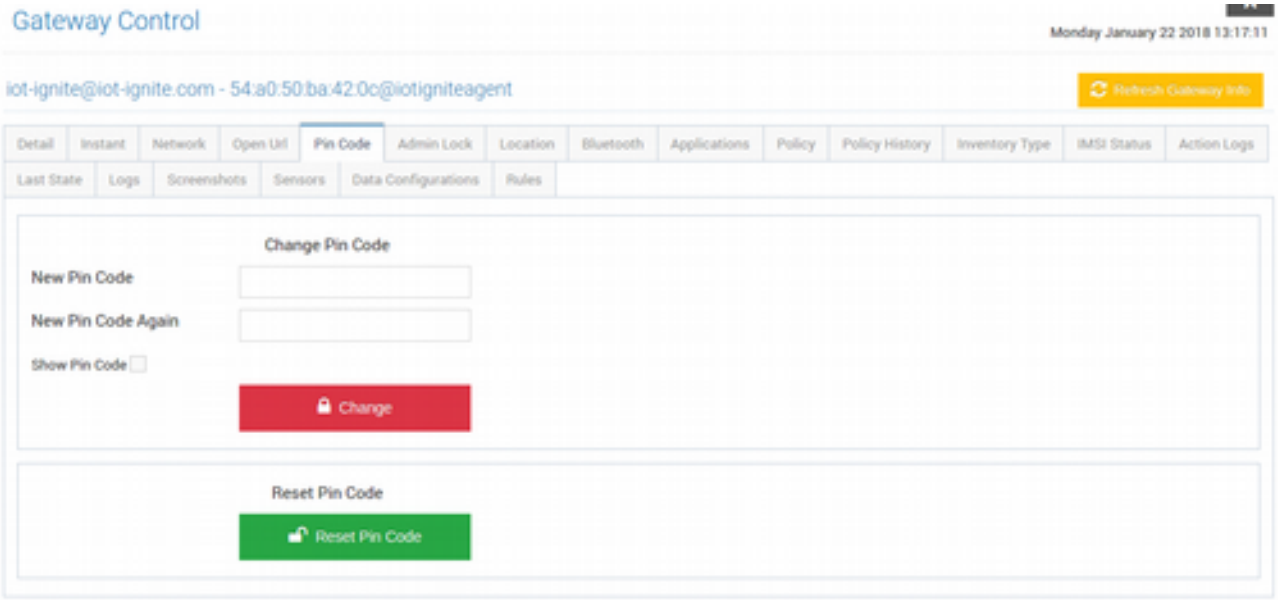
Yapılacak olan bu ağ işlemlerini yine Send Immetiately (Hemen Gönder) ve Send Later (Sonra Gönder) opsiyonları ile gerçekleştirebilirsiniz.

- **Open URL (URL Aç)**



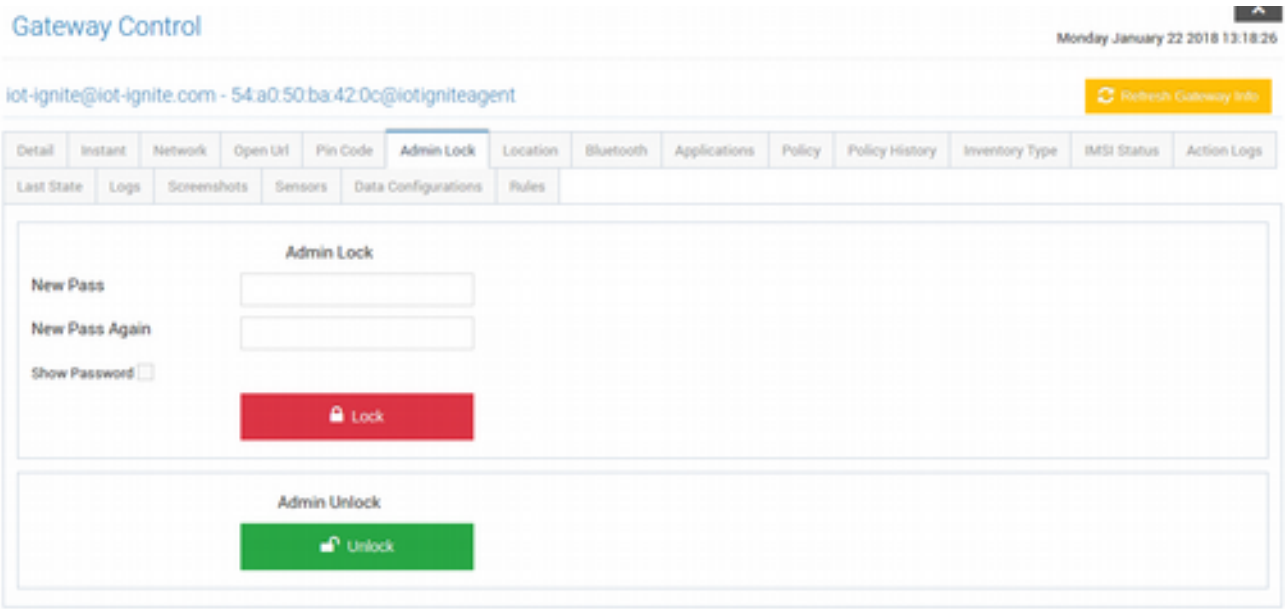
Gateway'e bir URL push edilir ve bu URL Gateway'in varsayılan tarayıcısında açılır.

- **Pin Code (Pin Kodu)**



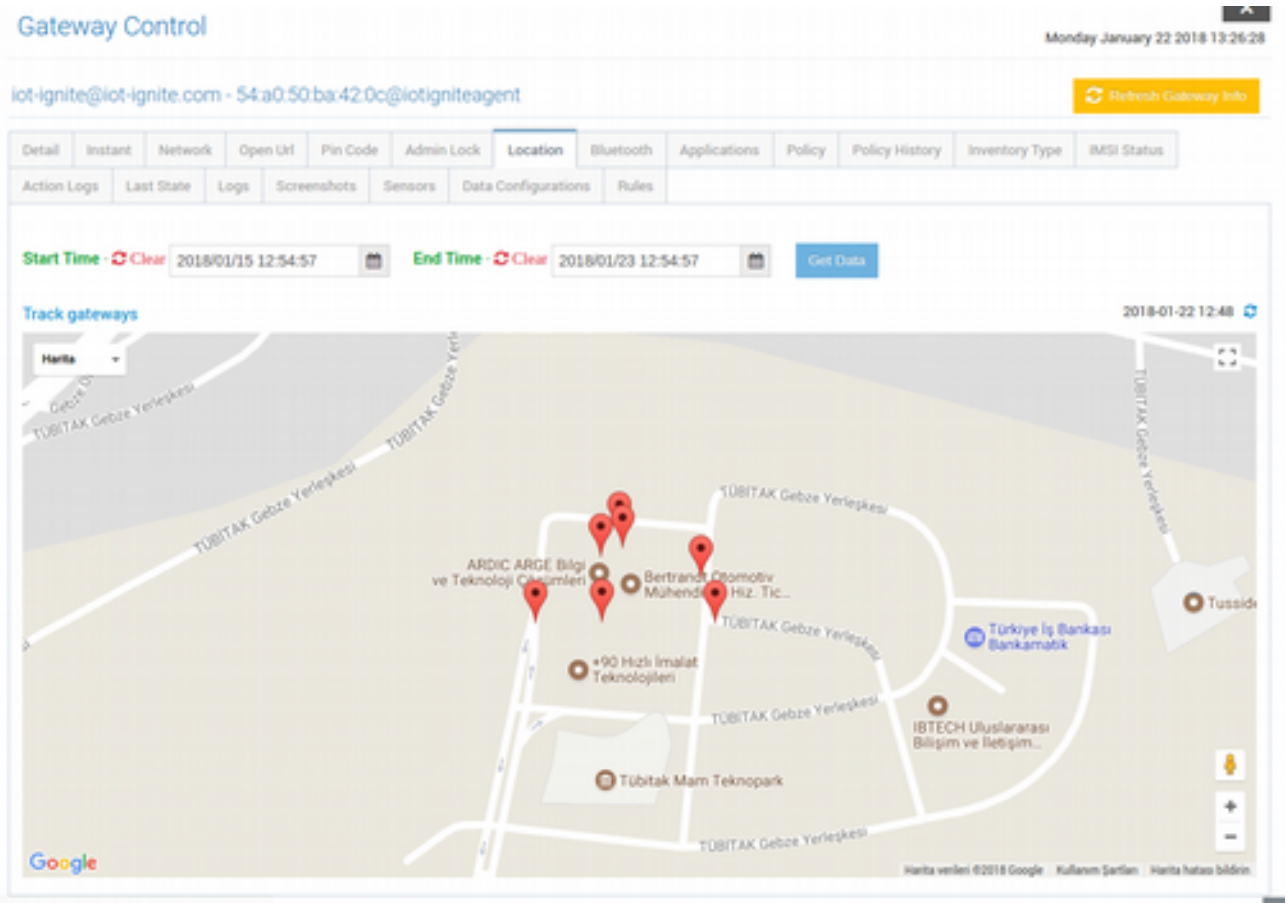
Gateway'in PIN kodunu değiştirebilir veya PIN kodunu sıfırlayabilirsiniz.

- **Admin Lock (Yönetici Kilidi)**



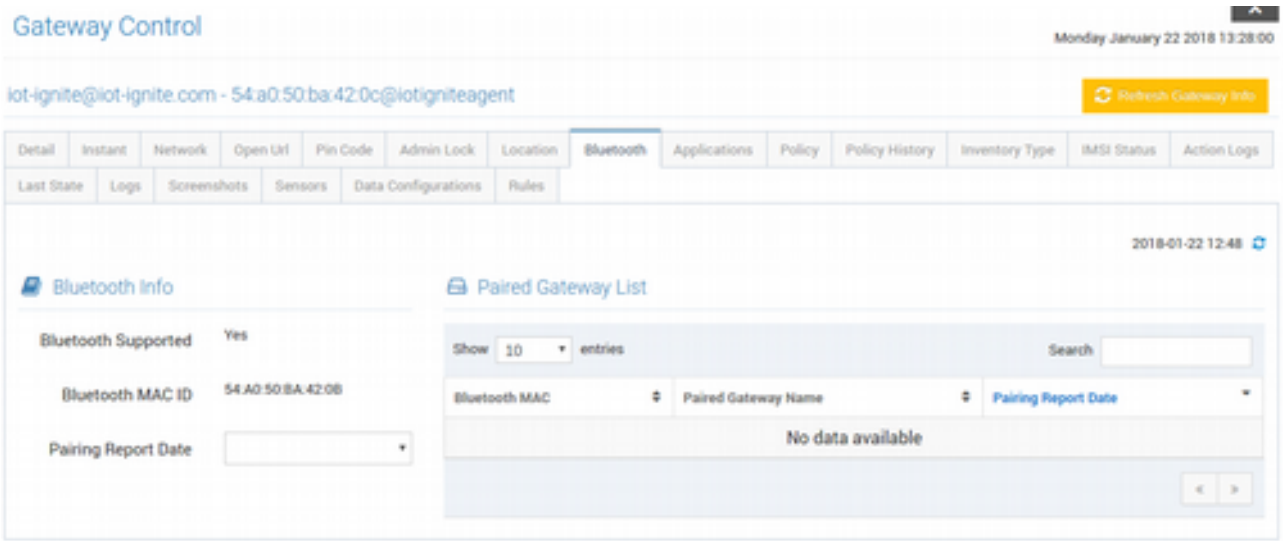
Gateway'in yönetici kilit ekran kodu ile ekranı kilitleyebilir veya tekrar serbest bırakabilirsiniz.

- **Location (Lokasyon)**



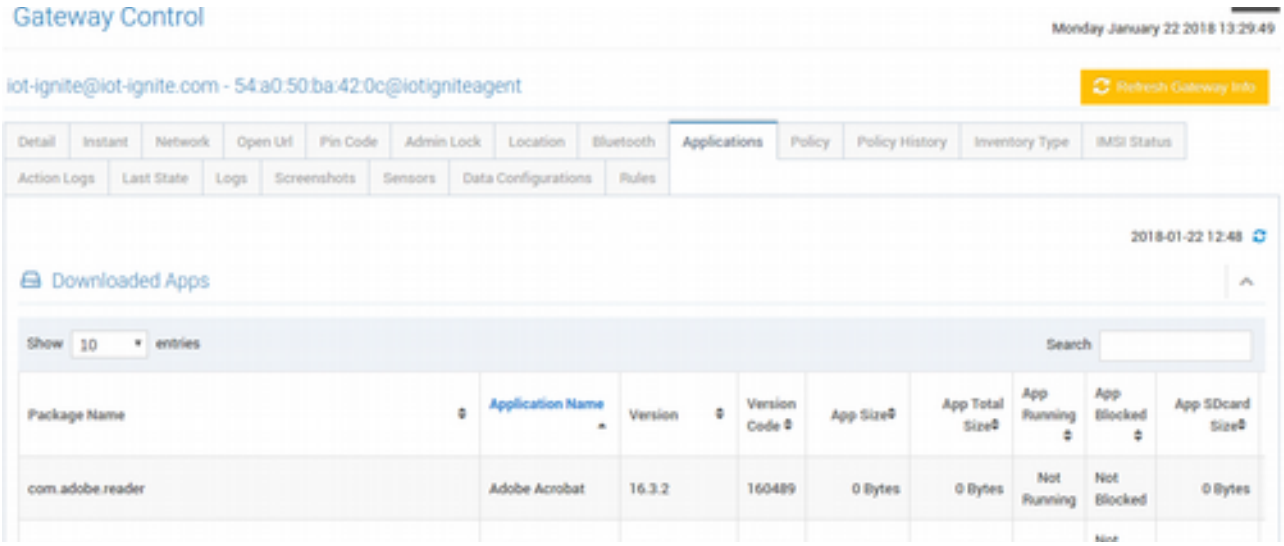
Gateway'in belirli tarihler arasında hangi konumlarda olduğunu görebileceğiniz bir harita arayüzü açar. Bu özelliğin çalışabilmesi için Gateway'in GPS'inin aktif olması gerekmektedir.

- **Bluetooth**



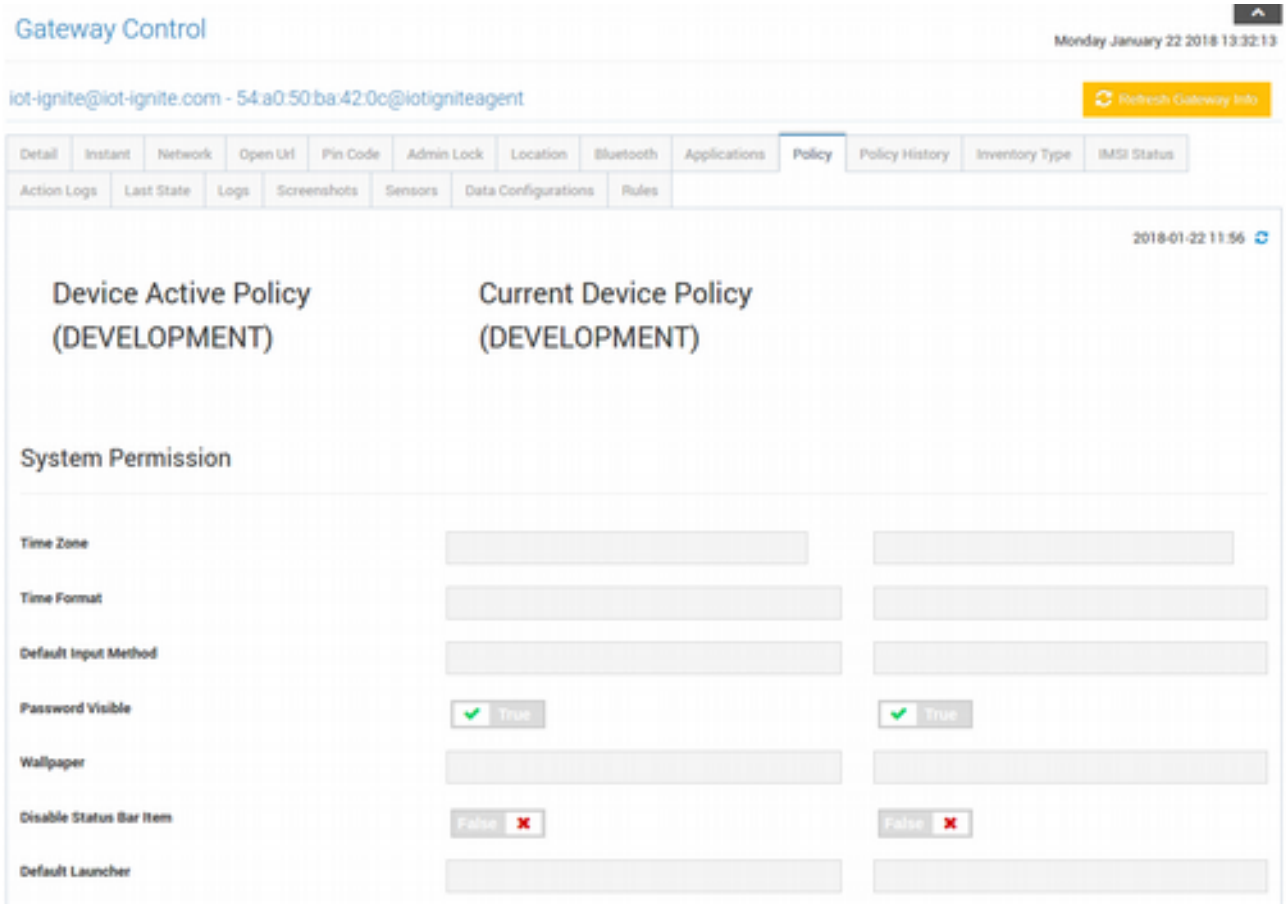
Gateway'in Bluetooth MAC ID'sini görebileceğiniz ve eşleşme rapor tarihlerini ve detaylarını görebileceğiniz bir liste açar.

- **Applications (Uygulamaları)**



Anlık olarak Gateway'de yüklü olan uygulamaların listesini, detaylarını, hangi uygulamanın çalıştığını, hangisinin bloklu olduğunu ve hangisinin ne kadar alan kapladığını görebileceğiniz bir liste açar. Listeyi yenilemek için Refresh Gateway Info butonuna tıklanarak Gateway'den güncel durum çekilebilir.

- **Policy (Politika)**

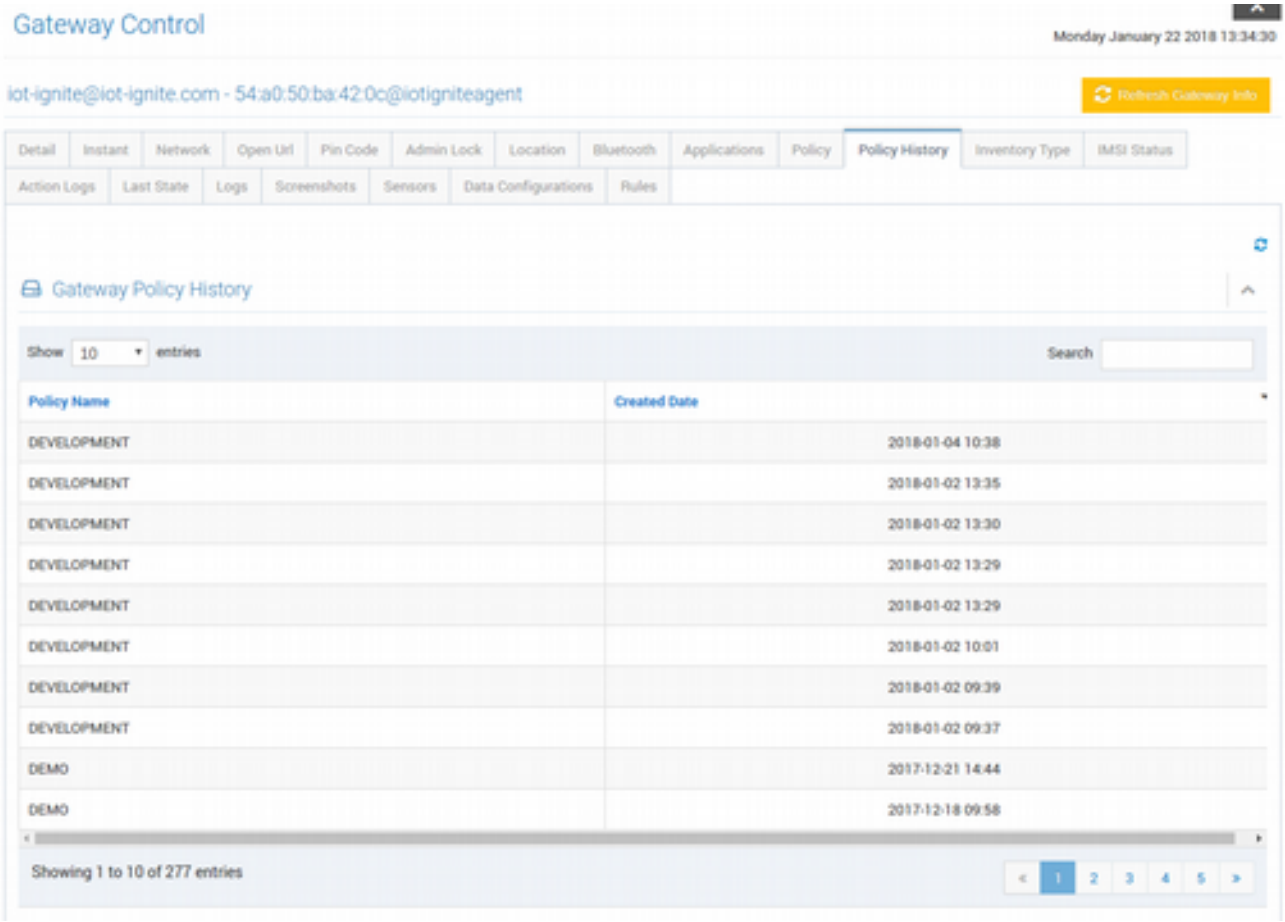


Mode içinde o an aktif olan Policy ve Gateway'de o an yüklü olan Policy'leri karşılaştırır. Bazı durumlarda EHUB'ta Policy güncellediği halde Gateway'e push edilmemiş, edilse de almamış

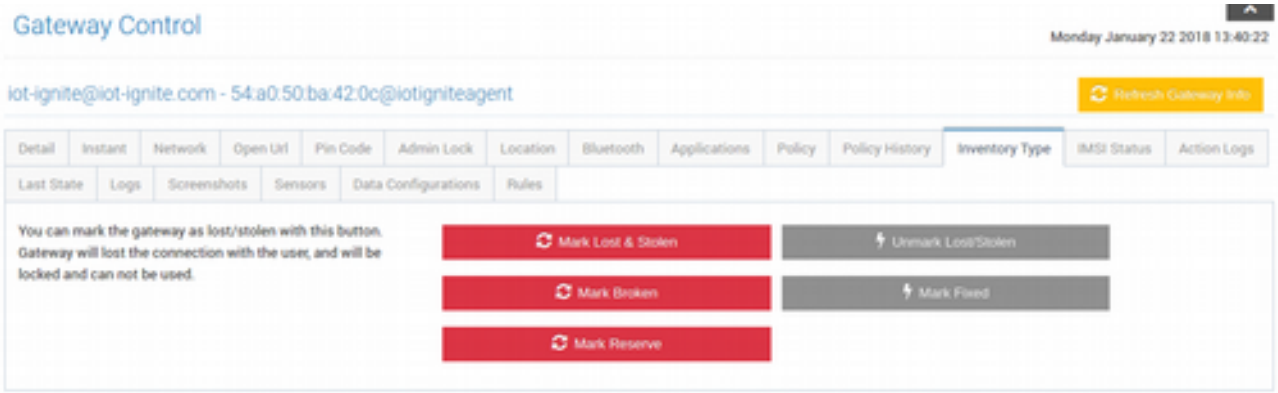
olabilir. Bu gibi durumlarda kontrol edilmek istendiğinde bir karşılaştırma listesi ile kontroller yapılabilir. Refresh butonuna tıklanarak Gateway'de yüklü olan Policy'nin son durumu çekilebilir.

- **Policiy History (Politika Geçmişi)**

Gateway'lerde hangi tarihlerde hangi policy'nin aktif (etkin) olduğunu gösteren bir liste yer almaktadır.



- **Inventory Type (Envanter Türü**)



Gateway'ler birer envanter ürünü olarak düşünülebilir. Gateway'lere verebileceğiniz envanter türleri şunlardır:

Mark Lost & Stolen: Gateway'in kayıp veya çalıntı olduğu işaretlenebilir. Gateway, kullanıcı ile olan bağlantısını kaybedecek, kilitlenecek ve kullanılmaz hale gelecektir.

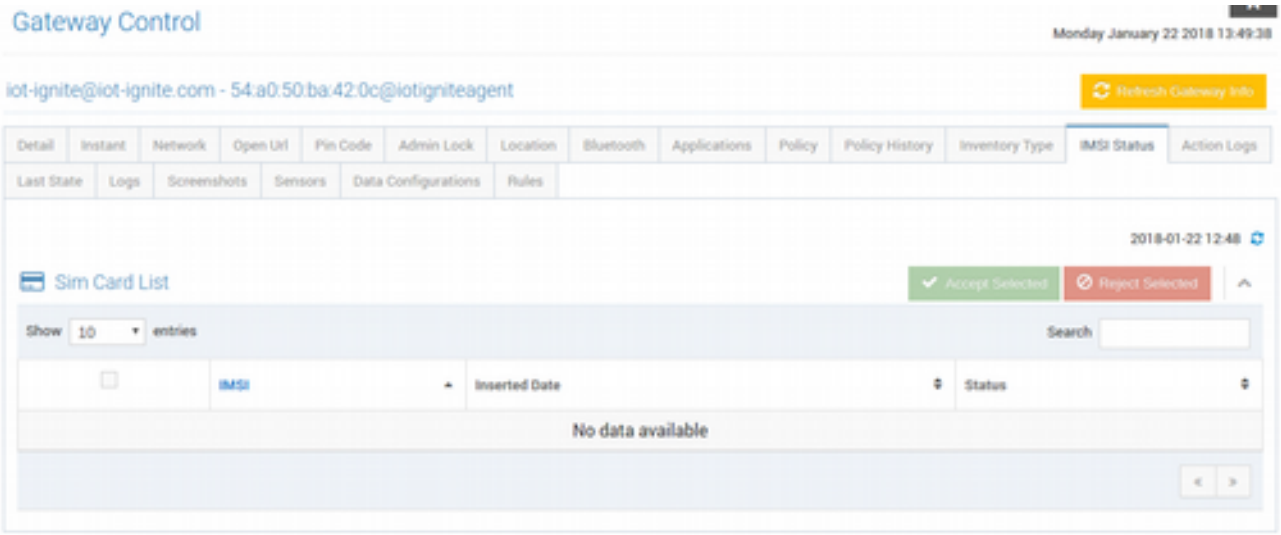
Unmark Lost & Stolen: Kayıp veya çalıntı olarak işaretlenen Gateway'in envanter türünü düzeltir.

Mark Broken: Gateway'in bozuk/çalışmıyor olarak işaretlenmesini sağlar. Gateaway, kullanıcı ile olan bağlantısını kaybedecek. Arızanın doğası nedeniyle Gateway ile iletişim kurulabiliyorsa ve tüm bilgiler silinecek olursa, Gateway varsayılan ayarlarına geri dönecektir.

Mark Fixed: Bozuk olarak işaretlenen Gateway'in envanter türü düzeltilir.

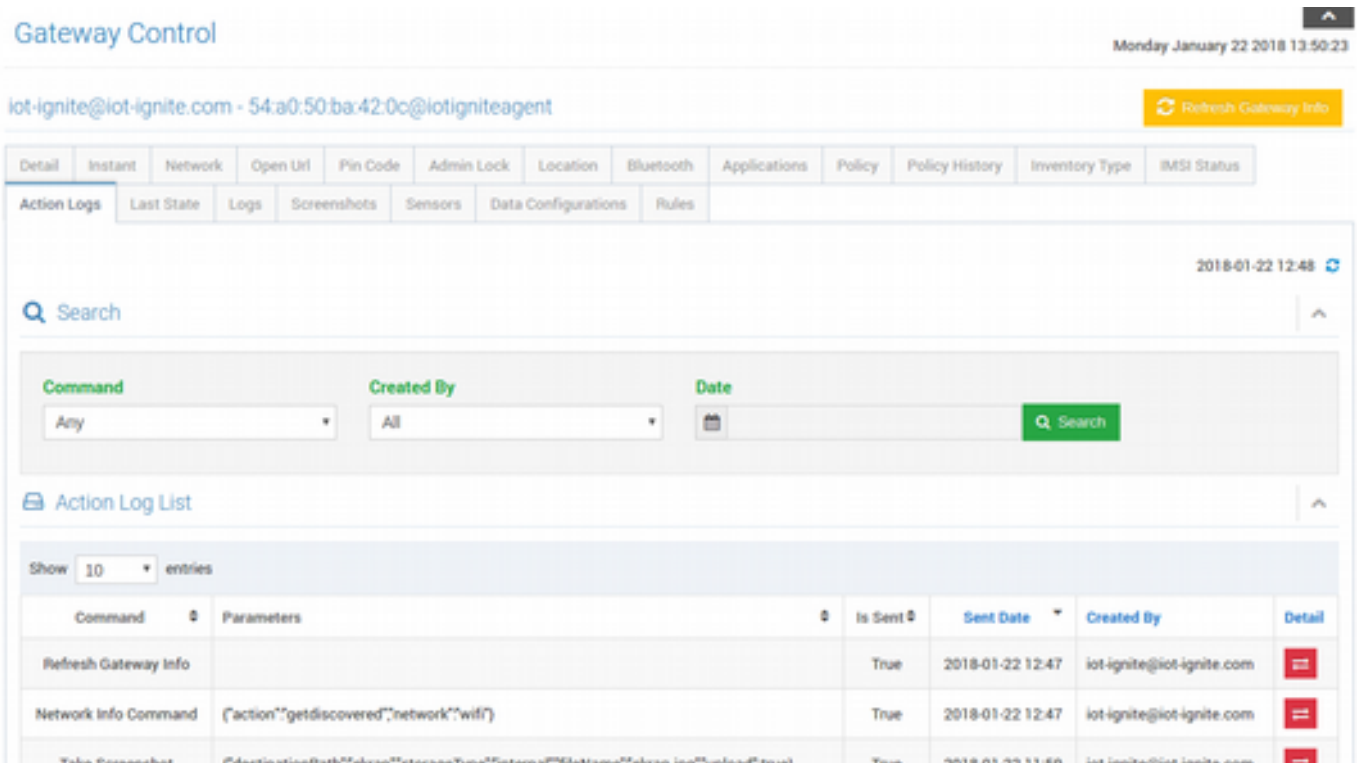
Mark Reserve: Gateway, yedek olarak işaretlenir. Gateway, kullanıcı ile olan bağlantısını kaybedecektir, varsayılan ayarlarına döndürülecek ve bilgileri silinecektir.

- **IMSI State (IMSI Durumu)**



Gateway'de takılı olan SIM Card'ları listeler. Listeden seçili olanı **Accept Selected** ile aktif olarak yapabilir veya **Reject Selected** ile de çıkarabilirsiniz.

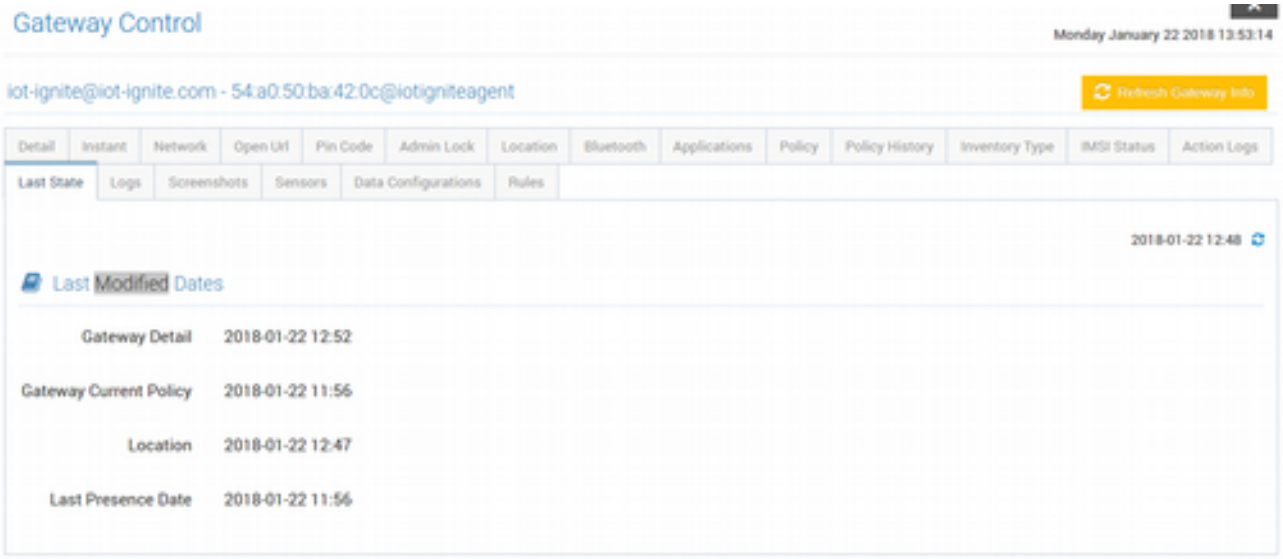
- **Action Logs (İşlem Listesi)**



Aksiyon kayıtlarını ve detaylarını görebileceğiniz bir liste açar (Devzone'da da görmüştük). Listeyi

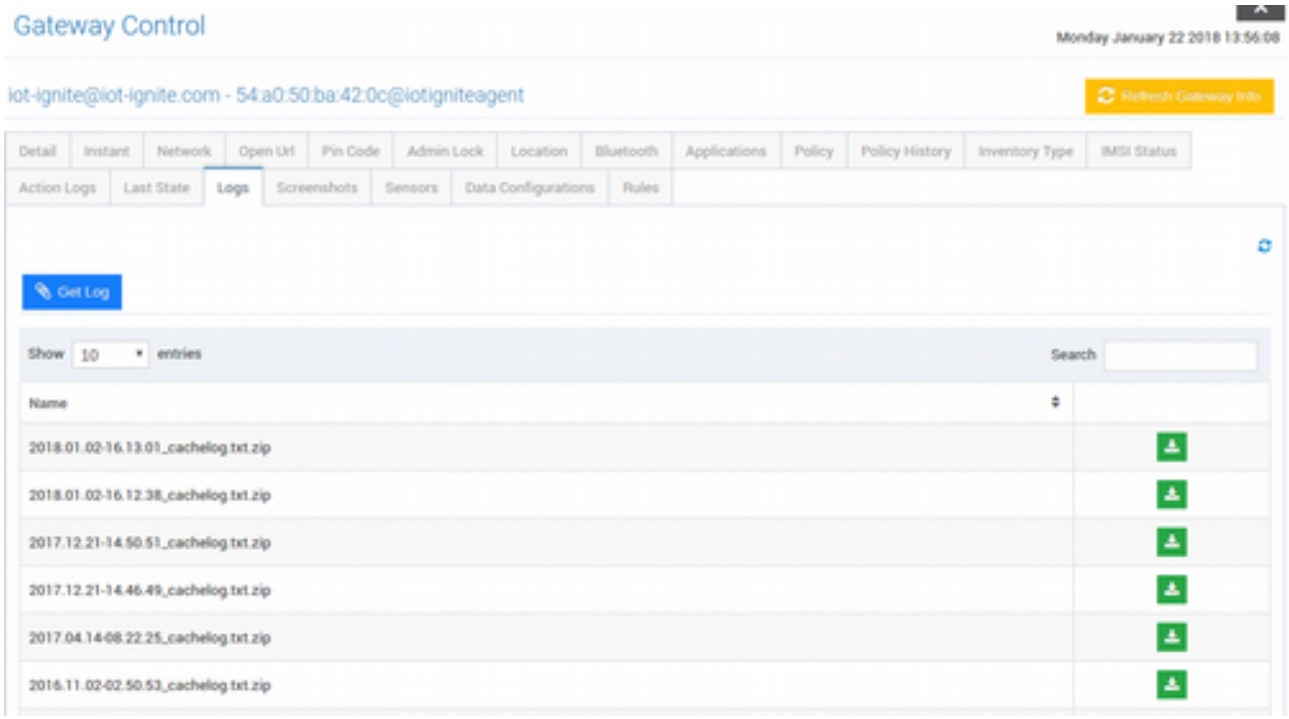
herhangi bir komut, End User veya tarihe göre filtreleyebilirsiniz. Her bir aksiyon kaydının detaylarını görmek için **Detail** (Detay) butonlarına tıklanabilir.

- **Last State (Son Durum)**



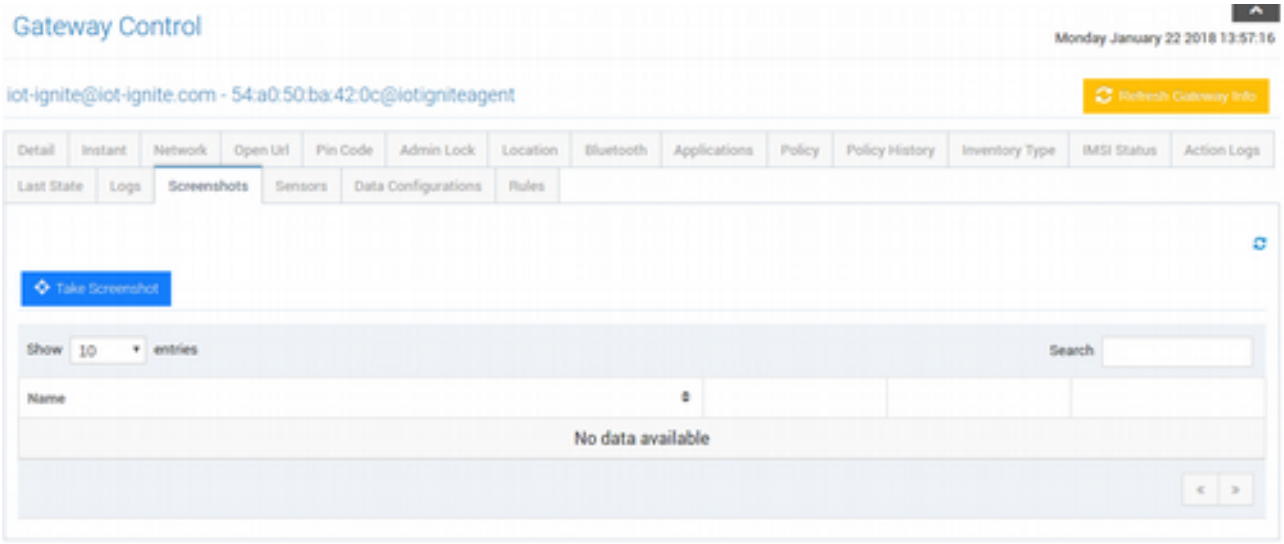
Gateway ile işlem yapılan işlemlerin son tarihlerini verir.

- **Logs (İşlem / Olay Listesi)**



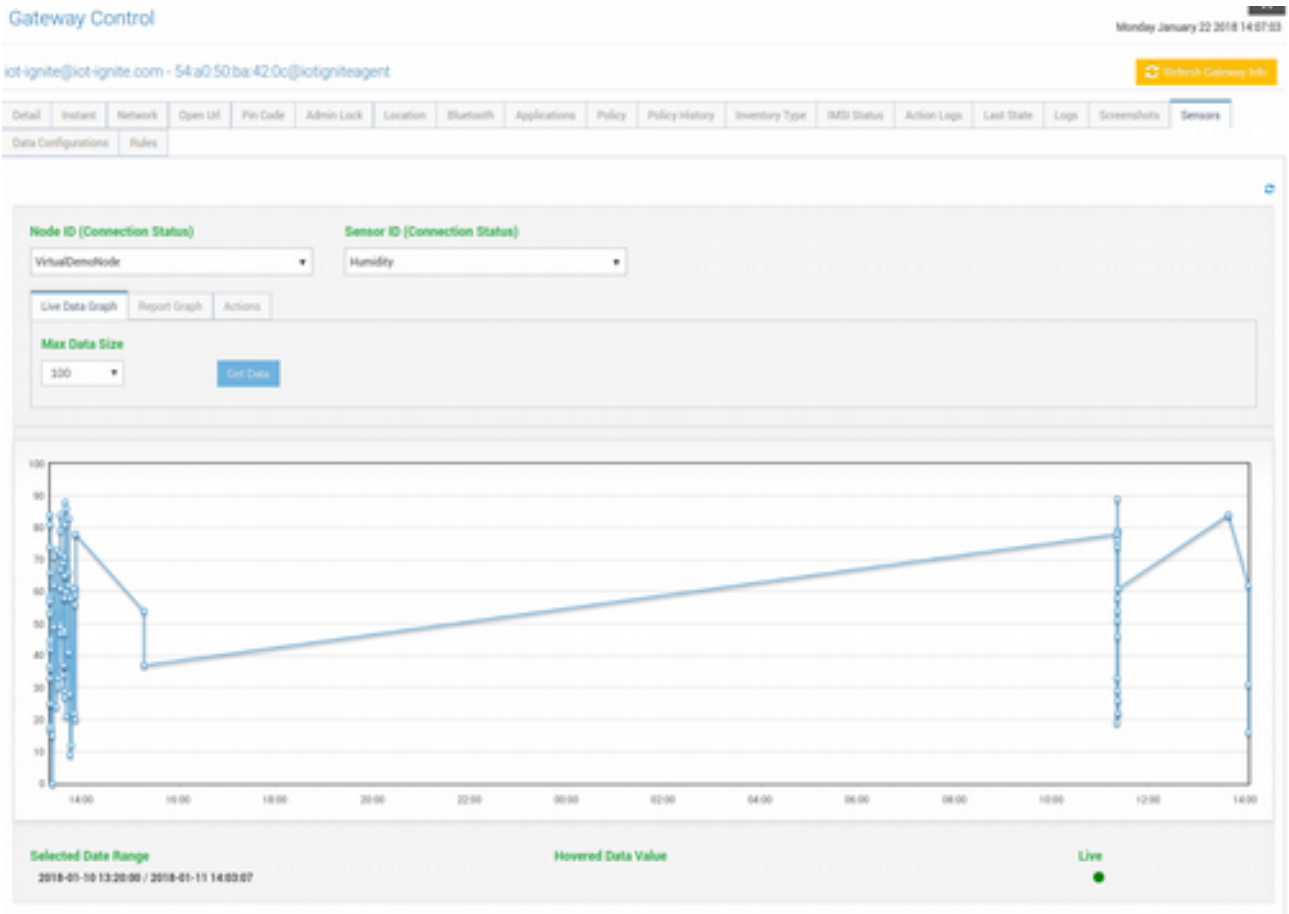
Log dosyalarını görebileceğiniz bir liste açar (Devzone'da da görmüştük). Her bir log kaydını incelemek **Download** (İndir) butonlarına tıklayarak **ZIP** formatlı log'u indirebilirsiniz. Eğer yeni bir log isteniyorsa, **Get Log** (Log Getir) butonuna tıklanabilir. Bu işlem biraz zaman alacaktır. **Refresh** butonuna da tıklayıp güncel Log listesi tekrar gösterilebilir.

- **Screenshots (Ekran Görüntüleri)**



Eğer Gateway **AFEX** yüklü bir Gateway ise; ekran görüntüsü alınabilir ve alınan ekran görüntüleri de bir liste içinde gösterilebilir.

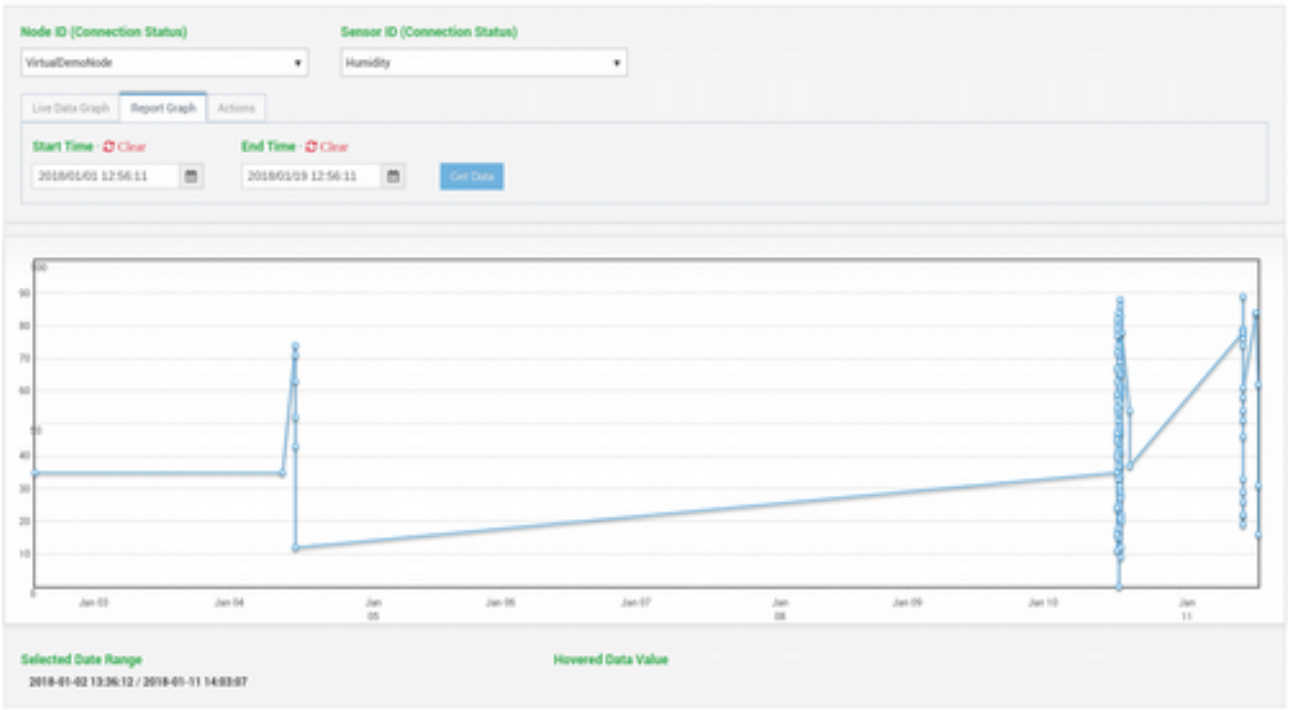
- **Sensors (Sensörler)**



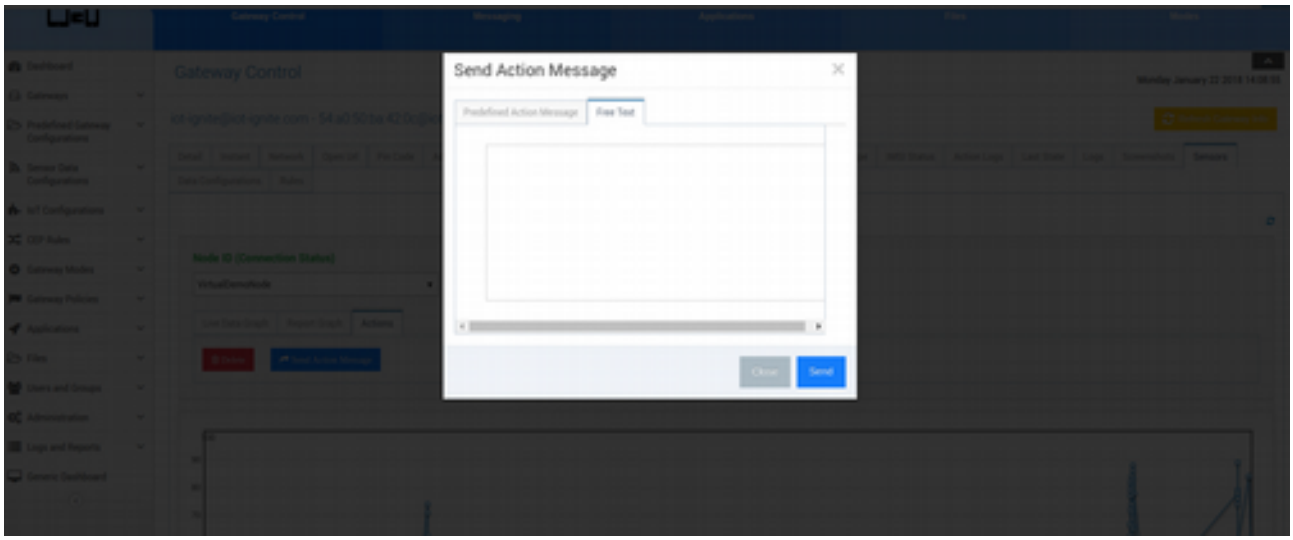
Gateway'e ait **Node** ve **Sensör**'leri listeleyen bir arayüzdür. Listede o an bağlı olan Sensör'ler yeşil, bağlı olmayanlar kırmızı, belirsiz olanlar da gri renkli görünür.

Live Data Graph (Canlı Veri Grafiği) sekmesindeyken listeden seçilen Sensöre'ün geçmiş verileri ile birlikte gerçek zamanlı olarak verileri bir çizgi grafikte gösterilir.

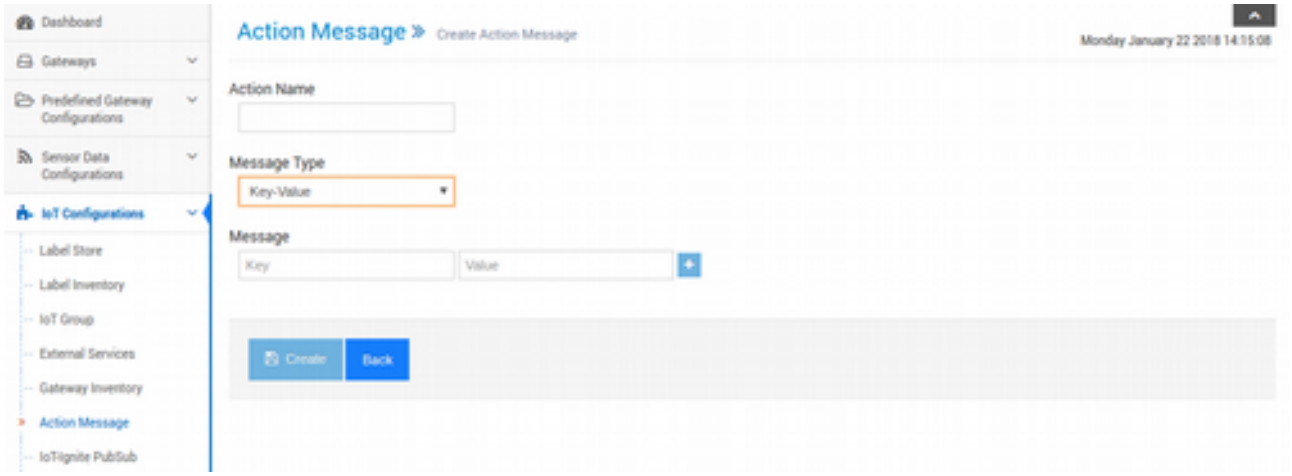
Eğer belirli bir döneme ait veriler görüntülenmek isteniyorsa, da **Report Graph** (Rapor Grafiği) sekmesine girilip başlangıç ve bitiş tarihleri seçilerek o tarihler arasında kalan veriler gösterilebilir.



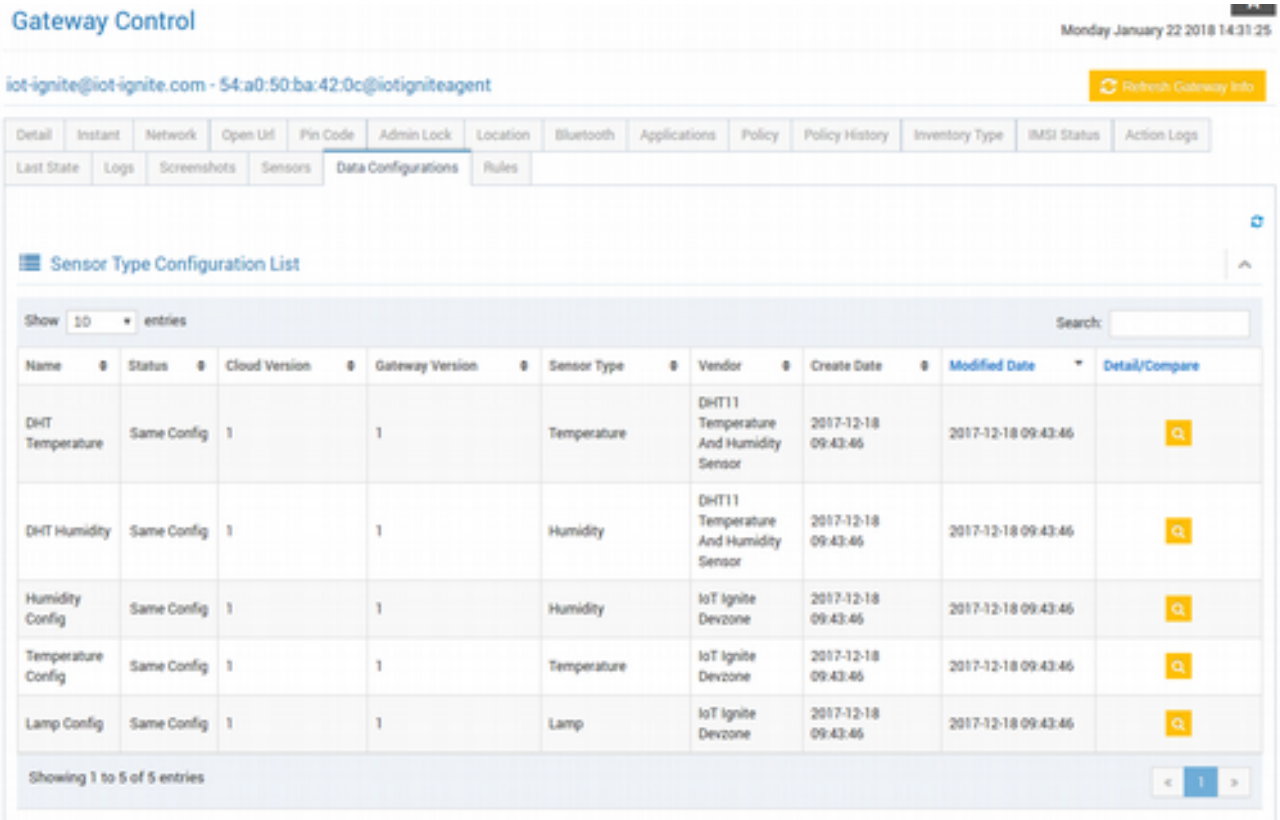
Eğer o sensöre bir aksiyon mesajı gönderilmek isteniyorsa, **Actions** (Aksiyonlar) sekmesine girilip **Send Action Message** (Aksiyon Mesajı Gönder) butonuna tıklanarak ya daha önceden tanımlı olan bir aksiyon mesajı, ya da serbest metin alanına girilecek olan aksiyon mesajı Gateway'e **Send** (Gönder) butonu ile gönderilebilir.



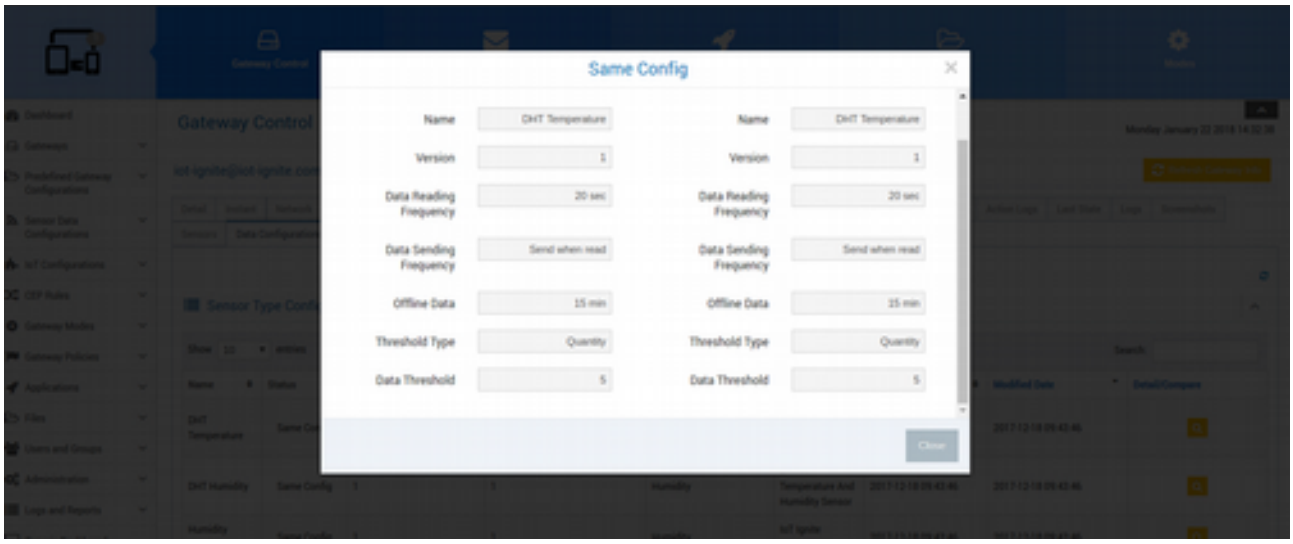
Aksiyon mesajları, menüden **IoT Configurations > Action Message** menüsü altında yer almaktadır. Burada aksiyon için bir **isim**, **mesaj türü** (key-value, yani JSON veya serbest metin) seçilir. Eğer Key-Value mesaj türü seçilirse Key ve Value'ları girebileceğiniz araçlar çıkar. Eğer serbest metin girilecekse de mesajı gireceğiniz bir metin kutusu çıkar. **Create** (Oluştur) ile tanımladığınız aksiyon mesajı, daha sonrasında Sensör (Actuator)'lere push edilebilir.



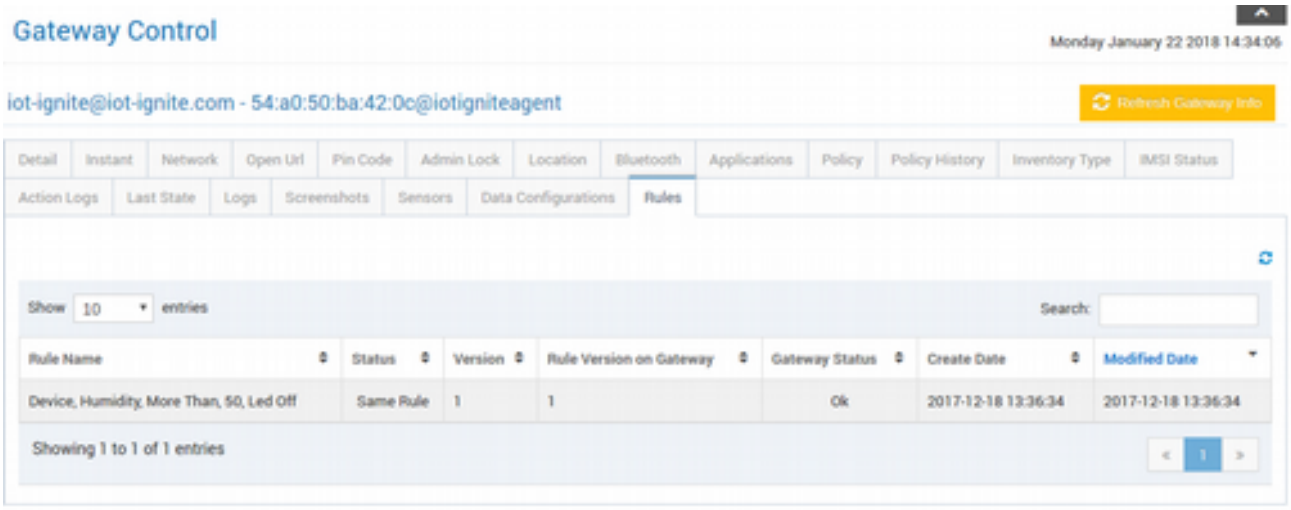
- **Data Configurations (Veri Yapılandırmaları)**



Gateway'de var olan sensör türü yapılandırmalarını listeler. Detay butonlarına tıklandığında, her bir veri yapılandırmasının hem **Gateway** hem de **Cloud** tarafındaki durumu karşılaştırılarak gösterilir.



- **Rules (Kurallar)**



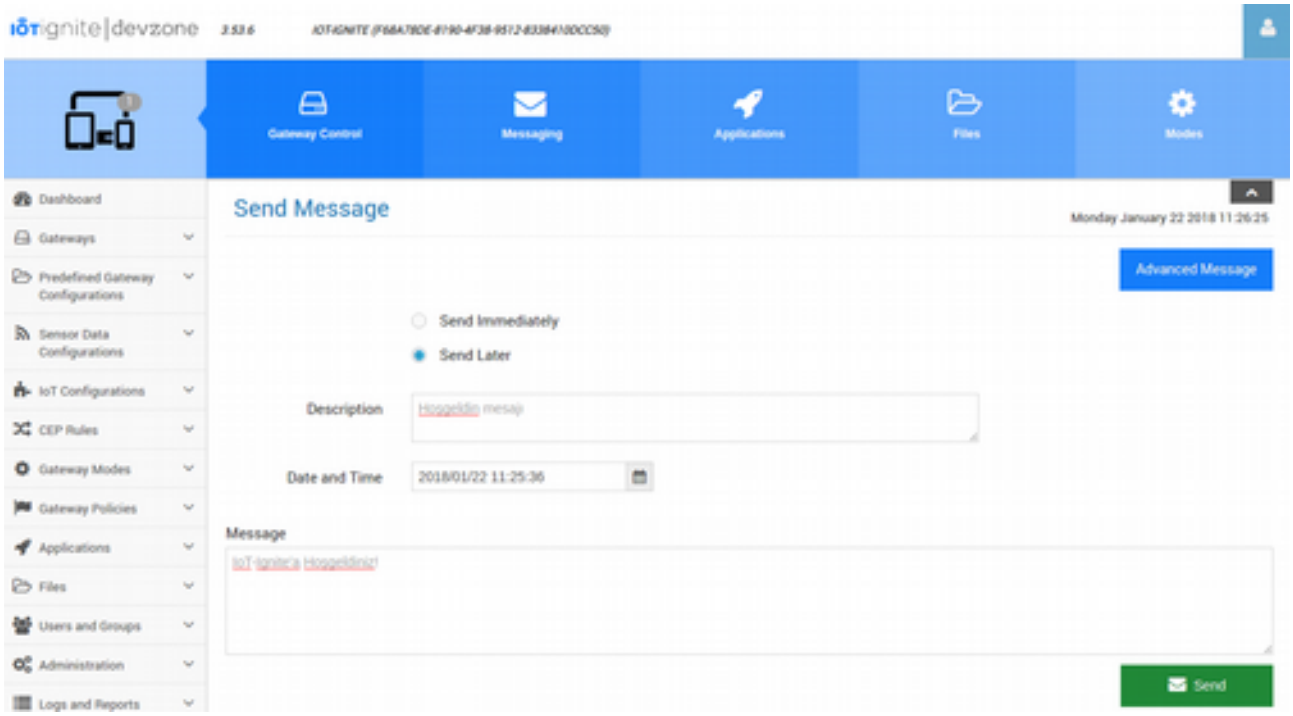
Gateway için tanımlanmış olan Gateway kurallarını listeler.

Gateway'lere Mesaj Göndermek

Working Set'teki Gateway'lere anlık veya ileri bir tarihte mesaj gönderebilirsiniz. Göndereceğiniz mesaj Gateway'in ekranında (varsa) görünecektir.

Working Set alanından Messaging (Mesajlaşma) butonuna tıklayın. Gelen arayüzden iki seçenekten birini seçmeniz beklenir. Bunlar;

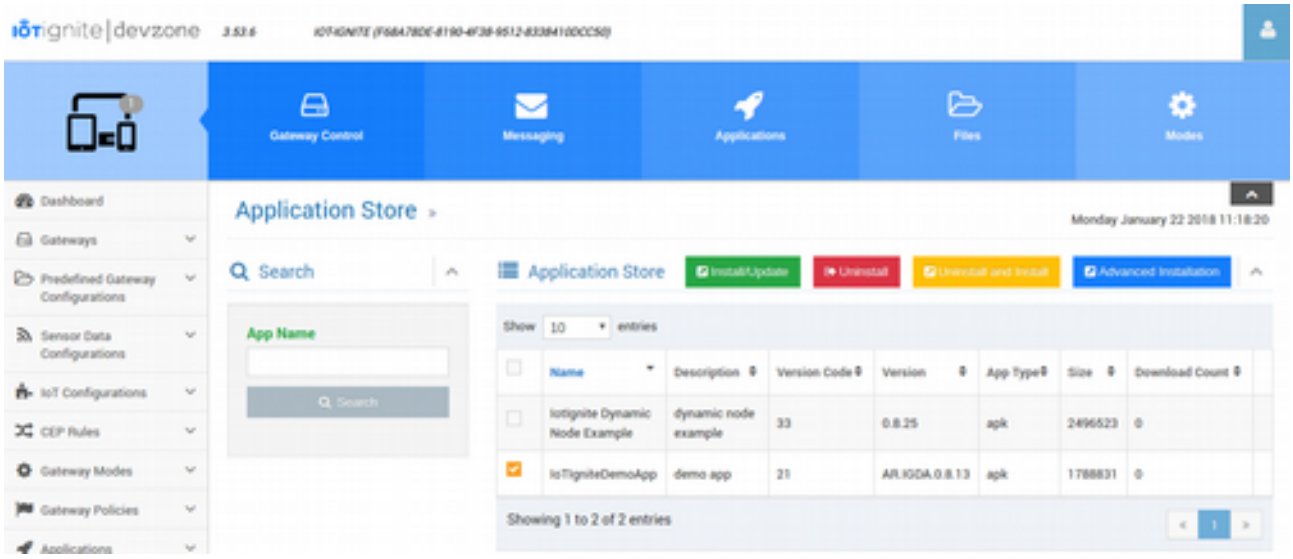
- **Send Immediately**: Mesajı hemen göndermenizi sağlar.
- **Send Later**: Mesajı, belirleyeceğiniz bir tarihte göndermenizi sağlar.



Uygulamaları Yönetmek

Working Set'teki Gateway'lerde yer alan uygulamaları (APK) güncelleyebilir, kaldırabilir, kaldırıp tekrar yükleyebilir veya harici kaynaklardan yükleme yapabilirsiniz.

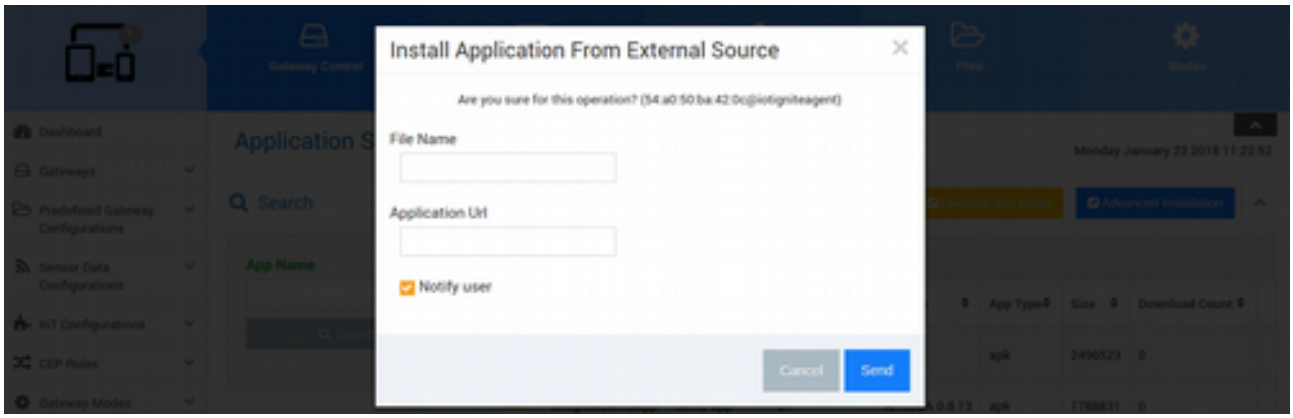
Working Set alanından **Applications** (Uygulamalar) butonuna tıklayın. Gelen arayüzde App Store'da yer alan uygulamalar listelenecektir.



Listenin üst kısmında yer alan butonlar ve görevleri:

- **Install/Update**: Listeden seçili olan uygulamayı Working Set'teki Gateway'lere gönderir. Uygulamayı alan Gateway'de eğer uygulama yüklü değilse yüklenir. Eğer Gateway'de zaten yüklü ise, gönderilen uygulamanın versiyonu Gateway'dekinden daha yüksekse, uygulama güncellenir.
- **Uninstall**: Working Set'teki Gateway'lerde eğer seçili olan uygulama yüklü ise, kaldırılır.

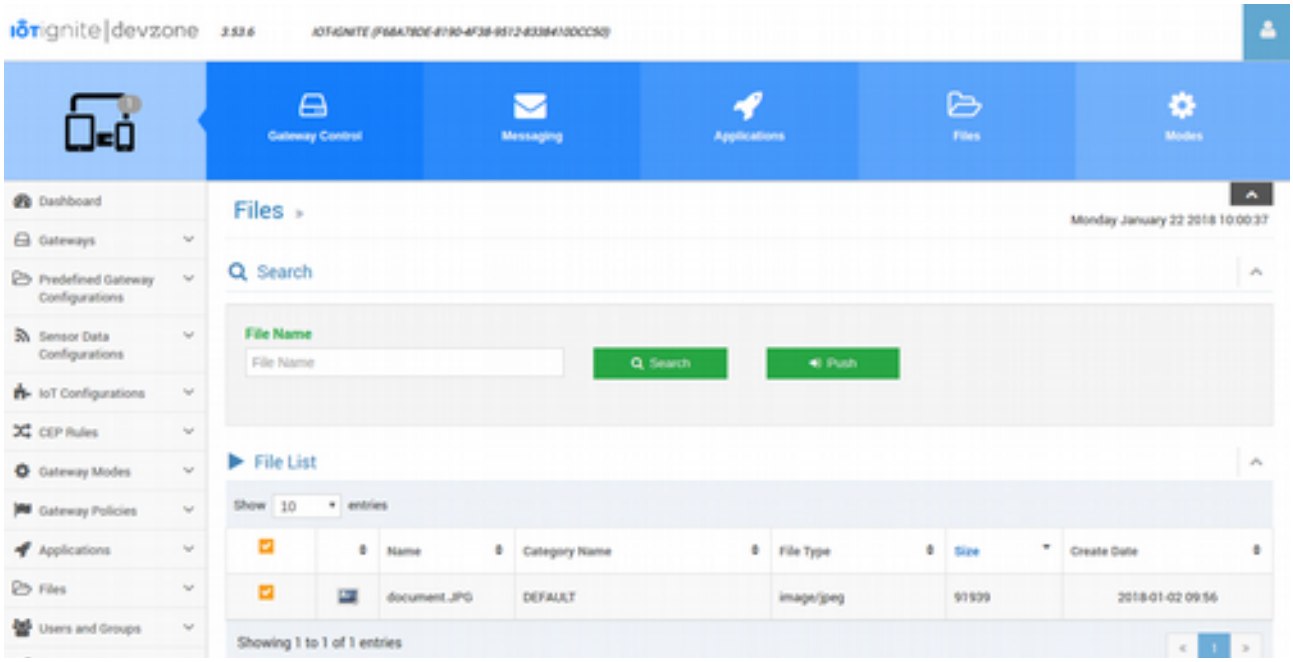
- **Uninstall and Install**: Listeden seçili olan uygulamayı Gateway'den kaldırır (yüklüyse) ve uygulamanın gönderilen sürümünü yükler. Install/Update işleminden farkı; bu işlemde versiyon numarası düşük olsa bile yükleme yapılabilmesidir. Yani eski versiyona geri dönüş yapılabilmesini sağlar.
- **Advanced Installation**: Servisinizin uygulama dükkanında yer almayan, harici bir kaynaktaki uygulamayı Gateway'e yükleyebilmenizi sağlar. Bu butona tıklandığında, karşınıza basit bir form gelecektir. Form'da uygulamanın adını ve URL'ini girip **Send** (Gönder) butonuna tıkladığınızda o uygulama Gateway'e indirilip kurulacaktır. Eğer **Notify User** (Kullanıcıyı Bilgillendir) opsiyonu seçili ise, kurulumdan önce kullanıcı onayı alınacaktır. Eğer Gateway'lerde **AFEX** yüklü ise; kullanıcı onayı alınmadan direkt kurulum yapılabilir.



Dosya Göndermek

Working Set'teki Gateway'lere toplu olarak içerik dükkanındaki (Content Store) yer alan dosyaları gönderebilirsiniz.

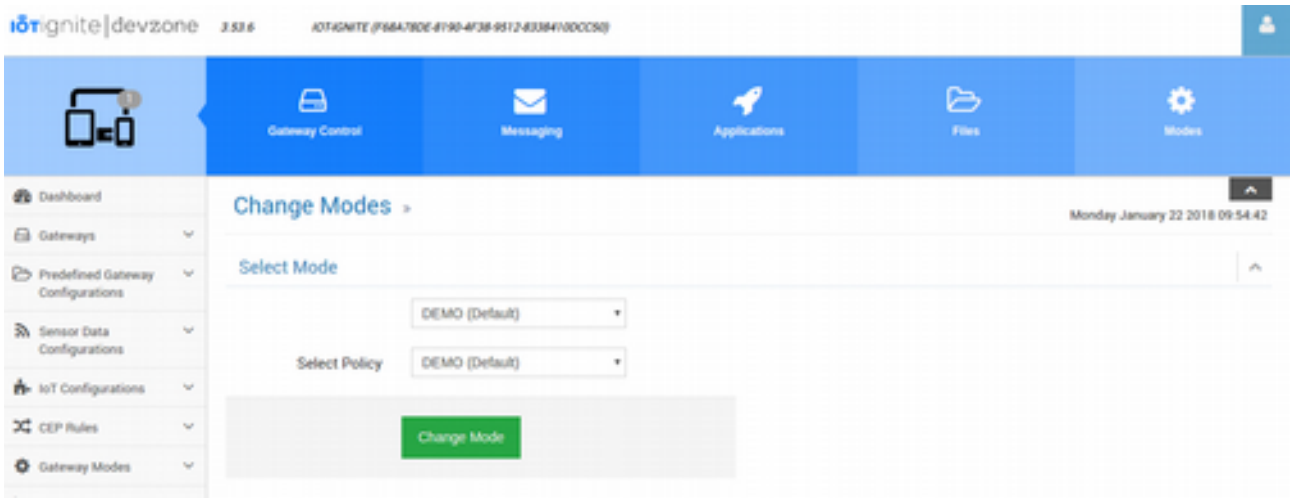
Working Set alanından **Files** (Dosyalar) butonuna tıklayın. Gelen arayüzde **File Store**'da yer alan dosyalar listelenecektir. Listeden, Working Set'teki Gateway'lere göndermek istediğiniz dosyaları seçin ve ardından Push (Gönder) butonuna tıklayın.



Gateway’lerin Mode’larını ve Policy’lerini Değiştirmek

Working Set’te yer alan Gateway’lerin Mode’larını ve Mode’a bağlı olarak Policy’leri toplu olarak basitçe değiştirebilirsiniz.

Working Set alanından **Modes** (Modlar) butonuna tıklayın. Gelen arayüzden de önce Mode, sonra da Mode’a bağlı olan Policy’yi seçin ve **Change Mode** (Mod Değiştir) butonuna tıklayın. İşlem sonunda Working Set’teki Gateway’lerin Mode ve Policy’leri değiştirilecek, Gateway’lere de gönderecektir.



Sensör Veri Tanımlamaları Oluşturmak, Düzenlemek ve Silmek

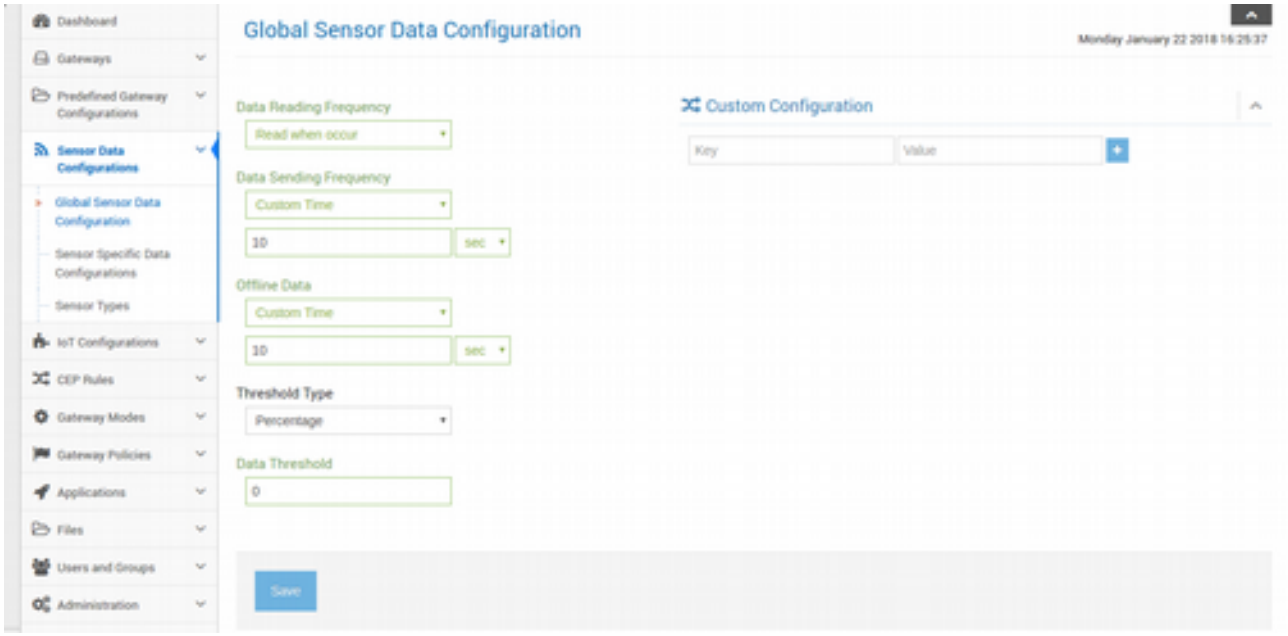
Daha önce Devzone’da Sensör’ler için veri yapılandırmalarını görmüştük. Şimdi de EHUB’ta veri yapılandırmaları ile ilgili olarak daha gelişmiş işlemlere bakalım...

Genel Sensör Veri Tanımlamaları

Hatırlayacağınız üzere her bir sensör için farklı farklı veri yapılandırmaları yapmıştık. Date Time ve Geolocation gibi özel sensörler dışında kalan genel yapılı sensörler için varsayılan olarak ortak kullanılabilecek bir veri yapılandırması tanımlayabiliriz. Böylece veri yapılandırılması yapılmamış olan sensörler de Cloud’a veri gönderebilir hale gelecektir.

> Genel sensör veri yapılandırmaları yaparken oldukça dikkatli olmak gerekmektedir! Çünkü bütün sensörler aktif olarak Cloud’a veri göndereceği için Gateway’in enerji tüketimi ve ağ trafiği fazla olacaktır. Bu nedenle genel sensör veri yapılandırması yaparken buna gerçekten ihtiyacınız var mı, varsa da ortalama olarak ne kadar bir sürede veri göndermeniz gerektiğini, verilerin saklanıp saklanmayacağını dikkatlice düşünmelisiniz.

Genel sensör veri yapılandırması yapmak için menüden **Sensor Data Configuration**s (Sensör Veri Yapılandırmaları) > **Global Sensor Data Configuration** (Genel Sensör Veri Yapılandırması) butonuna tıklayın ve ilgili arayüze girin.



Genel yapılandırma ayarını, sensörlerden veri geldikçe oku ve 10 saniyede bir buluta gönder şeklinde tanımlayıp **Save** (Kaydet) butonu ile kaydettik. Bu yapılandırma sonucunda aşağıdaki gibi bir Version bilgisi ile yapılandırma isimlendirilecek. Tanımlanan bu yapılandırma ayarları, **bütün Mode'larda ortak** kullanılmaktadır.

Eğer daha sonraları bu genel yapılandırma ayarlarını değiştirmek isterseniz, aynı sayfada parametreleri yeniden düzenleyip **Update** (Güncelle) butonu ile güncelleyebilirsiniz.

Genel yapılandırma ayarlarını Gateway'lerde kullanmak için iki farklı yöntemimiz var.

Birincisi; güncellenen Policy ile birlikte Mode'u Gateway'e göndermek gerekecektir. **Gateways > All Gateways** sayfasına girip güncel Policy'i hangi Gateway'lere göndermek istiyorsak onları seçip **Working Set**'e aktarmamız gerekiyor. Sonrasında da üst kısımda açılacak olan mavi alanda **Modes** sekmesine tıklayıp, listeden güncel Mode'u Working Set'teki Gateway'lere gönderebilirsiniz.

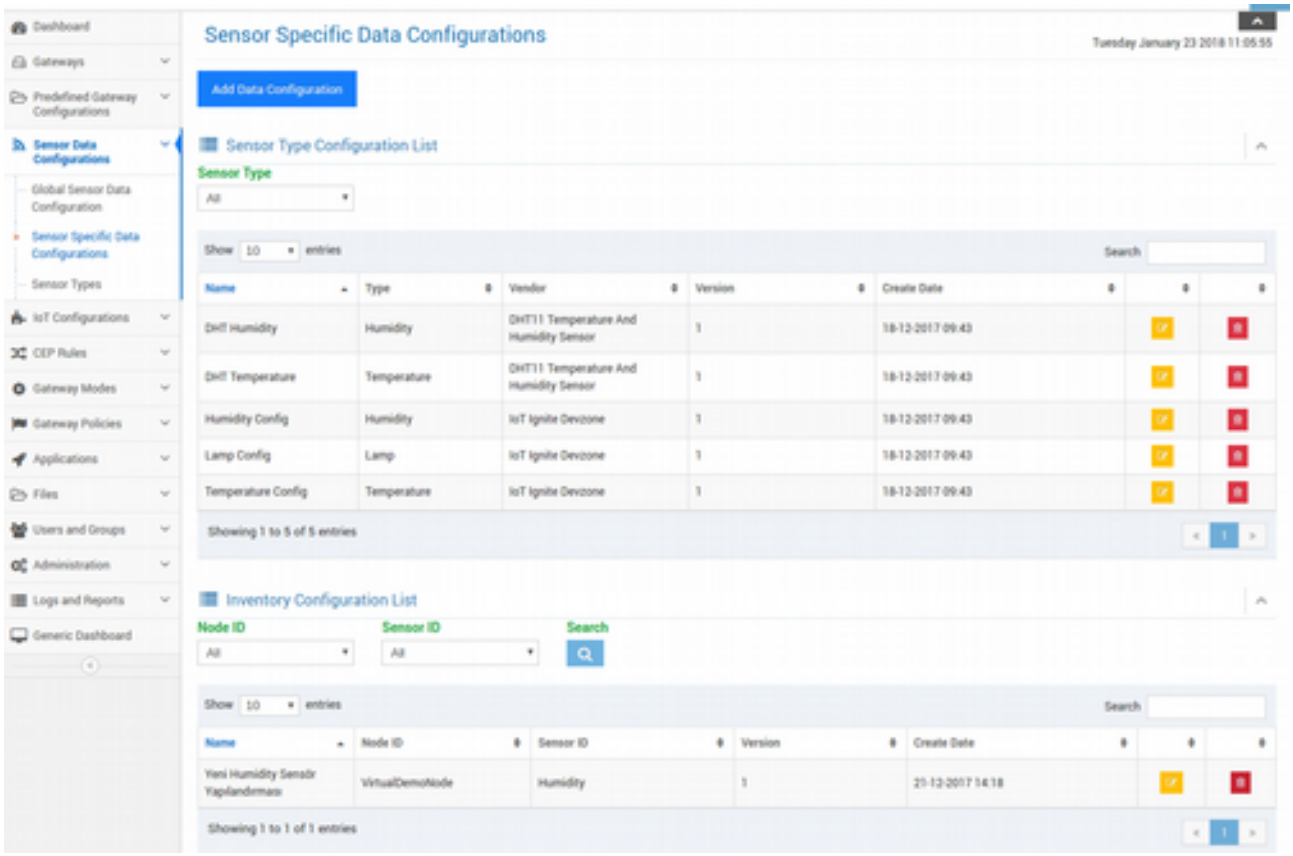
İkinci yöntemde ise; Gateway'de yüklü olan **Ignite Agent** uygulamasını açıp, menüden **Update** (Güncelle) butonuna tıklayıp güncelleme işlemini başlatmaktır. Yapılan bu işlemle, Cloud'ta Policy güncellenmiş ise güncel Policy'i Gateway'in kendi kendine alması sağlanır.

Birinci yöntemde çoklu olarak Gateway'lere gönderilmesi sağlanırken, ikinci yöntemde her bir Gateway'in ayrı ayrı Policy güncelleme işlemini gerçekleştirmesi gerekmektedir.

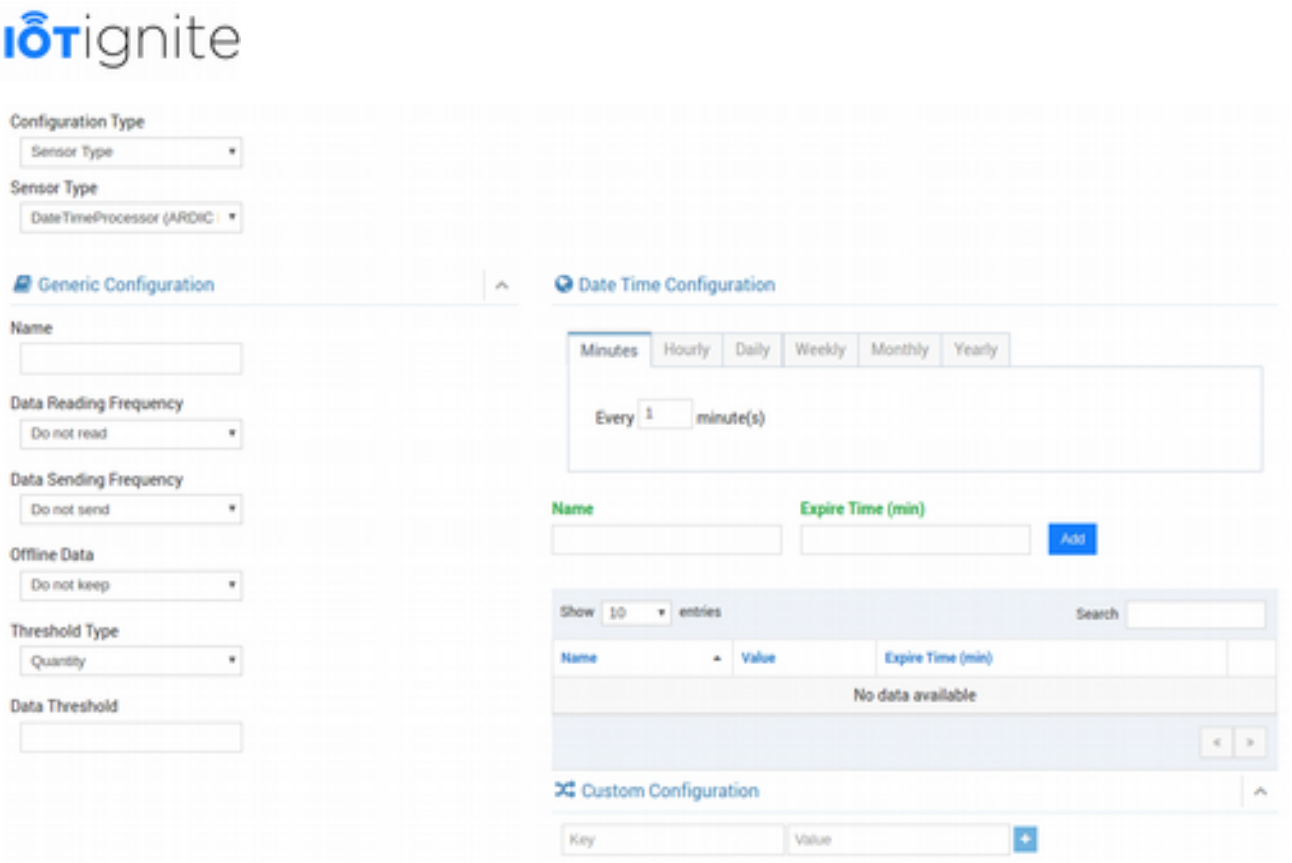
Sensöre Özel Veri Yapılandırmaları

Devzone'da çalışırken Sensör'ler için veri yapılandırmaları konusunu görmüş, EHUB'ta da yapılan bu işlemlerin sonucunu incelemiştik. Devzone'da bir sensör için yapılan veri yapılandırmaları, aynı

şekilde **Sensor Data Configurations** (Sensör Veri Yapılandırmaları) > **Sensor Specific Data Configurations** (Sensöre Özel Veri Yapılandırmaları) sayfasından da yapılabilir.



Add Data Configuration (Veri Yapılandırması Ekle) butonuna tıkladığınızda, açılan formda Node ve Sensör seçip veri yapılandırması yapabilirsiniz.



Ayrıca, daha önce oluşturulmuş olan veri yapılandırmaları bir listede görebilir, istediğinizi yeniden düzenleyebilir veya silebilirsiniz.

Sensör Tipleri

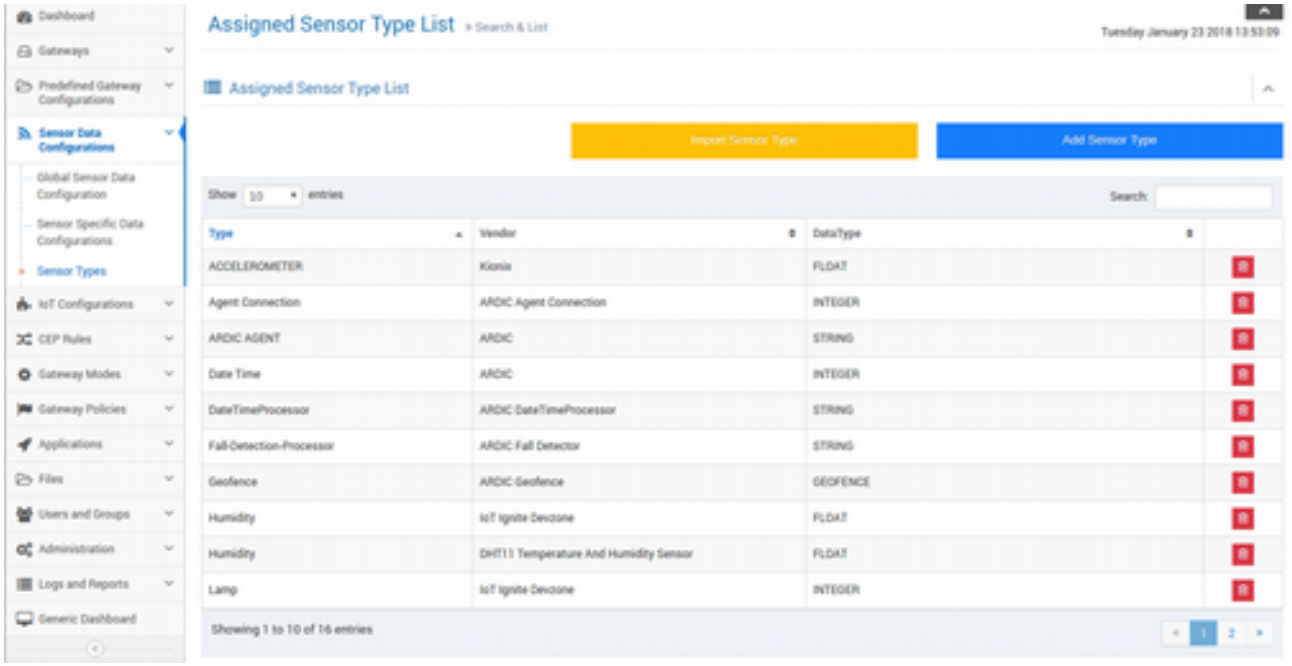
Gateway'lerde yer alan Sensor ve Actuator'lerin mutlaka Ignite Cloud'ta bir karşılığını bulunması gerekmektedir. Bu şekilde, Gateway'den gelen veri yığınları içinden hangi verinin hangi sensör'e ait olduğu tanımlanabilir.

Devzone'dan üyelik açtığınızda varsayılan olarak bir Android Gateway'de yer alacak temel sensör'ler ve veri tipleri tanımlı halde gelmektedir. Bunlar aşağıdaki tablodaki gibidir.

Sensör Tipi	**Vendor (Üretici Firma)**	**Veri Tipi**
ACCELEROMETER	Kionix	FLOAT
Agent Connection	ARDIC Agent Connection	INTEGER
ARDIC AGENT	ARDIC	STRING
Date Time	ARDIC	INTEGER
DateTimeProcessor	ARDIC DateTimeProcessor	STRING
Fall-Detection-Processor	DARDIC Fall Detector	STRING
Geofence	ARDIC Geofence	GEOFENCE
Humidity	IoT Ignite Devzone	FLOAT
Humidity	DHT11 Temperature And Humidity Sensor	FLOAT

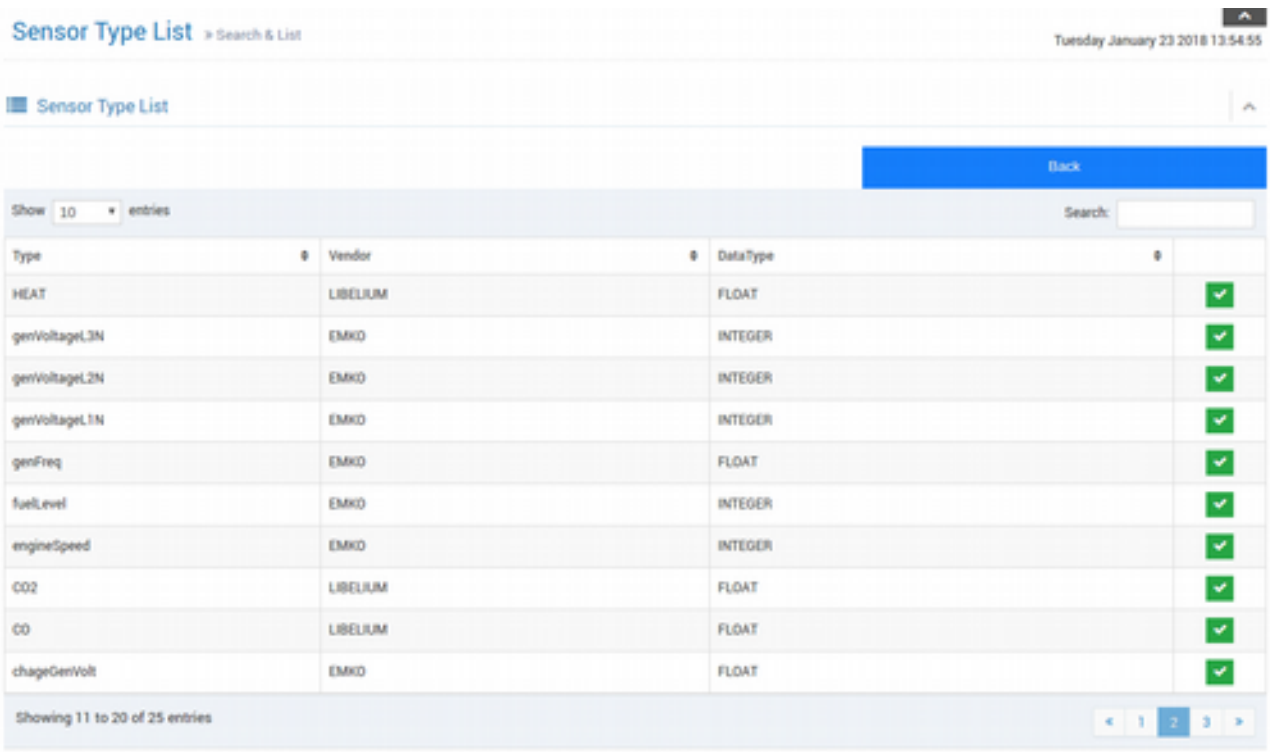
Lamp	IoT Ignite Devzone	INTEGER
MAGNETIC_FIELD	Asahi Kasei Microdevices	FLOAT
ORIENTATION	Asahi Kasei Microdevices	FLOAT
ROTATION_VECTOR	Asahi Kasei Microdevices	FLOAT
Temperature	IoT Ignite Devzone	FLOAT
Temperature	HDT11 Temperature And Humidity Sensor	FLOAT
WiFi	ARDIC Wifi	STRING

Menüden Se**nsor Data Configurations > Sensor Types** (Sensör Tipleri)'ine girdiğinizde, yukarıdaki tabloda yer alan Sensör türü tanımlamalarını listede göreceksiniz. Bunlardan herhangi birini sildiğinizde, Gateway'den o sildiğiniz sensöre ait veriler geldiğinde, bulut tarafında işlenmez.



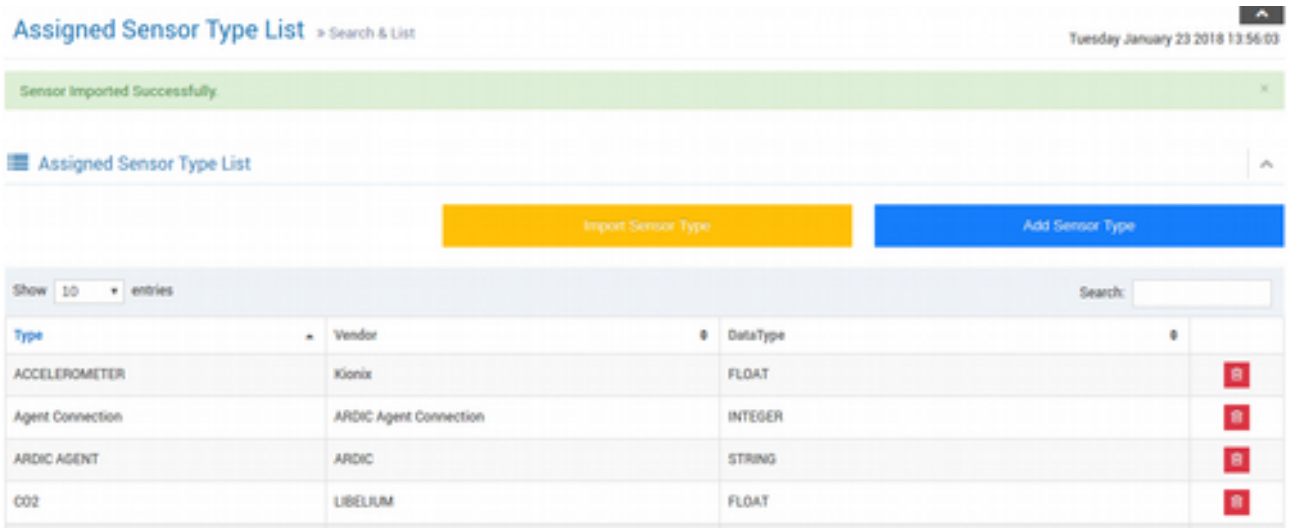
Listede yer alan sensörler dışında başka bir sensör kullanmak istediğinizde (Özellikle MQTT ile Raspberry, Arduino ile çalışırken sıklıkla ihtiyaç duyacaksınız) **sanal veya gerçek** sensörleri bu arayüz ile tanımlamanız gerekmektedir. **MQTT dışında ise Gateway tarafında zaten sensör tipini ve sensörü (node) tanımlayacağınız için bulut ortamında (EHUB'ta) yeniden tanımlamanıza gerek yok** (İlerleyen bölümlerde uygulamalı olarak bu konuyu göreceksiniz).

Eğer tanımlamak istediğiniz sensör sıklıkla kullanılan ve bilindik bir sensör ise **Import Sensor Type** (Sensör Türü Al) butonuna tıklayıp, açılan listede eklemek istediğiniz sensörü bulup **check** butonuna tıklayarak ekleyebilirsiniz.

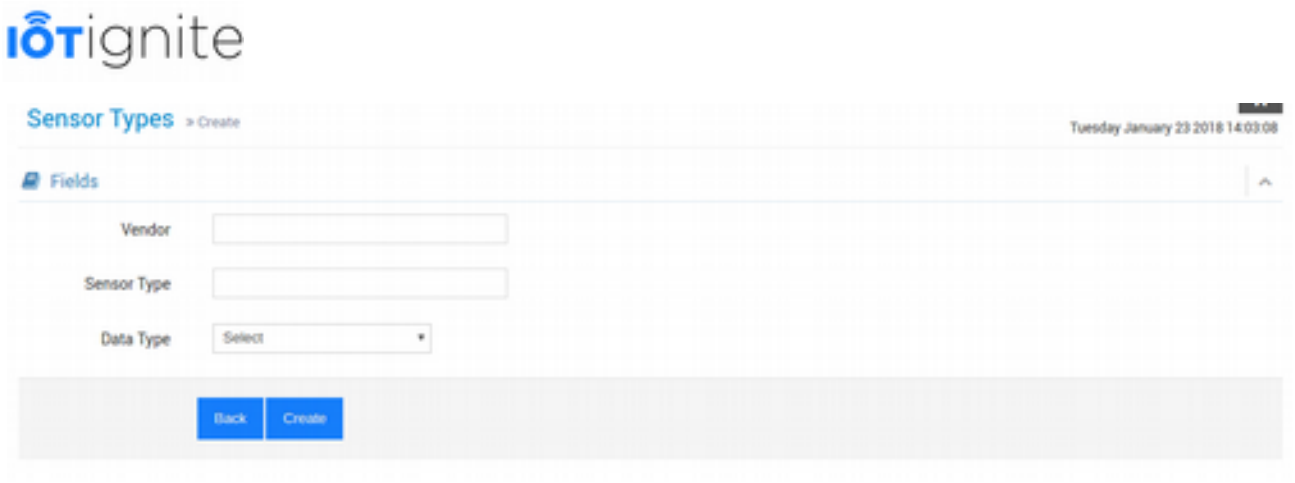


Örneğin; **LIBELIUM** firmasına ait **CO2** sensörünü **FLOAT** veri tipi ile ekleyelim...

Ekleme işleminden sonra tekrar Sensör Tipleri sayfasına geçilecek. Listede artık CO2 sensörümüz de görünecektir.



Eğer aradığınız sensör listede yoksa veya kendiniz bir sanal sensör oluşturacaksanız (hatta listede olsa bile kendiniz de tanımlayabilirsiniz), **Add Sensor Type** (Sensör Tipi Ekle) butonuna tıklayıp, açılan formda **Vendor**, **Sensor Type** ve **Data Type** alanlarını doldurup **Create** (Oluştur) butonuna tıklamanız yeterlidir.



Burada tanımlayacağınız özel sensörünüzün aynı vendor ile aynı sensor ismiyle ve aynı veri tipi ismiyle Gateway'de de sensor tanımlamaları yapılmalıdır. Yani Gateway ve bulut bu üç bilgi ile eşleşmelidir. Gateway'de nasıl sensör tanımlanacağını Devzone bölümü incelerken kısaca görmüştük.

Demo APP'i Git'ten klonlayıp VirtualDemoNode'ları altında yer alan Temperature, Humidity ve Lamp sensörlerinin tanımlamalarıyla, Cloud'taki tanımlamaları karşılaştırabilirsiniz.

```
git clone https://github.com/IoT-Ignite/android-example-IoTIgniteDemoApp.git
```

android-example-IoTIgniteDemoApp/app/src/main/java/com/ardic/android/iotignitedemoapp/

VirtualDemoNodeHandler.java

304..306 satır aralığına bakıldığında Sensor ve Actuator'lerin tanımlamaları görülmektedir.

```
303     private void initIgniteVariables() {
304         mTempThingType = new ThingType("Temperature", "IoT Ignite Devzone", ThingDataType.FLOAT);
305         mHumThingType = new ThingType("Humidity", "IoT Ignite Devzone", ThingDataType.FLOAT);
306         mLampThingType = new ThingType("Lamp", "IoT Ignite Devzone", ThingDataType.INTEGER);
```

```
// Create node and things and register them
private void initIgniteVariables() {

   mTempThingType = new ThingType("Temperature", "IoT Ignite Devzone",
ThingDataType.FLOAT);

   mHumThingType = new ThingType("Humidity", "IoT Ignite Devzone",
ThingDataType.FLOAT);

   mLampThingType = new ThingType("Lamp", "IoT Ignite Devzone",
ThingDataType.INTEGER);


   ...


}
```

Yukarıdaki kodlarda sadece veri tipi tanımlanmıştır. Bunun haricinde her bir Node.IotIgniteManager.NodeFactory.createNode() metodu ile oluşturulmalı ve .register() metodu ile de Cloud'a kaydedilmelidir.

```
Node myNode = IotIgniteManager.NodeFactory.createNode("Security Node", "Home Security Node", NodeType.GENERIC, "HSN1", new NodeListener() {
    @Override
    public void onNodeUnregistered(String s) {
        Log.i(TAG, "Security Node unregistered!");
    }
  });
);
```

```
myNode.register();
```

Ortak olarak **IoT Ignite Devzone** Vendor ismi kullanılmış. EHUB'tan **Vendor** ismine göre sıraladığımızda, yukarıdaki tanılamalar ile buluttaki tanımlamaların eşleşmiş olduğu görülmektedir.

Örneğin listeden Humidity sensörünün Vendor'ü farklı olsa veya sensör ismi yanlış yazılsaydı, bulut tarafında Humidity verilerini karşılayacak bir sistem olmayacaktı.

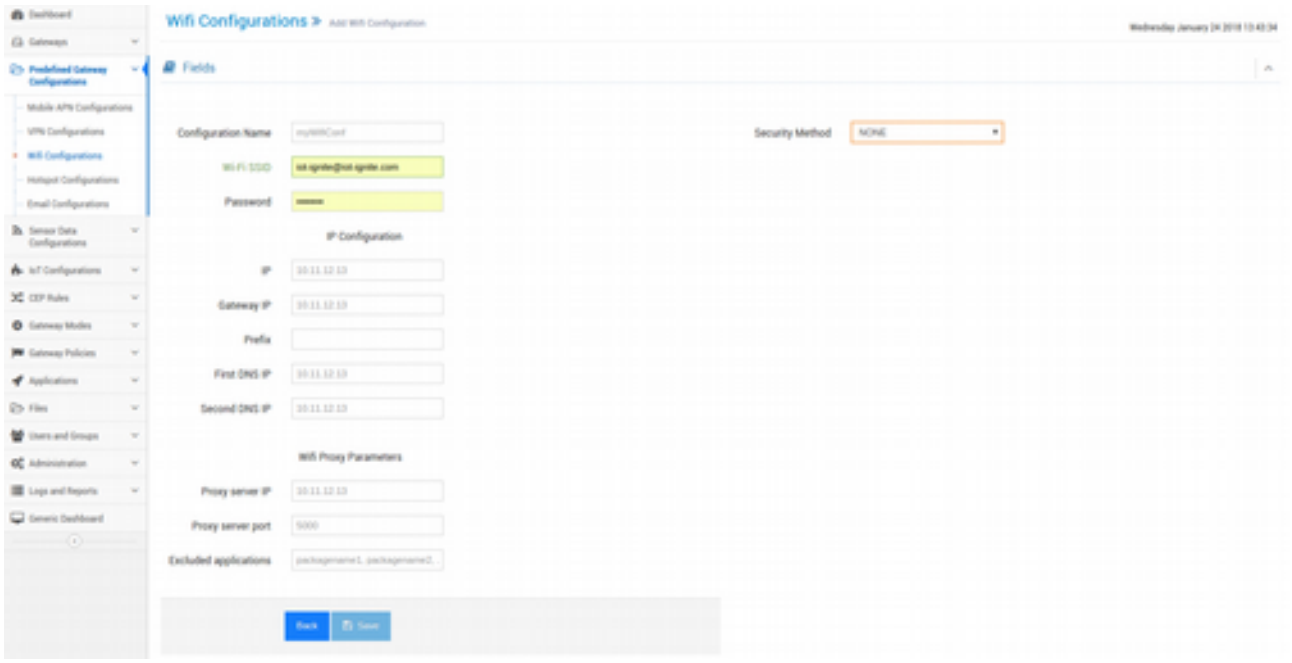
İlerleyen bölümlerde ise artık gerçek donanım ve gerçek sensörlerle çalışırken ilk olarak bu sayfadan sensör tiplerini tanımlamanız gerektiğini unutmayın.

Öntanımlı Gateway Tanımlamaları Oluşturmak, Düzenlemek ve Silmek

Yeni bir Mode oluştururken veya var olan bir Mode'u düzenlerken, söz konusu Mode'da bir takım ayarların Gateway'lere gönderilmesi gerekebilir. Bu ayarlar APN, VPN, Wifi ve Hotspot ayarları olabilmektedir. Şimdi bu ön tanımlı ayarlar nedir, nasıl oluşturulur ve bir Mode'a nasıl alınır sırasıyla görelim...

Wifi

Menüden **Mobile APN Configurations > Wifi Configurations**'a tıklayın. **Create Wifi Configuration** (Wifi Yapılandırması Oluştur) butonuna tıklayın.



Wifi yapılandırma ayarlarına bakıldığında genel olarak bir yapılandırma ismi, **Wifi SSID** ve **parolası** ve **güvenlik** türünü belirteceğimiz alanlar bulunmaktadır. Ek olarak **IP** ve **DNS** yapılandırma ayarları ile birlikte Proxy parametreleri de bulunmaktadır. Bilgileri girdikten sonra bir de bu yapılandırma ayarları için **Configuration Name** (Yapılandırma İsmi) belirleyip **Save** (Kaydet) butonu ile kayıt işlemini tamamlayın.

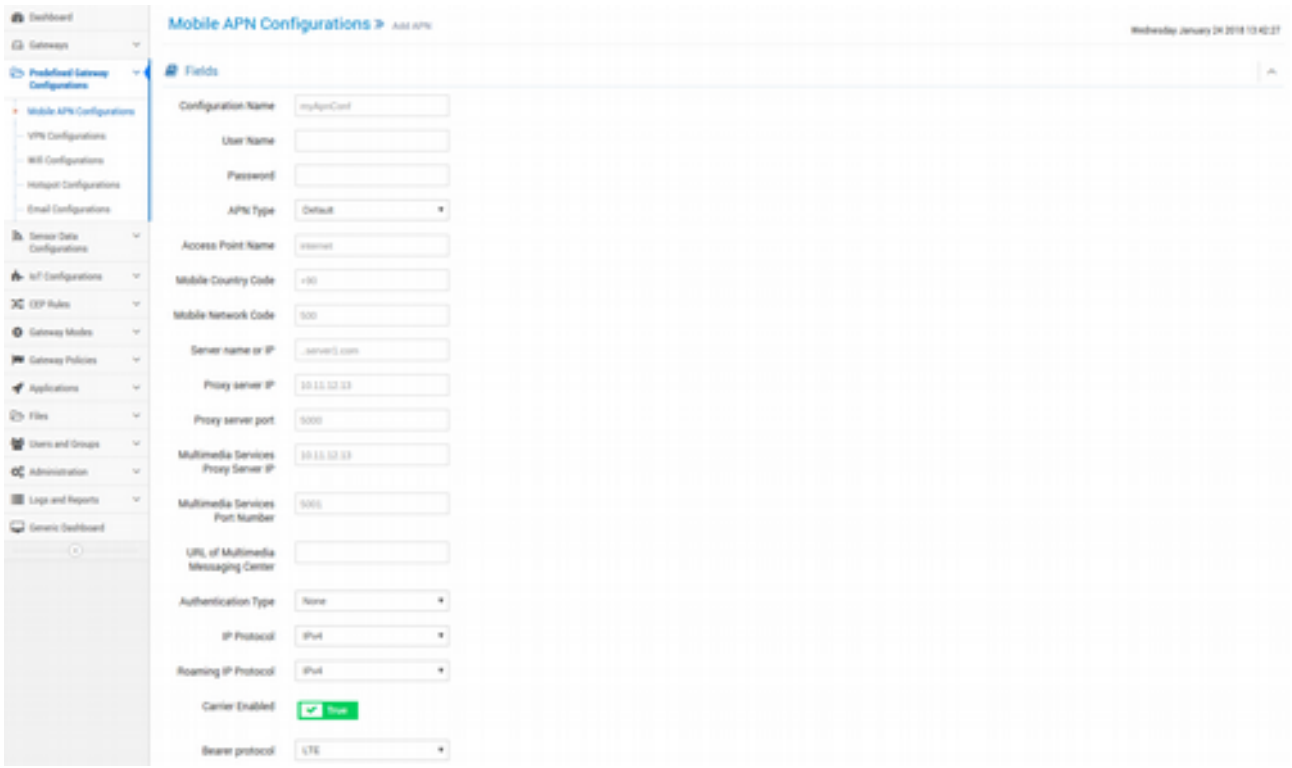
Gateway'in normal şartlarda bir network'e kablolu veya kablosuz olarak bağlı olması gerektiğini biliyoruz. Ancak Gateway'in yeni bir Wifi ağına bağlantı kurmasını istediğimizde bu işlemi ya doğrudan Gateway üzerinden ya da EHUB'tan bir Wifi bağlantı noktası yapılandırması tanımlayıp Gateway'e göndermemiz gerekmektedir. Göndereceğimiz Wifi yapılandırması Gateway tarafından alındığında, Wifi bağlantı noktaları listesinde de otomatik olarak kayıtlı kalacaktır. Sonrasında operatör tarafından gönderilen Wifi noktasına bağlantı kurulabilir.

Veya Gateway'in bu yeni Wifi noktasına bağlantı kurulması isteniyorsa, Gateway alınır ve **Working Set**'te **Gateway Control** sekmesinden **Network** sekmesine girilir. Sonrasında sağ tarafta yer alan **Type'**tan **Wifi** seçilir ve **Name** alanına da bu yeni tanımlanan Wifi bağlantı noktasının **SSID**'si girilip **Connect** butonuna tıklanarak Gateway'e hangi ağa bağlanacağı bilgisi gönderilir ve bağlantı hemen sağlanabilir.

Mobil APN

Sadece AFEX ve SAFEX'te çalışmaktadır.

Menüden **Mobile APN Configurations > Mobile APN Configurations**'a tıklayın. **Create APN** (APN Oluştur) butonuna tıklayın.



Mobil APN (Access Point Name, Erişim Noktası Adı), mobil servis sağlayıcılarının (Trcell, Turk Telekom, Vodafone...) sağlamış olduğu GPRS, Edge ve 3G/4G/LTE gibi mobil iletişim altyapısı üzerinde kurumlara sanal özel ağ kurulmasını sağlayan teknolojidir. Bu teknoloji sayesinde yetkilendirilmiş kullanıcıların (yani yetkilendirilmiş SIM Card'lar) erişimine açık bir özel ağ servis edilmektedir. Telefonların "Mobil Ağlar" menüsü altında APN ayarları bulunmaktadır.

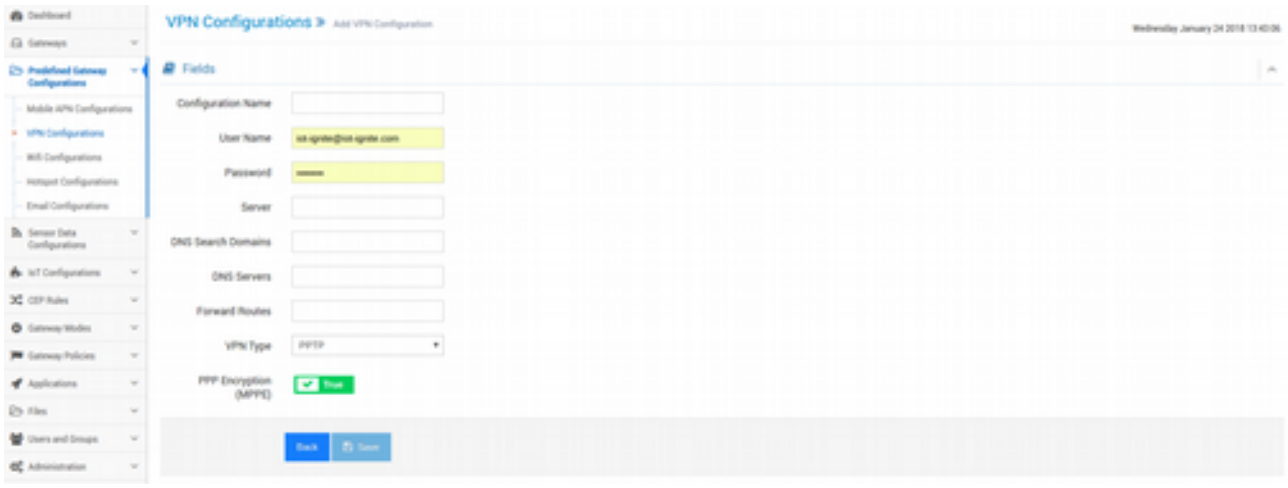
APN erişim bilgileri, mobil servis sağlayıcıları tarafından özel olarak kurumlara verilmektedir. Verilen bu bilgilerle doğrudan Gateway üzerinden APN girişi yapılabilir.

Yapacağınız öntanımlı APN yapılandırması Mode içinde Gateway'e gönderildiğinde, doğrudan bağlantı sağlanmaz, sadece APN ayarlarını uzatan tanımlanmış olur. Eğer Gateway'lerin hemen APN ile bağlantı kurması isteniyorsa; EHUB'tan APN bilgileri girilip kaydedildikten sonra Gateway'lerin bu APN üzerinden ağa erişebilmesi için, yine Gateway'ler Working Set'e alınmalı. Sonrasında Gateway Control sekmesi altından Network ayarlarından Type'ı APN seçilip, sonrasında bu hazırladığımız yapılandırma seçilerek Connect butonuna tıklanması yeterli olacaktır.

VPN

Sadece AFEX ve SAFEX'te çalışmaktadır.

Menüden **Mobile APN Configurations > VPN Configurations**'a tıklayın. **Create VPN Configuration** (VPN Yapılandırması Oluştur) butonuna tıklayın.



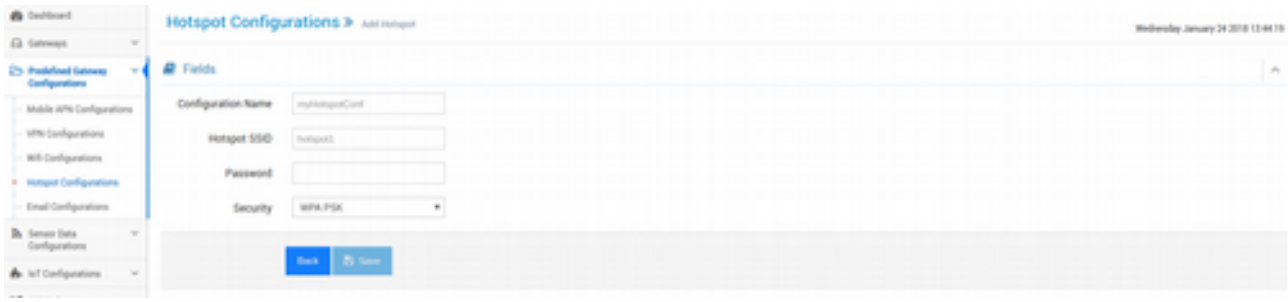
VPN (Virtual Private Network, Sanal Özel Ağ); uzakta bulunan fiziksel bir ağa sanki gerçekten fiziksel olarak bağlanmayı sağlayan teknolojidir. Çok merkezli veya merkez ağa uzaktan bağlanması gereken istemcilerin (client) olduğu yapılarda kullanılır. VPN'in çalışma mantığında, bir yerel ağ ile uzaktaki istemcilerin bağlanabilmesi için şifreli özel bir tünel açılır. Tünel ile erişim sağlandığında IP ve giden-gelen veriler gizlidir. Bu nedenle de VPN doğru bir amaçla ve kurumsal olarak kendi ağınızı kullandığınızda güvenli bir bağlantı kurulmasını sağlar.

Yapacağınız öntanımlı VPN yapılandırması Mode içinde Gateway'e gönderildiğinde, doğrudan bağlantı sağlanmaz, sadece VPN ayarlarını uzatan tanımlanmış olur. Eğer Gateway'lerin hemen VPN ile bağlantı kurması isteniyorsa; EHUB'tan VPN bilgileri girilip kaydedildikten sonra Gateway'lerin bu VPN üzerinden merkezdeki ağa erişebilmesi için, yine Gateway'ler Working Set'e alınmalıdır. Sonrasında Gateway Control sekmesi altından Network ayarlarından Type'ı VPN seçilip, sonrasında bu hazırladığımız yapılandırma seçilerek Connect butonuna tıklanması yeterli olacaktır.

Hotspot

Sadece AFEX ve SAFEX'te çalışmaktadır.

Menüden **Mobile APN Configurations > Hotspot Configurations**'a tıklayın. **Create Hotspot** (Hotspot Oluştur) butonuna tıklayın.



Hotspot; açık alanda Wifi teknolojisini kullanarak cihazın kendi bağlantısı üzerinden diğer cihazlara internet bağlantısını paylaştırma teknolojisidir. Bağlantısını paylaşan, yani hotspot olan cihaz aynı bir modem gibi davranış sergiler. Bu nedenle de hotspot olarak bir **SSID** (Ağ ismi), **parola** ve **WPA PSK / WPA2 PSK** olarak güvenlik ayarı bulunmaktadır.

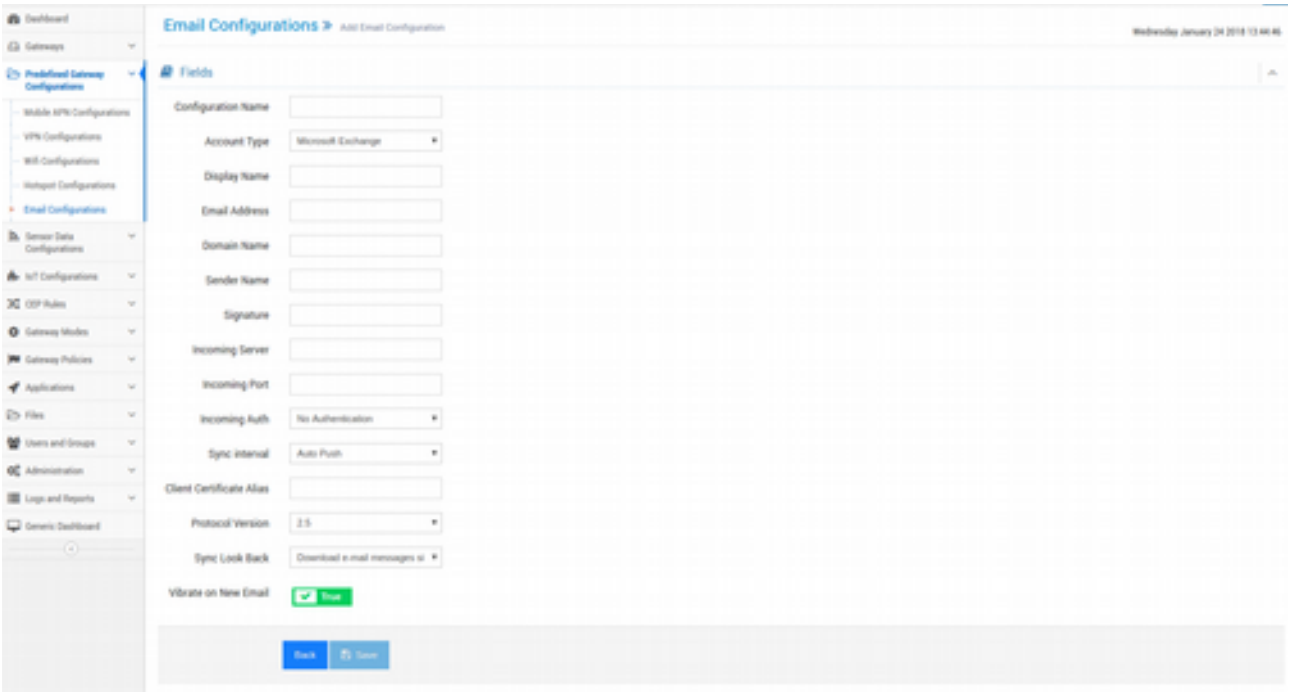
Eğer Gateway'in hotspot yapabilme özelliği varsa; bir hotspot yapılandırması tanımlayıp bunu Mode içine ekleyip Gateway'e gönderdiğimizde, Gateway'de hotspot ayarları girilmiş olur. Sonrasında Gateway'de hotspot açıldığında, bu tanımlamalara göre erişim noktası açmış olacaktır.

Başka gateway'lerin bu hotspota bağlanmaları için gereki Wifi tanımı yapılarak bunların hotspot desteği verin gateway üzerinden ağa bağlanmaları sağlanabilir.

Email

Sadece AFEX ve SAFEX'te çalışmaktadır.

Menüden **Mobile APN Configurations > Email Configurations**'a tıklayın. **Create Email** (Eposta Oluştur) butonuna tıklayın.



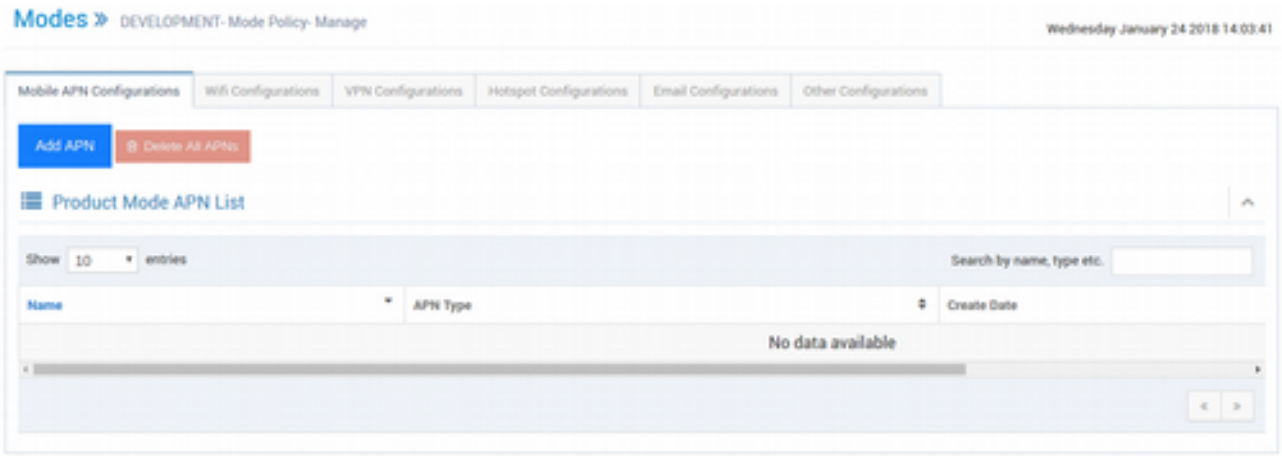
E-posta yapılandırması ile Gateway'de E-posta Hesapları kısmında yeni bir hesap açılır. E-posta yapılandırması **Microsoft Exchange**, **POP3** ve **IMAP** olarak yapılabilmektedir. E-posta için tanımlayacağınız bilgiler, Gateway'e Mode içerisinde gönderdiğinizde Gateway'in ekranında bir uyarı mesajı çıkar. Bu mesajda Gateway'de tanımlanmış olan e-posta hakkında bilgi verilir ve bu e-posta için bir **parola** tanımlanması beklenir (Eğer **Display Name** (e-posta'nın görünen ismi) tanımlı değilse, onu da girmesi beklenir). Kullanıcının işlemi onaylamasıyla birlikte e-posta hesabı Gateway'de aktif olur.

Öntanımlı Yapılandırma Ayarlarının Mode İçine Alınması ve Gateway'lere Gönderilmesi

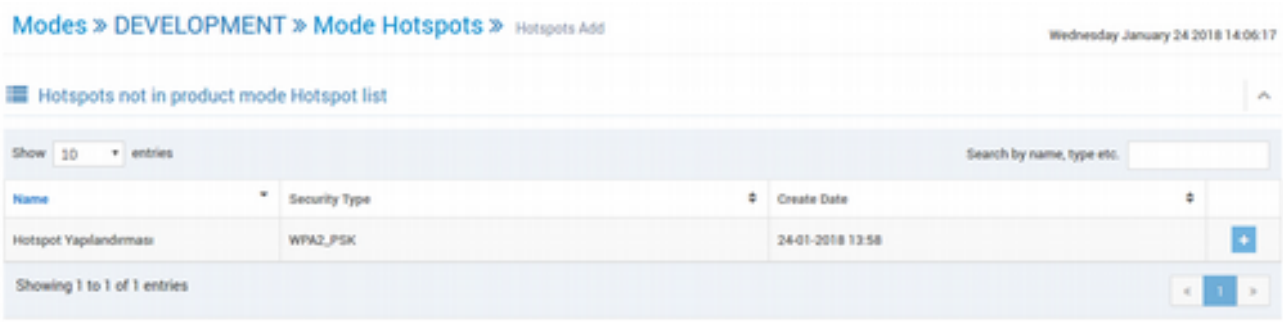
APN, VPN gibi hazırlamış olduğunuz öntanımlı yapılandırma ayarlarını istediğiniz herhangi bir **Mode** içine alıp, bu Mode'u da **Working Set** içine aldığınız Gateway'lere "**Push"** edebilirsiniz.

Menüden **Gateway Modes > Mode Store (Mod Deposu)**'a tıklayın. Servisinizde yer alan bütün Mode'lar liste halinde görünecektir. İstediğiniz Mode'un **Configuration** (Konfigurasyon) butonuna tıklayın. Örneğimizde DEVELOPMENT Mode'unu kullanalım...

DEVELOPMENT Mode'unun konfigürasyonuna girdiğimizde aşağıdaki gibi bir arayüz ile karşılaşacağız.



Üst kısımda yer alan sekmelerde görüldüğü gibi çeşitli kategorilerde yapılandırma ayarları ekleyebiliyoruz. Örnek olması amacıyla **Hotspot Configurations**'a giriyoruz. (Örnek için daha önce oluşturmuş olduğumuz bir Hotspot yapılandırması bulunmaktadır). Daha sonra **Add Hotspot** (Hotspot Ekle) butonuna tıklayıp devam ediyoruz. Bu işlem ile daha önce öntanımlı olarak hazırladığımız Hotspot yapılandırmaları listesi gelecektir. Listeden birini seçip + butonuna tıklayarak devam ediyoruz.



Ekleme işleminden sonra sayfa otomatik olarak tekrar **Hotspot Configurations** sekmesine geçecektir ve listede **Hotspot Yapılandırma**sı isimli konfigürasyonumuz görünecektir. Artık bu Mode'u Gateway'e push edebiliriz.

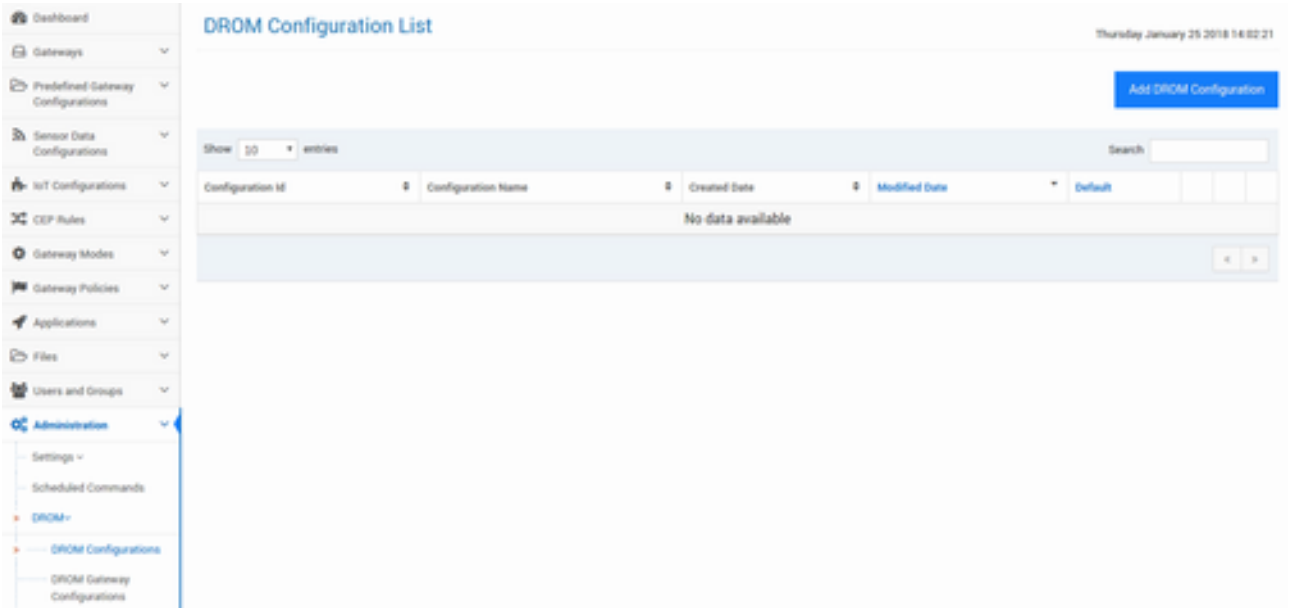
Working Set'e **Android** Gateway'imizi alıp, **Modes** sekmesine geçerek listeden **DEVELOPMENT** Mode'unu seçerek gönderiyoruz.

DROM

Dynamic ROM olarak açılımı olan DROM, Gateway'lerin lisanslanmasında servis sahibi tarafından öntanımlı olarak belirtilen konfigürasyonlarla Gateway'lerin otomatik olarak lisanslanması anlamına gelmektedir. Konuyu biraz daha açacak olursak şöyle bir senaryo düşünün. IoT-Ignite altyapısını kullanarak bir ürün geliştirdiğimizi ve bu ürünlerden 1000 adet üretildiğini, her bir ürünün de Gateway ID'sini bildiğimizi varsayalım. Ürünümüzü alan müşteri aktivasyon işlemi ile uğraşmasın, Gateway internete bağlandığında doğrudan Ignite Cloud'una erişip kendi kendini lisanslasın ve kullanıma hazır olsun istiyoruz. İşte bu ihtiyacımızı karşılamak için DROM kullanılmaktadır.

DROM Yapılandırması Hazırlamak

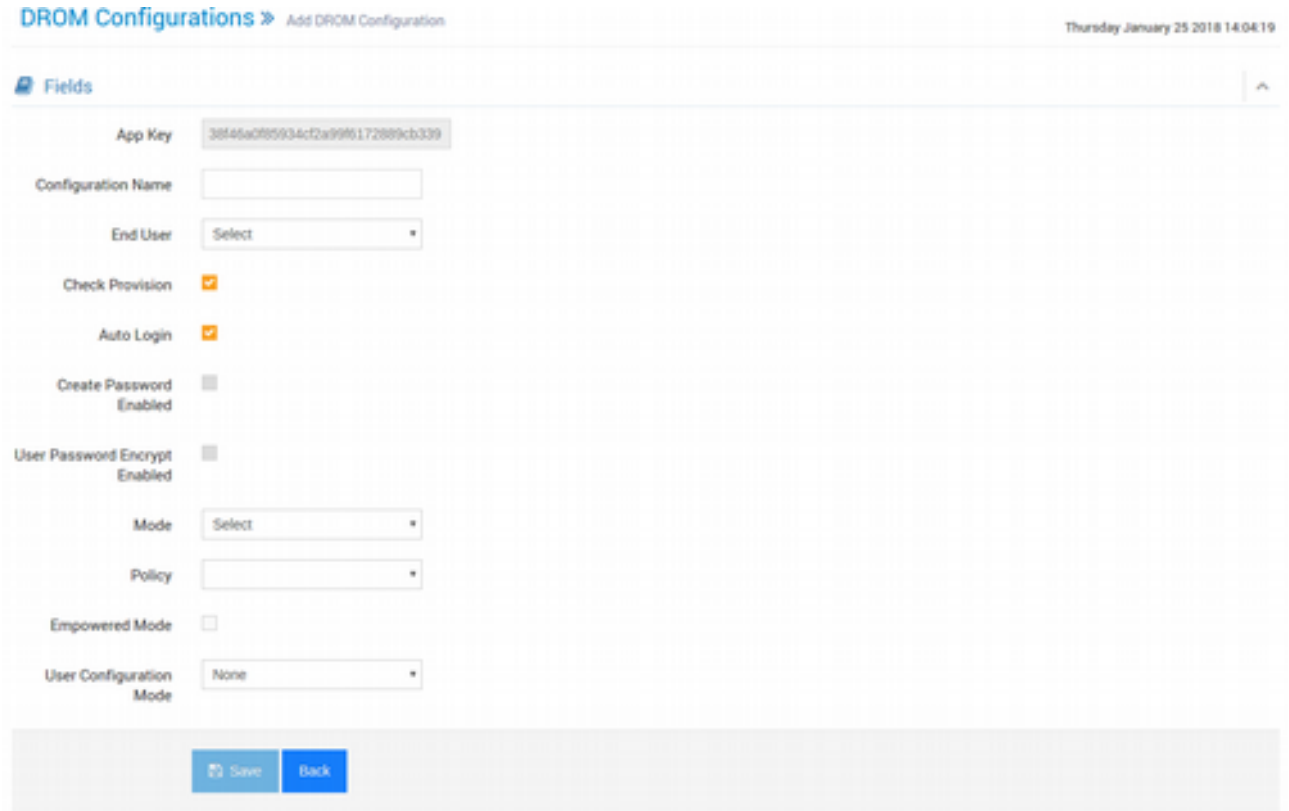
Menüden **Administration > DROM > DROM Configurations**'a girin.



Eğer daha önceden DROM tanımladıysanız, açılan bu sayfada listelenecektir. Listede yer alan DROM'u yeniden düzenleyebilir veya silebilirsiniz.

DROM'da yapılan ayarlar Gateway'ler bir ağa bağlandığında doğrudan Ignite Cloud'a kendi Gateway ID'si ile sorgu gönderir. Bu sorguda kendisi için bir DROM tanımlaması yapılmış mı diye bakar. Eğer gerçekten kendi ID'si ile bir DROM tanımlanmışsa, hemen lisans ayarlarını alır ve kendini lisanslar.

Yeni bir DROM oluşturmak için **Add DROM Configuration** (DROM Yapılandırması Ekle) butonuna tıklayın. Karşınıza aşağıdaki gibi bir form gelecek.



Formda önemli olan değerlerden biri **App Key**'dir. Devzone'u incelerken App Key'e değinmiştik. Bu anahtar servisinize ait olan tekil bir anahtardır. Gateway tarafında da lisans eşleştirmesi sağlanabilmesi için geliştirme yapılırken bu App Key de DROM içinde Gateway'lere gönderilir.

Her yapılandırmada olduğu gibi Configuration Name için bir isim belirtin.

Gateway'in hangi **End User**'a (Son Kullanıcıya) lisanslanacağını da End User listesinden seçebilirsiniz. Eğer yeni bir End User'a lisanslamak istiyorsanız menüden **Users and Groups > Gateway User**'a girip yeni bir End User (Diğer ismi ile Gateway User) tanımlayabilirsiniz. Eğer listeden herhangi bir End User seçilmezse, Gateway lisanslama yapamaz. Ancak Gateway'in ekranında bir aktivasyon kodu girilmesi istenecektir. Bu aktivasyon kodu da End User'a ait olan 6 haneli bir sayıdır (Gateway Users sayfasına girildiğinde her user'a ait farklı bir **Activation Code** görülmektedir). Gateway kısmında ekranda hangi End User'a lisanslama yapılacaksa onun aktivasyon kodu girilir ve onaylanır. Bu durumu şöyle bir senaryo ile düşünebilirsiniz. Mesela 1000 adet ürününüz var ve siz de 1000 adet End User oluşturdunuz. Her bir ürün kutusunun üzerine de End User'lara ait olan 6 haneli Activation Code'ları ayrı ayrı yazılı olarak verilmiştir. Gateway, herhangi bir End User'a bağlı olmayan DROM'u aldığında, yetkili bir kişi de bu aktivasyon kodunu girerek lisanslama işlemini gerçekleştirir.

Diğer bir kritik ayar ise **Check Provision**'dur. Bu opsiyon aktif olursa, Gateway'in daha önceden lisanslanıp lisanslanmadığı kontrol edilir. Eğer lisanslanmışsa doğrudan eski ayarlara geri getirilerek o lisanslandığı **End User**'a tekrar lisanslanır.

Auto Login ile otomatik olarak yetkilendirme alınır.

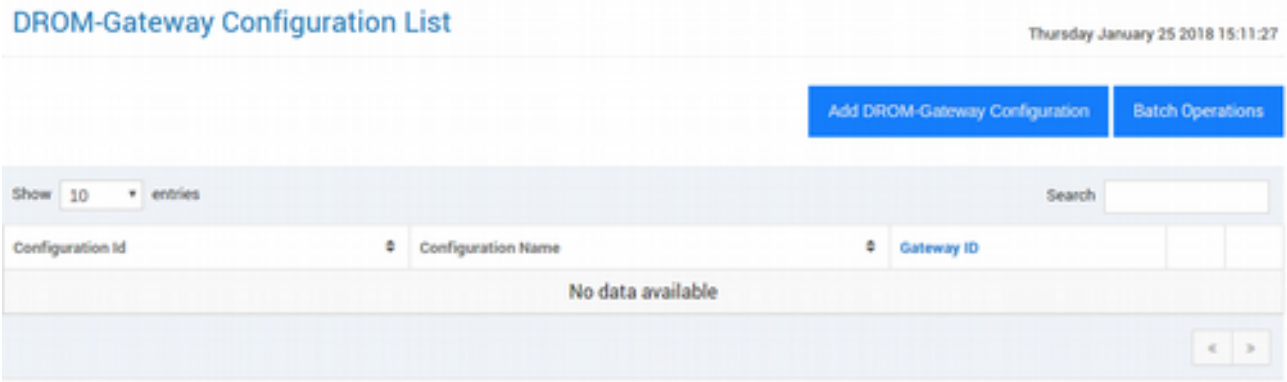
Son olarak da **Mode** ve **Policy** seçimi yapılır. Böylece Gateway lisanslandığı anda kendisine Mode ve Policy'i de çeker. Eğer buradan bir Mode veya Policy seçilmezse, End User'ın varsayılan Mode ve Policy'sini alır. Örnek olması amacıyla **DROM Ayarları 1** ismi ile **dev@iot-ignite.com** End User'ına lisanslanması için (Eğer bir End User'a birden fazla Gateway lisanslanma izni varsa, **Administration > Settings > API Configuration Settings** sayfasından kontrol edilebilir. Eğer **CAN_USE_SAME_ACTIVATION_CODE** değeri **1** ise aynı End User'a birden fazla lisanslama yapılabilir.) ve **Auto Login** opsiyonu ile **DEVELOPMENT** Mode'unun **DEVELOPMENT** Policy'si ile formu dolduruyoruz.

Formu doldurun ve **Save** (Kaydet) butonunu tıklayarak DROM'u oluşturun.

DROM Gateway Yapılandırmaları

Az önce bir DROM yapılandırmasının nasıl yapılacağını gördük. Şimdi de bu DROM ayarlarını hem tek bir Gateway'e hem de çoklu olarak Gateway'lere nasıl eşleştireceğimizi görelim...

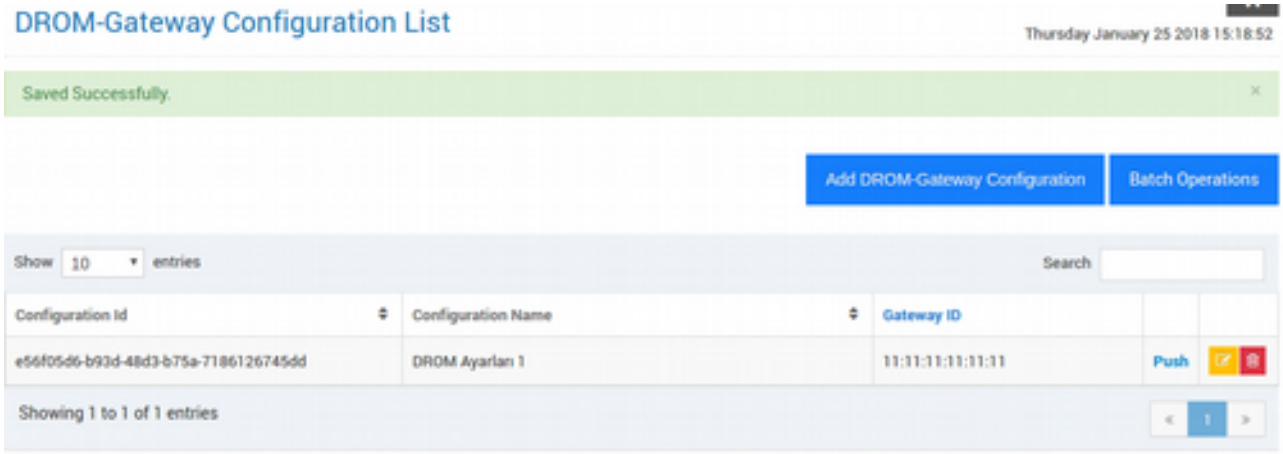
Menüden **Administration > DROM > Drom Gateway Configurations** sayfasına girin. Eğer daha önceden Gateway'lere DROM atadıysanız, açılan bu sayfada listede o Gateway'ler ve sahip olduğu DROM'lar görünecekti. Listeden her birinin DROM'unu yeniden verebilir veya kaldırabilirsiniz.



Bir Gateway’e DROM Atamak

Tek bir Gateway’e DROM atanacaksa, **Add DROM-Gateway Configuration** butonuna tıklayın. Açılan basit form’da DROM konfigürasyonunu seçin ve Gateway’in ID’sini girip **Add** butonu ile işlemi tamamlayın.

Örnek olarak **DROM Ayarlar 1** isimi yapılandırmayı **11:11:11:11:11:11** ID’li Gateway için tanımladık.

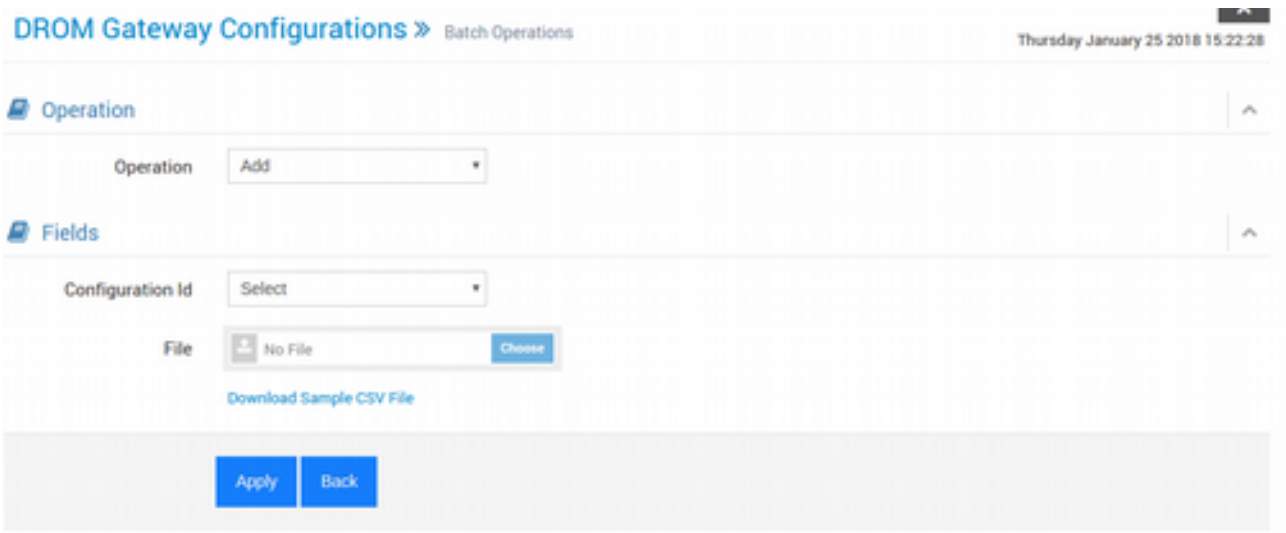


> Normal veya DROM ile lisanslama işlemini yaparken Devzone hesabınız ile toplamda 5 adet lisansa sahip olduğunuzu unutmayın! Lisanslanmış bir Gateway’in lisansını kaldırsanız da, kullanmış olduğunuz lisans hakkı geri gelmeyecektir. Eğer daha fazla lisansa ihtiyacınız olursa, daha fazla lisans satın alınabilir.

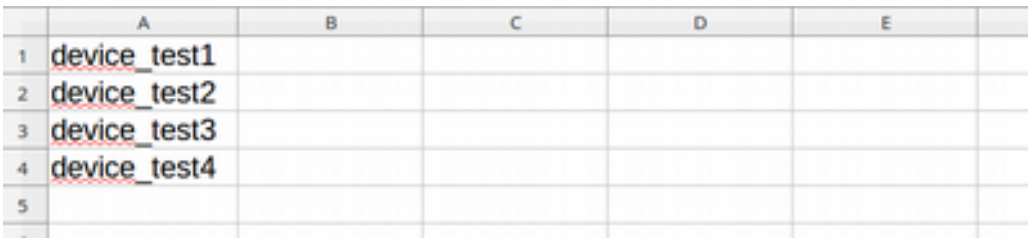
Çoklu Olarak Gateway’lere DROM Atamak

Eğer bir değil de yüzlerce Gateway’e DROM tanımlamak isterseniz, az önce gördüğümüz metot her ne kadar işe yarıyor olsa da tek tek Gateway tanımlamak zaman alacak. Bu nedenle toplu olarak DROM tanımlama işlemi yapılması gerekmektedir.

Batch Operations (Toplu İşlemler) butonuna tıklayın. Karşınıza aşağıdaki gibi bir arayüz gelecek.



Operation (İşlem) listesinde **Add**, **Delete** ve **Update** işlemleri bulunmaktadır. **Add** ile Gateway'lere DROM atanır. **Delete** ile Gateway'lerden lisans kaldırılır. **Update** ile de güncel DROM ayarları ile Gateway yeniden lisanslanır. **Configuration ID**'de yine daha önce oluşturduğumuz DROM ayarı seçilir. **File** (Dosya) kısmında ise bir EXCEL dosyası (.csv) ile Gateway'leri liste olarak yüklemeniz istenecektir. **EXCEL** dosyası da belli bir formatta olmak zorundadır. Örnek bir **.csv** dosyasını **Download Sample CSV File** butonuna tıklayarak indirebilirsiniz. İnen EXCEL dosyası aşağıdaki gibidir:



	A	B	C	D	E
1	device_test1				
2	device_test2				
3	device_test3				
4	device_test4				
5					

Görüldüğü üzere sadece **A** sütununda satır satır **Gatewa ID**'leri yer almaktadır. Formu doldurduktan sonra **Apply** (Uygula) butonuna tıkladığınızda, çoklu olarak DROM eşleştirmesi yapmış olacaksınız.

BÖLÜM 6

Gateway Katmanı

Gateway

IoT-Ignite ile IoT uygulamaları geliştirmek için bazı fonksiyonel cihazların sisteme Gateway olarak kayıt edilmesi gerekiyor. Gateway cihazları IoT-Ignite ile iletişim kurabilen donanımlardır. IoT sisteminde bulunan sensörlerden toplanan veriler bu cihazlar ile buluta gönderilir. Gateway olarak; Android yüklü cihazlar, evimizdeki bilgisayarımız veya Raspbery Pi gibi cihazlara kullanabileceğimiz gibi, sadece bu işlemi yapmak için geliştirilmiş cihazları da kullanabiliriz.

IoT-Ignite, Gateway olarak Android'e daha büyük önem vermektedir. Ancak diğer bahsettiğimiz cihazları da rahatlıkla kullanabilirsiniz.

Yukarıdaki şekilde görüleceği üzere IoT-Ignite platformunda Android, PilarOS, Industrial Gateways ve MQTT, Gateway olarak kullanılmaktadır.

IoT-Ignite platformu daha önce de bahsettiğimiz üzere Edge Computing (veya Fog Computing) mantığına sahip olacak şekilde tasarlanmıştır. IoT sistemlerinin geniş coğrafi alanları yayılması zorunluluğu ve beraberinde milyarlarca cihazın aynı sisteme bağlanması, gerçek zamanlı eylem veya anlık veri analizini zorunlu kılmıştır. Bulut sistemleri böyle bir yapıda kesinlikle tutunamaz. Tabi bu bulut sistemlerinin kullanılmayacağı anlamına gelmiyor. Ancak karmaşık sistemlerde anlık veri analizinin önemli olduğu durumlarda, Edge Gateway kavramı işlemlerin anlık olarak yapılması için kaçınılmazdır. Örnek bir Edge Gateway şeması aşağıdaki gibidir.

Çevre birimler merkezde bulunan bir cihaza (Edge Gateway) bağlanarak, kablosuz ağ protokolü ile bulut sistemlerine bağlanmıştır. Edge Gateway, hem çevre birimleri anlık olarak yönetmekte hem de sistemin bir bütün içinde çalışabilmesi için kritik verileri buluta iletmeyi sağlamaktadır. Akıllı Edge Gateway ile dinamik iş kararlarının anlık olarak icra edilmesi sağlanır. Ayrıca birçok farklı servis aynı ağ geçidi (gateway) üzerinde rahatlıkla çalışabilir.

Akıllı Edge Gateway ile çalışmanın bizlere sağladığı avantajları şu şekilde sıralayabiliriz:

- Daha düşük bant genişliği ile çalışabilme imkanı,
- Daha hızlı veri erişimi,

- Kuralları hızlı bir şekilde uygulaması,
- Etkili karar verme mekanizması oluşturmayı sağlaması,
- Bulut bağlantısının olmadığı veya sağlanamadığı durumlarda sistemin kesintiye uğramadan çalışmaya devam etmesi,
- ve basitleştirilmiş veri yönetimi gibi avantajları bulunmaktadır.

IoT-Ignite Edge Gateway Fonksiyonel Bileşenleri

IoT-Ignite ile çalışırken Edge Gateway ile çevre birimlerden veri alabilir ve anlık işlemleri gerçekleştirebiliriz. Edge Gateway kavramı, yukarıda değindiğimiz üzere gerçek zamanlı eylemlerde bulunmak veya bulut ile bağlantının olmadığı durumlarda sistemin duraksamadan çalışmasını sağlamak adına oldukça önemlidir.

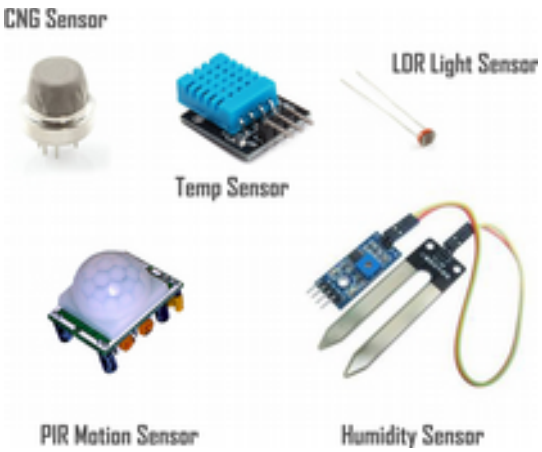
Edge Gateway ile kontrol edebileceğimiz veya veri okuyabileceğimiz temel fonksiyonel bileşenler bulunmaktadır. Bunlar genel olarak **Actuators** ve **Things** olarak ifade edilmektedir.

Actuator; fonksiyonel bileşenleri, bir mekanizmayı veya sistemi kontrol eden veya hareket ettiren bir tür motor olarak tanımlanmaktadır. Buraya örnek olarak servo motorları verebiliriz.

IoT sistemlerinde bulunan Gateway cihazları bunları kontrol etmek için kullanılır. Örneğin servo motora bağlı olan bir kamerayı Edge Gateway ile uzaktan kontrol edebilirsiniz. Bu sayede görmek istediğiniz alana doğru kameranızı 360° döndürebilirsiniz.

Things; IoT-Ignite platformunda sensör gibi donanımları temsil etmek için kullanılmaktadır. Bilindiği üzere sensörler IoT sistemlerinin vazgeçilmez bileşenleridir. Özellikle bu donanımların ucuza üretilmesi ile beraber IoT sistemleri dünyada hızla yayılmaktadır. Tabii bunların artması beraberinde ağa bağlı donanım sayısının artması anlamına da geliyor. Bundan dolayı Edge Gateway kavramı geliştirilmiştir. Bu sayede sistemde bulunan sensörlerden elde edilen veriler anlık olarak analiz edilmektedir.

IoT-Ignite sisteminde kullanabileceğimiz bazı temel sensörler aşağıdaki gibidir.



Bu sensörleri kullanarak IoT-Ignite ile uygulamalar geliştirebilirsiniz. Bu sensörlerden gelen verileri okumak, analık analiz veya verileri buluta göndermek adına Gateway kavramını anlamak oldukça önemlidir.

Virtual Edge Gateway Platformu

IoT-Ignite ile Edge Gateway kavramını etkili bir şekilde kullanmaktayız. Gateway listesi içinde hizmetimize sunulan MQTT'yi Virtual Edge Gateway olarak kullanabiliyoruz.

MQTT; **M2M** uygulamalarında tercih edilen veri dağıtım protokolüdür. Machine to Machine (M2M) kavramı, **Industial Internet of Things** (IIoT) kavramı içinde gelişen önemli bir bileşendir.

M2M, veri alışverişinde makinelerin birbirleriyle nasıl konuştuğu üzerine odaklanır. Diğer bir deyişle M2M teriminde, insan eylemleri olmaksızın gerçek zamanlı veri alışverişini sağlayan tüm teknolojilere ve kablosuz ağlara atıfta bulunuyoruz.

MQTT kütüphanesini kullanarak M2M tabanlı IoT uygulamaları geliştirebiliriz. Mesaj tabanlı bir protokol olan bu kütüphane ile telemetri, bildirim sistemleri ve akıllı ev sistemlerini rahatlıkla geliştirebiliriz.

MQTT hakkında ayrıntılı bilgiyi 1. bölümünde aktarmıştık. Ayrıca 10. bölümde **MQTT Client Kütüphanesi**'ni kullanarak **Virtual Gateway** kavramının ne olduğunu örnek bir uygulama ile inceleyeceğiz.

IoT-Ignite Agent

IoT-Ignite platformu, IoT uygulamaları geliştirme için kullanılan bir PaaS çözümüdür. Tasarımın arkasında en iyi mimariye sahip modern uygulamalar yer almaktadır. Ignite Agent, IoT ve MDM'i (Mobile Device Management) bir araya getiren çok fonksiyonlu bir uygulamadır. Bu uygulama ile, sensör verileri buluta gönderilebilir veya cihaz üzerinde işlenebilir. Ayrıca kurallar, verilerle ilişkili

olarak tanımlanabilir. Yönetim araçlarına sahip olan bu uygulama ile aygıtınızı ve bu aygıtta bulunan sensörleri kurallar dahilinde kontrol edebilirsiniz. Ignite Agent, cihaz yönetimini sağlamak için Android ile gelen cihaz yönetici izinlerini kullanır.

IoT-Ignite Agent uygulaması ile:

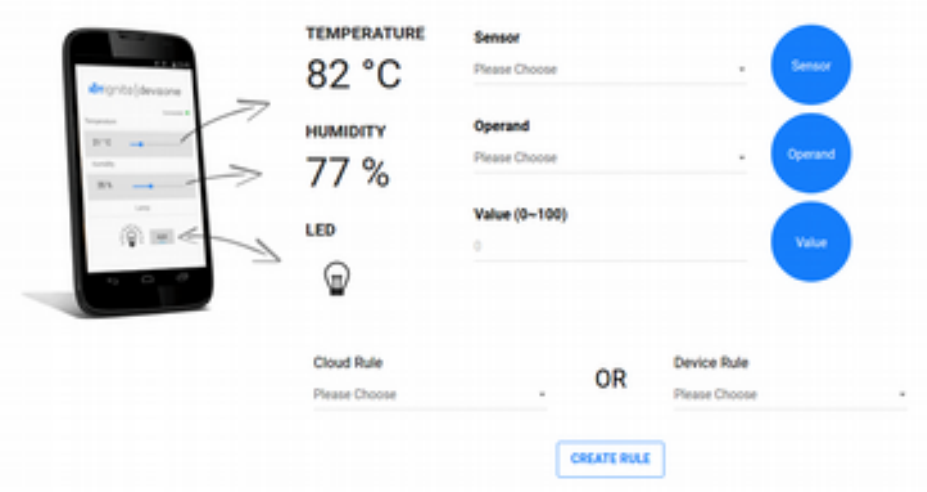
- Şifre kontrolü,
- Uzaktan ekran kilidini açıp/kapamak,
- Depolama birimlerini şifrelemeyi veya var olan şifreyi çözmek,
- Cihaz kamerasının açıp/kapamak,
- Cihazı yeniden başlatmak,
- Ekran görüntüsü almak,
- Cihazı çaldırmak,
- Diğer uygulamaları durdurmak veya çalıştırmak,
- Cihaza bağlı çevre birimlerini yapılandırmak (WiFi, Bluetooth vb.),
- Tüm verileri silmek gibi işlemleri gerçekleştirebiliriz.

IoT-Ignite Kural Yönetimi

IoT-Ignite kural yönetimi, herhangi bir altyapıyı yönetmek zorunda kalmadan, bağlı cihazlar tarafından global ölçekte üretilen verileri toplamak, işlemek, analiz etmek ve bunlara göre hareket eden IoT uygulamaları oluşturmak için geliştirilmiştir. Bu kurallar IoT-Ignite bulutunun yanı sıra, kenar geçitlerde (Gateway) de çalışabilir. Bunlara sırasıyla "Cloud (Bulut) Kuralları" veya "Gateway (Ağ Geçidi) Kuralları" diyoruz.

Kural (Rule) Nedir

Kurallar; belirli koşulların sağlanması durumunda belirli işlemlerin yapılması için tanımlanan ifadeyi belirtir. Kurallar sadece bir koşula bağlı olabileceği gibi, birden fazla koşula da bağlı olabilir. Ayrıca sağlanan koşul ile bir veya birden fazla eylem de gerçekleştirilebilir.



Kuralları iki şekilde gerçekleştirebiliriz; Cloud ve Gateway.

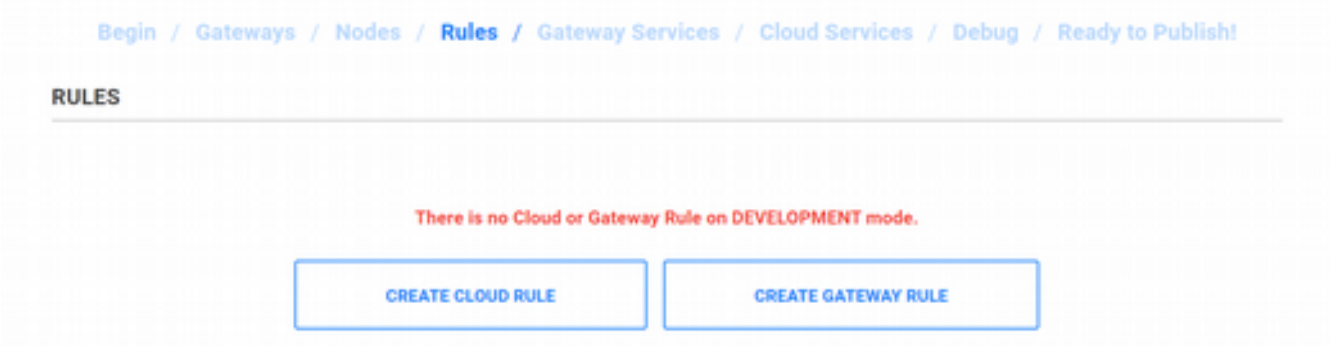
Cloud Rules (Bulut Kuralları)

IoT-Ignite için yayınlanan iletileri değerlendiren ve bunları tanımladığınız iş kurallarına dayalı olarak başka bir aygıta veya bulut hizmetine dönüştürmeyi sağlamak için kullanılan kurallardır. Bir veya birden fazla cihazdan gelen veri akışlarına bir kural uygulanabilir ve paralel olarak bir veya birden çok işlem yapılabilir.

Gateway Rules (Ağ Geçidi Kuralları)

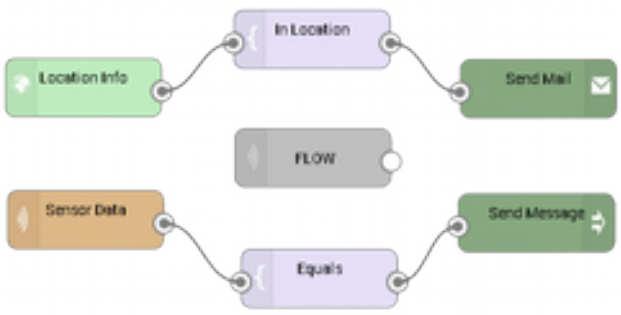
Anlık veri analizi ve gerçek zamanlı eylemde bulunmak için Gateway cihazları, IoT uygulamaları için vazgeçilmez bileşenlerdir. Kenar aygıtları bulut ile sürekli bağlantıda olmayı engelleyip sadece ihtiyaç olan durumlarda buluta başvurmayı sağlar. Böylece gerçek zamanlı eylemler yapılabilir. Gateway için kullandığımız cihazlar için kural tanımlayabiliriz. Tanımlanan kurallar Cloud'ta çalışmaz böyle durumlarda kural veya kurallar ağ geçitine gönderilir. Özellikle Gateway ve Cloud arasındaki bağlantı koptuğu zaman Gateway kuralları sistemin çevrimdışı olarak çalışmasını sağlar. Bulut trafiğini azaltmak adına böylesi bir kullanım oldukça önemlidir.

Kuralları tanımlamak için Devzone web sitesine giriş yapıp Rules sekmesini kullanmanız gerekiyor.



Kural Editörü Hakkında

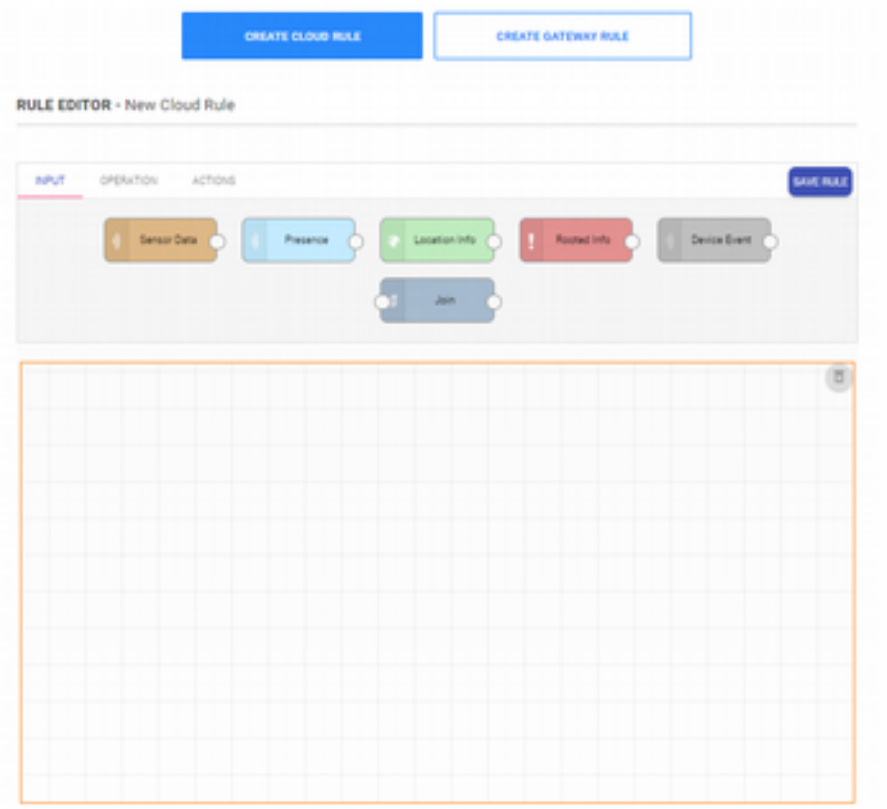
IoT-Ignite ile gelen Kural Editörü ile veri kaynağınızı, durumunuzu ve hareket bileşenlerini sürükleyip bırakarak kurallarınızı görsel olarak tasarlayabilirsiniz. Bileşenlerin birbirine bağlanarak kuralların oluşturulmasını sağlayan bu arayüz ile kural tanımlama işlemini pratik bir şekilde gerçekleştirebilirsiniz.



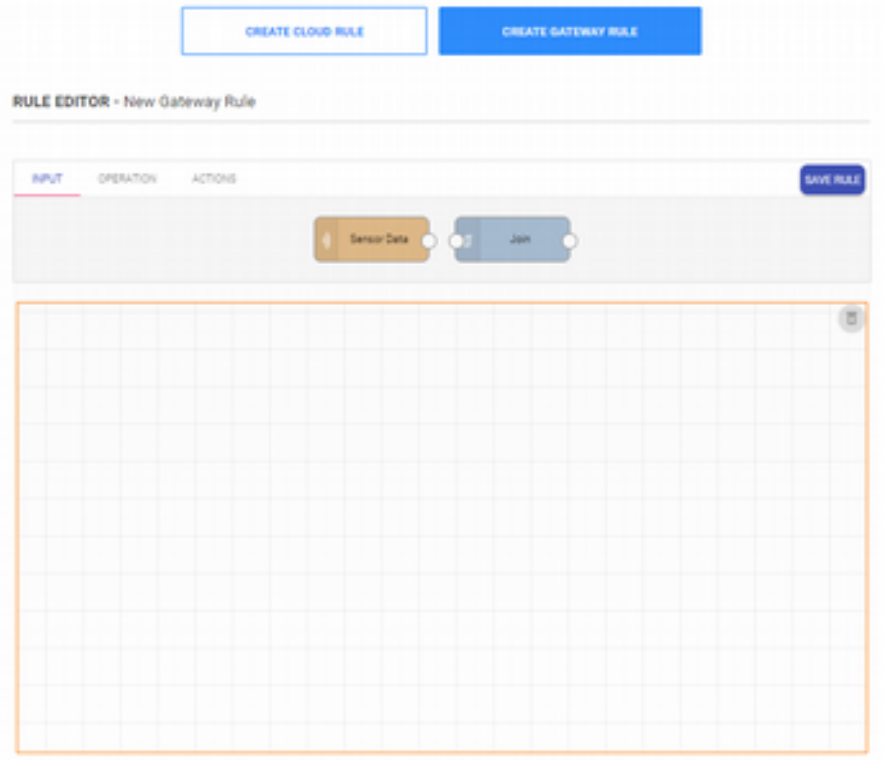
Editör yardımıyla istediğiniz kadar kural tanımlayabilirsiniz. Ayrıca istediğiniz zaman kuralları etkinleştirip devre dışı bırakabilirsiniz.

Cloud ve Gateway kurallarınız oluşturmak için kural editörünü kullanabilirsiniz. Bunun için Devzone'a girmeniz gerekiyor. Rules sekmesinde görüleceği üzere Cloud ve Gateway olarak kural tanımlayabilirsiniz.

Cloud kuralları tanımlamak için aşağıda verilen arayüzü kullanırız.



Gateway kuralları tanımlamak için aşağıda verilen arayüzü kullanırız.



Kural'ların Önemi

IoT sistemleri ile ilgili uygulama geliştirirken kurallar ile çalışmak oldukça önemlidir. Kurallar IoT sistemlerinin gücünü kontrol altında tutmak için en önemli kavramlardan biridir. Örneğin aracınız çalıştığı zaman evinizin sizin geleceğini algılayıp oda sıcaklığını ideal seviyeye çıkarmak için kombi veya klimayı başlatması güzel bir uygulama. Ancak böyle bir sistemden beklenen oda sıcaklığının ideal seviyeye geldikten sonra kombi veya klima gibi donanımları kapatması beklenir. Bu yapılmadığı zaman tasarruftan ziyade harcama maliyetlerinin artışı meydan gelir. Bu ve benzer uygulamalarda kuralları kullanarak IoT sistemlerinin daha verimli ve otomatik bir şekilde çalışması sağlanır.

IoT-Ignite Karmaşık Olay İşleme (CEP)

IoT-Ignite Kural Yönetimi sayesinde IoT-Ignite platformunda Complex Event Processing (CEP - Kompleks Olay İşleme) yapılabilir. Burada CEP hakkında bilmemiz gereken bilgileri sizlerle paylaşacağız.

CEP Nedir

CEP, entegre sensörler, sanal sensörler, işlemciler veya müşteri uygulamalarından (IoT-Ignite Cloud platformunda belirlenen ve cihaza gönderilen) her bilgiyi toplayan ve değerlendirmeyi sağlayan bir uygulamadır. Değerlendirme süreci sonrasında, yakalanan kalıplar (patterns) olay olarak tanımlanır. IoT-Ignite ile önceden tanımlanmış kurallar için şartlar sağlandıktan sonra, CEP bu kurallara göre önceden tanımlanmış olay eylemlerini harekete geçirir. Meydan gelen bu olay ve eylemler bulut ortamına kaydedilir ve IoT-Ignite Agent Application veya diğer uygulamalar yoluyla müşteri başvurusuna gönderilir. Ayrıca bu bilgiler ajan aracılığıyla Ignite Cloud'a gönderilir.

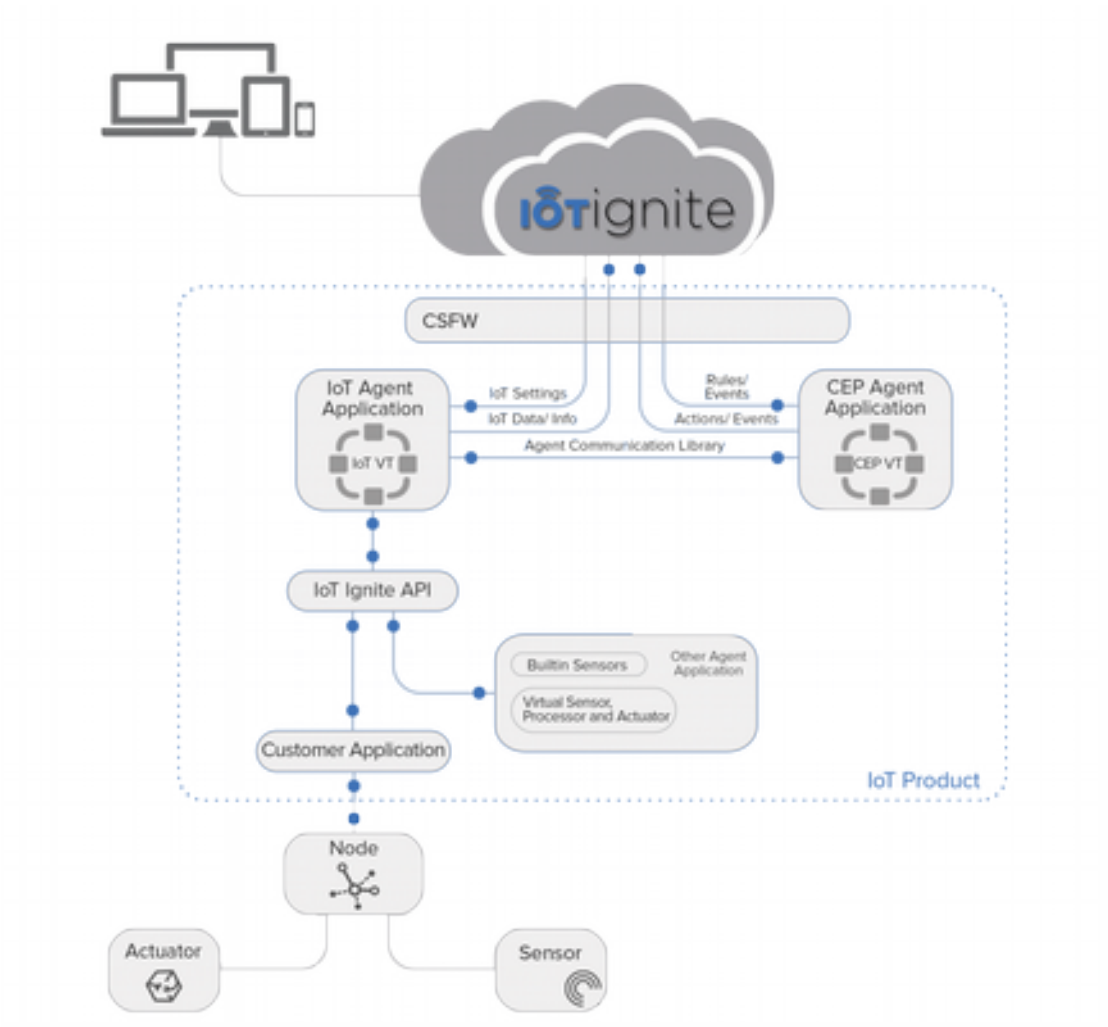
CEP Akış Şeması

Çevre birimlerden gelen toplanan verilerin CEP ile nasıl toplandığını ve değerlendirildiğini anlamak için CEP'in akış şemasını incelememiz gerekiyor.

Aşağıda verilen CEP akış şemasından şu çıkarımlarda bulunabiliriz:

- CEP, IoT-Ignite platformu ile veri alışverişnde bulunan bir uygulamadır.
- CEP ile IoT-Ignite Agent uygulaması arasında bir iletişim bulunmaktadır. İletişim için kullanılan Agent Communication Library'ı kullanılmaktadır.
- CEP'ten alınan veriler IoT-Ignite ajan uygulaması aracılığıyla IoT-Ignite bulutuna gönderilir.
- Müşteri uygulamalarının bağlı olduğu Actuator veya Sensörlerden alınan veriler IoT-Ignite API ile IoT-Ignite Agent uygulamasına gönderilir.
- IoT Agent uygulaması, bulut ile iletişim halinde olup IoT uygulamalarından elde edilen ayarlar veya veriler Agent ile buluta gönderilir.
- Müşteri uygulamalarından toplanan veriler değerlendirilmek üzere CEP'e gönderilir. CEP elde edilen verilere göre belirli şartlar meydana geldiğinde bazı eylemlerde bulunmayı sağlar. Böylece IoT sistemlerinde olayların işlenmesi pratik bir şekilde icra edilmiş olur.

IoT uygulamalarında eylemleri gerçekleştirmek için kullanılan CEP'i akış şeması özetle aşağıdaki gibidir.



IoT-Ignite Kütüphane Arabirimi Üzerinden App Programlama

Arabirim kütüphanesi, sanal sensörler ve bağlantı işlemleri için müşteri uygulamaları, aracı uygulamaları ve API'ler sağlamaktadır. IoT-Ignite, müşteri uygulamasını veya cihaza kurulan uygulamalar IoT Agent ile bağlamak için ihtiyaç duyulan arayüzleri sağlar.

- Uygulamalar, IoT Agent ile bağlantı kurmak için kullanıcı kimlik doğrulamasına ihtiyaç duymaktadırlar. Kimlik doğrulama işlemi başarılı olduktan sonra, uygulama aşağıda verilen hizmetlerden faydalanabilir:
- Node ekleme, silme ve güncellemek,
- Sensör ekleme, silme ve güncellemek,
- Ignite Cloud'a sensör bilgisi göndermek,
- Sensör ve node geçmişini arayabilmek,
- Sensör ve node bilgisi aramak,
- Farklı müşteri aygıtları arasında iletişimde bulunmayı sağlamak,
- Hata ayıklamak,
- Olay oluşturmak,
- Durum raporu hazırlamak,

- Complex Event Processing (CEP-Karmaşık Olay İşleme),
- Ignite Cloud'a kayıtlı aygıt, node ve sensör hakkındaki bilgilere erişmek.

Eğer uygulamanız için kimlik doğrulama işlemini yapmazsanız, sadece aşağıda verilen işlemleri gerçekleştirebilirsiniz.

- LAN'da bulunan node'lar ile iletişim kurmak,
- Ağ sorunlarını çözmek,
- Bağlı cihazlar için durum raporu oluşturmak,
- Desteklenen mikro denetleyiciler için otomatik algılamak,
- Desteklenen mikro denetleyicilere özel olarak tasarlanmış arabirimler aracılığıyla bilgi toplamak.

IoT Agent Bağlantı Arabirimi, kullanıcı doğrulamasından sonra müşteri uygulamasına veya diğer aracı uygulamalara erişebilmek için uygulama anahtarı sağlar. Uygulamalar, bu uygulama anahtarını kullanarak IoT Agent ile iletişime geçebilir. Ancak şunu bilmekte özellikle fayda var: Eğer IoT-Ignite ile çalışmak istiyorsanız bunun için **igniteManager** nesnesinin üretilmesi gerekiyor.

IoT-Ignite Agent Uygulaması Üzerinden Uygulama Programlama

IoT Agent uygulaması ile kurulan iletişim dolaylı olarak sağlanmaktadır. Müşteri uygulaması ve Ignite Cloud arasında yapılan veri iletimi, IoT Agent uygulaması ile gerçekleşir. Eğer müşteri uygulaması Ignite Cloud'a bağlanmak isterse, kimlik doğrulama işleminin ***appKey*** ile yapılması gerekiyor. Müşteri uygulaması ile ilgili her işlem, appKey'in paket adı aracılığıyla işlenir. Kimlik doğrulama işleminden sonra, müşteri uygulaması, node'lara bağlı her sensör için bir dinleyici atanmasını sağlar. Bu dinleyici yapılandırma işlemi sensöre gönderilebilir, ayrıca sensörlerden gönderilen olay mesajlarını alabilir.

Müşteri uygulama bilgileri IoT Agent uygulamasına, IoT-Ignite yoluyla gelir. Node'lar ve sensörler IoT-Ignite'a kaydedilir ve tanımlı fonksiyonları buradan çalıştırılır. Tanımlanan her sensör, node yapısı üzerinde benzersiz olarak ayarlanır. Farklı düğümlerde aynı ada sahip sensör olabilir. Müşteri uygulaması güvenliği sağlamak için appKey ve paket adı kullandığı için cihazdaki bir uygulama diğer uygulamaların sensörlerine ve düğümlerine erişemez. Bundan dolayı aynı cihazda bulunan farklı müşteri uygulamaları, tek bir IoT Agent uygulamasını kullanabilir.

Sensörler için dört temel yapılandırma bulunmaktadır. Bunlar şu şekildedir:

- Veri gönderme sıklığı,
- Veri gönderme eşiği,
- Veri gönderme eşik türü: değişim yüzdesi veya değişim farkı,
- Veri tutma zaman aşımı sınırı.

Müşteri uygulaması, bu yapılandırmalar yoluyla uygulamalarını zenginleştirebilir. Bununla beraber özel bazı yapılandırmalar da oluşturulabilir. Ancak IoT Agent, nasıl davrandığına bakılmaksızın müşteri uygulaması yoluyla gönderilen yapılandırmalar ile çalışacaktır.

Bu bağlantı türünde, cihaz tarafında karmaşık olay işleme yapılabilir. Bu eylemler belirlenen kural veya kurallar vasıtasıyla yürütülür. Bir eylemin çalışması için birkaç kuralın tamamlanması gerekebilir. Ayrıca bir kural için birkaç işlem de gerçekleştirebiliriz. Bu, müşterinin uygulamanın düğmeye bağlı actuator'ı tetiklemesine, uygulamayı başlatmasına, cihazı kilitlemesine, politika yüklemesine, komut göndermesine, profilini değiştirmesine gibi işlemleri yapmasına olanak tanır.

Bu kapsamda, örnek verebileceğimiz bazı akış işlemleri aşağıdaki gibi olabilir:

- Konumunuz ofis ise ve zaman aralığı hafta sonu ise, cihazı kilitleyin.
- Şirket muhasebe uygulamasını yalnızca cihaz VPN ağ şirketlerine bağlıysa çalıştırın.
- Yurtdışındaysanız veri bağlantısını kapatın.
- Tatilde cihaz konum bilgilerini göndermeyi durdurun.
- Önceden tanımlanmış numaraya SMS mesajı gönderin.

Gateway Olarak Android

IoT-Ignite ile çalışırken size sunulan Gateway'lerden herhangi birini tercih ederek IoT uygulamaları geliştirebilirsiniz. Ancak bu platformlar içinde Gateway denilince akla gelen ilk cihaz şüphesiz Android yüklü akıllı telefon veya tabletlerdir. Bu cihazlardan herhangi birisini Gateway olarak sisteme kayıt edebilirsiniz.

Neden Android Cihazlar Gateway Olarak Kullanılmalı?

Android cihazların Gateway olarak kullanılması için birçok önemli neden bulunmaktadır. Özellikle bu cihazların dünya genelinde yaygın bir şekilde kullanılması Android'i, Gateway olarak tercih etmenin en önemli sebebidir. Android ile gelen geniş yelpazedeki sensör desteği ile birçok Android cihazı sıcaklık, nem, manyetometre, ivmeölçer ve daha birçok sensöre dahili (built-in) olarak sahip olmaktadır. Android cihazların Gateway olarak kullanılabilmesini sağlamaktadır.

Android cihazların Gateway olarak tercih edilmesinin sebepleri genel olarak şu şekildedir:

- Mobil ağ teknolojisi ile Mobile Edge kavramının yaygın bir şekilde kullanılması,
- Ağ tıkanıklığını azaltması ve veri trafiğini daha hızlı gerçekleştirmesi,
- Gecikmeyi en aza indirmesi,
- Yerel hizmetlerin verimli bir şekilde sunulmasını sağlaması,
- Gerçek zamanlı eylem veya anlık veri analizi için güçlü işlemcilere sahip olması,
- Üretim aşamasında dahili olarak sensörlere sahip olması,
- WiFi veya hücresel bağlantıları kullanarak bulut sistemleri ile sürekli etkileşim halinde olabilmesi,
- Bluetooth bağlantısı ile yerel düzeyde bulunan sensörler veya geliştirme kartları ile haberleşebilmesi,

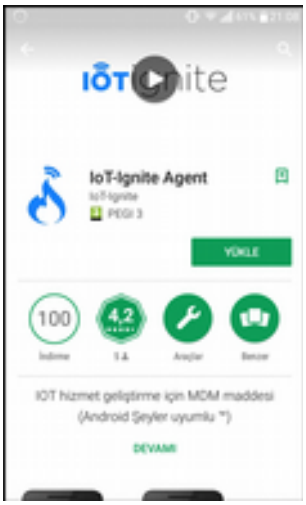
- Son kullanıcıların yaygın bir şekilde Android'i tercih etmesi ve dolayısıyla IoT uygulamalarını kontrol etmek için mevcut cihazları kullanma isteği.

Bu liste daha da uzatılabilir. Ancak burada şunu belirtmekte fayda var; 2020 yılından itibaren internete bağlı cihaz sayısında hatırı sayılır bir oranda artış meydana gelecektir. **IoT-Ignite platformu internet ortamında bulunan Android telefonları Gateway olarak kullanarak, hem ağa bağlı olan cihazların IoT uygulamaları için kullanılmasını sağlamakta, hem de son kullanıcıların maliyetleri düşürerek daha ucuza IoT uygulamaları geliştirmesini hedeflemektedir.**

IoT-Ignite Agent'ın Kurulumu

IoT-Ignite platformunun geniş bir kullanıcı ağı için geliştirilen IoT-Ignite aracılığı sayesinde herhangi bir Android telefon veya Android tablet bir ağ geçidine dönüştürülebilir. Bunun için **IoT-Ignite Agent** yazılımını **Play Store**'dan indirmek yeterli olacaktır. İndirme işleminden sonra okutacağınız bir **QR** kodu ile IoT-Ignite Cloud'a rahatlıkla bağlanabilirsiniz.

Öncelikle IoT-Ignite Agent uygulamasını indirmek için cihazınızın Play Store uygulamasını başlatınız ve uygulamayı aratınız. Aşağıdaki gibi bir ekran çıktısı ile karşılaşmanız gerekiyor.

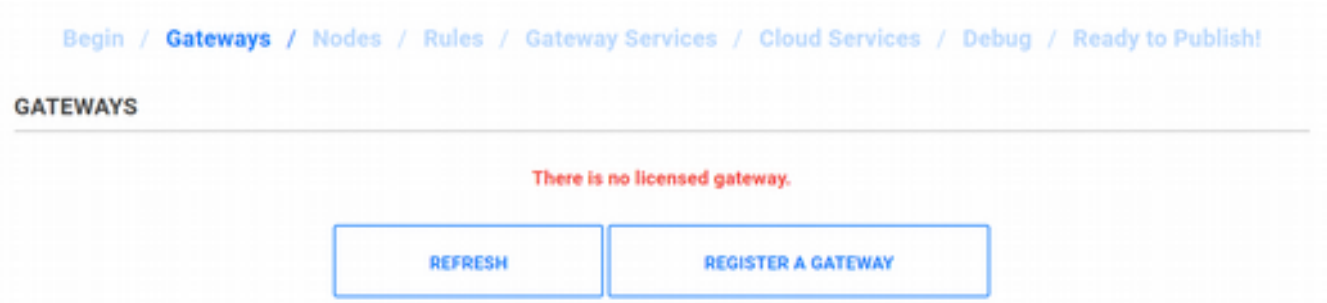


Buradan uygulamayı yüklüyorsunuz. Yükleme işlemi bittikten sonra aşağıdaki ekran bizi karşılayacaktır.

Başla butonuna tıkladıktan sonra aşağıdaki ekran ile karşılaşırız.

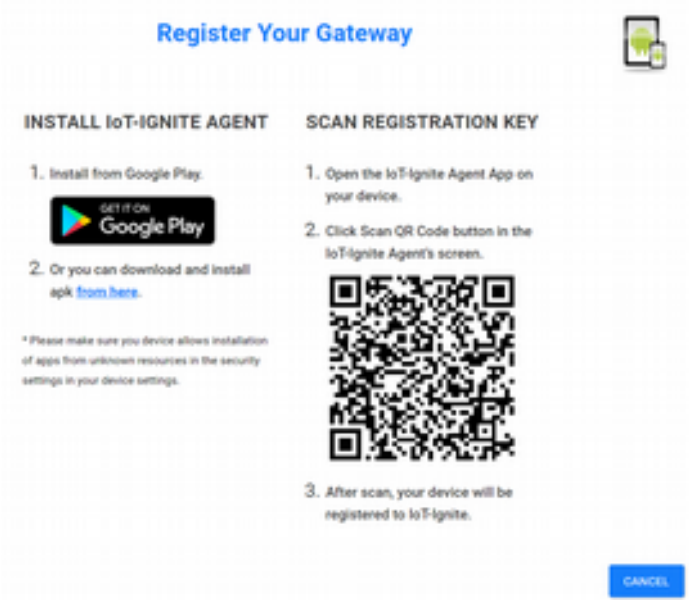
Burada kayıt işlemini tamamlamak için QR kodunu taratmamızı isteyen bir arayüz karşılamaktadır.

Bu noktada IoT-Ignite platformuna giriş yapalım ve Devzone alanında bulunan Gateway lisanslama arayüzüne gelelim. Aşağıda verilen ekranı görmeniz gerekiyor.



Burada bulunan **REGISTER A GATEWAY** seçeneğine tıkladığımızda aşağıdaki pencereden **Android** seçeneğini seçelim.

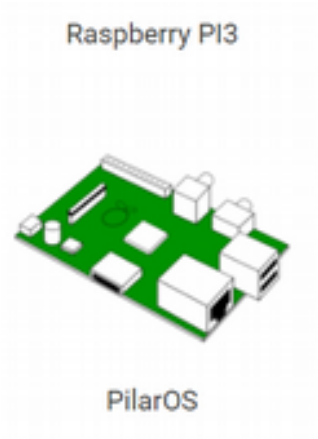
Android Gateway'ini seçtikten sonra aşağıda ekran açılır. Bu ekranda verilen QR kodunu akıllı telefonunuzda bulunan Agent uygulaması ile okuyup kayıt işlemini tamamlayabiliriz.



Kodu tarattıktan sonra Gateway kısmında aşağıda görüleceği üzere akıllı telefonun Gateway olarak kayıt edildiği görülmüş olur.

Raspberry Pi'yi Tanıtma

IoT-Ignite için kullanabileceğimiz diğer bir Gateway cihazı Raspberry Pi geliştirme kartıdır. Bu kartı kullanabilmek için ARDIC firmasının geliştirdiği **PilarOS** işletim sistemini kullanmamız gerekiyor. PilarOS'u Raspberry kartına yüklemeyi daha sonra göstereceğiz. Burada Raspberry kartının ne olduğu, yetenekleri ve şu ana kadar üretilmiş çeşitlerinden bahsedeceğiz.

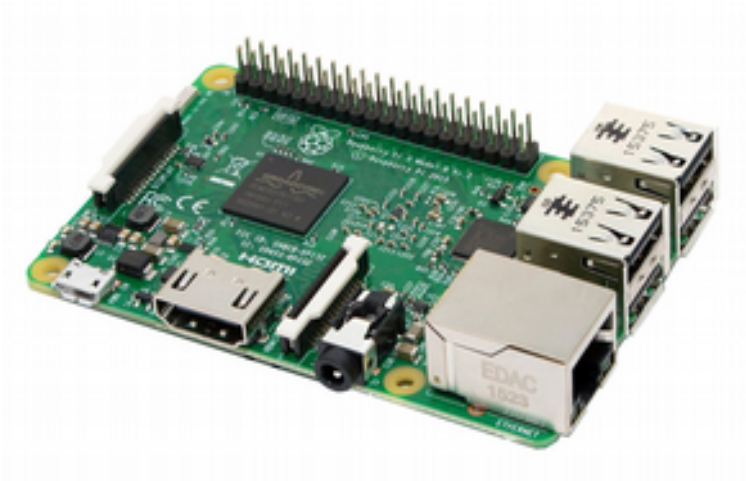


Rasperry Pi Nedir

Raspberry Pi, kredi kartı büyüklüğünde olan ve eğitim amaçlı geliştirilen mini bir bilgisayardır. Fiyatı oldukça uygun olan bu kart özellikle çocuklara programcılığı ve kodlamayı sevdirmek için üretilmiştir. Üniversite öncesi düzeyde programlama becerilerini ve donanım anlayışını geliştirecek bir aygıt olan Raspberry, dünya genelinde oldukça popüler bir geliştirme kartıdır. Özellikle dar gelirli grupların bilgisayar ihtiyaçlarını karşılayan bu aygıt normal bir masaüstü veya dizüstü bilgisayarın yapabileceği işlemleri yapabilmektedir.

Linux işletim sistemi ve çeşitlerini destekleyen bu kart IoT uygulamaları için oldukça kullanışlıdır. Dünya çapında en çok tercih edilen işletim sistemleri Raspbian (Debian tabanlı) ve Pidora (Fedora tabanlı) işletim sistemleridir. Bunlarla birlikte aşağıda verilen işletim sistemlerini de kullanabiliriz.

- PilarOS (ARDIC firmasının geliştirdiği Android tabanlı bir işletim sistemi),
- Android Things (Google'ın IOT işletim sistemi)
- Ubuntu Mate,
- Snappy Ubuntu Core,
- Windows 10 IOT Core,
- Pinet
- ve Risc OS işletim sistemleri.



Raspberry Pi kartına adı geçen işletim sistemlerini yüklemek için SD karta ihtiyacımız vardır. Burada en azından 8 GB'lik bir SD kart tercih etmeniz isabetli olacaktır.

Rasperry Pi'ın Yetenekleri

Raspberry Pi, mini bir bilgisayar olarak normal bir masaüstü veya dizüstü bilgisayarın tüm yeteneklerine asgari düzeyde sahiptir. Bu kartı bir bilgisayar olarak evinde veya iş yerinde kullananların sayısı oldukça fazladır. Özellikle internette gezmek veya paket programlar ile çalışmak için bu kartı bilgisayar olarak kullanabilirsiniz. Hatta kendiniz için bir oyun konsolu bile oluşturabilirsiniz.

Raspberry Pi kartı, dahili bir ekran kartı ile gelmektedir. Bundan dolayı HDMI bağlantısı ile kartınızı herhangi bir monitöre bağlayabilirsiniz. Monitörün HDMI veya VGA (dönüştürücü ile) bağlantısına sahip olması yeterlidir. Eğer internete bağlanmak isterseniz LAN bağlantısı veya Wireless ile kartınızı kolay bir şekilde internete bağlayabilirsiniz. Bluetooth ile çevrede bulunan donanım veya sensörlerle iletişime geçebilirsiniz. Raspberry'nin kablosuz protokolleri desteklemesi ve internete bağlanabilmesinden dolayı IoT uygulamalarında sıklıkla tercih edilmektedir. Yine sahip olduğu güçlü işlemci ile anlık veri analizi ve gerçek zamanlı eylemlerde çok kaliteli sonuçlar üretmektedir.

Raspberry Pi için üretilen kameraları kullanarak gözlem tabanlı uygulamalar da geliştirebilirsiniz. USB bağlantıları ile fare, klavye veya yazıcı gibi donanımları rahatlıkla kullanabilirsiniz.

Raspberry Pi kartının yeteneklerini daha kavramak adına bu kart ile geliştirilen en popüler uygulamalardan örnekler verelim:

- Elektrikli kaykay,

- Fotoğraf makinesi,
- Elektronik satranç tahtası,
- Robot kolu,
- Akıllı ayna,
- Akıllı kapı zili,
- ve son olarak arabalar için akıllı dikiz aynası gibi projeler verilebilir.

Rasperry Pi Çeşitleri

Raspberry PI kartları ilk defa 2012 yılının ilk çeyreğinde üretilmeye başlandı. Bu tarihten itibaren Raspberry kartının yayınlanmış birçok modeli bulunmaktadır. Bu modeller genel olarak aynı özelliklere sahip olmakla beraber aralarında hız ve yenilik olarak bazı farklılıklar bulunmaktadır. Şu ana kadar üretilen Raspberry modelleri aşağıdaki gibidir.

- Raspberry PI Model A
- Raspberry PI Model A+
- Raspberry PI Model B
- Raspberry PI Model B+
- Raspberry PI 2
- Raspberry PI 3
- Raspberry PI Zero
- Raspberry PI Zero W

Raspberry PI modelleri A ve B olmak üzere iki farklı isim altında üretilmeye başlandı. Model A, ilk üretilen Raspberry PI kartıdır ve 128 MB'lık bir RAM'e sahip olarak piyasaya sunulmuştur. Zaman içinde birçok modeli üretilen bu kartın en büyük çıkışı Raspberry PI 3 Model B ve Zero ile yapıldı.

Bundan dolayı özellikle Raspberry PI 3 ve Raspberry PI Zero hakkında kısaca bilgi edinelim. Çünkü şu anda en yaygın kullanılan Raspberry PI kartları bunlardır.

Raspberry Pi 3

Raspberry PI 2'den sonra üretilen bu model ile ilk kez Raspberry kartları dahili WiFi ve Bluetooth ile gelmeye başladı. Diğer bir özelliği ise 64 bitlik bir işlemciye sahip olmasıdır. Üçüncü nesil Raspberry PI olarak isimlendirilen bu kart, Şubat 2016 tarihinden beri Raspberry PI 2 yerine kullanılmaktadır.

Raspberry PI 2 ve Raspberry PI 3 arasındaki ayrımı net görmek adına aşağıdaki tabloyu inceleyebilirsiniz.

Özellik	Raspberry PI 2	Raspberry PI 3
İşlemci	A 900MHz quad-core ARM Cortex-A7 CPU	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
RAM	1GB	1GB
USB	4 Adet	4 Adet
Wireless	Bulunmuyor	BCM43438 wireless LAN on board
Bluetooth	Bulunmuyor	Bluetooth Low Energy (BLE) on board
GPIO	40 Pin	40 Pin
Full HDMI Port	1 Adet	1 Adet
Ethernet Port	1 Adet	1 Adet
Camera Interface	Var	Var
Micro SD slot	Var	Var

Raspberry Pi Zero

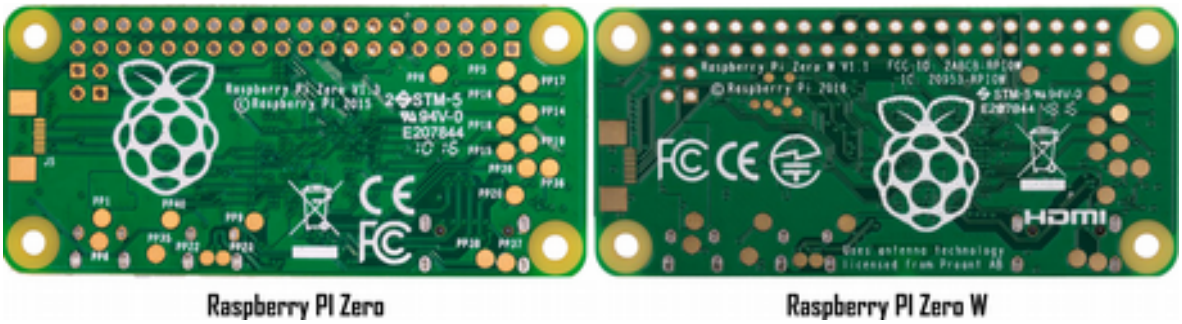
Şu ana kadar üretilmiş en küçük Raspberry modeli olan bu model ilk çıktığı zaman 5$'lık fiyatıyla dikkatleri üstüne çekmeyi başarmıştı. Boyutu küçük olduğu için tam boy USB yerine mikro USB OTG portu kullanılan bu modelin diğer bir çeşidi de Zero W'dur. Her iki modelin ortak özellikleri şu şekilde sıralanabilir.

- 1GHz single-core CPU
- 512MB RAM
- Mini HDMI port
- Micro USB power
- HAT-compatible 40-pin header
- Composite video and reset headers
- CSI camera connector

PI Zero ailesinin genişletilmiş sürümü olan Zero W, Şubat 2017 sonunda piyasaya duyuruldu. Orijinal PI Zero'nun tüm işlevlerine sahip olan Zero W'nun diğer özellikleri aşağıdaki gibidir.

- 802.11 b/g/n wireless LAN
- Bluetooth 4+ Low Energy (BLE)

Görüleceği üzere Zero W özellikle kablosuz ağ donanımlarına sahip olmasıyla farkını göstermektedir.



IoT uygulamalarında Raspberry PI 3 ve Raspberry PI Zero W modelleri kablosuz ağ desteğinden dolayı özellikle tercih edilmektedir.

PilarOS ROM Image'ları (RPI3 Pilaros Img)

PilarOS, ARDIC'ın 1800'den fazla ek API ile geliştirilmiş endüstriyel Android sürümüdür. Android cihazları Gateway olarak kullanmayı sağlayan IoT-Ignite özellikle Android işletim sistemine büyük önem vermektedir. PilarOS'ta Android tabanlı olduğu için bu işletim sistemini Raspberry Pi3 geliştirme kartına yükleyerek Gateway olarak kullanabiliriz.

ARDIC firmasının geliştirdiği bu işletim sisteminin mevcut olan imaj dosyalarına erişebilmek için aşağıdaki linki kullanabilirsiniz.

https://download.iot-ignite.com/Pilaros-rpi3/

Bu linki ziyaret ettiğiniz zaman aşağıda görüleceği üzere üç adet ROM Image ile karşılaşırız.

Bunlardan istediğinizi seçip bilgisayarınıza indirebilirsiniz. İlki **HDMI** bağlantılar için diğer ikisi ise **VGA** bağlantısı için kullanılmaktadır.

PilarOS'un Rasperry Pi 3'te Windows Ortamında Kurulumu

PilarOS işletim sistemini Raspberry kartından kullanabilmek için öncelikle bunu SD karta yüklememiz gerekiyor. Burada Windows ortamında PilarOS'u SD karta yüklemek için yapılması gereken işlemleri sizlere aktarmaya çalışacağız.

Donanım İhtiyaçları

PilarOS'u gateway olarak kullanabilmek için ihtiyacımız olan donanımların listesi şu şekildedir:

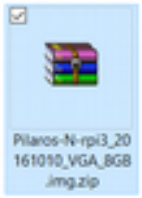
- Raspberry PI 3 Geliştirme Kartı,
- Mikro SD Kart (Minimum 4GB),
- SD Kart okuyucu.

Bunları temin ettikten sonra diğer adıma geçebiliriz.

PilarOS Image Dosyasının İndirilmesi ve Açılması

Image dosyasını indirmek için aşağıdaki linki kullanabilirsiniz. Burada bizlere sunulan üç adet imaj dosyası bulunmaktadır.

Bu imaj dosyalarından istediğinizi seçip indirebilirsiniz. Dosya indirme işleminden sonra aşağıdaki gibi bir ZIP dosyası ile karşılaşırız.



Yapılması gereken diğer bir işlem ZIP'li olan imaj dosyasını açmaktır. Bunun için 7-ZIP programını kullanabilirsiniz.

Dosyayı açtıktan sonra aşağıdaki gibi imaj dosyasını görmemiz gerekiyor.

İmaj dosyasını bu şekilde elde ettikten sonra mikro SD karta yazdırma adımına geçebiliriz.

MicroSD Kart'a Image'ın Yazdırılması

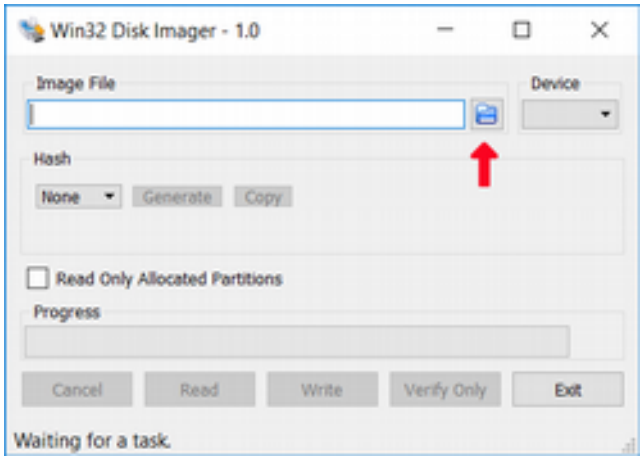
Burada indirdiğimiz imaj dosyasını SD karta yüklemek için izlememiz gereken adımlardan bahsedeceğiz.

- Öncelikle aşağıda verilen linkten **Win32 Disk Imager** programını indiriniz.

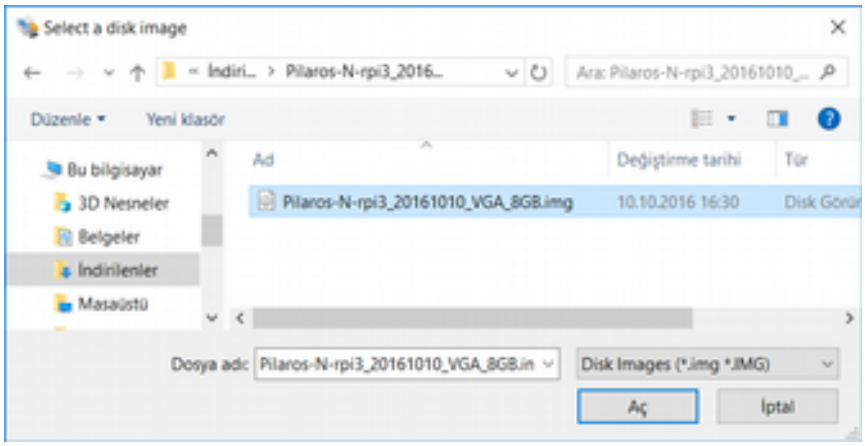
https://sourceforge.net/projects/win32diskimager

Programı indirdikten sonra kurulum işlemini yapınız.

- Mikro SD kartınızı kart okuyucunuza yerleştiriniz.
- Win32 Disk Imager programını yönetici olarak başlatınız. Aşağıdaki gibi bir ekran ile karşılaşmanız gerekiyor.

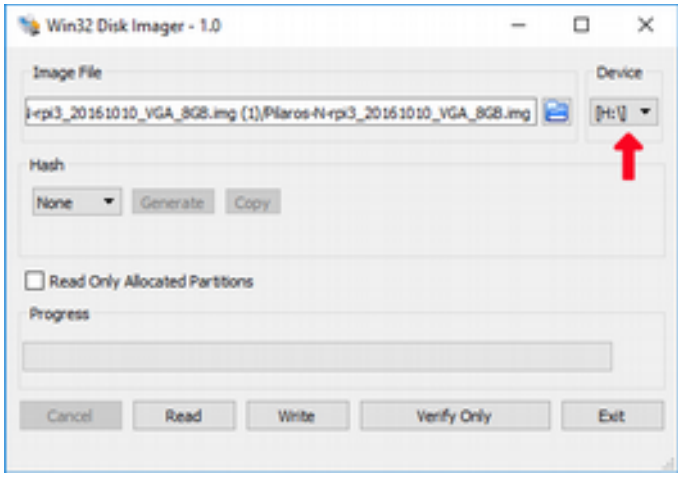


- Yukarıda verilen görselde ok ile gösterilen klasör simgesine tıklayınız. Açılan pencereden indirdiğiniz imaj dosyasını seçiniz.



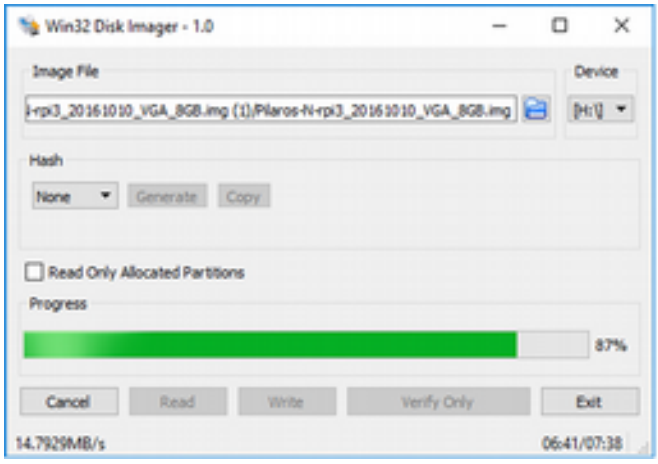
Dosyayı yukarıdaki gibi seçtikten sonra **Aç** butonuna tıklayıp seçme işlemini tamamlayabilirsiniz.

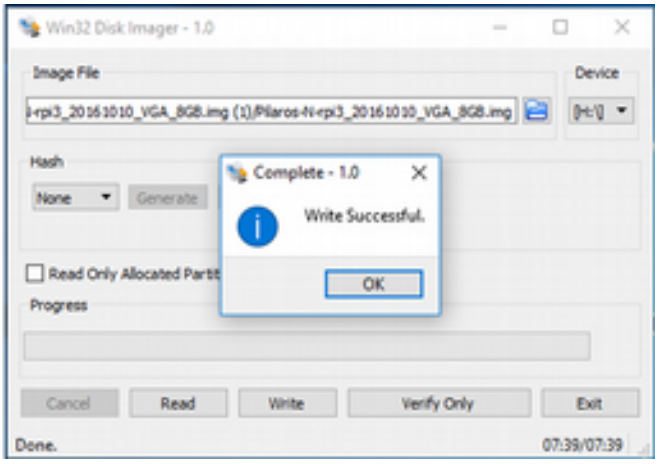
- Kart okuyucunuzu bilgisayarınıza bağlayarak aşağıdaki gibi Win32 ile seçili olduğundan emin olunuz. Kart okuyucu için bende tanımlanan sürücü ismi H'dir. Bu sizde farklı olabilir. Eğer birden fazla SD kartınız bağlı ise açılır pencere içinde istediğinizi seçebilirsiniz.



Yukarıdaki adımları izledikten sonra **Write** butonuna tıklayıp imaj dosyasını SD karta yazdırabilirsiniz.

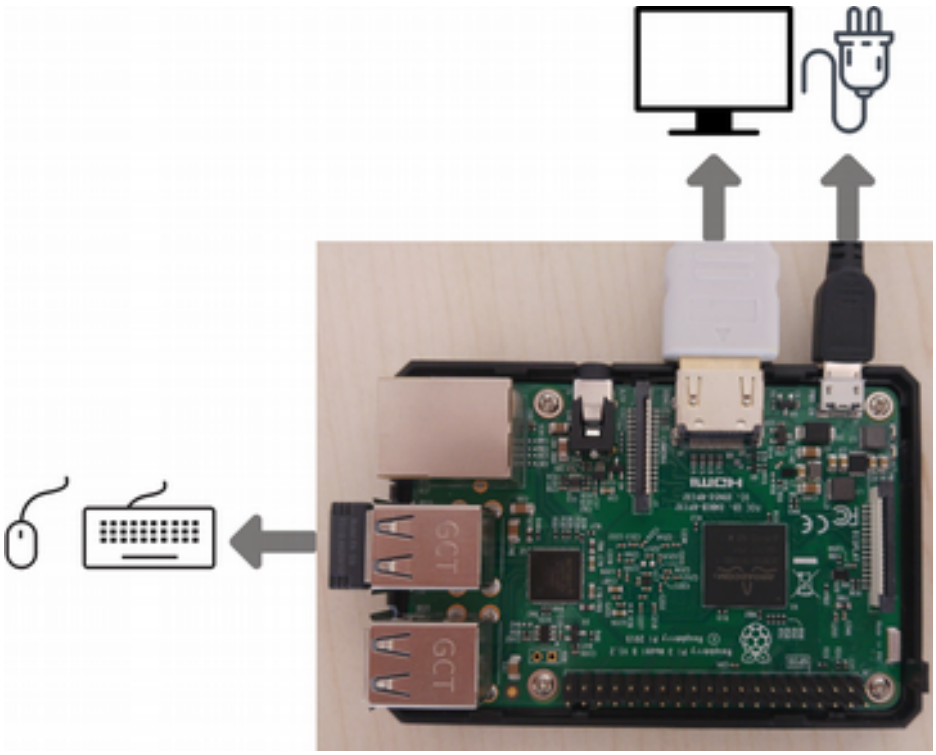
Yazma işlemi sırasında aşağıda verilen ekran çıktıları ile karşılaşırsınız.





MicroSD Karttan Raspberry Pi 3'e PilarOS'un Kurulması

Mikro SD karta yüklediğimiz işletim sistemini test etmek için SD kartımızı Raspberry PI kartına yerleştirmemiz ve aşağıda verilen bağlantıları kurmamız gerekiyor.



Yukarıda görüleceği üzere monitör, klavye, fare ve güç bağlantılarının nasıl yapılacağı gösterilmektedir. Raspberry PI'yi başlattıktan sonra bizi aşağıdaki ekran karşılar. Bu ekranı gördüğünüz zaman yapılan tüm işlemlerin başarıyla tamamlandığını anlayabiliriz.

Yukarıdaki karşılama ekranından itibaren ilk açılışta yaklaşık 5 dakika sonra aşağıda görüleceği üzere **PilarOS** işletim sistemini kolaylıkla kullanabilirsiniz. Daha sonra sistem açılmaları çok daha kısa sürecektir.

PilarOS'un Raspberry Pi 3 Linux Ortamında Kurulumu

Bu başlık altında Linux ortamında PilarOS'u SD karta yüklemek için yapılması gereken işlemleri sizlere aktarmaya çalışacağız.

Donanım İhtiyaçları

PilarOS'u Gateway olarak kullanabilmek için ihtiyacımız olan donanımların listesi şu şekildedir:

- Raspberry PI 3 Geliştirme Kartı,
- Mikro SD Kart (Minimum 4GB),
- Linux İşletim Sistemi ve SD Kart okuyucu.

Bunları temin ettikten sonra diğer adıma geçebiliriz.

Image Dosyasının İndirilmesi ve Açılması

Windows ortamında olduğu gibi aşağıda verilen listeden istediğiniz herhangi bir PilarOS versiyonunu indirmeniz gerekiyor.

Dosyayı indirdikten sonra yapılması gereken diğer bir işlem ZIP'li olan imaj dosyasını açmaktır. Linux ortamında bu işlemi yapmak için **unzip** komutunu kullanmamız gerekiyor.

Komut satırında yazacağınız aşağıdaki komut ile imaj dosyasını açabilirsiniz.

Örnek: **unzip Pilaros-N-rpi3_20160927_HDMI_8GB.img.zip**

MicroSD Kart'a Image'ın Yazdırılması

Linux ortamında PilarOS işletim sisteminin imaj dosyasını SD karta yazdırmak için aşağıda verilen adımları takip etmeniz yeterlidir.

- PC'nizdeki mikro kart okuyucuya Micro SD kart takın.
- Fdisk komutuyla SD kartın aygıt adını öğrenin.

  Örneğin: **fdisk -l**

- İmaj dosyasını SD Karta yazmak için aşağıda verilen komutu kullanmanız gerekiyor.

dd if=/home/ardic/Desktop/Pilaros-N-rpi3-20160927_HDMI_8GB.img of=/dev/sdc bs=4096

Bu işlem yaklaşık 5 dakika sürebilir. İşlem tamamlandıktan sonra **sync** komutunu icra ederek işlemi tamamen sonlandırabilirsiniz.

Bu son komutun bitmesinden sonra SD kartınızı çıkarabilirsiniz.

MicroSD Karttan Rasperry Pi 3'e PilarOS'un Kurulması

Windows işletim sisteminde takip edilen adımları aynen uyguladığınızda PilarOS işletim sistemini pratik bir şekilde Raspberry PI kartına yükleyebilirsiniz. Burada verilen bilgiler Windows ve Linux için aynı olduğu için sadece şu ekran çıktısı ile bu kısmı tamamlıyoruz.

IoT-Ignite API

Bu başlık altından IoT-Ignite platformunun geliştirilmesinde kullanılan API'ler hakkında bilgiler vereceğiz. Bu SDK'ler IoT-Ignite için oldukça önemlidir.

IoT-Ignite Device SDK - (JAVA DOC) Referansları

IoT-Ignite API'lerinin listesi aşağıdaki gibidir.

com.ardic.android.iotignite.build

Geçerli olan IoT-Ignite yapısı hakkında bilgi sağlayan sınıftır.

com.ardic.android.iotignite.callbacks

Bu arayüz, IoT-Ignite API ile kurulan bağlantı durumunu yönetmeyi sağlar.

com.ardic.android.iotignite.connection

Node ve Thing bağlantılarını kontrol etmeyi sağlayan sınıftır.

com.ardic.android.iotignite.constants

IoTIgnite uygulamalarında kullanılan bazı sabitlerin tanımlandığı sınıftır. Bilgi mesajı ve başlangıç değeri gibi sabitler yer alır.

com.ardic.android.iotignite.enumerations

Çeşitli alt sınıflardan meydana gelen bu sınıf ile şu işlemler yapılabilir.

- NodeType ile Node örnekleri üretmek için Node tiplerinin numaralandırılması sağlanır.
- ThingCategory ile Thing nesneleri için kategori tanımlanır.
- ThingDataType ile Thing, yani sensörün türü belirlenir.
- ThresholdType ile eşik tipleri numaralandırılır.

com.ardic.android.iotignite.exceptions

UnsupportedVersionException tipinde numaralandırma sağlayan sınıftır. Genel olarak aşağıdaki işlemleri yapar.

- Kimlik doğrulaması yapılmadığı zaman exception gönderir.
- IoTIgnite kullanılırken client bağlantısı null olduğunda exception gönderir.
- IoTIgnite SDK ve IoT Agent uyum sorunu olduğunda exception gönderir.

com.ardic.android.iotignite.interfaces

Bu paket aşağıdaki arayüzler ile gelmektedir.

- IIotIgniteManager: IoTIgnite sınıfının giriş noktasıdır.
- INode: Things yani sensörler için Node arabirimi sağlar.
- IThing: IoT sensörü veya bir actuator için kullanılır.
- IThingManager: Thing nesneleri için yönetici arayüzü sağlar.

com.ardic.android.iotignite.listeners

Bu arayüz ile gelen arayüzler aşağıdaki gibidir.

- NodeListener: Node nesneleri için dinleyici tanımlar.
- ThingListener: Thing nesneleri için dinleyici tanımlar.

com.ardic.android.iotignite.nodes

Bu paket ile gelen sınıflar ve görevleri aşağıdaki gibidir.

- IoTIgniteManager: IoTIgnite kütüphanesinin giriş noktasıdır. İşlemler buradan yapılmaya başlar.
- IoTIgniteManager.Builder: IoTIgniteManager sınıfından bir nesne oluşturmayı sağlayan sınıftır.
- IoTIgniteManager.NodeFactory: Node oluşturmak için gerekli olan tanımlamaları yapan sınıftır.
- Node: INode arayüzünü uygulayan sınıftır.

com.ardic.android.iotignite.services

Arayüz, class ve enum yapılarından meydana gelen bir paktettir. Sahip olduğu yapılar ve görevleri şunlardır.

- IPinService: Pin için arayüz tanımlamayı sağlar.
- Gpio: Raspberry Pi 3 pinlerini haritalamak için kullanılan sınıftır.
- PinService: IPinService arayüzünü uygulayan sınıftır.

- Gpio.Mode, Gpio.Model, Gpio.Number ve Gpio.Value enum yapıları ise Gpio ile ilgili işlemlerde kullanılır.

com.ardic.android.iotignite.things

Bu paket ile gelen sınıflar ve görevleri aşağıdaki gibidir.

- Thing: IoTIgnite platformunda son noktaları temsile eden bir sınıftır. Thing kavramı sensor ve actuator için kullanılır.
- ThingActionData: Actuator gibi etkin olan Thing nesneleri için kullanılan eylem verilerini tanımlamak için kullanılır.
- ThingConfiguration: Thing nesnesi için yapılandırma ayarlarını tanımlamak için kullanılır.
- ThingData: ArCloud ortamına veri göndermeyi sağlayan ThingData oluşturmayı sağlamak için kullanılır.
- Bunların dışında Thing ile ilgili diğer işlemler için ThingManager, ThingManager.ThingFactory ve ThingType sınıfları da kullanılmaktadır.

com.ardic.android.iotignite.utils

Bu paket ile gelen temel iki sınıf aşağıdaki gibidir.

- ListUtils: Listeler ile çalışmayı sağlayan yardımcı fonksiyonlar sağlayan sınıftır.
- LogUtils: Loglama işlemlerinde kullanılan sınıftır.

IoT-Ignite Kütüphanesi Nasıl Kullanılır

IoT-Ignite API'sini kullanmak için bilmemiz gerekenleri iki başlıkta altında ele alacağız.

IoT-Ignite Device (Cihaz) SDK

Ignite Device SDK'nın üç temel adım vardır;

- IoT Ignite SDK ile uygulama geliştirme alt platformunu oluşturmak,
- IoT Ignite API tanıtımı,
- IoT Ignite platformunda çalışmaya hazır sensörleri edinmek ve Gateway ile bütünleşmesini sağlamak.

Yukarıdaki işlemleri Eclipse ve Android Studio ortamında şu şekilde yapabiliriz:

Eclipse

```
IoTIgnite
https://repo.iot-ignite.com/content/repositories/releases

com.ardic.android
IoTIgnite
0.8.2
```

Android Studio

Geliştirdiğiniz uygulamanın **build.gradle** dosyasında **Repositories** ve **Dependencies** bloklarına aşağıdaki satırları eklemeniz gerekiyor.

```
1    repositories {
2        mavenCentral()
3        maven {
4            url "https://repo.iot-ignite.com/content/repositories/releases"
5        }
6    }
```

```
7    dependencies {
8        compile 'com.ardic.android:IoTIgnite:0.8.2'
9        compile 'com.google.code.gson:gson:2.7'
10   }
```

IoT-Ignite Manager

IoT-Ignite SDK kurulumunu yukarıdaki yöntemlerden biriyle yaptıktan sonra kodlama kısmına geçebiliriz. Burada kullanacağımız IoT-Ignite Manager sınıfı Ignite SDK'nin giriş noktasıdır.

IoTIgniteManager nesnesi oluşturmak için iki parametre gereklidir:

- context: Android uygulama context nesnesidir.
- connectionCallBack: IoT-Ignite Agent bağlantısını bildiren dinleyicidir.

```
40  try {
41        IotIgniteManager mIotIgniteManager = new IotIgniteManager.Builder()
42              .setContext(getApplicationContext()).setConnectionListener(new
ConnectionCallback() {
43
44                  @Override
45                  public void onConnected() {
46                      Log.i("IgniteSample", "IoT-Ignite Connected!");
47                  }
48
49                  @Override
50                  public void onDisconnected() {
51                      Log.i("IgniteSample", "IoT-Ignite Disconnected!");
52                  }
53              }).build();
54
55            } catch (UnsupportedVersionException e) {
56      if (UnsupportedVersionExceptionType.UNSUPPORTED_IOTIGNITE_AGENT_VERSION.
57              toString().equals(e.getMessage())) {
58          Log.e(TAG, "UNSUPPORTED_IOTIGNITE_AGENT_VERSION");
59      } else {
60          Log.e(TAG, "UNSUPPORTED_IOTIGNITE_SDK_VERSION");
61      }
62 }
```

IgniteManager'ı bağlamadan, IoT-Ignite'da herhangi bir işlem yapılamaz. IoT-Ignite'ın onConnected() metodu tetiklendikten sonra Node, NodeFactory kullanılarak oluşturulabilir.

Node sınıfı, sensor ve actuator gibi bileşenleri kontrol etmek için kullanılır. Bunlar, mikro denetleyici birimi (MCU) gibi tek bir fiziksel aygıta veya bir yazılım tanımlı node aygıta bağlanabilir. Örneğin, bir mobil cihaza bağlı ArduinoYun kartı, bir Node olarak kaydedilebilir ve

sensörler ve aktüatörler bunlara Thing olarak bağlanabilir. Her Node veya Thing bileşeni ayrı ayrı yapılandırmak mümkündür.

Örnek bir Node aşağıdaki gibi oluşturulur ve kaydedilir:

```
40    Node myNode = IotIgniteManager.NodeFactory.createNode("Security Node",
"Home Security Node", NodeType.GENERIC, "HSN1", new NodeListener() {
41     @Override
42     public void onNodeUnregistered(String s) {
43         Log.i(TAG, "Security Node unregistered!");
44     }
45  });
```

```
46  myNode.register(); // Node bağlantısı başarıyla sağlanırsa, true döner
```

nodeID her aygıt için benzersiz olmalıdır. Dahili sensörler ve dahili İşlemciler gibi ayrılmış nodeID'ler bulunmaktadır. Bu ayrılmış kimlikler IoT-Ignite Agent App tarafından kullanılır, dolayısıyla diğer uygulamalar bu ID bilgilerini kullanamazlar.

Node kaydı yapılmadığı durumlarda **onNodeUnregistered** metodu tetiklendi. Tüm Node işlemleri, o Node örneği içinde işlenir. myNode.setConnected(true) metodu, Node'un bağlantı durumunu IoT-Ignite Cloud'da çevrimiçi olarak ayarlar. Bir node çevrimiçi olmadığı sürece, o node altında kayıtlı sensör ve actuator bileşenlerine herhangi bir işlem uygulanamaz.

Thing

Sensörler veri üreten aygıtlar olarak kullanılırken, actuator'lar ise bir mesaj veya komuta cevap veren ve IoT sistemlerinde bazı eylemlerde bulunan bileşenler olarak kullanılır. Thing bileşenler genelde donanım olarak kullanılır ancak sanal sensör ve actuator de oluşturulabilir. Örneğin, bir uygulama tarafından sanal sensör tanımlamak mümkündür. Tanımlanan her bir Thing bileşeni fiziksel olsun veya sanal olsun fark etmez, mutlaka bir Node'a aygıta bağlanmalıdır. Bir Thing oluşturmak için IoT-Ignite bağlantısına ve kayıtlı bir Node aygıtına ihtiyaç vardır.

Thing'ler kayıtlı bir node ile kaydedilebilir.

Android işletim sistemli cihazda bir sensör kayıt etmek için aşağıdaki kodları kullanırız.

```
1 Thing myAndroidThing=myNode.createThing(myAndroidSensor,new ThingListener(){
2       @Override
3       public void onConfigurationReceived(Thing thing){
4
5       }
6
7       @Override
8       public void onActionReceived(String nodeId, String thingID, ThingActionData,
thingActionData){
9
10      }
11
12      @Override
13      public void onThingUnregistered(String nodeId,String thingID){
14
15      }
16 });
```

```
1  myAndroidThing.register();
```

Herhangi bir sensör veya actuator Android sensörlerinin dışındaysa:

```
1  ThingType type = new ThingType("myType","vendor", ThingDataType.STRING);
```

```
6  Thing myOtherThing = myNode.createThing("myUniqueThingID", type,
ThingCategory.EXTERNAL, false, new ThingListener() {
7         @Override
8         public void onConfigurationReceived(Thing thing) {
9
10        }
11
12        @Override
13        public void onActionReceived(String nodeID, String thingID, ThingActionData
thingActionData) {
14
15        }
16
17        @Override
18        public void onThingUnregistered(String nodeId, String thingID) {
19
20        }
21
22 });
```

Thing bileşenlerin bağlantı durumu myThing.setConnected(true) metodu tarafından belirlenir ve Cloud'a bildirilir. Eğer bu ifade kullanılmazsa, sensör veri gönderse bile offline olarak görünür.

ThingListener, ThingConfiguration, Actuator

IoT-Ignite platformunda kayıtlı olan her bir Thing için bir listener bulunmaktadır. Listener yardımıyla, Thing bileşenler, IoT-Ignite Cloud ile kolaylıkla kontrol edilebilir. Cloud, veri gönderme periyodu, yerel saklama alanındaki veri önbellekleme süresi gibi yapılandırma parametrelerini sunabilir.

Eğer thing bileşen bir sensör değil de actuator olarak kayıt edilmişse, eylem ile ilgili işlemler onActionReceived(String nodeID, String thingID, ThingActionData thingActionData) metodu tarafından bilgilendirilir.

Eğer thing ya da node silinirse, onThingUnregistered(String nodeId, String thingID) metodu otomatik olarak tetiklenir.

ThingType, ThingDataType, ThingCategory, ThingActionData

Thing nesnesi oluşturulurken ThingType ve ThingCategory parametreleri kullanılır. Bu parametrelerin görevi aşağıdaki gibidir:

- **ThingType**: Sensör türünü belirler. Bu parametre, sensör tipi ve satırı/üretici hakkındaki bilgileri depolar.
- **ThingCategory**: Sensör kategorisini belirler. İki kategori türü bulunmaktadır. Gateway cihazda dahili olarak bulunan sensörler BUILTIN kategorisinde, cihaz dışında yer alan yani harici sensörler EXTERNAL kategorisinde yer alır.

ThingDataType, sensör verilerinin cihaz veya bulutta nasıl yorumlanacağını belirtir. Şu anda FLOAT, INT, STRING ve GEOFENCE, ThingDataType için desteklenmektedir.

ThingActionData, bazı olaylardan sonra oluşan eylemleri gerçekleştirmek için kullanılan bir mesajlaşma yapısı olarak tanımlanabilir. IoT-Ignite Cloud'daki bu yapı kural oluşturucu ve yazılım geliştirici arasında bulunur ve herhangi bir standart adım gerektirmez.

Örneğin, bir LED'nin bir aktüatör olarak kaydedildiğini varsayalım. LED’i hedefleyen kural belirli bir eylem mesajıyla tanımlanabilir. İşlem mesajı JSON formatında olmalıdır. Amacınız bu LED’i açıp kapamak ise sırasıyla “1” veya “0” değerlerini LED’e gönderebiliriz. Cihaz uygulaması böyle bir mesaj aldığında, bu mesajın amacını ve hedefini anlayıp gerekli işlemleri yapabilmelidir.

BÖLÜM 7

Node Katmanı (Sensors & Actuators)

Node

ARDIC'ın geliştirdiği IoT-Ignite ile geliştirilen IoT uygulamalarında bulunan sensörlerden veriyi alıp Gateway'e gönderen cihazlar Node olarak tanımlanmaktadır. Gateway cihazlar, Node'dan gelen verileri buluta göndermeyi sağlarken, Node aygıtlar ise sensör ve actuator denilen çevre birimlerden gelen verileri toplamayı sağlamaktadır.

Node olarak kullanabileceğimiz aygıtlar oldukça fazladır. En çok tercih edilen Node aygıtlar aşağıdaki gibidir:

- Arduino
- NodeMCU
- Raspberry Pi

Dinamik Node Örneği

Bu uygulama IoT-Ignite tarafından sağlanan örnek bir uygulamadır. Örnek kodlara GitHub üzerinden erişim sağlayabilirsiniz. Dinamik Node uygulamasında, sıcaklık ve nem sensörleri ile elde edilen veriler IoT-Ignite platformuna wifi üzerinden gönderilmesi sağlanır. Bunun için Esp8266 NodeMCU geliştirme kartını kullanacağız. Bu kartın en önemli özelliği; sensörlerden aldığı verileri wifi ile internet ortamına gönderebilmesidir.

Hazırlıklar

Hazırlık aşamasında uygulamaya başlamadan önce yapılması gereken temel bazı ayarlarmalar yer almaktadır. Özellikle bunları temin ettikten sonra uygulamaya geçmemiz gerekiyor. Hazırlık aşamasını donanım ve yazılım olarak iki başlık altında ele alabiliriz.

Donanım

Dinamik Node uygulamasında kullanacağımız donanımların tam listesi aşağıdaki gibidir. Öncelikle bu donanımları temin etmeniz gerekiyor.

- Esp8266 NodeMCU
- DHT11 sıcaklık ve nem sensörü
- LED
- 1 adet Breadboard
- 1 adet Push Button
- 1 adet 10K Ohm direnç
- 1 adet 330 Ohm direnç

- 5 veya 6 adet M-M (Erkeke-Erkek) Jumper kablo
- Android IoT Gateway

Kurulum ve çalıştırma aşamasında bunları nasıl bağlayacağımızı örnek bir şema üzerinde göstereceğiz.

Yazılım

Dinamik Node uygulaması için donanımları temin ettikten sonra, sıra ihtiyacımız olan yazılımları temin etmeye gelir. Öncelikle bu yazılımları elde etmeniz gerekiyor.

- Android Studio Geliştirme Ortamı
- Arduino IDE
- IoTIgniteAgent Uygulaması
- Service Provider Application (SPA)
- Arduino IDE ESP Board Kurulumu

Yazılım Kütüphaneleri

IoT-Ignite ile uygulama geliştirdiğimizde Android ve Arduino kısmı için ihtiyacımız olan bazı yerel kütüphaneler bulunmaktadır. Bunları şu şekilde sıralayabiliriz.

- Android için geliştirilen kütüphaneler (Güncel versiyonları “Sürümler” sayfasında bulabilirsiniz).
  - IoT-Ignite
  - HwNodeAppTemplate
- Arduino için geliştirilen kütüphaneler:
  - IgniteIotLibs (Arduino IDE kurulumu sonrası oluşturulan "Arduino/libraries/" klasörüne kopyalanması gerekiyor.)
  - Esp8266 Sketch Data Upload Plugin

Kurulum ve Çalıştırma

Kurulum işlemi için yapılması gereken ilk işlemler Gateway’e IoT Ignite Agent uygulamasını kurup lisansladıktan sonra NodeMCU’yu Arduino ile programlamaktır. Bu işlemler kısaca 11 maddede anlatıldıktan sonra her biri ayrıntılı şekilde resimlerle aşağıda bahsedilecektir.

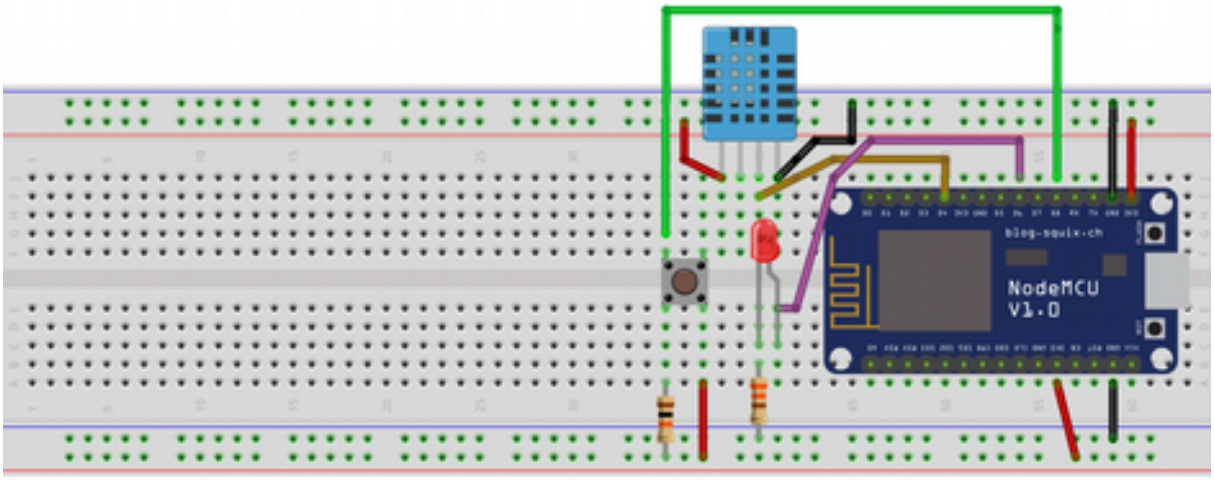
1. Gateway olarak kullanacağınız Android cihaza IoT-Ignite Agent uygulamasını indirilir ve lisanslanır. https://play.google.com/store/apps/details?id=com.ardic.android.iotigniteagent
2. Arduino IDE kurulduktan sonra “Esp8266 Sketch Data Upload Plugin” kurulur. (https://github.com/esp8266/arduino-esp8266fs-plugin)
3. Örnek uygulamayı (https://github.com/IoT-Ignite/arduino-sketch-dynamic-node-example) DynamicNodeExample.ino olarak derleyip NodeMCU’yu programlamanız gerekiyor.
4. Derleme için Arduino IDE’sinin belirttiği gerekli kütüphaneler eklenir

(Timer,ArduinoJson,Adafruit_Unified_Sensor, DHT_sensor_library, arduino-library-IgniteIotLibs gibi.)

arduino-library-IgniteIotLibs kütüphanesi https://github.com/IoT-Ignite/arduino-library-IgniteIotLibs adresinden indirilebilir.

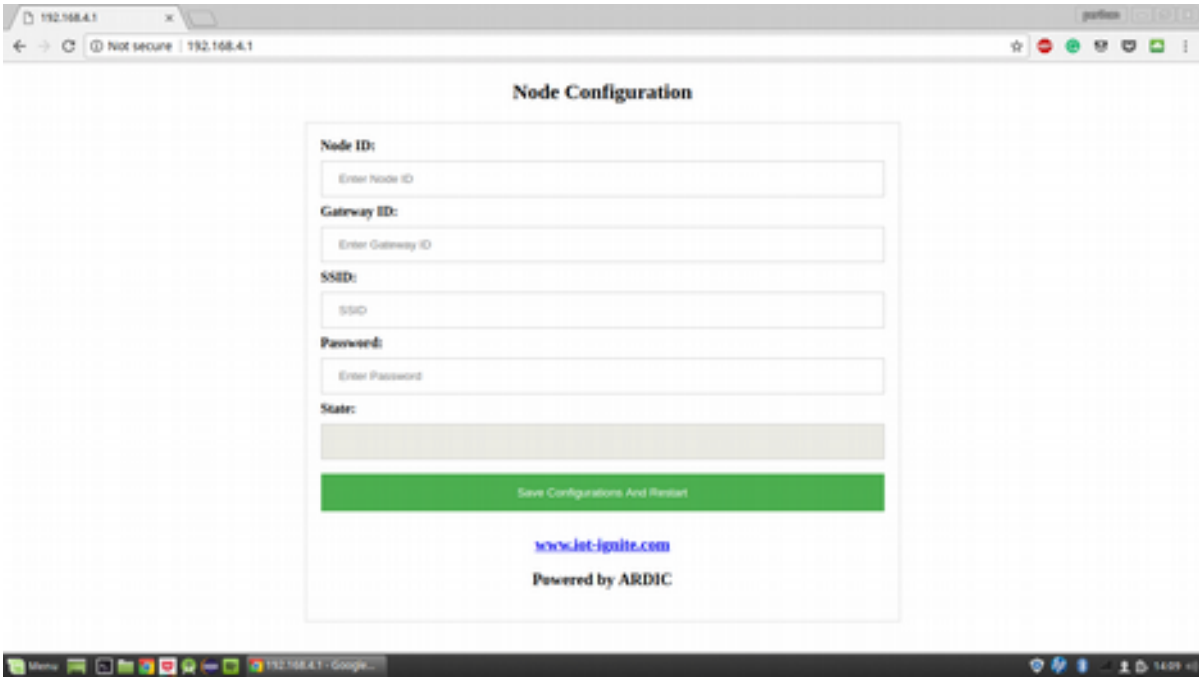
Diğer kütüphaneleri internette kolayca bulabilirsiniz.

5. NodeMCU Hazırlanır. Dinamik Node uygulaması için yapılması gereken işlemlerden bir diğeride aşağıda verilen devreyi kurmaktır.



6. DynamicNodeExample.ino derlenip upload edilince NodeMCU modülü yanıp sönecektir. Yanıp sönmeden sonra NodeMCU, AccessPoint modunda başlatılmış olur. AccessPoint modu ile NodeMCU, Hotspot ismi “Ignite” ön ekiyle başlamaktadır. Bu sayede ağda birden fazla cihaz olsa bile bağlanacağınız Node aygıtını hızlıca tespit edebilir ve kayıt altına alınabilir. Kayıt için iki yöntem vardır. İlki Web Service yardımıyla kayıt olmak bir diğeri ise SPA uygulaması yardımıyla kayıt olmak. Android 6.0 ve üzeri versiyonlar için Web Service kullanımı tavsiye edilir (Sadece Android 5.0’da SPA Node’ları bulabilir). Kullanılacak yöntemi belirleyip bu kayıt işlemini öncelikle yapmanız gerekiyor.
7. Bu kısımda Web Service yardımıyla kayıt olmayı gösterelim 8. maddede ise bunun alternatifi olan SPA uygulaması açıklanacaktır.

Herhangi bir cihazdan “Ignite” önekiyle HotSpot olan NodeMCU cihazına bağlanılır. Ardından aşağıdaki gibi 192.168.4.1 adresine gidilip bilgiler girilir.



Node ID: Belirleyeceğiniz node ismi.

Gateway ID: Gateway cihazınızda Ignite Agent uygulamasını açtığınızda Cihaz No kısmında yazan ID. (Diğer yöntem olan SPA uygulamasını kullanırsanız bu bilgi otomatik olarak gönderilecektir)

SSID ve Password: Ağ SSIDsi ve parolası.

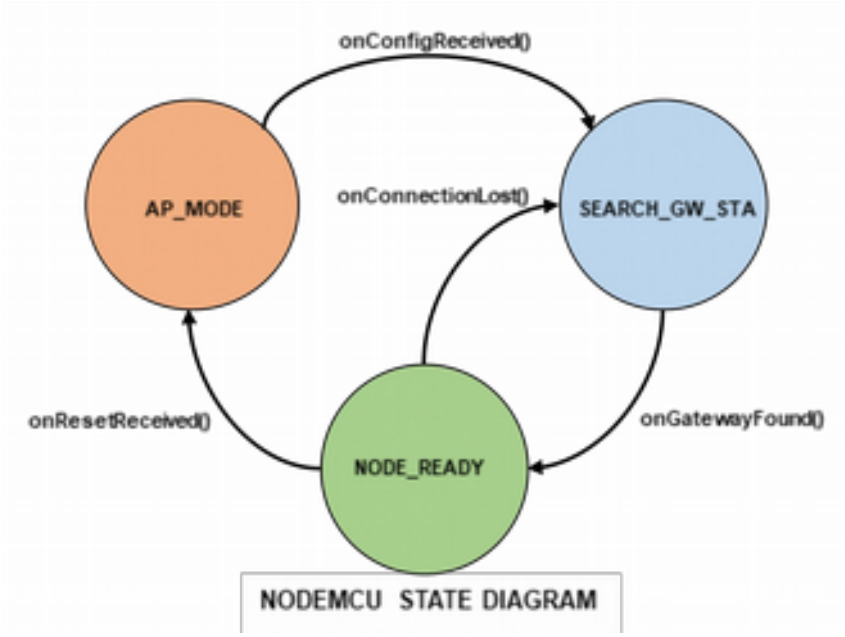
Save Configuration and Restart butonuna bastığınız zaman NodeMCU cihazınızı ve bağlı sensörlerini kaydetmiş olursunuz. SPA uygulamasını Gateway veya herhangi bir Android cihaza kurabilirsiniz.

https://download.iot-ignite.com/ServicePlatformApp/

Gateway'inizin kayıtlı olduğu hesap bilgilerinizle giriş yapıp gatewayinize access point olan NodeMCU node'unu ekleyebilirsiniz. Böylece Node'unuzu gatewaye kaydetmiş olursunuz.

8. NodeMCU, belirtilen SSID ve şifre bilgilerini kullanarak yerel ağa bağlanır. Eğer bağlantı sırasında bir hata meydana gelirse tekrar hotspot moduna döner. Eğer bağlantı sağlanırsa bu durumda, ID bilgisi verilen Gateway aygıtı ağda taranır. Ağ geçidi arama işleminde maksimum bir seviye deneme sayısı var. Bu seviye 30'dur. Eğer bu sayıya ulaşılırsa tekrar hotspot moduna dönülür. Ağ geçidi bulunduğu zamanlarda ise, Node aygıtı kendisiyle ilgili verileri ağ geçidine gönderir ve veri konfigürasyonu için kısa bir süre bekler.

NodeMCU ile Gateway arasında veri iletimi JSON formatında gerçekleşir. JSON formatının kullanılması oldukça mantıklı bir seçim. Çünkü TCP/IP iletişim protokolünde veriler genellikle bu formatta iletilir. Bu şekilde veri iletimi standartlara uygun olarak gerçekleşir.

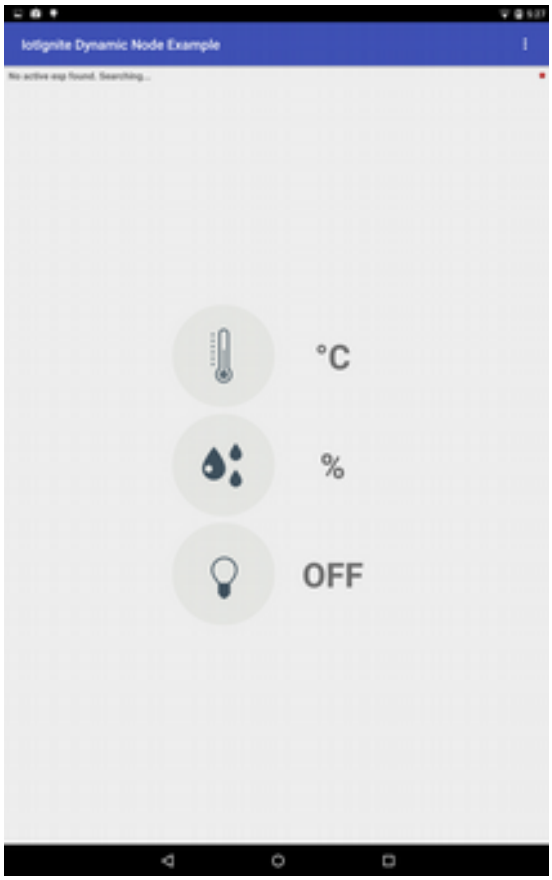


Yukarıda verilen diyagramda NodeMCU ile Gateway arasındaki iletişim kurulurken izlenen adımlar tek tek gösterilmiştir. Bağlantı işlemi kurulana kadar sürekli olarak devam eden bir döngü vardır. Bu döngünün amacı NodeMCU ile Gateway arasındaki bağlantıyı sağlamaktır.

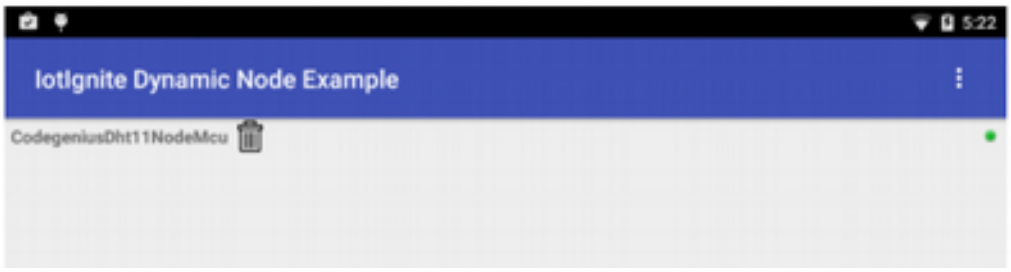
9. Gateway'e DynamicNodeExample uygulamasını (https://github.com/IoT-Ignite/android-example-DynamicNode/releases) adresinden indirip kurmanız gerekiyor. Bir değişiklik yapmak isterseniz repoyu indirerek de düzenleyebilirsiniz. Bu uygulama Dinamik Node uygulaması için geliştirilen özel bir Android uygulamasıdır. Temel amacı NodeMCU ile

kablosuz olarak bağlantı sağlamak ve NodeMCU'ya bağlı olan sensör verilerini kullanıcıya sunmaktır.

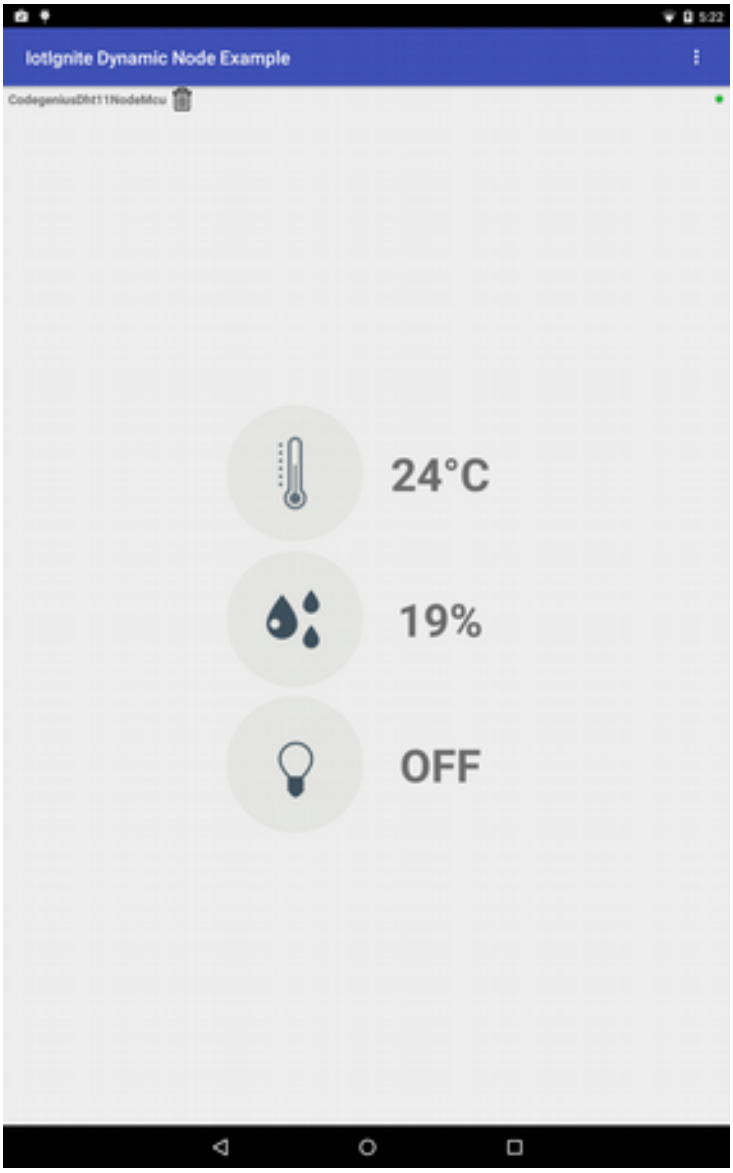
DynamicNodeExample uygulamasını ilk açtığınızda ve herhangi bir Node bağlamadığınızda aşağıdaki gibi bir ekran ile karşılaşırız.



DynamicNodeExample App'ine bir Node bağladığımız zaman ise aşağıdaki gibi bir ekran çıktısı ile karşılaşırız.



Uygulamanın en tepesinde Node bağlantısının kurulduğunu gösterene bir mesaj okumaktayız. Ayrıca bağlantı işleminden sonra NodeMCU'dan gelen sensör verilerini ekrandan aşağıdaki gibi okuyabiliriz.



Kurulum ve çalıştırma işlemleri pratik olarak bu şekilde gerçekleşmektedir. Bir diğer konu da ayrıntılarda gizli. İhtiyacımız olan kodları açıklamak adına bir sonraki başlığı inceleyelim.

Ayrıntılar

Bu başlık altında uygulama kodlarından ve arkaplanda yaptığımız bazı işlemlerden bahsedeceğiz.

Bağlantı işlemini yapan Arduino kodumuz aşağıdaki gibidir.

```
#include <Arduino.h>
#include "IgniteEsp8266WifiManager.h"
#include "IgniteEsp8266ThingHandler.h"

IgniteEsp8266ThingHandler handler;
IgniteEsp8266WifiManager manager(&handler);

void setup() {
  manager.setup();

void loop() {
  Serial.println(WiFi.localIP());
  manager.loop();
}
```

Bağlantı işlemini kurduktan sonra sıra Node ile bağlantıda olan sensörlerden veri okumaya geldi. Burada kullanılabilecek bir kütüphane var. Bu kütüphane ile ağ geçidi bağlantısı ve veri iletişimi ile ayrıca uğraşmak zorunda kalmazsınız. InventorySetup isminde bir fonksiyon ile tanımlanan bu kütüphaneyi kullanarak, sensörden gelen verileri kolay bir şekilde okuyabilirsiniz. Bu fonksiyon, hem yapılandırma işlemlerini hem de müşteri uygulamasına veri göndermeyi sağlamak için kullanılmaktadır. Bunun için kurulum talimatlarında IgniteIotLibs kütüphanesini "Arduino/libraries/" yoluna eklemiştik.

Sensör yapılandırması ve sensörlerden gelen verileri okumak için ihtiyacımız olan Arduino kodu aşağıdaki gibidir:

```
void IgniteEsp8266ThingHandler::inventorySetup() {

  addThingToInventory(SENSOR_DHT11_TEMPERATURE,
                      TYPE_TEMPERATURE,
                      PIN_DATA_DHT11_SENSOR,
                      NOT_ACTUATOR,
                      VENDOR_DHT11,
                      DATA_TYPE_FLOAT, new IgniteEsp8266Timer(readDHTTemperature));

  addThingToInventory(SENSOR_DHT11_HUMIDITY,
                      TYPE_HUMIDITY,
                      PIN_DATA_DHT11_SENSOR,
                      NOT_ACTUATOR,
                      VENDOR_DHT11,
                      DATA_TYPE_FLOAT, new IgniteEsp8266Timer(readDHTHumidity));

  addThingToInventory(ACTUATOR_BLUE_LED,
                      TYPE_LED,
                      PIN_DATA_BLUE_LED,
                      ACTUATOR,
                      VENDOR_BLUE_LED,
                      DATA_TYPE_STRING, new IgniteEsp8266Timer(readLedData));

}

void IgniteEsp8266ThingHandler::unknownMessageReceived(String msg) {
}

void  IgniteEsp8266ThingHandler::readDHTTemperature() {
  String packet = "";
  String tempData = "";
  float t = dht->readTemperature();
  if (isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  tempData = String(t);

  StaticJsonBuffer<100> jsonBuffer;
  JsonObject& root = jsonBuffer.createObject();
  JsonArray& data = root.createNestedArray("data");

  root["messageType"] = DATA_RESPONSE;
  root["thingId"] = SENSOR_DHT11_TEMPERATURE;
  data.add(tempData);

  root.printTo(packet);

  Serial.println("Temperature :");
  Serial.println(packet);
  packet += "\n";
  sendMessage(packet);
}

void  IgniteEsp8266ThingHandler::readDHTHumidity() {
  String packet = "";
  String humData = "";
  float h = dht->readHumidity();
  if (isnan(h)) {
    Serial.println("Failed to read from DHT sensor!");
```

```
    return;
  }
  humData = String(h);

  StaticJsonBuffer<100> jsonBuffer;
  JsonObject& root = jsonBuffer.createObject();
  JsonArray& data = root.createNestedArray("data");

  root["messageType"] = DATA_RESPONSE;
  root["thingId"] = SENSOR_DHT11_HUMIDITY;
  data.add(humData);

  root.printTo(packet);
  Serial.println("Humidity :");
  Serial.println(packet);
  packet += "\n";
  sendMessage(packet);

}
```

Node uygulamasının Arduino kısmı yukarıdaki gibidir. Bir de Android Studio tarafında daha sonra geliştireceğimiz müşteri uygulaması bulunmaktadır. Burada bizi ilgilendiren bazı kod bloklarına kısaca değineceğiz.

DynamicNodeExample uygulaması çalışırken, arkaplanda Network Service Discovery (NSD – Ağ Hizmeti Bulma) hizmeti başlatılır. Ayrıca gelen bağlantılar için bir TCP/IP sunucusunda başlatılır. Node aygıtı ağa bağlandığı zaman IoT-Ignite platformuna kayıt edilir ve node verileri HwNodeAppTemplate akışıyla senkronize olarak okumaya başlanır. Bu örnek uygulamada, müşteri uygulaması yalnızca arayüz bölümünü icra eder. Geri kalan tüm işlevler HwNodeAppTemplate kütüphanesi tarafından işlenir.

Bilmemiz gereken ve bizi ilgilendiren kod blokları aşağıdaki gibidir.

```
167       @Override
168       protected void onCreate(Bundle savedInstanceState) {
169           super.onCreate(savedInstanceState);
170           setContentView(R.layout.activity_main);
171           // Servis HwNodeAppTemplate içinde başlatılır.
172           startService(new Intent(this, WifiNodeService.class));
173           // Uyumluluk olay dinleyicisi tanımlanır.
174           WifiNodeService.setCompatibilityListener(this);
175           Log.i(TAG, "Dynamic Node Application started...");
176           // Kullanıcı arayüzü başlatılır.
177           initUIComponents();
178           initSensorDatas();
179           /* İletişimi sağlamak için Genric Device ve Generic Wifi Node yöneticisi
başlatılır.*/
180           initEspDeviceAndNodeManager();
181       }


401       private void initEspDeviceAndNodeManager() {
402           espManager = GenericWifiNodeManager.getInstance(getApplicationContext());
403           // Ağda tespit edilen cihaz alınır ve işlemler yapılır.
404           espManager.addWifiNodeManagerListener(this);
405
406           for (GenericWifiNodeDevice dvc : espManager.getWifiNodeDeviceList()) {
407               checkAndUpdateDeviceList(dvc);
408           }
409
410       }
```

Yukarıda verilen kod içinde yer alan WifiNodeManager ile ağda olan Node aygıtlar kolay bir şekilde işlenir. HwNodeAppTemplate kütüphanesi ağda yer alan tüm düğümleri almakta

yeteneklidir. Ancak bu uygulama için bize lazım olan Node aygıtları **DYNAMIC NODE - DHT11 SENSOR** tipinde olmalıdır.

DYNAMIC NODE - DHT11 SENSOR tipine sahip aygıtları almayı sağlayan kod bloğumuz aşağıdaki gibidir:

```
422       private void checkAndUpdateDeviceList(BaseWifiNodeDevice device) {
423           if
(DynamicNodeConstants.TYPE.equals(device.getWifiNodeDevice().getNodeType())) {
424
425           Log.i(TAG, "New node found adding to list.");
426           espNodeList.add(device);
427
428           updateActiveEsp();
429       }


523       private void updateActiveEsp() {
524       // Eğer sadece bir aygıt bulunursa bu aygıt aktif duruma getirilir.
525               if (espNodeList.size() == 1) {
526                   activeEsp = espNodeList.get(0);
527               }
528
529               if (activeEsp.getNode() != null) {
530                   activeEsp.addThingEventListener(espEventListener);
531                   setUINodeId(activeEsp.getNode().getNodeID());
532                   isActiveEspConnected = activeEsp.getNode().isConnected();
533                   setConnectionState(isActiveEspConnected);
534               }
535       }
```

Müşteri uygulaması hakkında daha sonra ayrıntılı bilgiler verip uygulamanın kaynak kodlarını inceleyeceğiz.

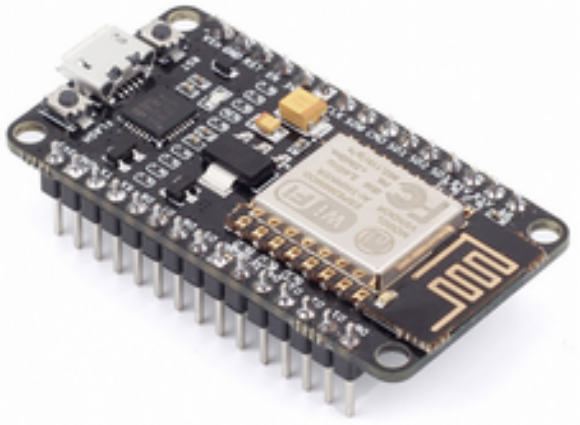
ESP8266 ve ESP32 Modülleri

IoT-Ignite platformunda Node aygıtı olarak sıklıkla kullanacağımız modüllerin başında ESP8266 ve ESP32 gelmektedir. Arduino IDE geliştirme ortamında programlanan bu modelleri hakkında bilgi vermekte fayda var.

ESP8266

ESP8266, Espressif tarafından üretilen bir Wifi modülüdür. Arduino gibi geliştirme kartı ve mikro denetleyicilerin çoğuyla kullanılabilir. Popülerdir ve iyi bir fiyat / performans oranı sağlamaktadır. ESP8266 bir Wi-Fi modülüdür. -40 ile +125 derece arasında çalışabilir.

Bu modülün örnek görüntüsü aşağıdaki gibidir.



Tam internet bağlantısı sağlamak için tasarlanmış oldukça küçük bir yapısı bulunan bu modül ile çevre birimlerden toplanan veriler kolaylıkla ağ ortamına gönderilebilir. IoT-Ignite ile çalışırken bu modülü, sensörlerden gelen verileri Gateway cihaza iletmek için kullanırız.

Bu modülün teknik özellikleri aşağıdaki gibidir:

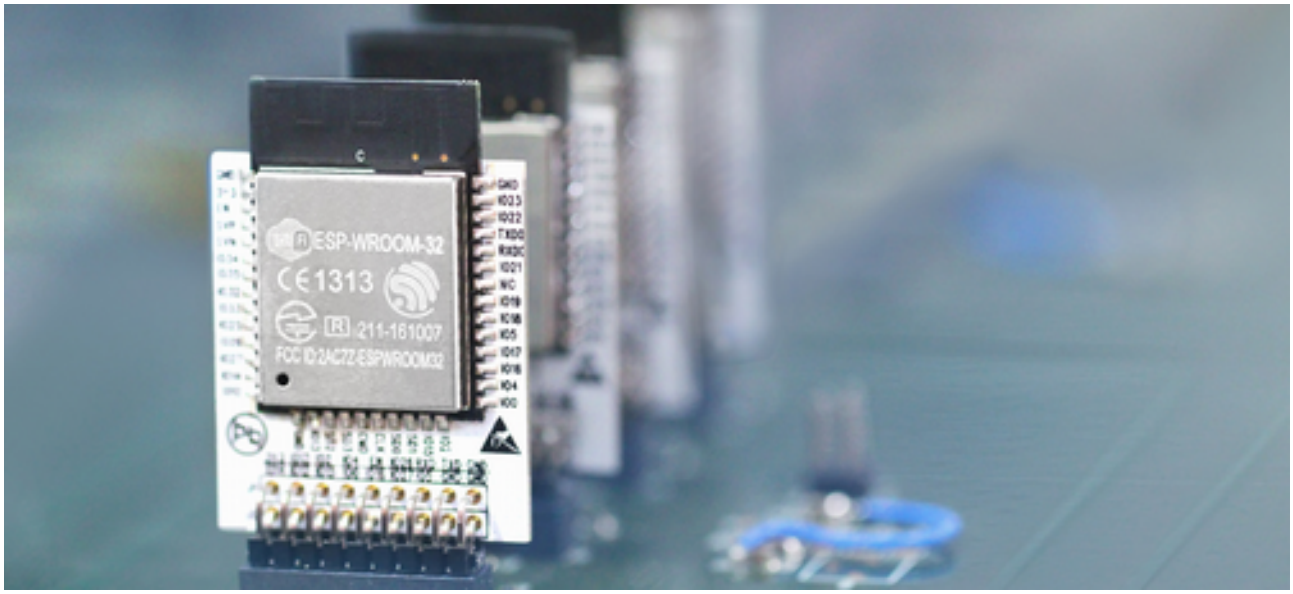
- 802.11 b / g / n
- Wi-Fi Direct (P2P)
- Dahili TCP / IP protokol desteği
- Dahili voltaj regülatörü ve güç yönetim bileşenleri
- 802.11b modu + 19.5 dBm çıkış gücü
- Dahili sıcaklık sensörü
- Dahili düşük güç tüketimini sağlayan 32-bit işlemci
- SDIO 2.0, SPI, UART
- Veri paketlerini aktarmak için düşük zaman avantajı (2ms'de)

Burada verilen Wifi desteği, düşük güç gereksinimi ve veri aktarımının 2 ms'de yapılması gibi avantajlarından dolayı IoT uygulamalarında sıklıkla tercih edilmektedir.

ESP32

Espressif tarafından üretilen ESP32, ESP serisinin en yeni modülüdür. Giriş/Çıkış pinlerinin çoğu, kolay bir arabirim için her iki tarafın pin başlıklarına yönlendirilir. Geliştiriciler pinlere gerektiği gibi çevre birimlerini kolay bir şekilde bağlayabilir. Standart başlıklar uygulama geliştirmeyi kolay ve kullanışlı bir hale getirmeyi sağlar.

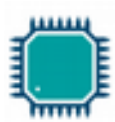
Bu modülün örnek görüntüsü aşağıdaki gibidir.



ESP32'nin resmi web sitesinden elde ettiğimiz bilgilere göre bu modülü diğer modüllerden ayıran temel özellikler 4 kategoride incelenebilir:

Hybrid Wi-Fi & Bluetooth Chip

Wi-Fi ve Bluetooth sağlamak adına diğer sistemler ile SPI/SDIO veya I2C/UART arabirimleri aracılığıyla bağlantı oluşturabilmesidir. Böyle bir destek ile IoT uygulamalarında, ana MCU'dan bağımsız veya bağımlı bir cihaz olarak çalışabilir. Bağımsız olduğu durumlarda uygulamanın merkezinde yer alan cihazın işlemcisinde iletişim desteğini azaltır. Çünkü bu gibi durumlarda sadece gereken ve ihtiyaç duyulan bilgileri merkezi cihaza gönderir. Acil olmayan işlemleri kendisi yerel olarak hızlı bir şekilde gerçekleştirebilir.



High Level of Integration

ESP32, dahili anten anahtarları, RF balun, güç amplifikatörü, düşük gürültülü alıcı amplifikatör, filtreler ve güç yönetimi modülleri ile yüksek seviyede entegre bir modüldür. Özellikle asgari Baskı Devre Kartı (PCB) gereksinimleriyle uygulamalarınıza paha biçilemez işlevsellik ve çok yönlülük katmayı sağlamak için tasarlanmıştır.

Ultra-Low Power Consumption

Mobil cihazlar, giyilebilir elektronik ve IoT uygulamaları için tasarlanan ESP32, çeşitli özel yazılım türlerinin birleşimi ile ultra düşük güç tüketimi sağlar. Düşük güç tüketimini sağlamak için çeşitli güç modları ve dinamik güç ölçekleme gibi en gelişmiş özelliklere sahip olacak şekilde tasarlanmıştır.

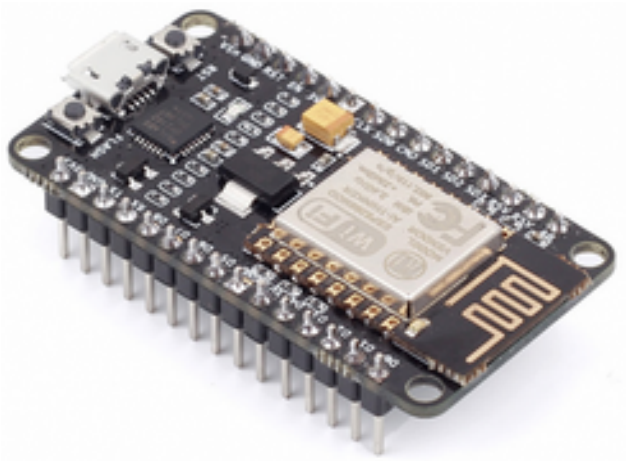
Robust Design

ESP32, -40 °C ile + 125 °C arasında değişen bir çalışma sıcaklığına sahip endüstriyel ortamlarda güvenilir bir şekilde kullanılabilir. Gelişmiş kalibrasyon devreleriyle güçlendirilen ESP32, harici devre kusurlarını dinamik olarak giderebilir ve harici koşullardaki değişikliklere hızlıca uyum sağlayabilir.

Bu özelliklerinden ve yeniliklerinden dolayı IoT uygulamalarında sıklıkla tercih edilmektedir.

NodeMCU ve Arduino IDE

ESP8266 NodeMCU, eLuo tabanlı ve Espressif tarafından geliştirilen bir Wifi modülüdür. Ürün yazılımı, Espressif NON-OS SDK'sini temel alır ve spiff'lere dayalı bir dosya sistemi kullanır. Kod deposu %98.1 C kodundan oluşur.



Bu başlık altında NodeMCU ile Arduino karşılaştırmasını ve ESP8266'yı programlamak için bilmemiz gerekenleri sizlere aktarmaya çalışacağız.

NodeMCU Arduino Core

NodeMCU Arduino Core, kısaca Arduino ortamına ESP8266 yongası desteği getirmeyi sağlayan bir projedir. Bu çalışma ile birlikte, Arduino ile gelen işlevleri ve kütüphaneleri kullanarak ESP8266 modülünün programlanması sağlanmaktadır. Ayrıca bu yol ile herhangi bir harici mikro denetleyiciye ihtiyaç duymadan kodları kolay bir şekilde ESP8266'ya yükleyebiliriz. ESP8266 Arduino Core, Wifi üzerinden iletişim kurmak için ihtiyaç duyulan tüm kütüphaneleri de sağlamaktadır.

NodeMCU Arduino Core ile aşağıda verilen işlemleri ESP8266 üzerinde gerçekleştirebiliriz:

- WiFi ile TCP/UDP, HTTP ve DNS kullanımı
- OTA güncellemeleri yapabilmek
- Flash bellekte dosya sistemi ile çalışabilmek
- SD kart kullanımı
- Servo motor kullanımı
- SPI ve I2C çevre birim desteği

Ayrıca şunları bilmekte de fayda var:

1. Kütüphane Desteği:
    a. WiFi için ESP8266WiFi kütüphanesi
    b. Ticker, belli bir süreyle işlev çağrısı yapmak için geliştirilen kütüphane
    c. Veri yazma veya okuma işlemi için EEPROM kütüphanesi
    d. 450KHz'ye kadar ana mod desteği sağlayan I2C kütüphanesi
    e. Tüm Arduino SPI API'sini destekleyen SPI kütüphanesi
    f. Hem STA hem de AP modlarında kullanılabilen basit bir DNS sunucusu uygulamayı sağlayan DNS Server kütüphanesi
    g. Servo motor kontrolünü sağlayan Servo kütüphanesi
2. File System (Dosya Sistemi):
    a. Flash düzen desteği

   b. Dosyaları dosya sistemine yükleyebilme desteği
   c. SPIFFS desteği
   d. Dosya sistemi hakkında bilgi edinebilme desteği
   e. Dizin içinde yer alan dosyalar üzerinde işlem yapabilme desteği
3. OTA (Over the Air), seri port'tan ziyade Wifi bağlantısı ile firmware yüklemeyi sağlamak için geliştirilmiştir. OTA şu işlemler için yapılabilir:
   a. Arduino IDE: Arduino IDE'den kablosuz olarak modüller yüklemek tasarlanmıştır.
   b. Web Browser: Aynı ağda bulunmak şartıyla tarayıcıla ilgili işlemler için tasarlanmıştır.
   c. HTTP Server: ESPhttpUpdate sınıfı güncellemeleri kontrol edebilir ve HTTP web sunucusundan bir ikili dosya indirebilir.

NodeMCU Arduino Core kütüphanesinin kodları GitHub üzerinden paylaşılmaktadır. Erişim için aşağıda verilen linki kullanabilirsiniz.

https://github.com/esp8266/Arduino

Arduino'ya Göre NodeMCU Platformunun Avantajları ve Dezavantajları

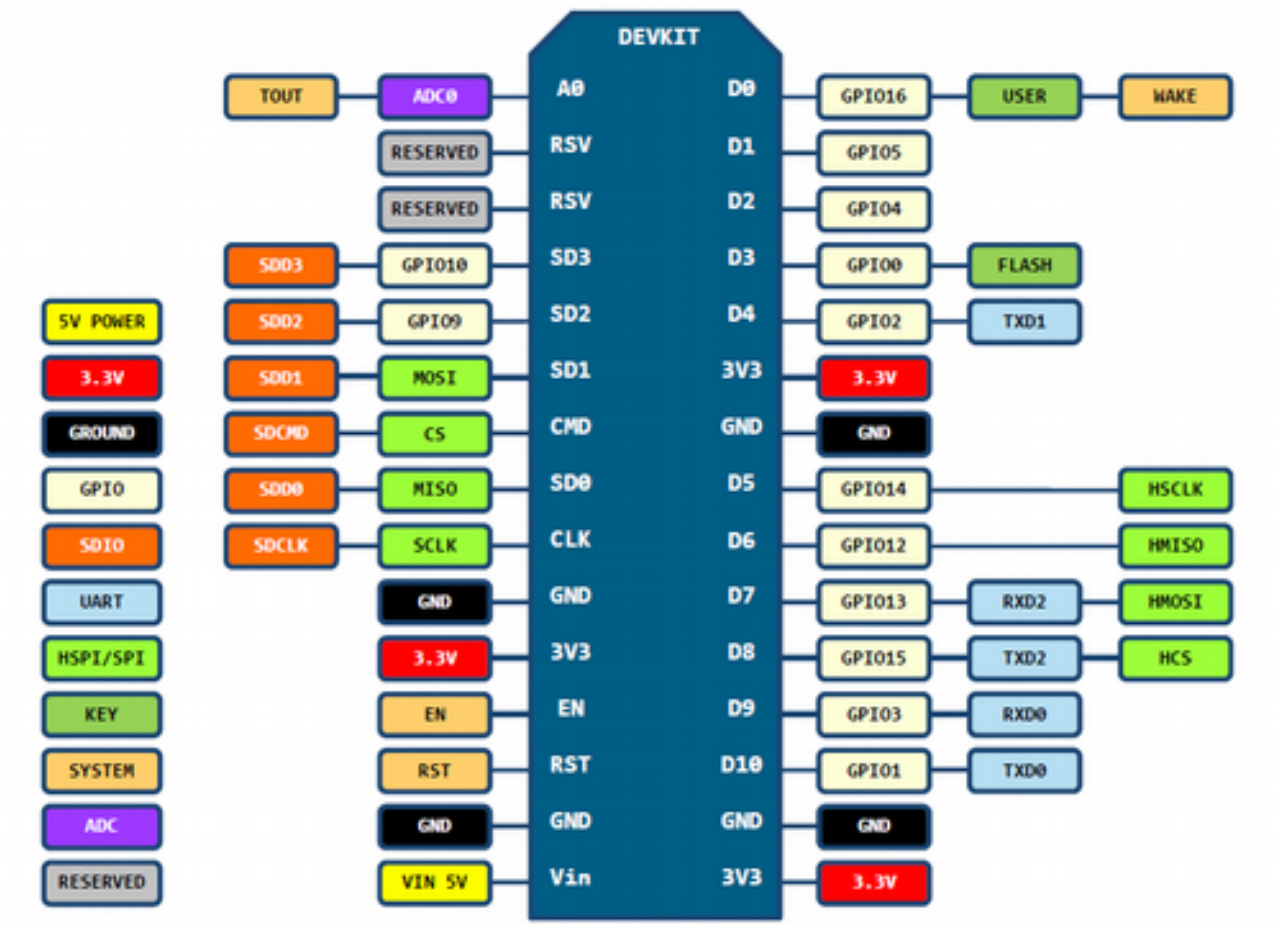
NodeMCU'nun avantajları şu şekilde sıralanabilir:

- Düşük maliyetli olması
- Wifi bağlantı desteğinin olması
- Kartın oldukça küçük olması
- Düşük enerji tüketimi sağlaması

Yukarıda verilen avantajlarla birlikte bazı dezavantajları da şu şekilde sıralanabilir:

- Yeni bir dil ve IDE öğrenmeyi gerektirmesi
- Kartın küçük olmasıyla birlikte pin sayısının azalması
- Kaynak eksikliği

NodeMCU ve Arduino programlama oldukça kolaydır. Temel fark, aşağıda açıklanan yönetim kurulunun sabitlenmesidir:



Bu şemayı incelediğinizde altıdan fazla GPIO olduğunu görebiliriz. Ayrıca NodeMCU projesi dosyaları Flash Chip'te saklamak için SPIFFS dosya sistemini kullanıyor.

NodeMCU ile Başlarken…

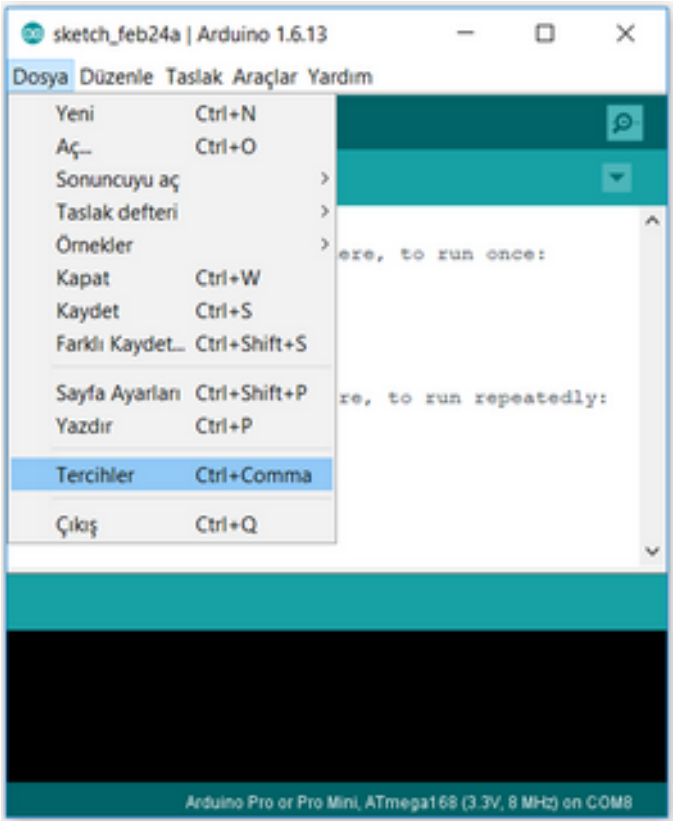
IDE kullanmadan önce, USB sürücüsünü işletim sistemine takın. Kartın sürümüne bağlı olarak CH340 veya CP2102 tercih edebilirsiniz. Bunları yaptıktan sonra şu adımları takip etmeniz gerekiyor.

Arduino IDE

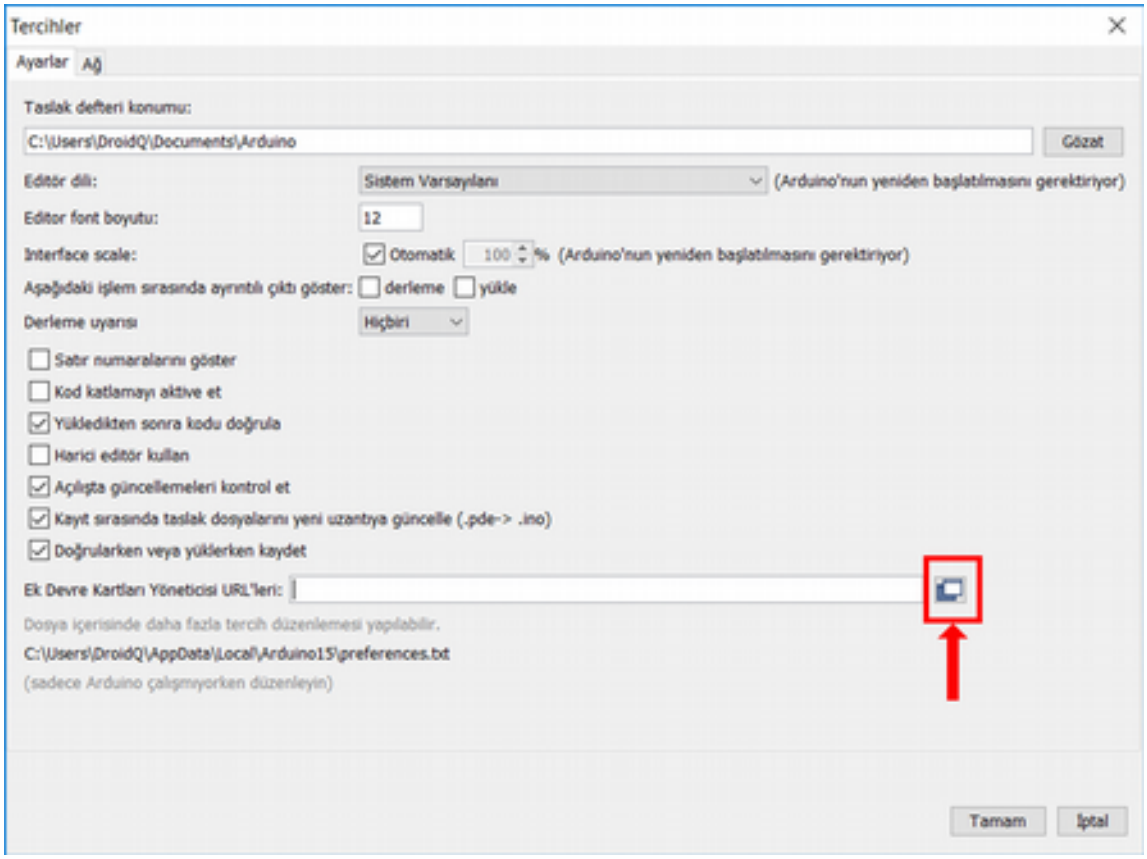
Yukarıdaki işlemleri yaptıktan sonra, Arduino IDE'yi çalıştırın. Eğer sizde bu yazılım kurulu değilse aşağıda verilen linkten temin edebilirsiniz.

https://www.arduino.cc/en/Main/Software

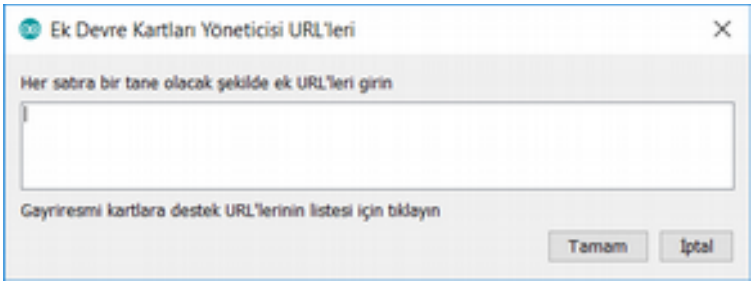
Arduino IDE menüsünde aşağıdaki adımları takip ediniz.



Yukarıda görüleceği üzere **Dosya > Tercihler** yolunu takip edelim.



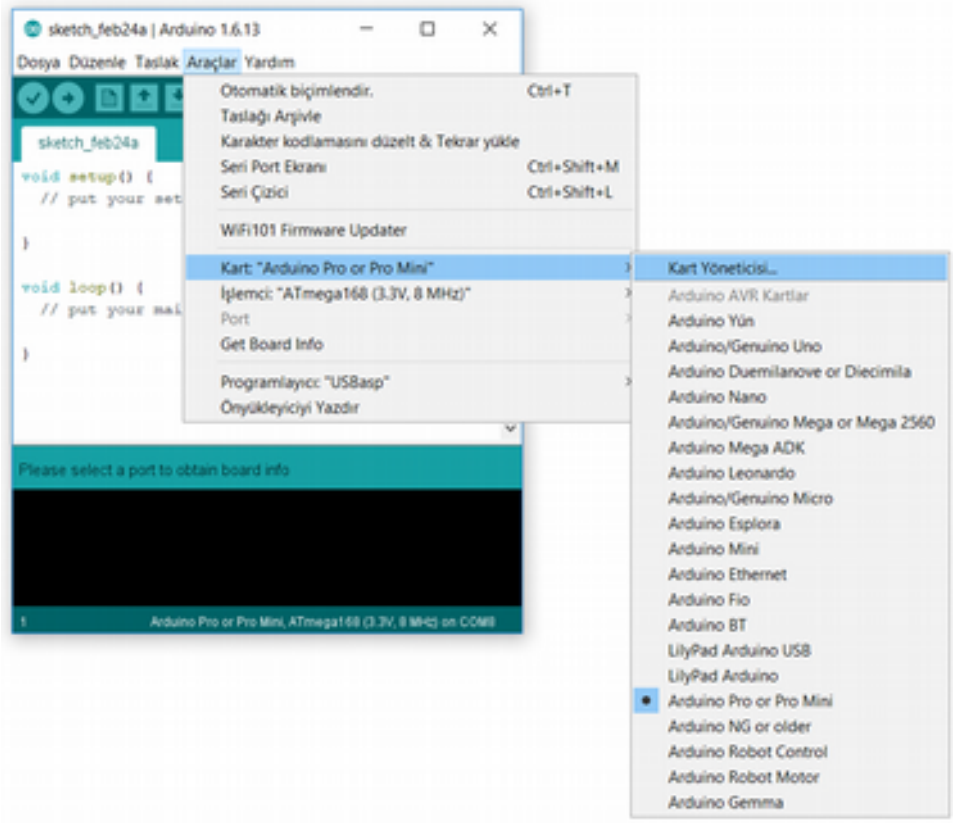
Yukarıda görüleceği üzere **Tercihler** penceresi açılır. Pencerede işaretlenen simgeye tıklayalım.



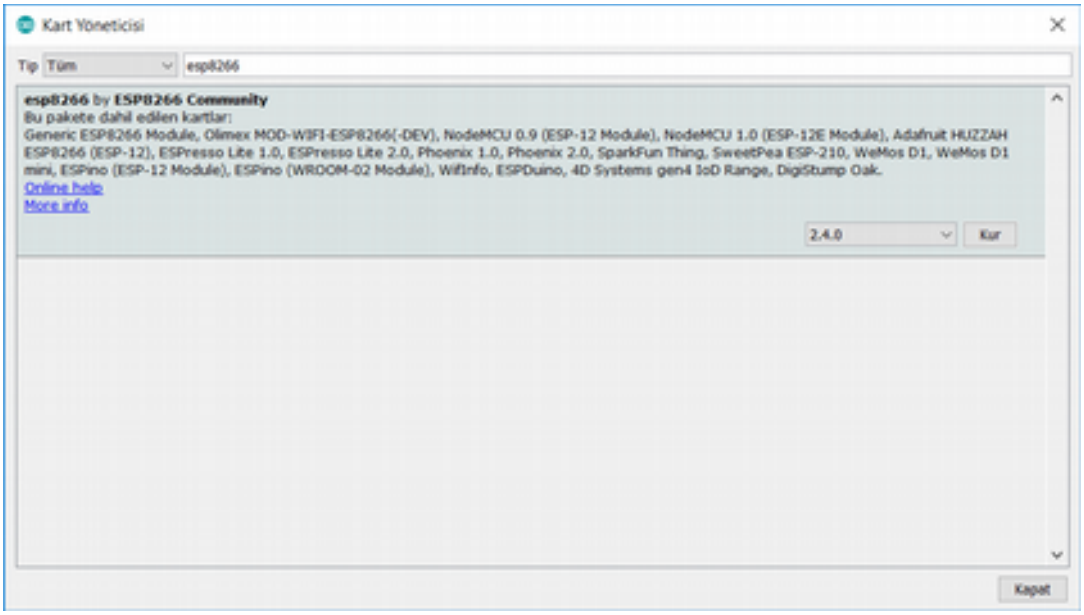
URL girin denilen yerin hemen altındaki metin kutusunu aşağıdaki **URL** bilgisini ekleyip **Tamam** butonuna tıklayınız.

http://arduino.esp8266.com/stable/package_esp8266com_index.json

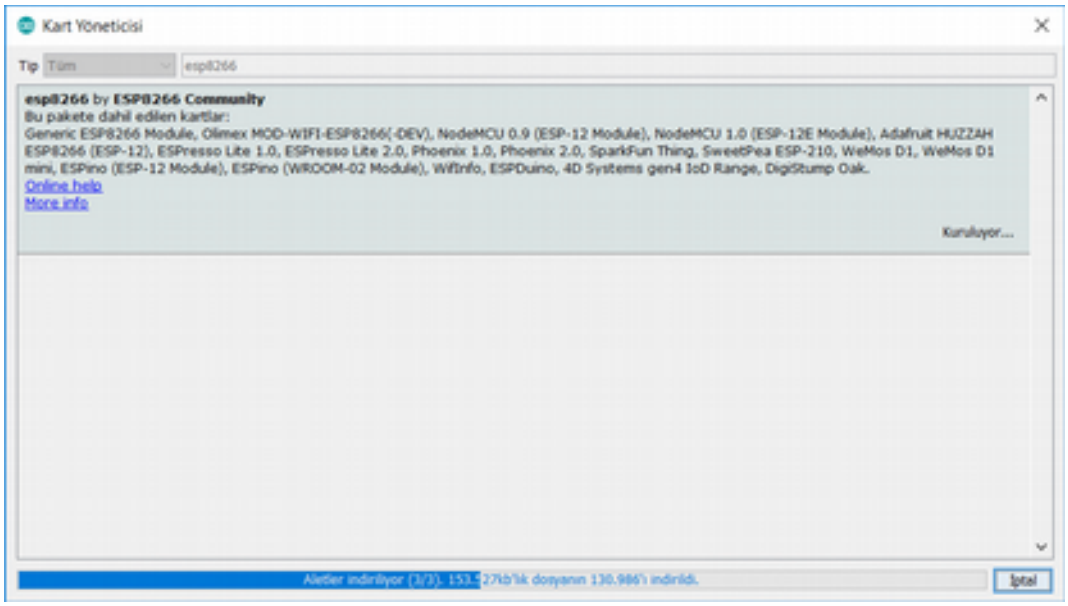
Bunu yaptıktan sonra esp8266 kütüphanesini indirmemiz gerekiyor. Bunun için aşağıda verilen yolu takip ediniz. Ayrıca dikkat edilirse kart listesinde ESP8266'nın olmadığı da görülür.



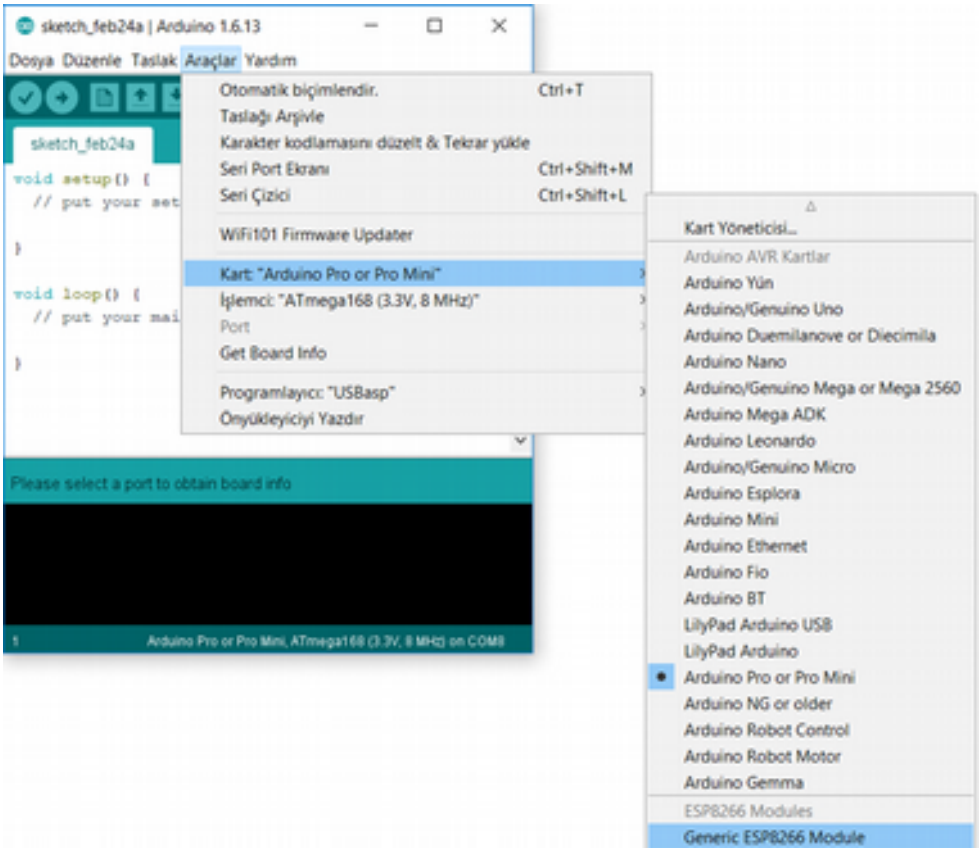
Bu işlemle birlikte **Board Manager** (Kart Yöneticisini) aşağıdaki gibi açılır. Metin alanına **esp8266** yazarak bulduğunuz dosyanın kurulmasını sağlamak için **Kur** seçeneğine tıklayınız.



Bu işlem ile birlikte aşağıdaki gibi ihtiyacımız olan dosyaların indirildiğini görmemiz gerekiyor.



İndirme işleminden sonra aşağıda verilen yolu takip ettiğimizde, ihtiyacımız olan ESP8266 kartının Arduino içinde tanımlandığını kesinlikle doğrulamamız gerekiyor. Bunu görmeden işlemleri gerçekleştiremeyiz.



Listenin en altında ESP8266 modülünü gördüyseniz, bu aşamadan sonra yazdığınız kodları ESP8266 kartına rahatlıkla yükleyebilirsiniz.

ESP8266 Spiffs Dosya Sistemi

Spiffs dosya sistemi, düşük RAM ile kısıtlı gömülü sistemler üzerinde kullanılmak için tasarlanmıştır. Özellikle düşük RAM kullanımı için kullanılan bu dosyalama sisteminde, dosya sayısından bağımsız olarak statik boyutlu RAM tamponları kullanılır.

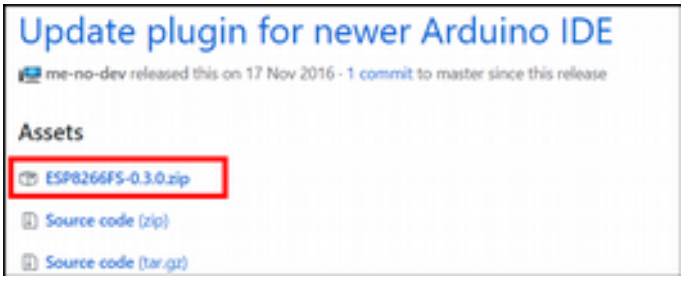
ESP8266 ile proje geliştirirken bizlere sunulan Spiffs dosyalama sistemi, seyrek olarak değişen verileri depolamak için kullanılabilir. Özellikle web sayfaları, yapılandırmalar, sensör kalibrasyon verileri gibi veriler için bu dosya sistemi idealdir.

Arduino ortamında Spiffs dosya sistemi ile çalışabilmek için öncelikle daha önce yaptığımız gibi ESP8266 modülünü Arduino ortamına kurmanız gerekiyor. Bu işlemi yapmadan Spifss ile çalışamayız. Bunu yaptıktan sonra spiffs kurulum işlemine geçebiliriz. Aşağıda verilen adımları takip ederek kurulum işlemini yapabilirsiniz.

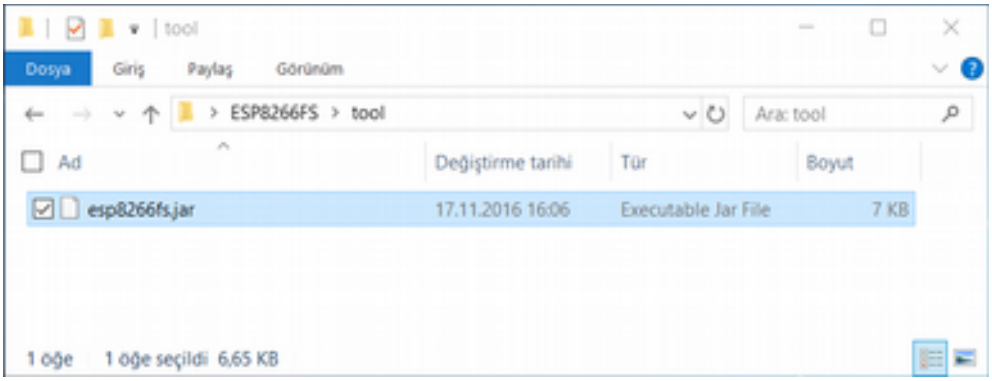
- Aşağıda verilen linkten ihtiyacımız olan dosyayı indiriniz.

https://github.com/esp8266/arduino-esp8266fs-plugin/releases/tag/0.3

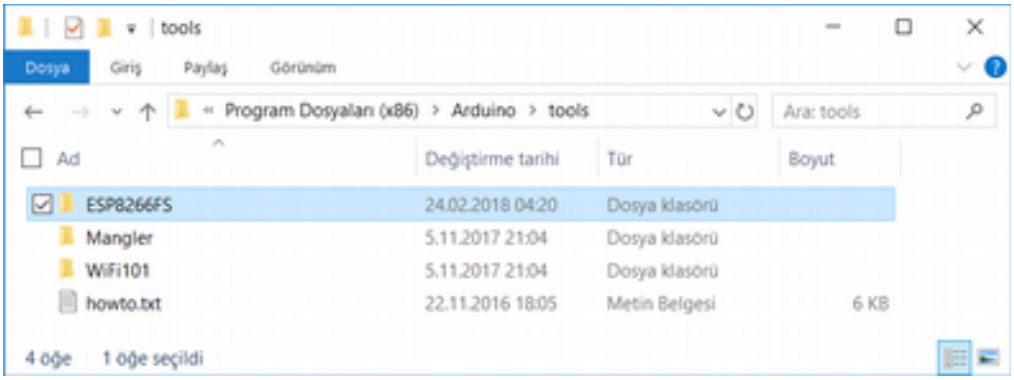
Bu linki kullandığınızda aşağıdaki gibi bir sayfa bizi karşılar. Buradan işaretli olan dosyayı bilgisayarınıza indiriniz.

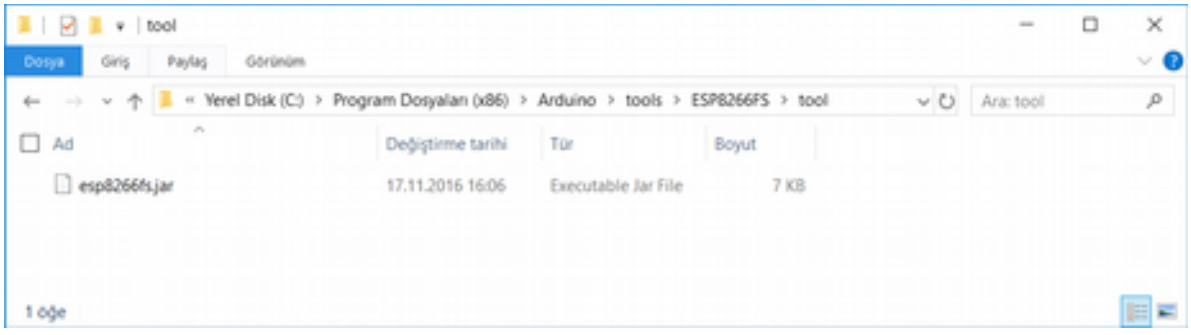


- İndirdiğiniz zip dosyasını, unzip'leyiniz. Açılan dosyada aşağıda görüleceği üzere .jar uzantılı bir dosya olması gerekiyor.



- İndirdiğimiz klasörün ismi ESP8266FS'dir. Bu klasörü kopyalayıp **Arduino** klasöründe bulunan **tools** klasörüne aşağıdaki gibi kopyalayınız.

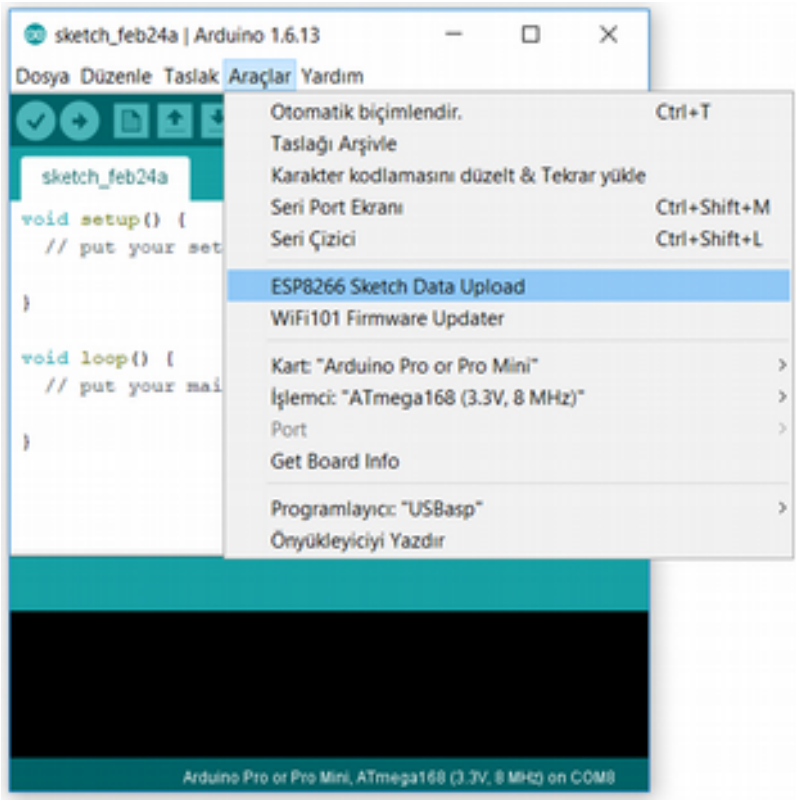




Klasör içinde aşağıdaki gibi **.jar** uzantılı dosyanın olmasına da kesinlikle dikkat ediniz.

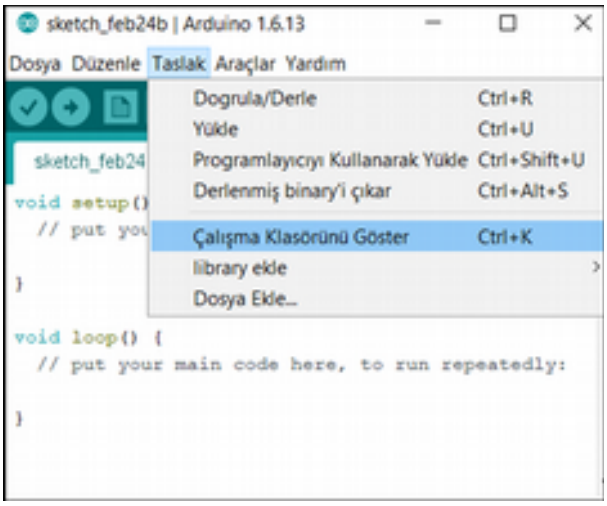
Bu işlemi yaptıktan sonra Arduino IDE uygulamasını yeniden başlatmanız gerekiyor.

- Arduino IDE uygulamasını yeniden başlattıktan sonra aşağıda verilen yolda gösterilen seçeneğin olduğunu görmemiz gerekiyor. Eğer bunu görürseniz yukarıda verilen işlemleri başarıyla gerçekleştirdiğimizi söyleyebiliriz.

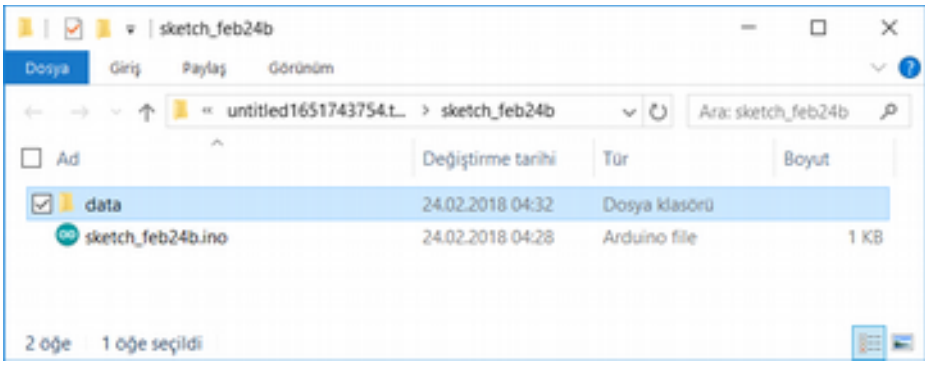


Kurulum işlemini yaptıktan sonra şimdi örnek bir uygulamada bunu nasıl kullanabiliriz, adım adım görelim.

- Arduino ortamında yeni Sketch oluşturun ve uygulamayı kayıt ediniz.
- Sketch klasörüne gitmek için aşağıda verilen yolu takip ediniz.

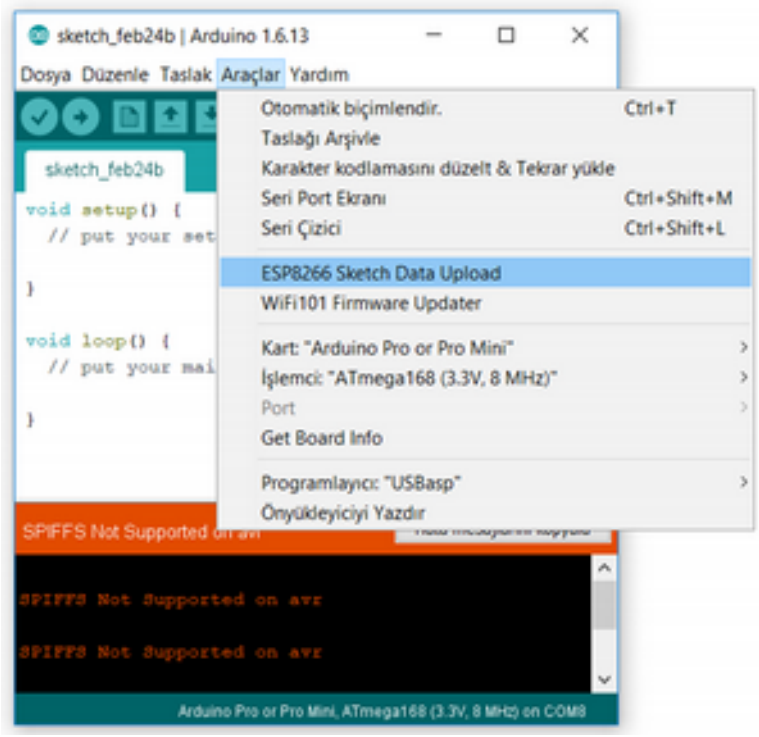


Bunu yaptıktan sonra açılan uygulama klasöründe ismi **data (veya "tools")** olan bir klasör oluşturalım.



data klasöründe istediğiniz dosyaları oluşturabilirsiniz.

- Arduino ile ESP8266 kartınızın bağlantısını sağlayınız.
- Son olarak aşağıda verilen adımı takip ediniz. Bu işlem, dosya sistemini yüklemeyi başlatır. İşlem tamamlandıktan sonra IDE durum çubuğu SPIFFS yükledi mesajı görüntüler.



Örnek Kodlar

NodeMCU ile çalışırken kullanabileceğimiz bazı kodlardan örnek vermek iyi olacaktır. Öncelikle örnek bir Arduino koduna bakalım.

```
pin = 1
gpio.mode(pin,gpio.OUTPUT)
gpio.write(pin,gpio.HIGH)
gpio.mode(pin,gpio.INPUT)
print(gpio.read(pin))
```

Yukarıda verilen kod örnek bir IO erişimi sağlamaktadır.

MQTT Broker'a Bağlanma

MQTT Broker'a bağlanmak için kullanacağımız örnek bir kod ise aşağıdaki gibidir.

```
-- init mqtt client with keepalive timer 120sec
m = mqtt.Client("clientid", 120, "user", "password")
-- setup Last Will and Testament (optional)
-- Broker will publish a message with qos = 0, retain = 0, data = "offline"
-- to topic "/lwt" if client don't send keepalive packet
m:lwt("/lwt", "offline", 0, 0)
m:on("connect", function(con) print ("connected") end)
m:on("offline", function(con) print ("offline") end)
-- on publish message receive event
m:on("message", function(conn, topic, data)
print(topic .. ":" )
if data ~= nil then
print(data)
end
end)
-- for secure: m:connect("192.168.11.118", 1880, 1)
m:connect("192.168.11.118", 1880, 0, function(conn) print("connected") end)
-- subscribe topic with qos = 0
m:subscribe("/topic",0, function(conn) print("subscribe success") end)
-- or subscribe multiple topic (topic/0, qos = 0; topic/1, qos = 1; topic2 , qos = 2)
-- m:subscribe({["topic/0"]=0,["topic/1"]=1,topic2=2}, function(conn) print("subscribe
success") end)
-- publish a message with data = hello, QoS = 0, retain = 0
m:publish("/topic","hello",0,0, function(conn) print("sent") end)
m:close();
-- you can call m:connect again
```

Spifss Dosya Sistemi

Spifss dosya sistemi ile çalışmak için kullanacağımız kod örneği aşağıdadır:

```
#include"FS.h"
void setup() {
  Serial.begin(115200);
  bool ok = SPIFFS.begin();
  if (ok) {
    Serial.println("ok");
    boolexist = SPIFFS.exists("/index.html");
    if (exist) {
      Serial.println("The file exists!");
      File f = SPIFFS.open("/index.html", "r");
      if (!f) {
        Serial.println("Some thing went wrong trying to open the   file...");
      }
      else {
        int s = f.size();
        Serial.printf("Size=%d\r\n", s);
        // USE THIS DATA VARIABLE
        String data = f.readString();
        Serial.println(data);
        f.close();
      }
    }
    else {
      Serial.println("No such file found.");
    }
  }
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Arduino Kullanımı

Arduino, kodları fiziksel etkilere dönüştüren dijital ve analog giriş / çıkış (I / O) pinlerine sahip bir geliştirme kartıdır. IoT-Ignite platformunda Arduino kartına sensörleri bağlayarak bir Node olarak kullanabiliriz. Arduino katkıda bulunmak isteyenler için açık kaynak bir donanım kartıdır. Sahip olduğu geniş kütüphane desteği ve dünya çapında geliştirilen örnek projeler sayesinde dünyada en çok kullanılan mikrodenetleyici kartlardan biri olmayı başarmıştır.

Arduino geliştirme kartlarında Atmel AVR mikrodenetleyicisi yer almaktadır. Genel olarak kullanılan Atmel AVR mikrodenetleyiciler aşağıdaki gibidir:

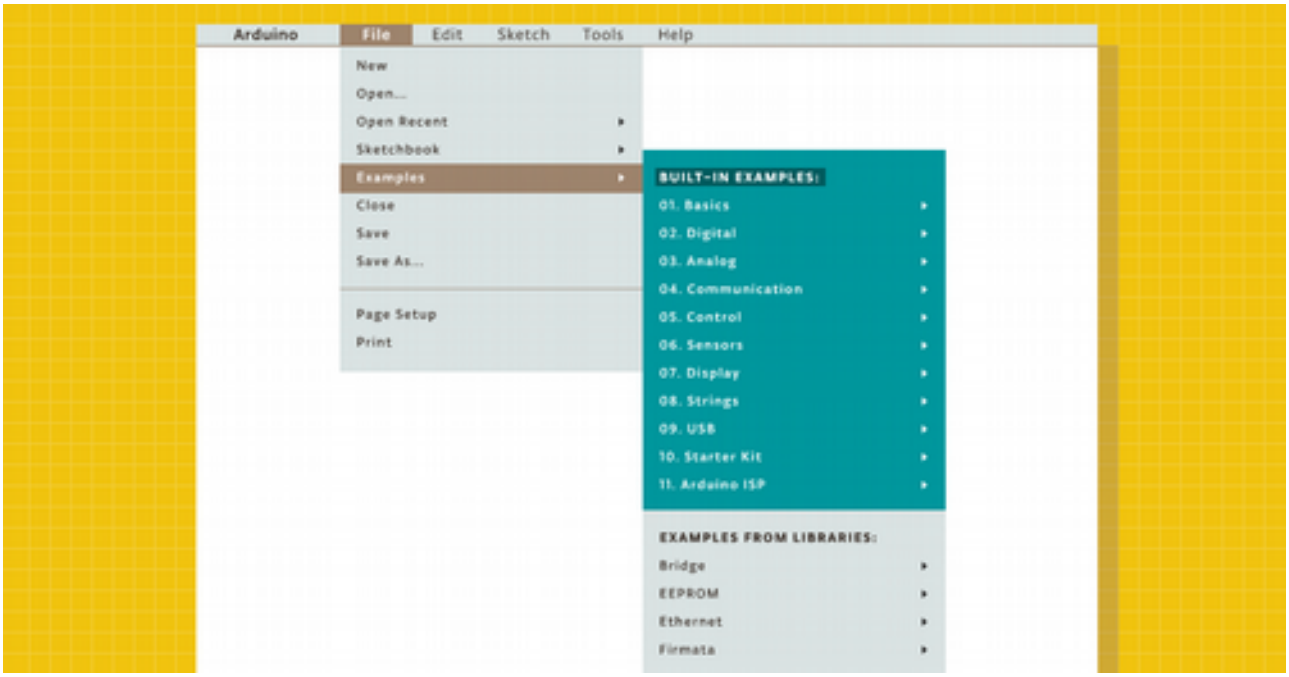
- ATmega328
- ATmega2560
- Atmega32u4

Bunlarla birlikte kart ile çevre birimler arasında bağlantı işlemini yapabilmek için diğer yan birimler de bulunmaktadır. En az bir 5 V doğrusal regülatör ve 16 MHz kristal osilatör ile çalışan bu kartların mikro denetleyicisi bir önyükleme yükleyicisi ile önceden programlanmıştır. Böylece harici bir yonga programcısına gerek duymadan kartınızı rahatlıkla programlayabilirsiniz.

Arduino ile Neler Yapılabilir

Arduino kütüphaneleri herkesin kolayca program yapmasına izin verecek şekilde hazırlanmıştır. Ayrıca çevreden alınan analog ve dijital sinyaller kolaylıkla işlenebilir. Bundan dolayı sensör sinyallerini kullanarak interaktif robotlar veya sistemler tasarlayabilirsiniz. Yine sisteminizin hareket halinde, sese veya ışığa yanıt vermesini sağlayan işlevleri de projenize ekleyebilirsiniz.

Arduino'nun resmi web sayfasında bizlere sunulan bazı hazır projeler bulunmaktadır. Sadece bu projeleri kullanarak evinizde, iş yerinizde hatta aracınızda çalışan IoT uygulamaları geliştirebilirsiniz. Bu projeler Arduino IDE ile gelmektedir. Projelerin genel olarak listesi aşağıdaki gibidir.



Arduino, birçok farklı gereksinimi karşılamak adına çok çeşitli kart modellerine sahiptir. Her kart özellikle belirli bir işlemi gerçekleştirmek için üretilmiştir. Arduino'nun resmi sitesinde yer alan ve şu ana kadar üretilen tüm modeller aşağıdaki gibidir:

Listeyi incelediğinizde her amaç için farklı bir Arduino modelinin olduğunu görebilirsiniz. Arduino kartlarını kullanarak aşağıda verilen projeleri rahatlıkla geliştirebilirsiniz. Bu projeler Arduino kartlarının ne kadar güçlü olduğunu göstermek adına çarpıcı örneklerdir:

- 3D Printers
- Fighting Robot
- Laser Harp
- Led Cubes
- Weather App
- Fingerprint Scanner
- Mute Project for TV
- Robot Arm
- Functional Computer Control Panel

Burada verilen uygulamalar, en yaygın örneklerden birkaçıdır.

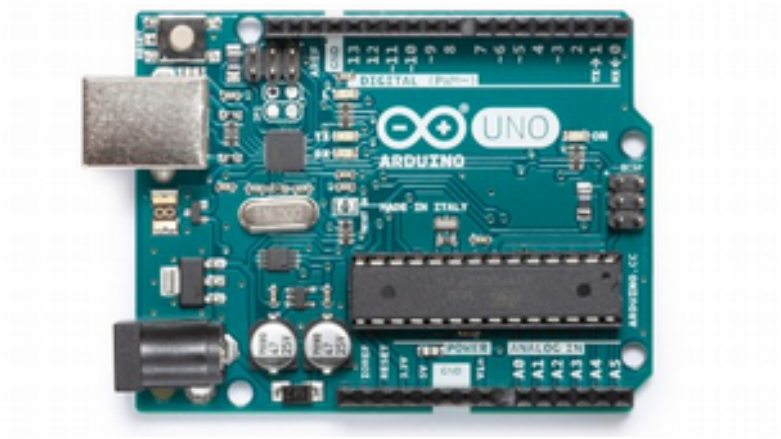
Popüler Arduino’lar ve Teknik Özellikleri

Arduino kart çeşitlerini yukarıda bir şemada sizlere sunmuştuk. Bu listeden projenizin ihtiyaçlarına göre doğru Arduino'yu seçmeniz gerekiyor. Giriş/Çıkış pinleri, Analog Giriş, EEPROM bellek ve benzeri özellikler Arduino kartlarını seçmenize yardımcı olmaktadır. Ayrıca amacınız ağ üzerinden kontrolü sağlamak ise LAN , Wifi ve Bluetooth ihtiyaçlarını gideren Arduino kartlarını ve Shield'lerini seçmeniz gerekiyor.

Arduino ile proje geliştirirken genellikle UNO ve Mega modelleri sıklıkla kullanılır. Hatta yeni başlayan geliştiriciler için Arduino UNO tavsiye edilmektedir. UNO ile hem geniş çaplı hem de ufak çaplı projeler geliştirebilirsiniz. Ancak amacınız daha dar alanda daha küçük boyutlu projeler geliştirmek ise, bu durumda Arduino Nano oldukça kullanışlıdır. Burada yaygın olarak kullanılan UNO ve Nano kartları hakkında bilgi vermekte fayda var.

Arduino UNO

Arduino’nun resmi web sayfasında, Arduino UNO, elektronik ve kodlamaya ilk defa başlayanlar için en iyi geliştirme kartı olarak ifade edilmektedir. Eğer ilk defa proje geliştirmeye başlayan biriyseniz ilk tercihiniz kesinlikle UNO olmalıdır. UNO, Arduino ailesinin en çok kullanılan geliştirme kartı ünvanına sahip olmuş ve bu isim altında belgelendirilmiştir.



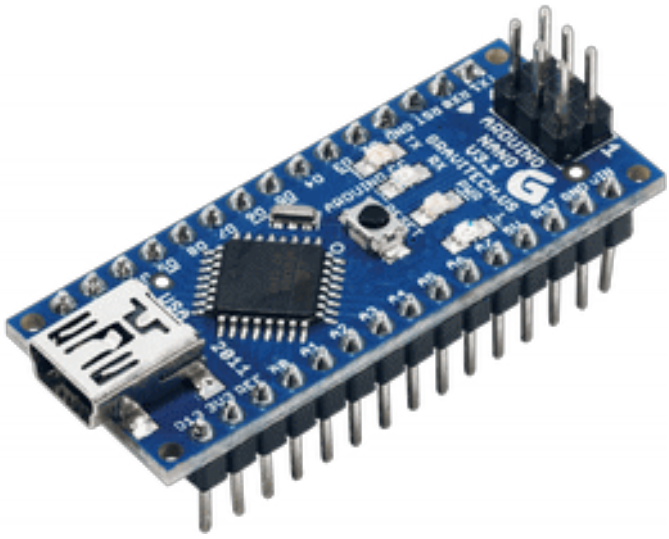
Arduino UNO, ATmega328P tabanlı bir mikrodenetleyici kartıdır. 14 tane dijital giriş/çıkış pini bulunmaktadır. Bu pinlerden 6 tanesi PWM çıkışı olarak kullanılabilmektedir. Ayrıca 6 tane analog girişe, 16MHz'lik bir kristale, USB bağlantısına, güç bağlantısına, ICSP header ve bir adet reset düğmesine sahiptir. Yani kısaca UNO, bir mikrodenetleyici için ihtiyaç duyulan tüm gereksinimleri sağlamaktadır. Kullanımı oldukça basit olan UNO'yu bilgisayarınıza USB ile bağlamanız ve bir adet AC-to-DC adaptörü ile güç sağlamanız yeterlidir.

UNO'nun teknik özellikleri genel olarak aşağıdaki gibidir:

- Atmega328P mikrodenetleyici
- Giriş voltajı 7-12 V arasında tavsiye edilmekle birlikte maksimum limit 6-20 V arasında olabilir.
- 14 tane dijital giriş/çıkış pinine sahiptir. Bunlardan 6 tanesi PWM çıkışı olarak kullanılabilir.
- Analog giriş pini 6 tanedir.
- Giriş/Çıkış pini başına DC akımı 20 mA'dir.
- 3.3 V pini için DC akımı 50 mA'dir.
- Flash Memory 32 KB olup bunun 0.5 KB'ı bootloader tarafından kullanılmaktadır.
- SRAM 2 KB'tır.
- EEPROM 1 KB'tır.
- Clock speed değeri 16 MHZ'dir.
- Dahili LED sayısı 13 tanedir.
- Kartın uzunluğu 68.6 mm'dir.
- Kartın genişliği 53.4 mm'dir.
- Ağırlığı 25 gr'dır.

Arduino Nano

Arduino Nano, ATmega328P tabanlı, küçük ve breadboard ile pratik bir şekilde kullanılan bir Arduino kartıdır. UNO'ya oldukça benzeyen bu geliştirme kartı daha kompakt bir yapıda geliştirilmiştir.



DC güç bağlantısı olmayan bu geliştirme kartı, küçük olması için Mini-B USB portuyla üretilmiştir. Küçük olmasından dolayı genellikle küçük çaplı projelerde kullanılmaktadır.

Arduino Nano'nun teknik özellikleri genel olarak aşağıdaki gibidir.

- ATmega328 mikrodenetleyici.
- AVR modeli tabanlı.
- 5 V çalışma gerilimine sahip olması.
- Flash Memroy 32 KB olup 2 KB'ı bootloader tarafından kullanılmaktadır.
- SRAM 2 KB'tır.
- Clock Speed 16 MHZ'dir.
- Analog giriş pin sayısı 8 tanedir.
- EEPROM 1 KB'tır.
- Giriş/Çıkış pini başına DC akımı 40 mA'dir.
- Giriş voltajı 7-12 V arasında olmalıdır.
- Dijital giriş/çıkış pin sayısı 22 tane olup bunların 6 tanesi PWM olarak kullanılabilir.
- Güç tüketimi 19 mA'dir.
- PCB boyutu 18 x 45 mm'dir.
- Ağırlığı 7 gr olup oldukça hafiftir.

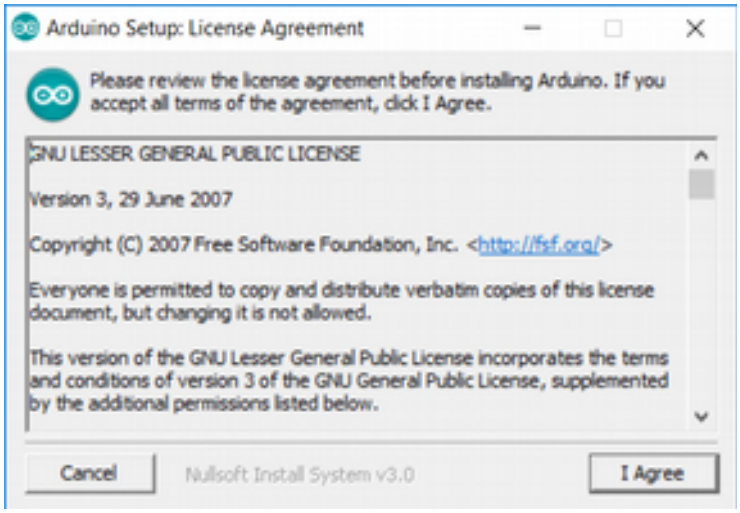
Arduino IDE'nin Windows'ta Kurulumu

Arduino kartlarını programlamak için Arduino IDE yazılımını kurmanız gerekiyor. Yazılımın son sürümünü aşağıda verilen linkten elde edebilirsiniz.

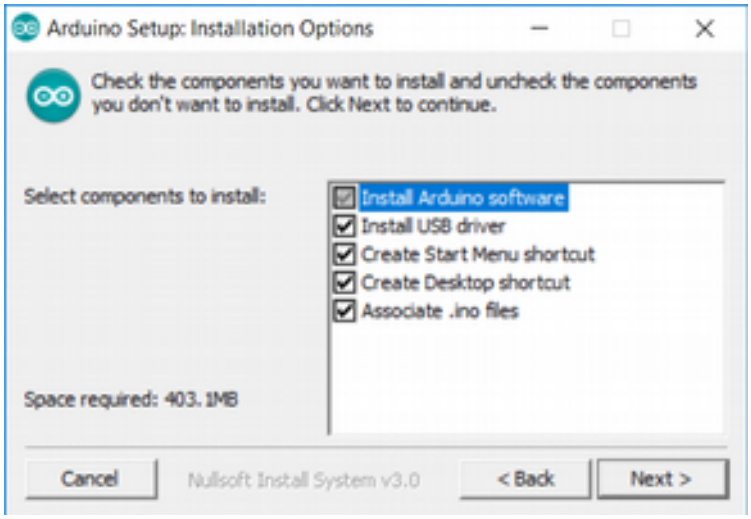
https://www.arduino.cc/en/Main/Software

Kurulum dosyasını yukarıdaki sayfadan indirebilirsiniz. Amacımız Windows ortamında kurulum işlemi yapmak olduğundan dolayı, **Windows** için geliştirilen setup dosyasını seçmemiz gerekiyor.

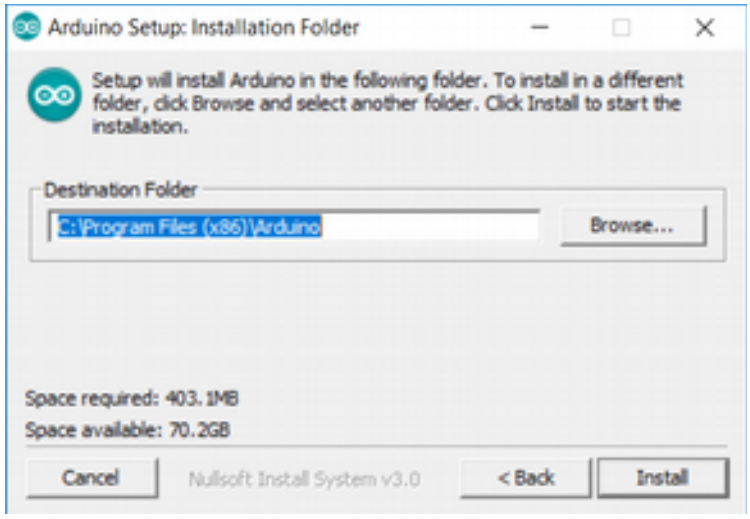
Setup dosyasını indirdikten sonra çalıştırdığımızda aşağıdaki gibi ekran açılır.



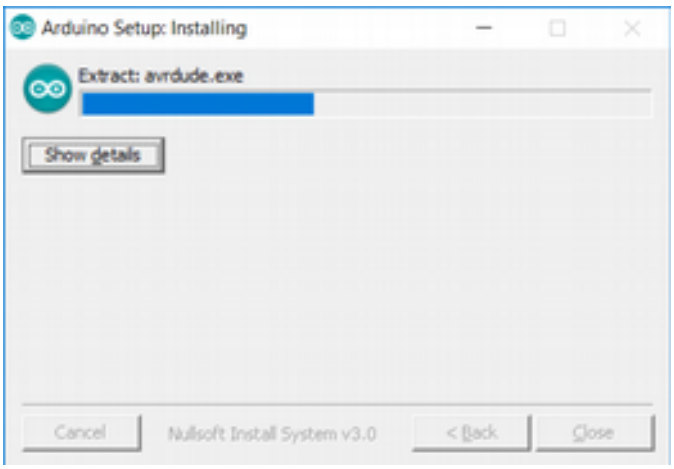
I Agree butonuna tıklayıp devam edelim.



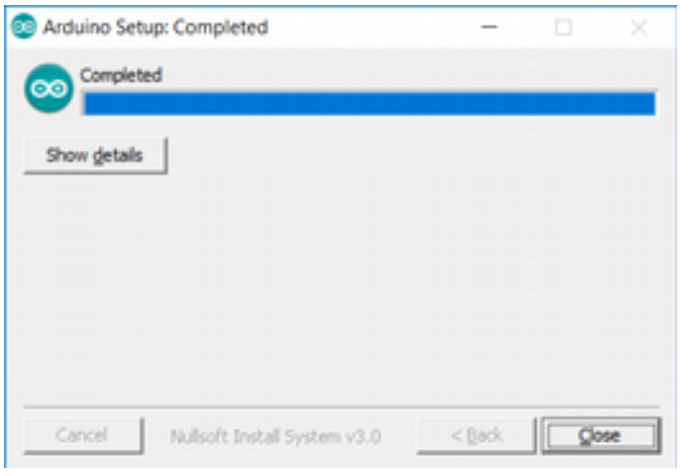
Yukarıda açılan pencerede herhangi bir değişiklik yapmadan **Next** diyelim.



Yukarıda açılan pencerede kurulumun nereye yapılacağı belirtilir. İsterseniz farklı bir sürücü de seçebilirsiniz. Biz herhangi bir değişiklik yapmadan **Install** deyip devam edeceğiz.

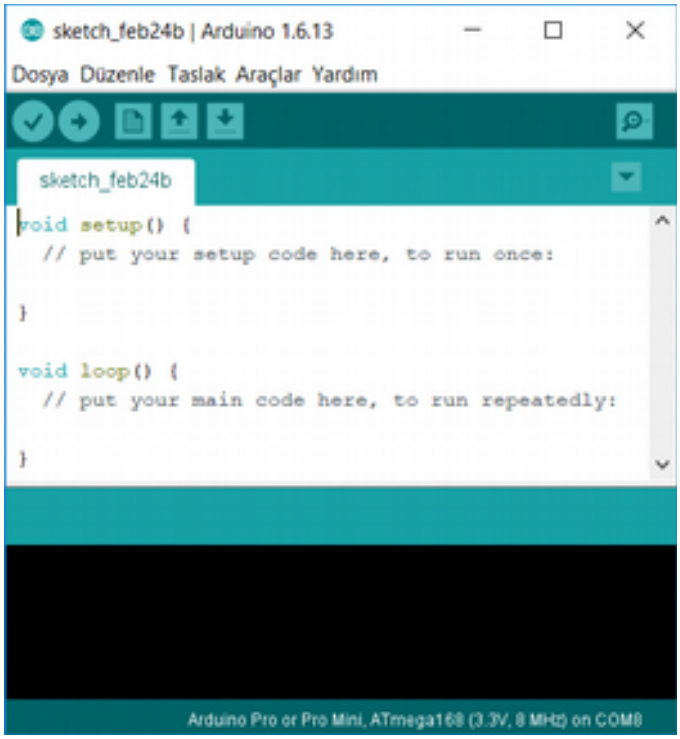


Install dedikten sonra kurulum kısa bir sürede tamamlanır.



Close butonunu tıklayarak kurulum işlemini tamamlayalım. Masaüstünde Arduino kısayoluna tıkladığımızda, öncelikle aşağıda verilen küçük bir ekran bizi karşılar.

Bu ekranda Arduino ile ilgili bazı tanımlayıcı bilgiler verildikten hemen sonra aşağıda görüleceği üzere IDE arayüzü açılır.



Kodlarımızı yazacağımı alanı yukarıda görebilirsiniz. Editör içinde yukarıda görüleceği üzere **setup()** ve **loop()** olmak üzere iki tane metot yer almaktadır. Bu metotların görevi kısaca şu şekildedir:

- **setup():** Arduino kartı ilk açıldığı zaman çalıştıracağımız kodları bu metot içinde yazarız. Burada çalışan kodlar sadece açılış sırasında ve yalnız bir defa icra edilir. Özellikle uygulama öncesi ihtiyaç duyulan yapılandırmalar bu blok içinde belirtilir. Örneğin değişken tanımlama ve pin atama gibi. Ayrıca isminden analaşılacağı üzere proje için ilk kurulum işlemleri burada yapılır.
- **loop():** Projemiz için ihtiyacımız olan temel kodları yazacağımı metodumuzdur. Bu metot, setup() metodundan hemen sonra başlar ve Arduino açık olduğu sürece çalışmaya devam eder. Uygulama için yazacağımı ve sensörlerden sürekli olarak veri almayı sağlayan temel kodların bu metot içinde yazılması bir zorunluluktur. Ayrıca isminden anlaşılacağı üzere bu bir döngüdür ve sürekli çalışır.

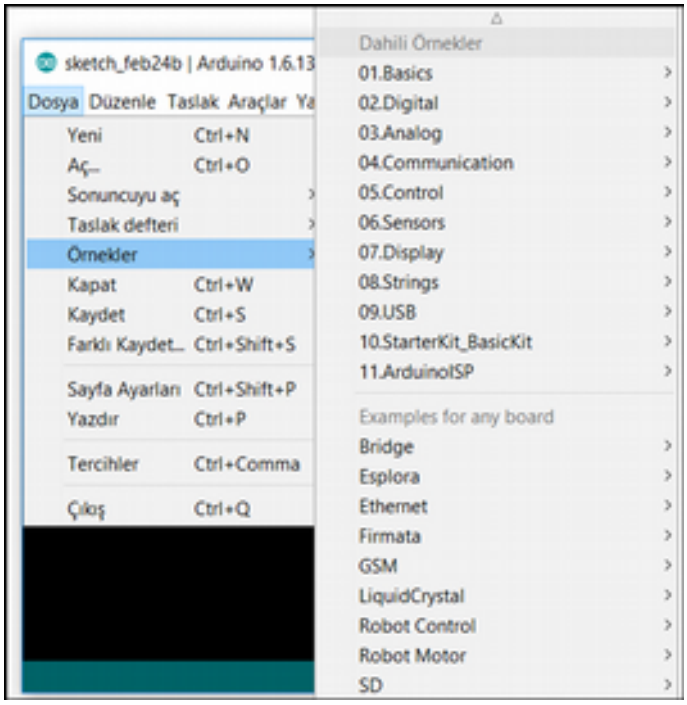
Arduino IDE Arayüzü

Arduino IDE geliştirme ortamında beş adet menu bulunmaktadır. Bu menülerin görevleri hakkında kısaca bilgi vermekte fayda var.

- **File**: Dosya işlemlerini yapabildiğimiz standart bir menüdür. Bu menü altında yeni dosya oluşturma, mevcut olan dosyayı açma, kaydetme, projeyi kapatma, yazıcıdan çıktı alma ve en önemlisi örnek uygulamaları erişmeyi sağlayan seçenekler yer almaktadır.

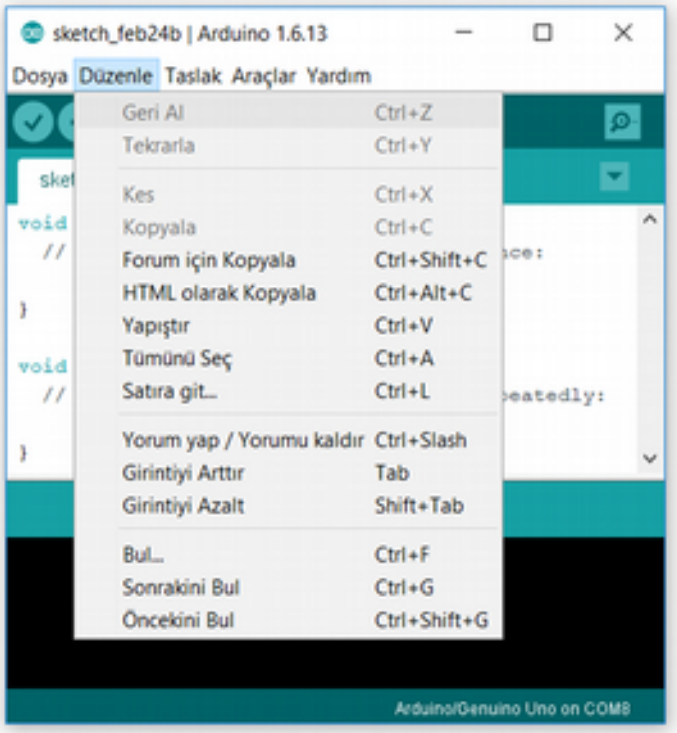
Menü seçenekleri genel olarak aşağıdaki gibi olup. Örnek kodlara erişim yolu özellikle gösterilmiştir. Örnek uygulamaları deneyerek kendi kendinize Arduino projeleri geliştirebilirsiniz.

Temelden başlayan örnek projelerde ilerledikçe Arduino işlevlerini daha net göreceksiniz.



- **Edit**: Metinler üzerinde düzenleme yapmayı sağlayan bu menü altında yapılan son işlemi geri almak ve bir işlemi tekrarlamak, kesme ve kopyalama yapabilmek, yapıştırma, kod başına girinti ekleme ve girinti kaldırma, kelime arama, yorum yapma ve kod blogu içinde belirli bir satıra gitmek gibi işlemler yapılabilir.

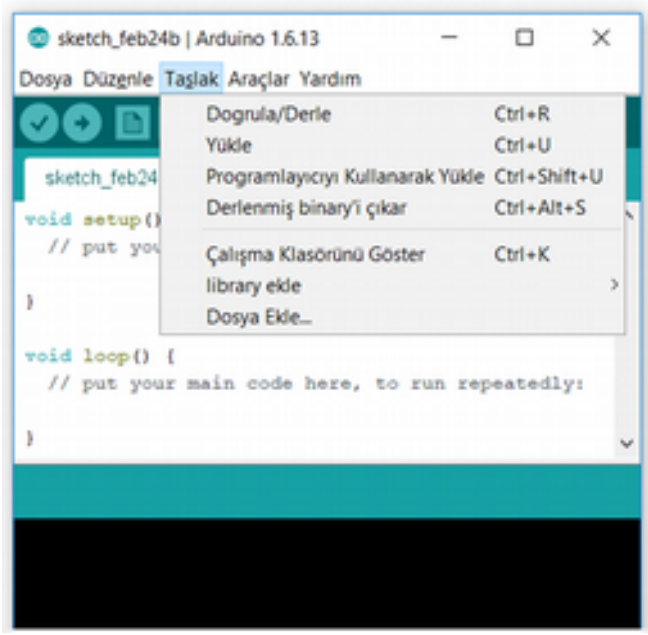
Bu seçenek altında yer alan tüm komutlar aşağıdaki gibidir.



- **Sketch** (Taslak): Yazdığımız kodu derleyen, kodu karta yüklemeyi sağlayan, uygulama klasörünü açan, uygulama için ihtiyaç duyulan kütüphaneleri program başına eklemeyi

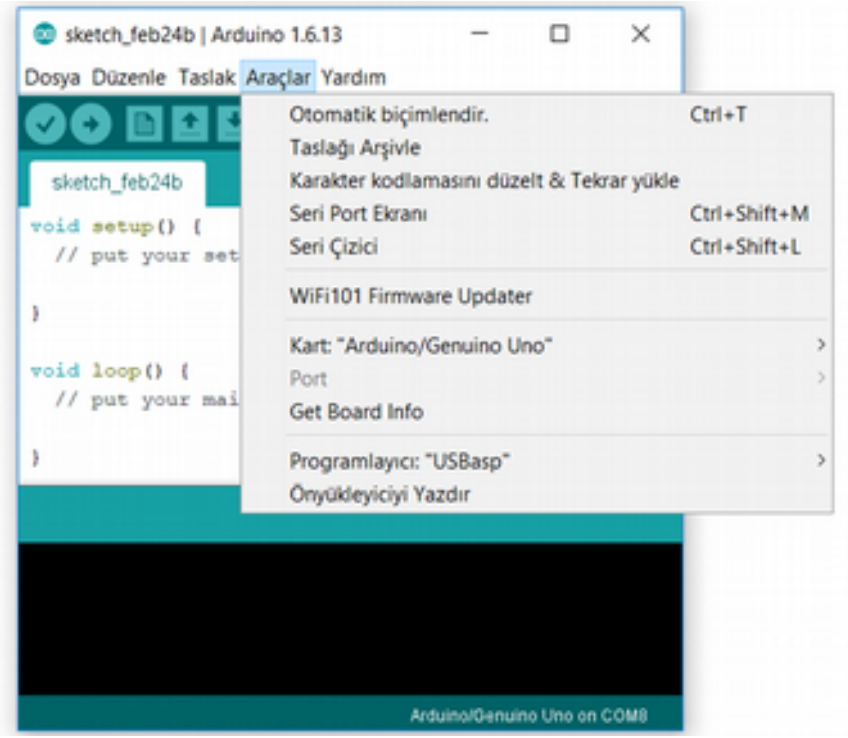
sağlayan, daha önce geliştirdiğimiz bir Arduino dosyasını mevcut uygulamaya eklemeyi sağlayan iş ve işlemleri gerçekleştiren menüdür.

Bu seçenek altında yer alan tüm komutlar aşağıdaki gibidir.



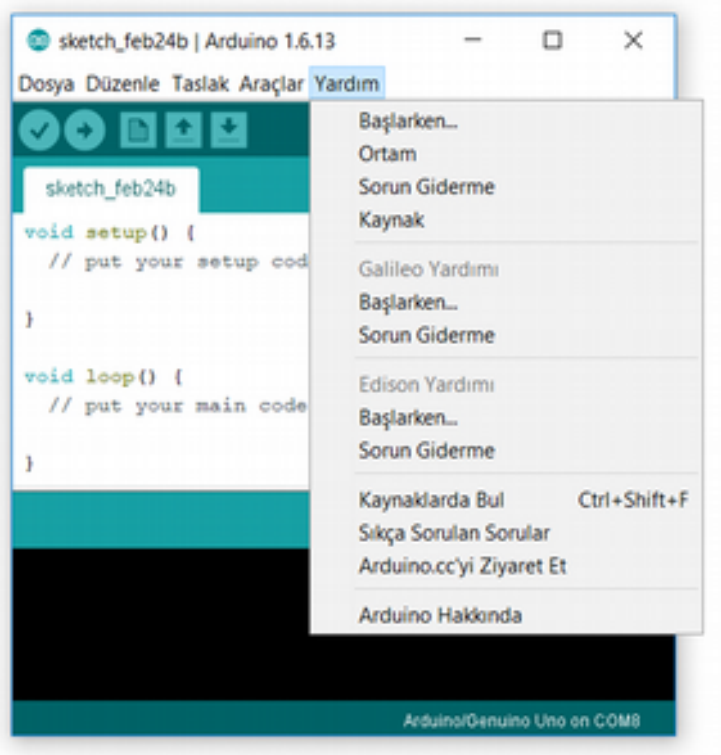
- **Tools** (Araçlar): Kodun daha iyi okunabilmesi için biçimlendirilmesini, yazdığımız kodu zip uzantılı olarak sıkıştırmayı sağlayan, karakter kodlamayla ilgili hataları düzelten, seri bağlantı ile gelen verileri görüntülemeyi sağlayan, proje için kullanılacak Arduino kartını seçmeyi sağlayan, bağlı olan kartın port bilgisini göstermek gibi iş ve işlemleri yapabilen bir menüdür.

  Bu seçenek altında yer alan tüm komutlar aşağıdaki gibidir.



- **Help**: Bu menü altından Arduino sitesi tarafından online olarak sağlanan bilgilerin çevrimdışı kaynakları bulunmaktadır. Buradan faydalanarak uygulamalarınızı daha kolay geliştirebilirsiniz.

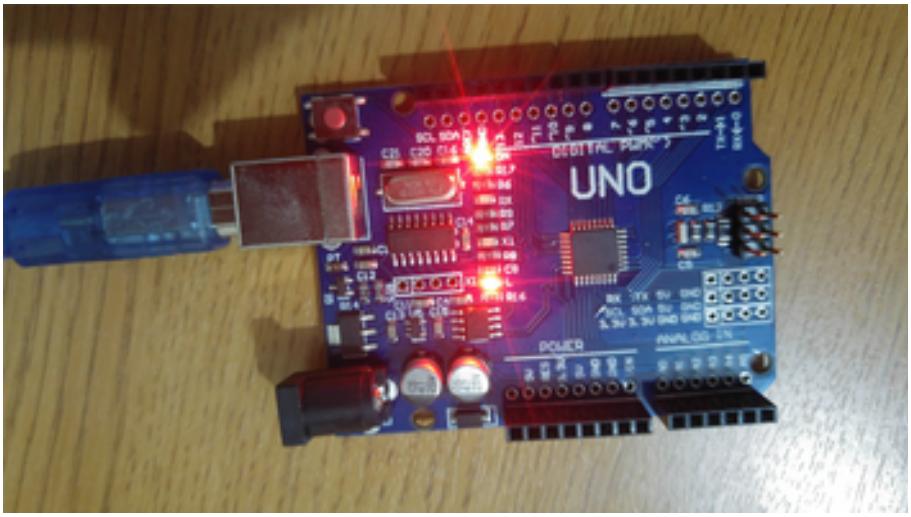
  Bu seçenek altında yer alan tüm komutlar aşağıdaki gibidir.



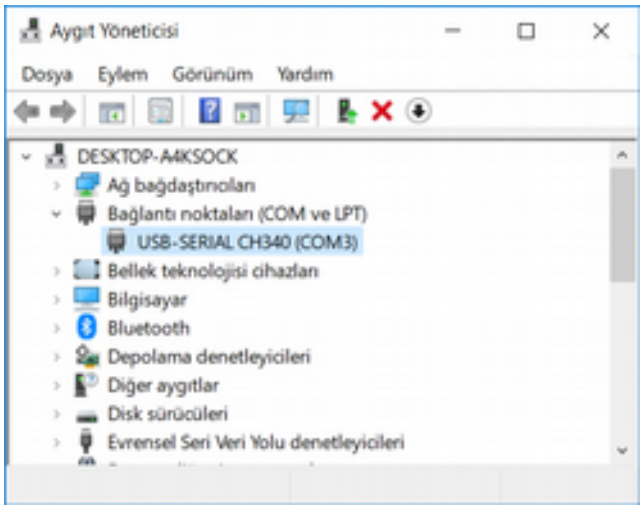
Yapılandırma Ayarları

Arduino arayüzünü tanıdıktan sonra sıra uygulama geliştirmeden önce yapılması gereken temel ayarlamaları yapmaktır. Bu ayarları bağladığınız her farklı kart için baştan kontrol etmeniz faydanıza olacaktır. Örneğin burada bilgisayara bağladığımız bir Arduino UNO kartını proje geliştirecek hale getirmek için yapılması gereken temel işlemler aşağıdaki gibidir.

- Öncelikle Arduino Uno kartınızı USB aracılığıyla bilgisayarınıza bağlayınız.

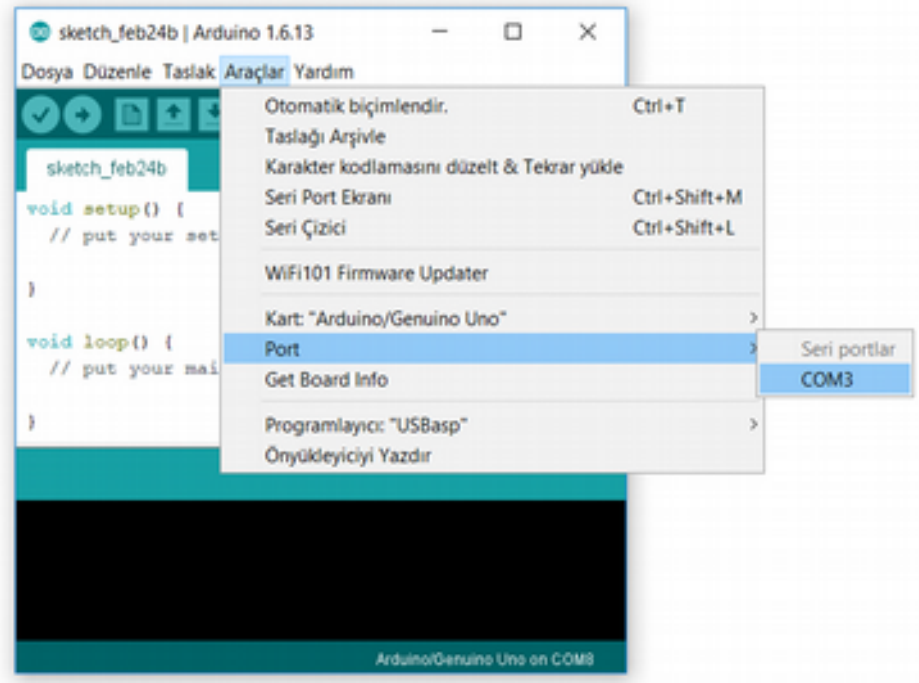


- Eğer aynı bilgisayarda birden fazla kartı programlamak için bağlantı kurduysanız, bu durumda çalışmak istediğiniz kartın bağlı olduğu **COM port** bilgisini öğrenmeniz gerekiyor. Bunu öğrenmek için aygıt yöneticisini başlatmanız gerekiyor.

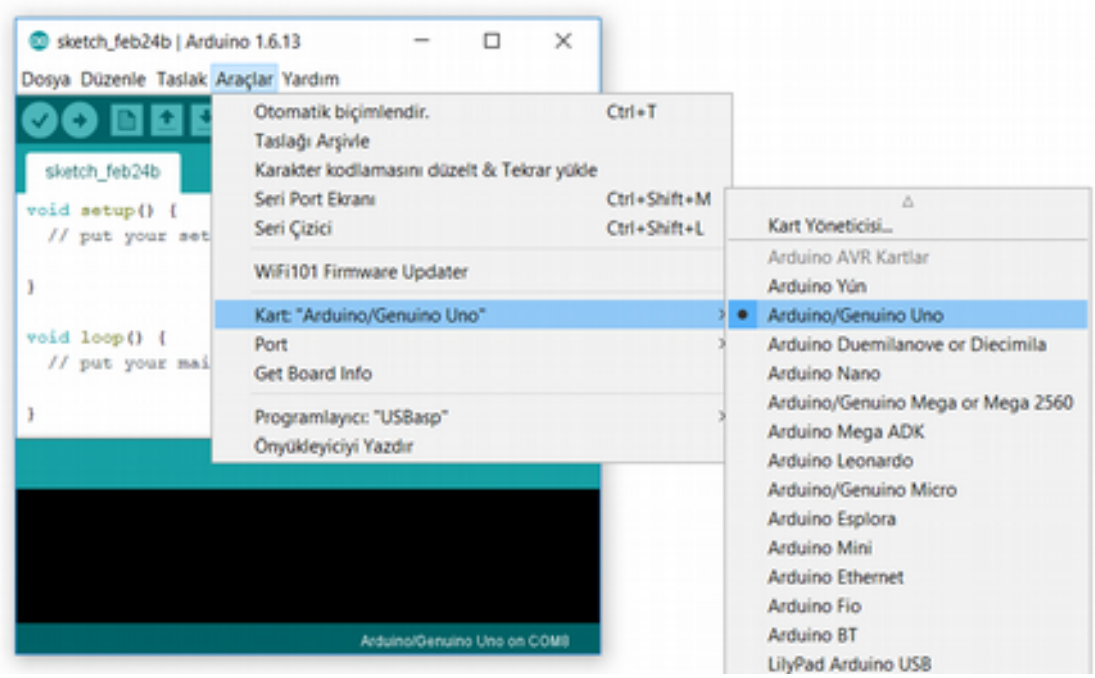


Bağlantı noktaları altında bağlı olan UNO kartımızın COM3'ü kullandığını görmekteyiz.

- **Tools** menüden bilgisayara bağlı olan Arduino kartınızın bağlı olduğu COM portunu seçmeniz gerekiyor.



- Son olarak **Tools** menüsünden çalışmak istediğimiz Arduino kartını aşağıdaki gibi seçmemiz gerekiyor.



Bu işlemden sonra Arduino UNO kartımızı programlayabiliriz.

Geliştirme ortamınız Linux ise "*can't open device "/dev/ttyUSB0": Permission denied*" benzeri bir hata alabilirsiniz. Bunun için ilgili porta linkte anlatıldığı gibi izin vermelisiniz.

https://www.arduino.cc/en/Guide/Linux#toc6

Referanslar

Arduino hakkında bilmeniz gereken temel bilgileri sizlerle paylaştık. Ancak daha fazla bilgi için Arduino'nun resmi web sayfasını incelemeniz tavsiye edilir.

https://arduino.cc/en

Kaynaklar

Bu bölümün başında NodeMCU ile ilgili örnek bir çalışma hakkında sizlere bazı bilgiler verdik. Bu başlık altında Arduino ile çalışırken **ESP8266** kartına yüklediğimiz kaynak kodlara nasıl erişebileceğiniz anlatacağız.

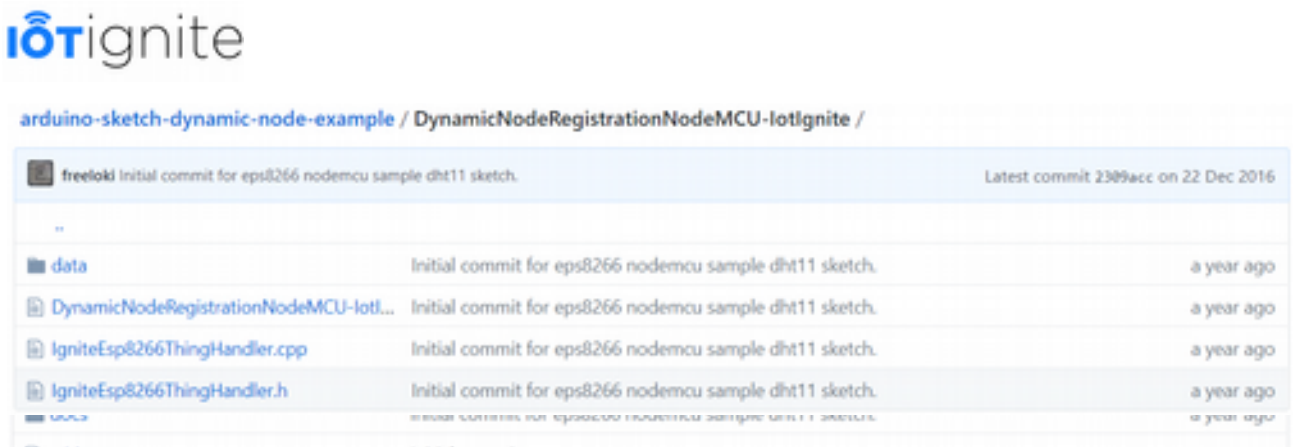
Github'ta Node MCU Firmware ve Şablonu

NodeMCU için ihtiyacımız olan örnek kodlar GitHub üzerinde paylaşılmaktadır. Aşağıda verilen linkten kodlara kolaylıkla erişim sağlayabilirsiniz.

https://github.com/IoT-Ignite/arduino-sketch-dynamic-node-example

Burayı kullanarak kaynak kodlarına erişebiliriz. DynamicNode ile başlayan seçeneği tıkladığınızda aşağıda görüleceği üzere kaynak dosyaları bizlere gösterilir.

Burada bulunan kodları dosya dosya ekleyelim.



DynamicNodeRegistrationNodeMCU-IotIgnite.ino

```
#include <Arduino.h>
#include "IgniteEsp8266WifiManager.h"
#include "IgniteEsp8266ThingHandler.h"

IgniteEsp8266ThingHandler handler;
IgniteEsp8266WifiManager manager(&handler);

void setup() {
  manager.setup();
}

void loop() {
  Serial.println(WiFi.localIP());
  manager.loop();
}
```

IgniteEsp8266ThingHandler.cpp

```
#include "IgniteEsp8266ThingHandler.h"

#include "IgniteEsp8266Timer.h"
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>

// Node Type

#define NODE_TYPE "DYNAMIC NODE - DHT11 SENSOR"

// Sensors
#define SENSOR_DHT11_TEMPERATURE "DHT11 Temperature Sensor"
#define SENSOR_DHT11_HUMIDITY "DHT11 Humidity Sensor"

#define ACTUATOR  true
#define NOT_ACTUATOR false

//Sensor Types
#define TYPE_TEMPERATURE "Temperature"
#define TYPE_HUMIDITY "Humidity"
#define TYPE_LED "Led"


//Actuators
#define ACTUATOR_BLUE_LED "Blue LED Actuator"

//ConnectedPinsForInfo
#define PIN_DATA_DHT11_SENSOR "D4"
#define PIN_DATA_BLUE_LED "D6"

#define PIN_DHT11_SENSOR D4
#define PIN_BLUE_LED D6
#define PIN_RESET_BUTTON D8

//reset button
#define PIN_DATA_RESET_BUTTON "D8"

//Vendors
```

```
#define VENDOR_DHT11  "DHT11 Temperature And Humidity Sensor"
#define VENDOR_BLUE_LED "Simple Diode"

//Data Types

#define DATA_TYPE_FLOAT "FLOAT"
#define DATA_TYPE_STRING "STRING"
#define DATA_TYPE_INTEGER "INTEGER"
#define DATA_TYPE_BOOLEAN "BOOLEAN"

//DHT Specific
#define DHTTYPE DHT11

#define CONFIG_REQUEST "configuration"
#define ACTION_REQUEST "action"
#define RESET_REQUEST "reset"
#define DATA_RESPONSE "data"
#define STATUS_REQUEST "inventory-status"


DHT *IgniteEsp8266ThingHandler::dht = new DHT(PIN_DHT11_SENSOR, DHTTYPE);

bool IgniteEsp8266ThingHandler::ledState;

long IgniteEsp8266ThingHandler::resetStateTime;

void IgniteEsp8266ThingHandler::setup() {
  initBlueLED();
  initResetButton();
  dht->begin();
}

IgniteEsp8266ThingHandler::IgniteEsp8266ThingHandler(): IgniteThingHandler(NODE_TYPE,
getMacAddress()) {

  ledState = false;
  resetStateTime = 0;
}

IgniteEsp8266ThingHandler::~IgniteEsp8266ThingHandler() {
}

void IgniteEsp8266ThingHandler::thingActionReceived(String thingId, String action) {

  Serial.println("Action Received For :");
  Serial.println(thingId);

  Serial.println("Action Message :");
  Serial.println(action);

  StaticJsonBuffer<250> jsonBuffer;
  JsonObject& root = jsonBuffer.parseObject(action);

  if (thingId.equals(ACTUATOR_BLUE_LED)) {
    float actionMsg = root["status"];
    setBlueLED(actionMsg);
  }

}

void IgniteEsp8266ThingHandler::inventorySetup() {

  addThingToInventory(SENSOR_DHT11_TEMPERATURE,
                      TYPE_TEMPERATURE,
                      PIN_DATA_DHT11_SENSOR,
                      NOT_ACTUATOR,
                      VENDOR_DHT11,
                      DATA_TYPE_FLOAT, new IgniteEsp8266Timer(readDHTTemperature));

  addThingToInventory(SENSOR_DHT11_HUMIDITY,
                      TYPE_HUMIDITY,
                      PIN_DATA_DHT11_SENSOR,
                      NOT_ACTUATOR,
                      VENDOR_DHT11,
                      DATA_TYPE_FLOAT, new IgniteEsp8266Timer(readDHTHumidity));
```

```
  addThingToInventory(ACTUATOR_BLUE_LED,
                      TYPE_LED,
                      PIN_DATA_BLUE_LED,
                      ACTUATOR,
                      VENDOR_BLUE_LED,
                      DATA_TYPE_STRING, new IgniteEsp8266Timer(readLedData));

}

void IgniteEsp8266ThingHandler::unknownMessageReceived(String msg) {
}

void  IgniteEsp8266ThingHandler::readDHTTemperature() {
  String packet = "";
  String tempData = "";
  float t = dht->readTemperature();
  if (isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  tempData = String(t);

  StaticJsonBuffer<100> jsonBuffer;
  JsonObject& root = jsonBuffer.createObject();
  JsonArray& data = root.createNestedArray("data");

  root["messageType"] = DATA_RESPONSE;
  root["thingId"] = SENSOR_DHT11_TEMPERATURE;
  data.add(tempData);

  root.printTo(packet);

  Serial.println("Temperature :");
  Serial.println(packet);
  packet += "\n";
  sendMessage(packet);
}

void  IgniteEsp8266ThingHandler::readDHTHumidity() {
  String packet = "";
  String humData = "";
  float h = dht->readHumidity();
  if (isnan(h)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  humData = String(h);

  StaticJsonBuffer<100> jsonBuffer;
  JsonObject& root = jsonBuffer.createObject();
  JsonArray& data = root.createNestedArray("data");

  root["messageType"] = DATA_RESPONSE;
  root["thingId"] = SENSOR_DHT11_HUMIDITY;
  data.add(humData);

  root.printTo(packet);
  Serial.println("Humidity :");
  Serial.println(packet);
  packet += "\n";
  sendMessage(packet);

}

void  IgniteEsp8266ThingHandler::readLedData() {

  String packet = "";
  StaticJsonBuffer<100> jsonBuffer;
  JsonObject& root = jsonBuffer.createObject();
  JsonArray& data = root.createNestedArray("data");

  String ledDataString = "";

  if (ledState) {
    ledDataString = "ON";
  } else {
    ledDataString = "OFF";
  }
```

```
  root["messageType"] = DATA_RESPONSE;
  root["thingId"] = ACTUATOR_BLUE_LED;
  data.add(ledDataString);

  root.printTo(packet);
  Serial.println("Blue LED :");
  Serial.println(packet);
  packet += "\n";
  sendMessage(packet);

}

String IgniteEsp8266ThingHandler::getMacAddress() {
  byte mac[6];
  WiFi.macAddress(mac);
  String cMac = "";
  for (int i = 0; i < 6; ++i) {
    cMac += String(mac[i], HEX);
    if (i < 5)
      cMac += ":";
  }
  cMac.toUpperCase();
  return cMac;
}

void IgniteEsp8266ThingHandler::initBlueLED() {
  pinMode(PIN_BLUE_LED, OUTPUT);
  digitalWrite(PIN_BLUE_LED, LOW);
}

void IgniteEsp8266ThingHandler:: initResetButton() {
  pinMode(PIN_RESET_BUTTON, INPUT);
  attachInterrupt(PIN_RESET_BUTTON, resetOn, RISING);
}

void  IgniteEsp8266ThingHandler::resetOn() {
  Serial.println("\nHold Reset Button 4 Sec.\n");
  detachInterrupt(PIN_RESET_BUTTON);
  attachInterrupt(PIN_RESET_BUTTON, resetOnFinal, FALLING);
  resetStateTime = millis();
}

void  IgniteEsp8266ThingHandler::resetOnFinal() {
  detachInterrupt(PIN_RESET_BUTTON);

  if ((resetStateTime + 4000) < millis()) {
    Serial.println("\nResetting...\n");
    reset();
  } else {
    Serial.println("\nFail to reset. Push 4 Sec.\n");
    attachInterrupt(PIN_RESET_BUTTON, resetOn, RISING);
  }
}

void IgniteEsp8266ThingHandler::setBlueLED(int msg) {

  Serial.println("LED MSG :");
  Serial.println(msg);
  String ledSyncMessage = "{\"ledState\":";

  if (msg == 1) {
    digitalWrite(PIN_BLUE_LED, HIGH);
    ledState = true;
    ledSyncMessage += "1}";

  } else if (msg == 0 ) {
    digitalWrite(PIN_BLUE_LED, LOW);
    ledState = false;
    ledSyncMessage += "0}";
  }

  //send syncronization message here.

  sendMessage(ledSyncMessage);
}
```

IgniteEsp8266ThingHandler.h

```
#ifndef _INCL_IGNITE_ESP8266THING_HANDLER
#define _INCL_IGNITE_ESP8266THING_HANDLER

#include "IgniteThingHandler.h"
#include "DHT.h"

class IgniteEsp8266ThingHandler : public IgniteThingHandler {

  public :

    IgniteEsp8266ThingHandler();
    ~IgniteEsp8266ThingHandler();
    virtual void thingActionReceived(String thingId, String action);
    virtual void inventorySetup();
    virtual void unknownMessageReceived(String msg);
    static void readDHTTemperature();
    static void readLedData();
    static void readDHTHumidity();
    virtual void setup();

  private :
    String getMacAddress();
    static DHT *dht;
    static bool ledState;
    static long resetStateTime;
    void initBlueLED();
    void initResetButton();
    static void resetOn();
    static void resetOnFinal();
    void setBlueLED(int msg);
};

#endif
```

Node Kayıt Yöntemleri

IoT-Ignite platformuna Node kaydını sağlamak için iki yöntem kullanılmaktadır. Bunlar şu şekildedir.

- MQTT ile NodeMCU Register (Kayıt) İşlemi
- SPA ile NodeMCU Register (Kayıt) İşlemi

Şimdi bunlar hakkında ayrıntılı bilgilere geçelim.

MQTT ile NodeMCU Register (Kayıt) İşlemi

NodeMCU donanımlarını kullanarak IoT uygulamaları geliştirebilmek için bunların Ignite platformuna kayıt edilmesi gerekiyor. Bu başlık altında DHT11 sıcaklık/nem sensöründen alınan verileri Iot-Ignite ortamına aktarabilmek için örnek bir NodeMCU kaydını oluşturacağız. Burada kayıt işleminde URL veya MQTT protokolünü kullanacağız. Yani herhangi bir programa gerek duymadan Node aygıtını bu yöntemle kayıt altına alabilirsiniz.

Bu yöntemde kayıt işlemi aşağıda verilen şekilde gerçekleşecektir:

- NodeMCU kablosuz bir ağa bağlanır,
- NodeMCU ağa bağlandıktan sonra, aynı ağda bulunan bir Gateway olup olmadığını tespit etmeye çalışacak (bu uygulamada Gateway olarak Android bir cihazı tercih ettik.)
- Gateway aygtını bulduktan sonra cihazın IP bilgisini kullanarak MQTT üzerinden bağlantı işlemi sağlanacaktır.

Kayıt işlemi tamamlandıktan sonra NodeMCU bulut ortamına veri gönderip alabilir. Bulut platformu da NodeMCU'dan gelen verileri kendi ortamına aktarıp bu verileri kullanabilir. Özetle bu işlemin nasıl yapıldığını ve ihtiyacımız olan kodları burada ele alacağız.

Yapılandırmalar

NodeMCU kayıt işlemini yapmadan önce aşağıda verilen işlemleri yapmamız gerekiyor:

1) IoT-Ignite bulut ortamını kullanabilmek için öncelikle kayıt oluşturmamız gerekiyor. Aşağıda verilen linki kullanarak kayıt işlemini yapabiliriz.

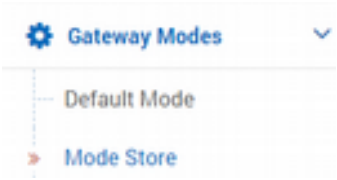
https://enterprise.iot-ignite.com/v3/access/login

2) Kayıt işleminden sonra yapılması gereken ilk işlem Gateway aygıtını sisteme kayıt etmektir. Bununla ilgili bilgileri kitabımızın 7.bölümünde aktarmıştık. Gateway kaydı için IoT-Ignite Agent uygulamasını aşağıda verilen linkten indirmemiz gerekiyor.

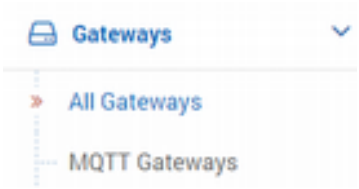
https://play.google.com/store/apps/details?id=com.ardic.android.iotigniteagent

3) Bu adımda IoTGate-MQTT uygulamasını yüklememiz gerekiyor.

4) Uygulamaları yükledikten sonra şimdi bir mod oluşturmamız ve bunu Gateway'e göndermemiz gerekiyor. Mod oluşturmak için Enterprise sayfasında aşağıda verilen yolu takip etmemiz gerekiyor.



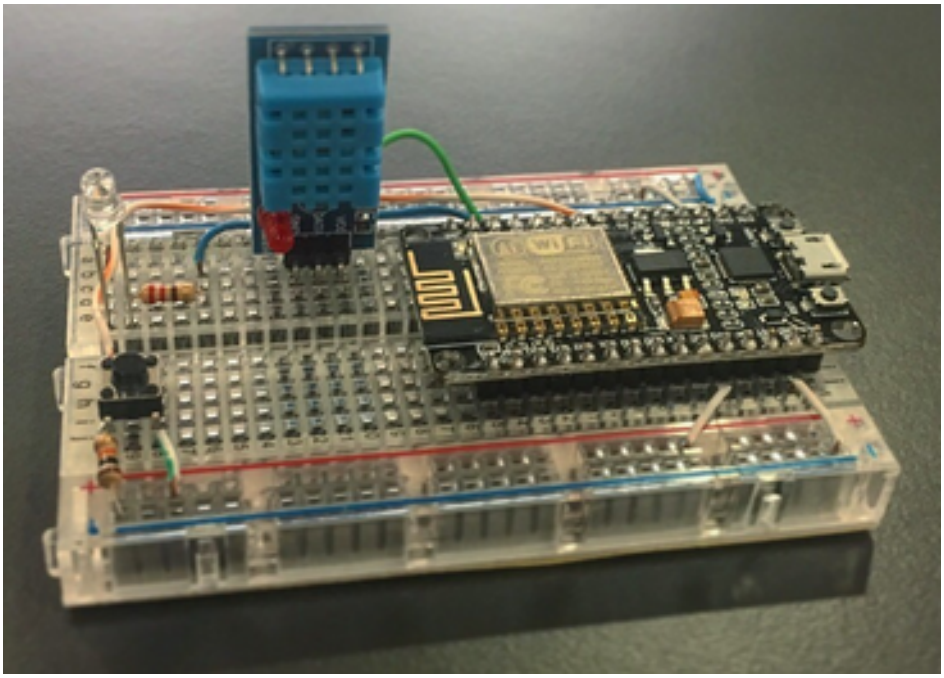
5) **Gateway Modes-> Mode Store** yolunu takip ettikten sonra açılan sayfada Add Mode butonuna tıklayarak ekleme işlemini yapabilirsiniz. Eğer varsayılan modlar ile çalışmak isterseniz, bu durumda yukarıda verilen resimde bulunan **Default Mode** seçeneğine tıklamanız gerekiyor. Hazırladığınız veya kullanmayı tercih ettiğiniz modu kayıtlı olarak Gateway'lere göndermek için aşağıda verilen yolu takip etmeniz gerekiyor.



Açılan sayfada kayıtlı olan Gateway cihazlardan istediğinizi seçebilirsiniz.

Seçtiğiniz Gateway'e modu göndermek için yukarıda verilen **Add To Working Set** butonuna tıklayınız. Tıklama ile birlikte açılan pencereden modunuzu seçip Gateway'e gönderebilirsiniz. Bu işlemlerden sonra Gateway cihazınız kullanıma hazır hale gelmiş olur.

Bu işlemden sonra şimdi Node ve Sensor yapılandırmamızı yapabiliriz. NodeMCU ile amacımız DHT11 sensöründen alınan sıcaklık ve nem verilerini bulut ortamına aktarmaktır. Öncelikle aşağıda verildiği üzere DHT11 ile NodeMCU bağlantısını sağlayalım.



DHT11 sensörünün bağlantısı aşağıdaki gibi yapılmalıdır.

- DHT11'in GND bacağını NodeMCU'da bulunan GND'ye bağlayalım.
- VCC bacağını 5V çıkışına bağlayalım.
- Data pinini de D0 pinine bağlayalım.

Bu bağlantıları sağladıktan sonra bu bölümde bulunan "NodeMCU ve Arduino IDE" başlığı altındaki işlemleri yaparak NodeMCU için ihtiyacımız olan yapılandırmaları sağlayalım. Bunun için ilgili başlığı incelemenizi tavsiye ediyoruz.

Uygulama için ihtiyacımız olan bazı kütüphaneler bulunmaktadır. Bunlar aşağıdaki gibidir:

- ESP8266WiFi
- ESP8266mDNS
- WiFiUdp
- FS
- DHT

Bunları yüklemek için Arduino IDE ortamında aşağıda verile yolu takip etmemiz gerekiyor.

Scketch->Include Library->Manage Library

Tüm bu işlemlerden sonra artık NodeMCU cihazını programlayabiliriz.

İhtiyacımız Olan Kodlar

Bu başlık altında geliştireceğimiz uygulama için ihtiyacımız olan kodları ve bunların hangi amaç için kullanıldığından bahsedeceğiz. NodeMCU kaydı ve programlanması için ihtiyacımız olan kodlar şu şekildedir:

Aşağıda bu uygulama için ihtiyacımız olan kütüphanelerin eklenmesini sağladık. Öncelikle bunu yapmamız gerekiyor.

```
#include <ESP8266WiFi.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
#include "dht.h"
#include <stdio.h>
#include <time.h>
#include <ESP8266mDNS.h>
#include "mdns.h"
#include <stdlib.h>
```

Kütüphaneleri ekledikten hemen sonra NodeMCU'nun bağlanacağı kablosuz ağın SSID ve şifre (password) bilgilerini belirtmemiz gerekiyor. Bunu aşağıdaki gibi yapabiliriz:

```
#define WLAN_SSID "ARDIC_GUEST"
#define WLAN_PASS "w1Cm2ardC"
```

MQTT bağlantısı kurabilmek için aşağıda verilen bilgilere ihtiyacımız var:

1) IoT-Ignite kullanıcı bilgileri,
2) Gateway ismi/IP bilgisi,
3) Port numarası.

Bu bilgiler ile Gateway'in çalışma anında keşfedilmesi sağlanacaktır.

Bunları elde ettikten sonra kodumuzu aşağıdaki gibi yazmamız gerekiyor.

```
char AIO_SERVER[100] = {0};
#define AIO_SERVERPORT 1883
#define AIO_USERNAME "ardic.smart.office"
#define AIO_CLIENTID "ardic.smart.office"
#define AIO_KEY "ardic12345"
```

MQTT sunucusuna bağlanmak için bir ESP8266 WiFiClient nesnesi oluşturup, WiFi istemcisinden, MQTT sunucusundan ve giriş ayrıntılarından geçerek MQTT istemcisini aşağıda verilen şekilde kurun.

```
WiFiClient client;
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_CLIENTID,
AIO_USERNAME, AIO_KEY);
```

Yayınlama ve abone olmak için kurulum bilgileri, bulut üzerinde oluşturulan düğüm (node) kimliği ve sensör kimliğine göre verilir. Bunları kullanarak MQTT yol (path) bilgisini elde ederiz.

```
#define NODE_ID "DHT11NODE"
#define PUBLISH_TEMPERATURE_DATA "things/" NODE_ID "/testthing/data"
#define PUBLISH_HUMIDITY_DATA "things/" NODE_ID "/testthing/data"
#define SUBSCRIBE_HUMIDITY_DATA "things/" NODE_ID "/testthing/config"
Adafruit_MQTT_Publish publishData = Adafruit_MQTT_Publish(&mqtt, PUBLISH_HUMIDITY_DATA,
MQTT_QOS_1);
```

```
Adafruit_MQTT_Subscribe configData = Adafruit_MQTT_Subscribe(&mqtt,
SUBSCRIBE_HUMIDITY_DATA, MQTT_QOS_1);
```

mSDN kütüphanesi, çalışma zamanında mevcut Gateway'in ismini veya IP'sini bulmak için kullanılır. Bunu yapabilmek için Gateway'in yerel ağda yayınladığı servis adını bilmemiz gerekir. Bu uygulama için **_iotgate._tcp .local** bilgisini kullanacağız.

```
#define QUESTION_SERVICE "_iotgate._tcp.local"
```

Bir mDNS paketi ayrıştırıldığında, bu geri arama sorgu başına bir kez çağrılır. **mdns :: Query** tanımı için **mdns.h** dosyasına bakabilirsiniz. Bu fonksiyon kullanılarak bir mDNS örneğini aşağıdaki gibi oluşturmamız gerekiyor.

```
void answerCallback(const mdns::Answer* answer);
mdns::MDns my_mdns(NULL, NULL, answerCallback, buffer, MAX_MDNS_PACKET_SIZE);
```

D0 pin'i DHT11'den sensör verilerini okumak için kullandık. **Mqtt_connect ()** metodu, verilen Wifi bilgilerini kullanarak yerel Wifi'yi bağlamayı sağlar. **setTimer (long delayTime)** metodu ise, sensör bilgilerini Gateway'e göndermeden önce bekleme süresini ayarlamak için kullanılır. İki sensör verisi gönderme arasındaki bekleme süresi bulut yapılandırma ayarları tarafından belirlenir.

```
#define dht_pin D0
void MQTT_connect();
void setTimer(long delayTime);
```

Kodun başında belirtilen giriş bilgilerini kullanarak kablosuz ağa bağlanmamız gerekiyor. Bunun için aşağıda verilen kodları kullanmamız gerekiyor.

```
WiFi.begin(WLAN_SSID, WLAN_PASS);
while (WiFi.status() != WL_CONNECTED) {
                delay(500);
                Serial.print(".");
}
```

Daha sonra, belirli bir hizmet için tüm ana bilgisayar bilgileri için sorgu ve aboneliği ayarlamalıyız. Bunu da aşağıdaki gibi yapabiliriz.

```
my_mdns.Clear();
struct mdns::Query query_mqtt;
strncpy(query_mqtt.qname_buffer, QUESTION_SERVICE, MAX_MDNS_NAME_LEN);
query_mqtt.qtype = MDNS_TYPE_PTR;
query_mqtt.qclass = 1;// "INternet"
query_mqtt.unicast_response = 0;
my_mdns.AddQuery(query_mqtt);
my_mdns.Send();
```

```
mqtt.subscribe(&configData);
```

loop() metodu yerel ağda kullanılabilir bir Gateway aramayı sağlar. Bağlantıyı sağlayabilmek için isim veya IP bilgisi kullanılır. Daha sonra, gelen mDNS paketlerinin yüzdesi hakkında geri bildirim vermeyi sağlarız. Bu işlem için kodlarımız da aşağıdaki gibidir.

```
my_mdns.loop();
#ifdef DEBUG_STATISTICS
      if (last_packet_count != my_mdns.packet_count && my_mdns.packet_count != 0) {
         last_packet_count = my_mdns.packet_count;
         Serial.print("mDNS decode success rate: ");
         Serial.print(100 - (100 * my_mdns.buffer_size_fail / my_mdns.packet_count));
         Serial.print("%\nLargest packet size: ");
         Serial.println(my_mdns.largest_packet_seen);
         sprintf(AIO_SERVER, "%s", hosts[0][HOSTS_SERVICE_NAME].c_str());
         Serial.println(AIO_SERVER);
      }
#endif
```

Eğer herhangi bir Gateway bulunursa, NodeMCU gerekli bilgileri kullanarak bir MQTT bağlantısı kurmaya çalışacaktır. Bunun için connect() komutunu kullanmalıyız.

```
MQTT_connect();
```

Ardından, IoT-Ignite bulut sistemi tarafından gönderilen konfigürasyon bilgisini okumamız gerekiyor. Bu uygulamada sensör bilgisi göndermek için zaman aralığı belirlenir.

```
Adafruit_MQTT_Subscribe *subscription;
 while ((subscription = mqtt.readSubscription(5000))) {
              if (subscription == &configData) {
              Serial.print(F("Got: "));
              Serial.println((char *)configData.lastread);
}
```

Aşağıda verilen kodlar yapılandırma bilgisi okunamadığı zaman geçici olarak kullanılır. Burada konsoldan bekleme süresi okunmaya çalışılır ve bu değer long'a dönüştürür. Bunu kullanarak zaman gecikmesi ayarlanır.

```
if(delayTime <= 0){
   Serial.println("Please enter delay time :");
   while (!Serial.available());
   for(int i = 0; Serial.available() > 0; ++i){
          input[i] = (long)Serial.read();
   }
   delayTime = atoi(input);
   Serial.print("delay time: ");
   Serial.println(delayTime);
}

setTimer(delayTime);
```

Tüm bu işlemlerden sonra yapılması gereken tek işlem DHT11 sensöründen okunan verileri IoT-Ignite ortamına aktarmak kalıyor. Bunu da aşağıdaki gibi yapabiliriz.

```
DHT.read11(dht_pin);
char data[100];
sprintf(data, "Humidity: %.2f, Tempereture: %.2f", DHT.humidity, DHT.temperature);
publishData.publish(data);
```

Buna ek olarak MQTT bağlantısını aktif tutmak için sunucuta ping atabiliriz.

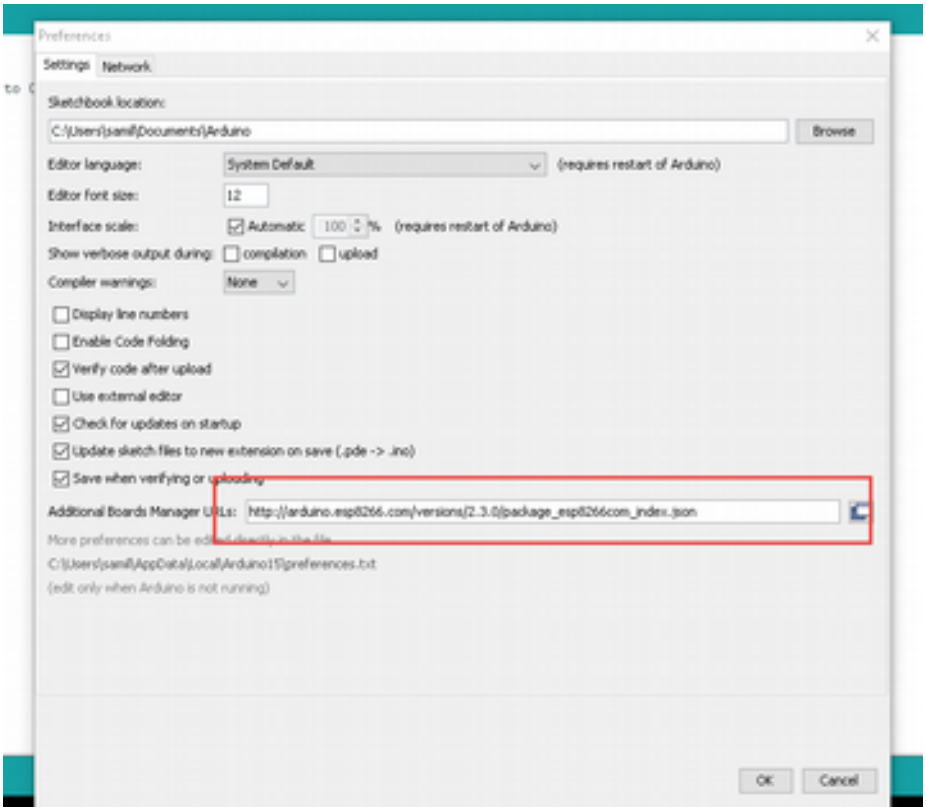
```
if(! mqtt.ping()) {
   mqtt.disconnect();
}
```

NodeMCU Kodlarının Yüklenmesi

Kodların açıklaması yukarıdaki gibidir. Şimdi bu kodu nasıl NodeMCU'ya yükleyebiliriz bundan bahsetmek istiyorum.

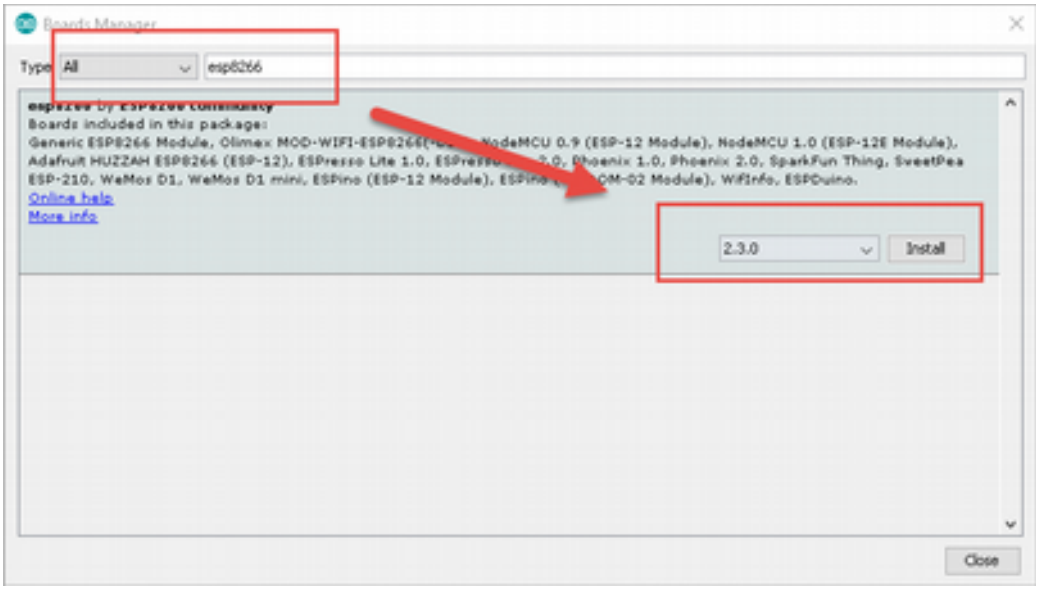
Bunun için öncelikle Arduino IDE ortamında **File > Preferences** yolunu takip edelim ve açılan yeni pencereye şu linki ekleyelim. Bu linkin amacı NodeMCU'yu Arduino IDE'ye tanıtmaktır.

http://arduino.esp8266.com/versions/2.3.0/package_esp8266com_index.json



Tool menüsünde bulunan **Board Manager** seçeneğine tıklayalım ESP8266 için ihtiyacımız olan

kütüphaneyi aşağıdaki gibi bulup yükleyelim.

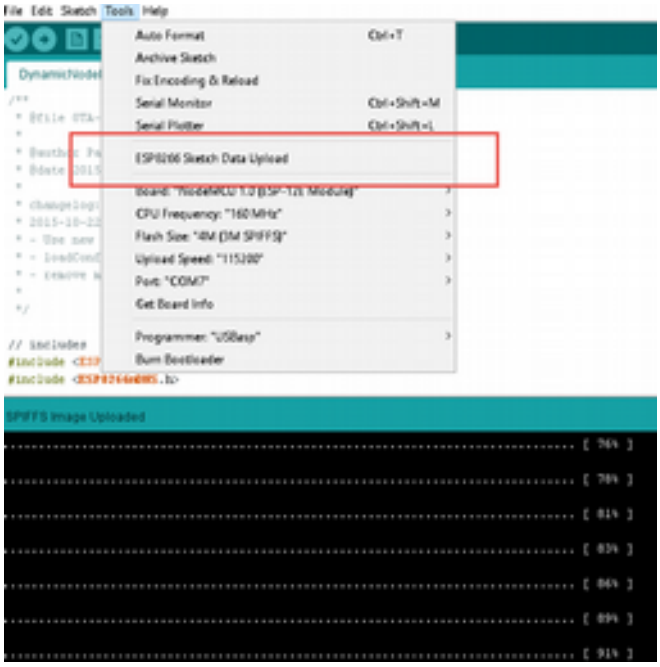


Yukarıdaki işlemleri yaptıktan sonra dosya sistemi ve ihtiyaç duyulan diğer kütüphaneleri kurmamız gerekiyor. Bunun için aşağıda verilen linkte bulunan ESP8266FS isimli dosyayı indiriniz.

https://github.com/esp8266/arduino-esp8266fs-plugin/releases/tag/0.2.0

Dosyaları aşağıda verilen yola ekleyiniz.

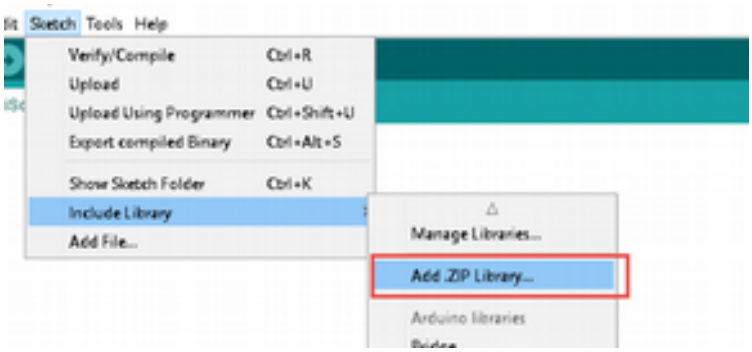
C:\Program Files (x86)\Arduino\tools\ESP8266FS\tool



Arduino IDE'yi yeniden başlatın ve **Tools** menüsü altında veri yükleme için hazır hale getirin. Burada uygulama için kullanılacak **Timer** kütüphanesini kurmamız gerekiyor.

https://github.com/JChristensen/Timer

Zip kütüphanelerini Arduino IDE ortamında yüklemek için aşağıda verilen yolu takip ediniz.



Açılan yeni pencereden zip dosyalarını kolaylıkla projenize ekleyebilirsiniz. Ayrıca **Sketch > Include Library > Manage Library** yolunu takip ederek aşağıda verilen kütüphaneleri de projenize yüklemeniz gerekiyor.

- ESP8266WiFi
- ESP8266mDNS
- WiFiUdp
- FS
- DHT

Tüm bu ön hazırlık ve kurulum işlemlerinden sonra NodeMCU'yu USB port üzerinden bilgisayarınıza bağlayınız. Bundan sonraki adımda uygulamanın temel kodunu yükleyeceğiz.

Öncelikle DynamicNodeRegistrationNodeMCU-IotIgnite kodunu aşağıdaki bağlantıdan indirelim...

https://github.com/IoT-Ignite/arduino-sketch-dynamic-node-example

Ve aşağıda belirtilen yolun altına yapıştıralım.

C:\Users\{name}\Documents\Arduino\DynamicNodeRegistrationNodeMCU-IotIgnite\data

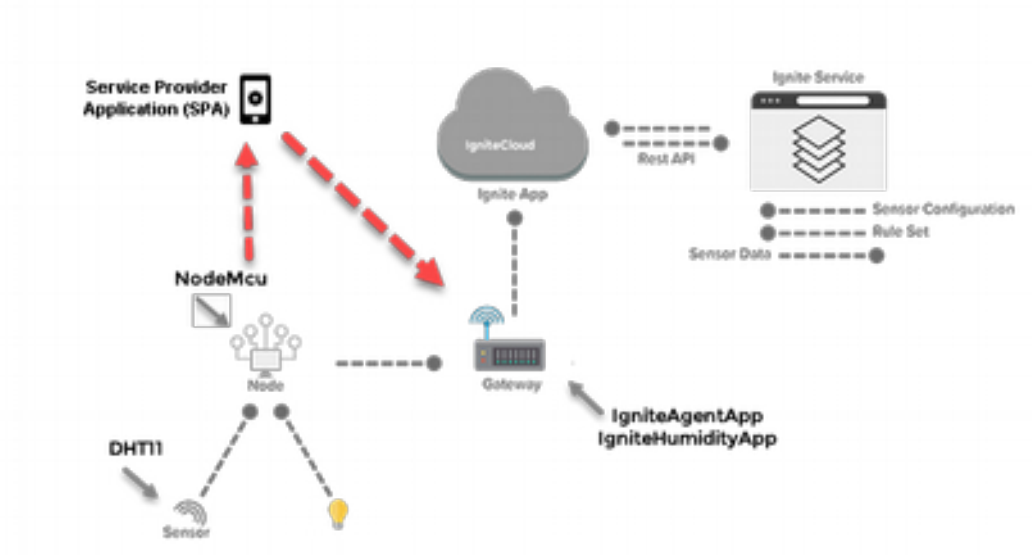
Yukarıda verilen kodu NodeMCU'ya yüklemek için Arduino IDE ortamında açınız.

Upload butonuna bastığımızda uygulama kodu NodeMCU'ya yüklenir ve NodeMCU bir erişim noktası olarak başlatılır.

SPA ile NodeMCU Register (Kayıt) İşlemi

"NodeMCU Kodlarının Yüklenmesi" başlığında NodeMCU uygulama kodunu yükledik ve aygıtımızı erişim noktası olarak başlattık. Bu adımda NodeMCU'yu SPA ile kayıt etmeyi göstereceğiz. SPA ile bir aygıtı Gateway'e kayıt edebiliriz.

Kayıt işlemininin nasıl yapıldığını analamak adına aşağıda verilen şemayı inceleyin.



SPA Uygulamasını İndirmek

Kayıt işlemini yapmak için şu adımları takip etmeliyiz:

- Aşağıda verilen linkten SPA'yı indiriniz.

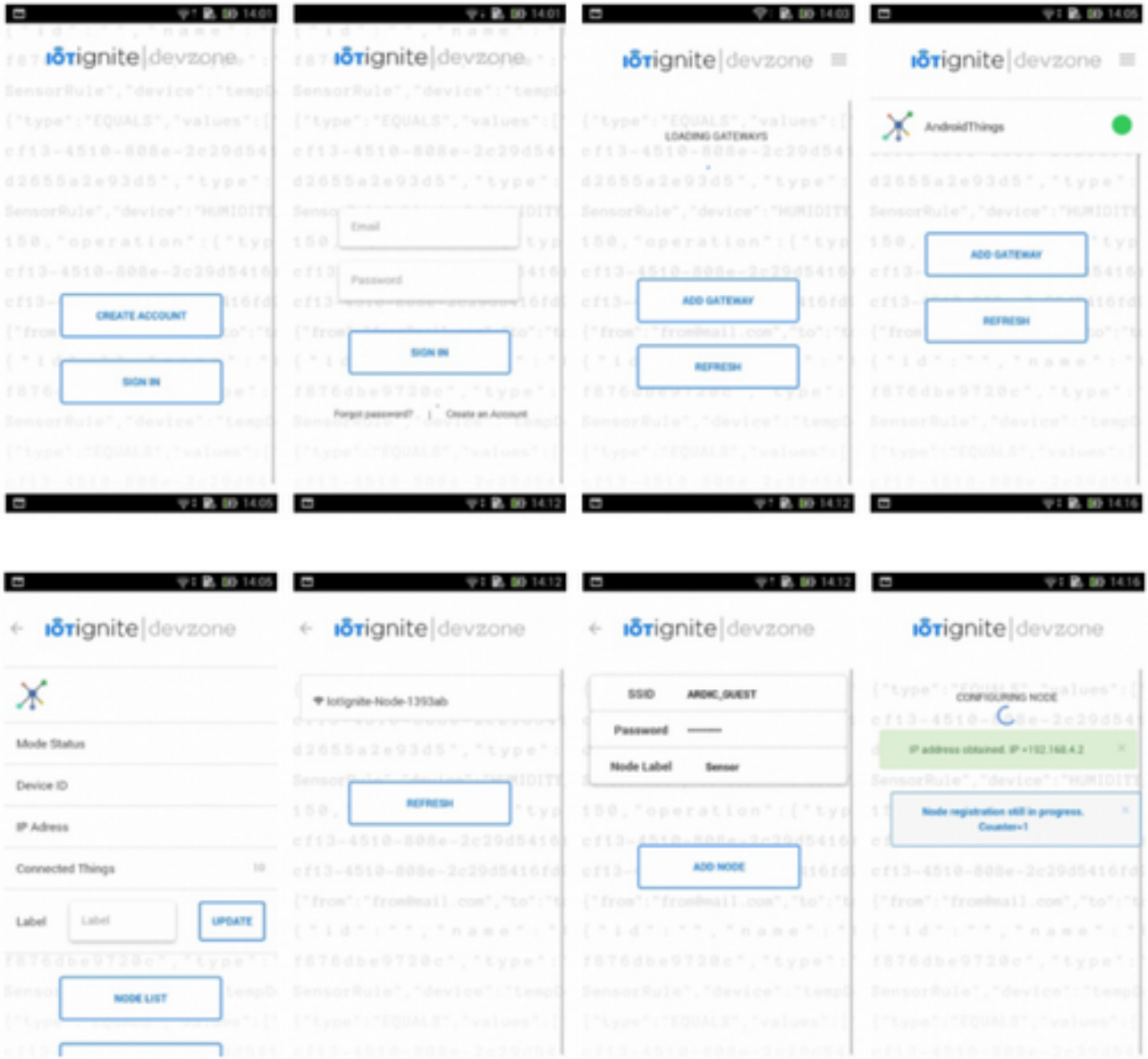
https://download.iot-ignite.com/ServicePlatformApp/

- Hizmetinizdeki Devzone kimlik bilgileriniz ile giriş yapın.

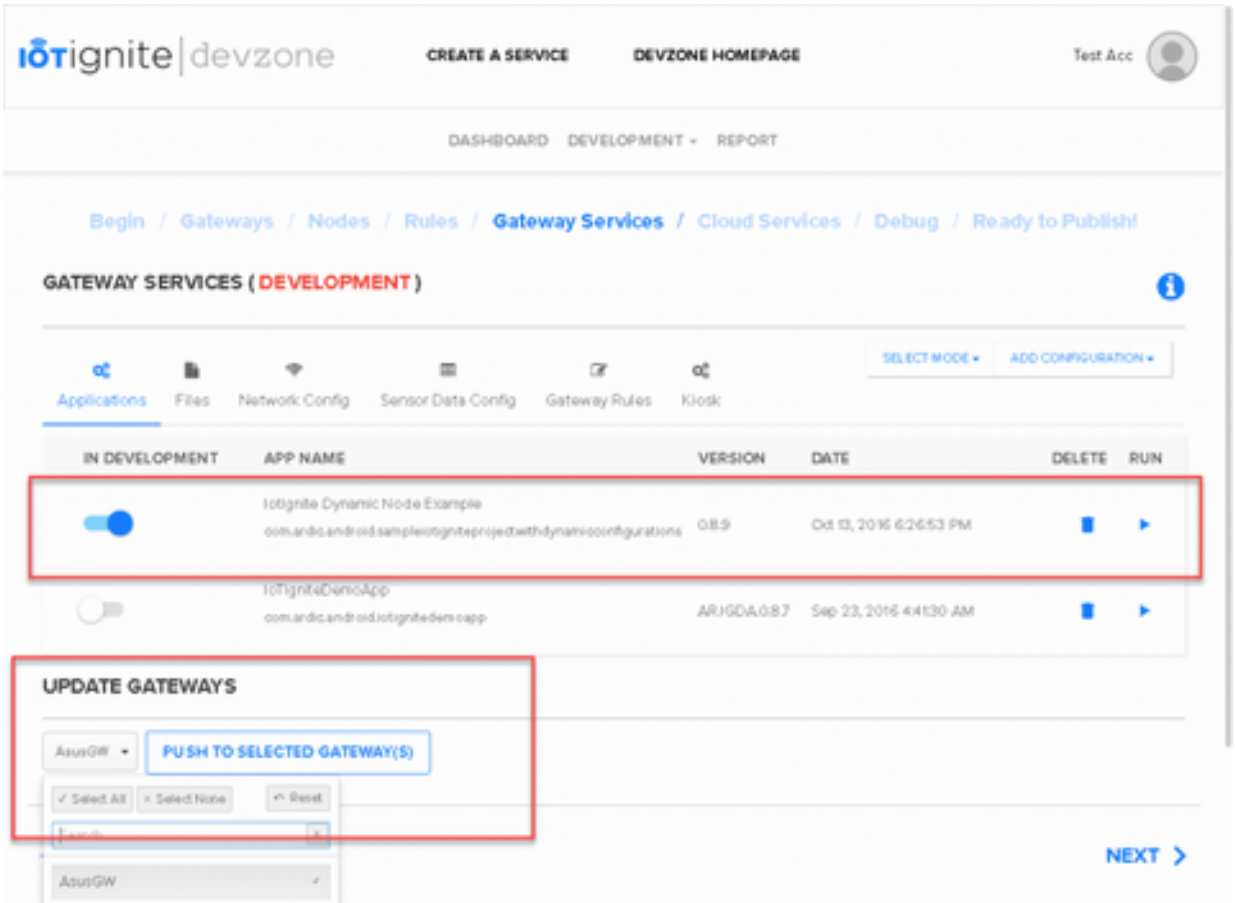
NodeMCU'nun Kayıt Edilmesi

Ignite ortamına giriş yaptıktan sonra **Gateways** seçeneğine tıklayalım ve açılan **Thing** listesinden **Add Node Manually** butonuna tıklayalım. NodeMCU'ya bağlanmak için kablosuz bilgilerinizi yazınız. SPA ve NodeMCU aynı ağa bağlandıktan sonra, SPA yazılımı Node aygıtını ağda taramaya başlar. Ağda bulduktan sonra Node aygıtı bir seçenek olarak size sunulur. Aygıta tıkladıktan sonra Node cihazı otomatik olarak sisteme kayıt edilir.

Bahsi geçen olayların görsel sonuçları aşağıdaki gibidir.



Akıllı telefon tarafında yukarıda görülen aşamaları başarıyla tamamladıktan sonra, Ignite platformunda **Node** aygıtını aşağıdaki gibi görebiliriz.



Yukarıdaki işlemle beraber Node aygıtı başarıyla sisteme kayıt edilir. Buradan itibaren Node aygıtının aldığı sensör verileri sisteme akmaya başlar.

Ağ bağlantısı sağlanırken, SPA uygulamasından Node'a gönderilen bazı parametreler şunlardır:

Node ID: Bu bilgi Node aygıtını IoT Ignite'ye kayıt ederken kullanılan benzersiz bir veridir. ID olarak ifade edilen bu veri, kimlik olarak kullanılmaktadır. Uygulama içinde çakışmaların önüne geçmek için aynı Gateway alanında bulunan tüm Node ID'lerin benzersiz olması gerekmektedir.

Gateway ID: Ağ geçidi kimliği, Node aygıtın bağlanacağı ağ geçidi kimliğini temsil etmek için kullanılır. Node aygıtı sadece belirtilen ağ geçidine bağlanabilir. Bu örnekte, ağ geçidi (gateway) olarak kullanılacak Android cihaza verilen ID.

SSID Credentials (SSID Kimlik Bilgileri): Node aygıtlar yerel ağa bağlanırken SSID adı ve şifre kullanır.

Müşteri Uygulaması (Customer App)

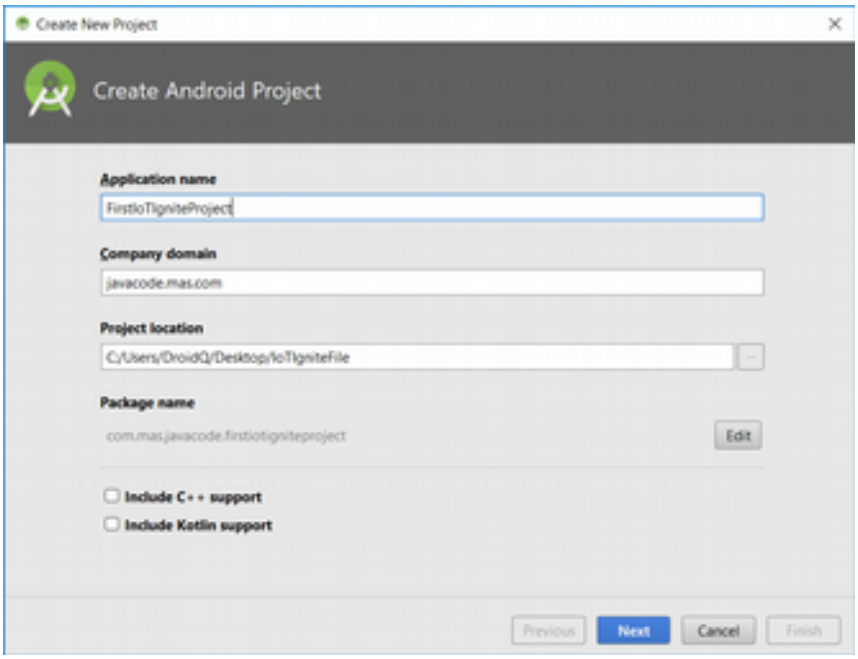
Bu başlık altında, Android Studio'da bir IoT-Ignite projesi nasıl oluşturulacağını ve IoT-Ignite bağımlılıklarını nasıl ayarlayacağımızı göstereceğiz.

Adım Adım Android Studio ile IoT-Ignite Uygulaması Geliştirmek

Android Studio ortamında IoT-Ignite destekli bir proje geliştirmek için takip edilmesi gereken adımlar şu şekildedir.

Adım 1: Yeni Proje Başlatmak

Android Studio geliştirme ortamını açınız ve yeni bir proje oluşturunuz.



Projeye **FirstIoTIgniteProject** ismini verdikten sonra **Next** butonuna tıklayalım.

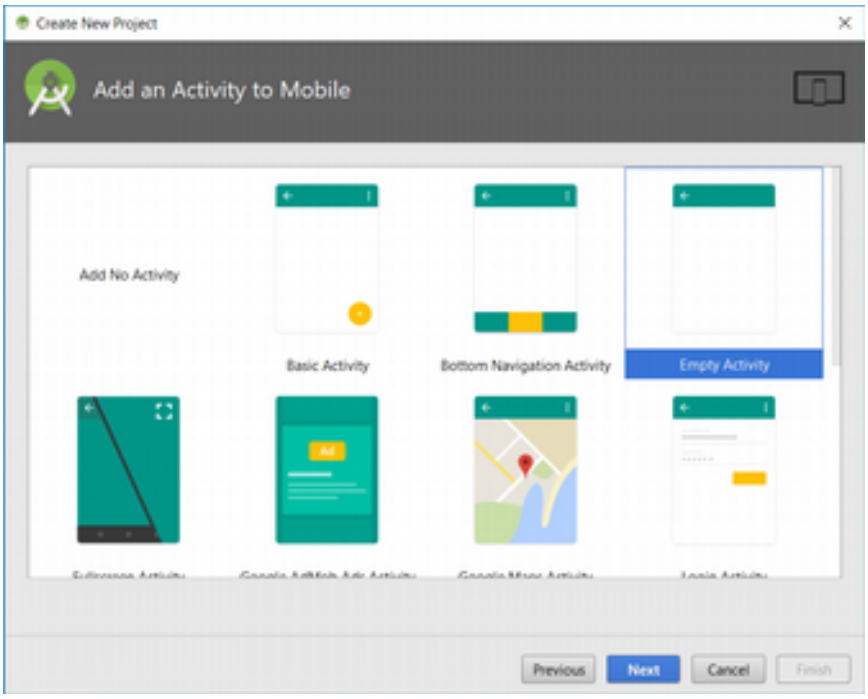
Adım 2: Hedef Aygıtları Ayarlamak

Uygulamanın çalışacağı hedef aygıtlar aşağıda açılan yeni pencerede belirlenir. Amacımız uygulamayı akıllı telefonlarda çalışacak şekilde kullanmaktır. Ayrıca minimum SDK veya API değeri olarak **API 24**'ü seçtik.

Hedef aygıtı belirledikten sonra **Next** butonuna tıklayıp devam edelim.

Adım 3: Main Activity Eklemek

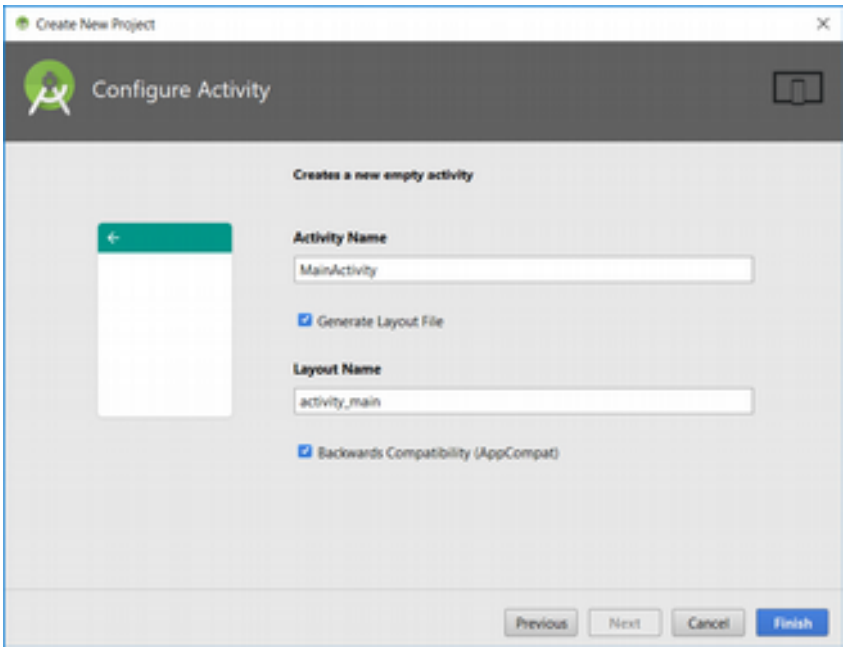
Açılan yeni pencerede yeni bir **Activity** eklememiz gerekiyor.



Empty Activity etkinliğini seçtikten sonra **Next** butonuna tıklayıp devam edelim.

Adım 4: Main Activity'i Düzenlemek

Activitiy'i seçtikten sonra aşağıda açılan pencerede etkinlik ismini değiştirebilir veya varsayılan isim ile devam edebilirsiniz.

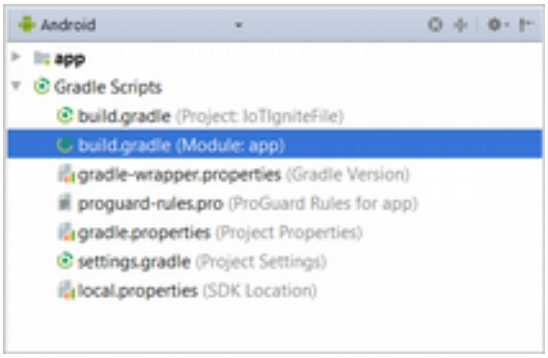


Finish butonuna tıklayarak projemizi oluşturabiliriz.

Adım 5: Depoları ve Bağımlılıkları Yapılandırmak

IoT-Ignite için oluşturduğumuz bu yeni projede Ignite ile uygulama geliştirmek için bazı ayarlamalar yapmamız gerekiyor. Bunları yapmadan IoT-Ignite için geliştirilen kütüphaneleri kullanamayız.

Öncelikle aşağıda verilen dosyayı açalım. Dosyayı kolay bir şekilde bulabilmek için proje görünümünü Android olarak değiştirmenizi tavsiye ediyoruz.



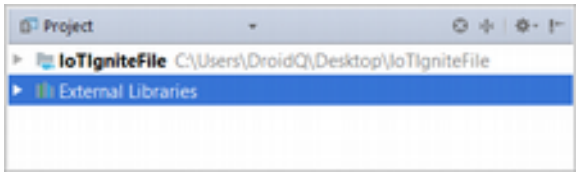
Dosya açıldıktan sonra aşağıda koyu renkli gösterilen satırları dosyanıza ekleyiniz.

```
20    repositories {
21        mavenCentral()
22        maven {
23            url "http://repo.maven.apache.org/maven2"
24        }
25        maven {
26            url "https://repo.iot-ignite.com/content/repositories/releases"
27        }
28    }
29


31    dependencies {
          ...
40        implementation 'com.ardic.android:IoTIgnite:0.8.2'
41        implementation 'com.google.code.gson:gson:2.8.5'
42    }
```

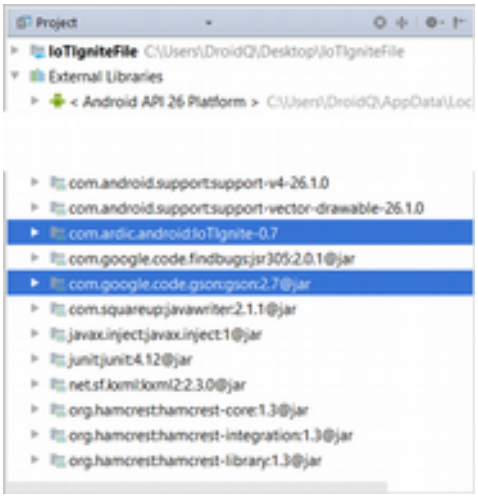
Yukarıda verilen **repositories** bloğu olduğu gibi eklenirken, **dependencies** içerisine ise sadece koyu renkli iki satır eklenmelidir. Dependencies içine eklenen ilk satır, Google tarafından geliştirilen **Gson** kütüphanesini eklemek için kullanılır. Bunu ekleme sebebimiz IoT-Ignite platformunda veri transferi için **JSON** yapısının kullanılmasıdır. Gson kütüphanesi, JSON verilerini ayrıştırmak için kullanılır. Bundan dolayı bunu mutlaka eklemeniz gerekiyor. Diğer satır ise **ARDIC** tarafından geliştirilen IoT-Ignite kütüphanesini projeye eklemek için kullanılmaktadır.

Bunları ekledikten hemen sonra **Build > Rebuild** yolunu takip ederek ilgili kütüphaneleri projeye ekleyebiliriz.

Kütüphanelerin projeye eklenip eklenmediğini kontrol etmek adına öncelikle Android görünümünden **Project** görünümüne geçiş yapalım.



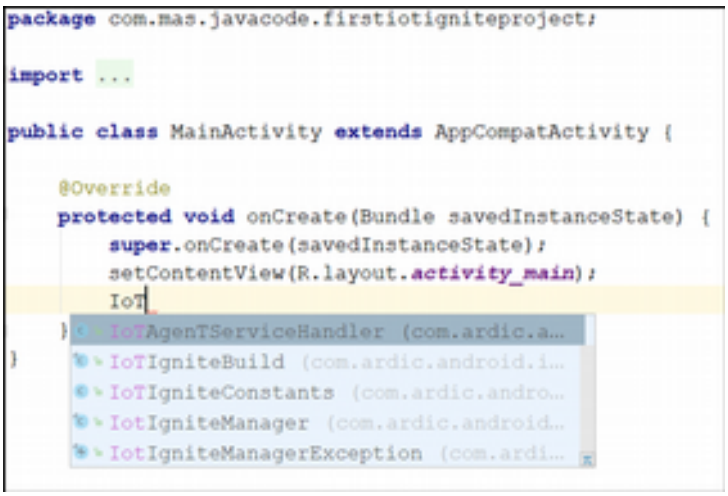
Project görünümü altında yer alan **External Libraries** açılır listesini genişlettiğimiz zaman, harici olarak eklenen kütüphaneleri görebiliriz. Aşağıda verilen listede eğer işaretli kısımları görüyorsanız kütüphanelerin projeye eklendiğinden emin olabilirsiniz.



Adım 6: Kurulum Testi

Kütüphaneleri ekledikten sonra ilk test aşamasını gerçekleştirebiliriz. Buradaki amaç editör alanında IoTIgnite ile eklenen kütüphanelere erişim sağlayıp sağlamadığımızı kontrol edebilmektir.

MainActivity içinde bulunan **onCreate()** metodunda **IoTIgnite** yazdığınızda aşağıda verilen sonuç ile karşılaşırsanız, bu durumda işlemlerin başarıyla gerçekleştiğini söyleyebiliriz.



Dinamik Node kütüphanesi başlığı altında kaldığımız yerden devam ederek, bölüm başında ekran görüntüsünü verdiğimiz müşteri uygulamasını geliştireceğiz.

HwNodeAppTemplate Kütüphanesi

Bu kütüphane IoT-Ignite ve Node arasında ve Wifi üzerinde kurulun bağlantıyı yönetmeyi sağlamaktadır. Kütüphane aşağıda verilen işlevleri kapsamaktadır:

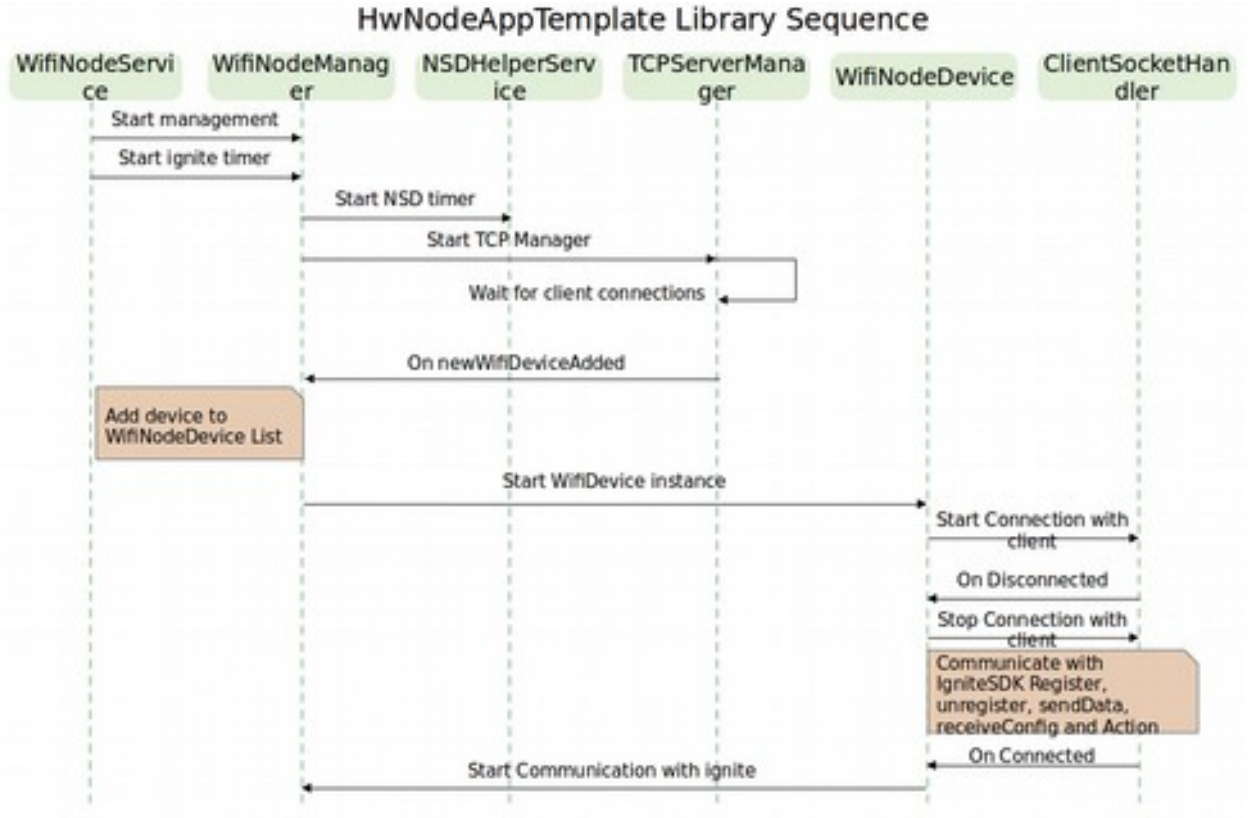
- Network Service Discovery (Ağ Hizmeti Bulma)
- TCP/IP Server

- IoTIgnite Connection Handler

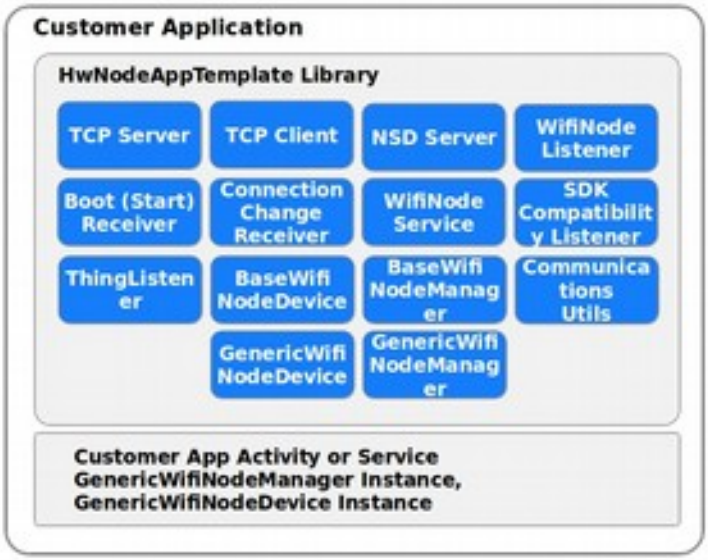
Bu kitaplığı Android Studio ortamına **.aar** uzantılı olarak eklememiz gerekiyor.

WifiNodeDevice sınıfı, IoTIgnite Wifi bağlantısı için gerekli temel işlevleri uygular. Bu kütüphane sayesinde, aşağıdaki işlemleri yapabiliriz:

- Belirli bir mDNS hizmeti, IP adresinizi ve portunuzu dinlemenizi sağlar. (Network Service Discovery Server – Ağ Hizmeti Bulma Sunucusu).
- Soketleri bağlı olan Node aygıtları dinleyen TCP/IP sunucusu.
- TCP / IP istemci soketi, Node verilerini okur, yapılandırmasını gerçekleştirir ve komutları yazmayı sağlar.
- Uygulamanız bir servis olarak kullanılıyorsa, cihaz önyüklemesi yapıldıktan sonra otomatik olarak başlar (Boot Completed Receiver – Önyükleme Tamamlandı Alıcısı).
- Wifi bağlantısında bir sorun meydana geldiğinde Wifi ağ bağlantı durumunu izlemek ve ilgili hizmetleri yeniden başlatmak (Connection State Changed Receiver – Bağlantı Durumu Değişti Alıcısı).
- IoT-Ignite Agent'da ve Customer App'da kullanılan IgniteSDK'ler çakıştığı zaman bu durum hemen ele alınır (Ignite SDK Compatibility Listener – Ignite SDK Uyumluluk Alıcısı).
- Soket ve IoT-Ignite bağlantılarını izleme ve bağlantı sorunlarını işleme (Socket and Ignite Connection Listener – Soket ve Ignite Bağlantı Alıcısı).
- Ignite platformuna veri göndermek ve ondan eylem ve yapılandırma işlemleri için yeni mesajlar almak.



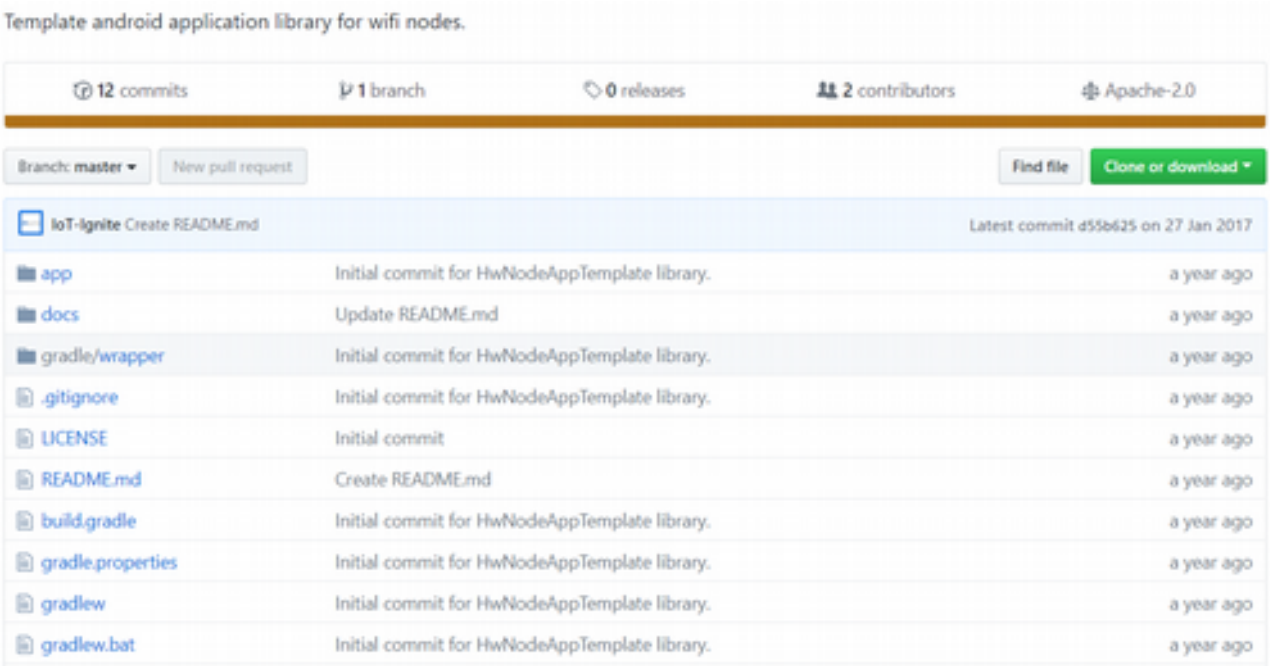
Müşteri uygulaması, bu kütüphanenin nasıl kullanıldığının en somut örneğidir. Bu uygulamanın modeli aşağıdaki gibidir.



Bu kütüphanenin kaynak kodları GitHub üzerinde paylaşılmaktadır. Aşağıda verilen adresten kodlara erişebilirsiniz.

https://github.com/IoT-Ignite/android-library-hwnodeapptemplate

Bu adresi ziyaret ettiğinizde proje dosyasını aşağıdaki gibi görebilirsiniz.



Kaynaklar

Buraya kadar anlattığımız konular için bilmemiz gereken iki temel kaynak vardır. Şimdi de bunları açıklayalım...

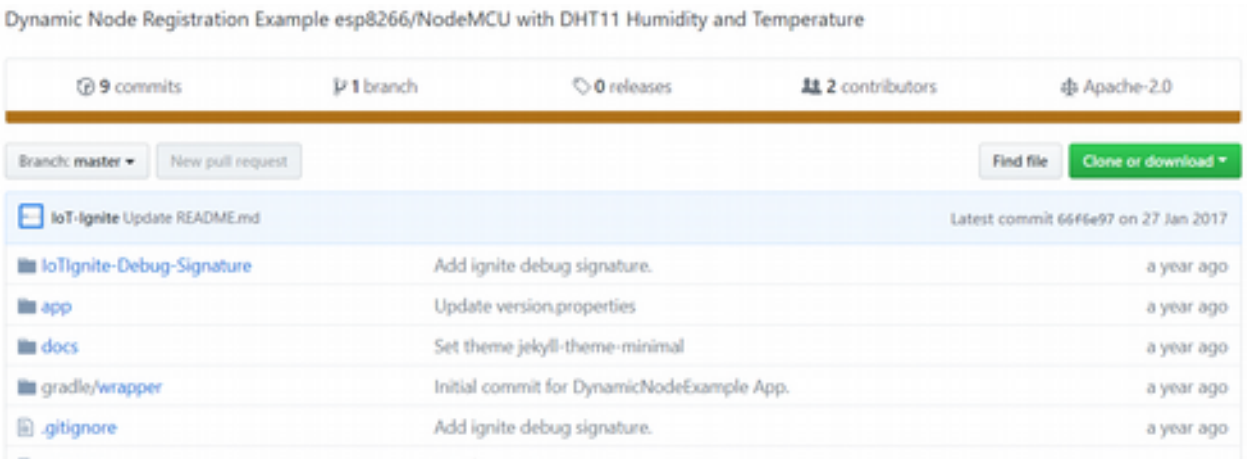
Dinamik Node Kütüphanesi

Bölümün başında **DynamicNodeExample** uygulamasının ekran çıktılarını görmüştük. Ayrıca müşteri uygulamasında yeni bir projenin nasıl oluşturulacağından bahsettik. Şimdi ise

oluşturduğumuz proje üzerinden Dinamik Node uygulaması için ihtiyacımız olan kodları tek tek gösterelim.

Kodlara geçmeden önce uygulamayı GitHub üzerinden paylaştığımızı belirtelim. Projeye erişim sağlamak için aşağıda verilen adresi kullanabilirsiniz.

https://github.com/IoT-Ignite/android-example-DynamicNode



Sağ üst köşede bulunan **Clone or download** seçeneği ile projenin tamamını bilgisayarınıza indirebilirsiniz. Bu bilgileri verdikten sonra kodlara geçebiliriz.

Öncelikle Android uygulaması için bir arayüz tasarlayalım.

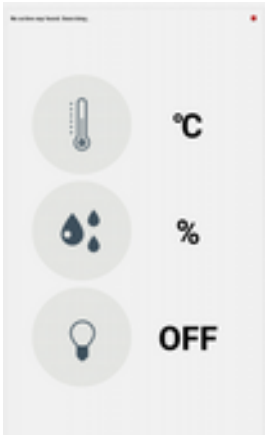
activity_main.xml

```
1     <?xml version="1.0" encoding="utf-8"?>
2     <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3         xmlns:tools="http://schemas.android.com/tools"
4         android:layout_width="match_parent"
5         android:layout_height="match_parent"
6         android:gravity="center"
7         android:orientation="vertical"
8
tools:context="com.ardic.android.sampleiotigniteprojectwithdynamicconfigurations.activit
y.HomeActivity">
9
10
11        <LinearLayout
12            android:layout_width="match_parent"
13            android:layout_height="wrap_content"
14            android:gravity="center_vertical"
15            android:orientation="horizontal">
16
17            <TextView
18                android:id="@+id/nodeIDTextView"
19                android:layout_width="wrap_content"
20                android:layout_height="wrap_content"
21                android:layout_margin="5dp"
22                android:gravity="start"
23                android:text="@string/no_available_node"
24                android:textStyle="bold" />
25
26            <ImageView
27                android:id="@+id/deleteNodeImgView"
28                android:layout_width="25dp"
29                android:layout_height="25dp"
30                android:src="@drawable/delete_icon"
31                android:visibility="gone" />
32
33            <LinearLayout
34                android:layout_width="match_parent"
35                android:layout_height="wrap_content"
```

```
                android:gravity="right">

                <ImageView
                    android:id="@+id/igniteStatusImgView"
                    android:layout_width="15dp"
                    android:layout_height="15dp"
                    android:layout_margin="5dp"
                    android:src="@drawable/disconnected" />
            </LinearLayout>

        </LinearLayout>


        <LinearLayout
            android:id="@+id/sensorDashboardLayout"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:orientation="horizontal">

            <!-- Icons Layout -->
            <LinearLayout
                android:id="@+id/temperatureLinearLayout"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:orientation="vertical">

                <ImageView
                    android:id="@+id/temperatureImageView"
                    android:layout_width="150dp"
                    android:layout_height="150dp"
                    android:layout_margin="5dp"
                    android:background="@mipmap/temperature16" />

                <ImageView
                    android:id="@+id/humidityImageView"
                    android:layout_width="150dp"
                    android:layout_height="150dp"
                    android:layout_margin="5dp"
                    android:background="@mipmap/waterdrop" />


                <ImageView
                    android:id="@+id/ledImageView"
                    android:layout_width="150dp"
                    android:layout_height="150dp"
                    android:layout_margin="5dp"
                    android:background="@mipmap/led2_off" />


            </LinearLayout>
            <!-- End of Icons Layout -->


            <!-- Text Layout -->
            <LinearLayout
                android:id="@+id/textLinearLayout"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:gravity="center"
                android:orientation="vertical">

                <TextView
                    android:id="@+id/temperatureTextView"
                    android:layout_width="150dp"
                    android:layout_height="150dp"
                    android:layout_margin="5dp"
                    android:gravity="center"
                    android:text="\u2103"
                    android:textSize="50sp"
                    android:textStyle="bold" />


                <TextView
                    android:id="@+id/humidityTextView"
                    android:layout_width="150dp"
```

```
                    android:layout_height="150dp"
                    android:layout_margin="5dp"
                    android:gravity="center"
                    android:text="%"
                    android:textSize="50sp"
                    android:textStyle="bold" />

                <TextView
                    android:id="@+id/ledTextView"
                    android:layout_width="150dp"
                    android:layout_height="150dp"
                    android:layout_margin="5dp"
                    android:gravity="center"
                    android:text="OFF"
                    android:textSize="50sp"
                    android:textStyle="bold" />

            </LinearLayout>
            <!-- End of Text Layout -->
        </LinearLayout>

    </LinearLayout>
```

Uygulamanın arayüzü aşağıdaki gibidir.



Uygulamanın Java kısmında ihtiyacımız olan bazı sabitler bulunmaktadır. Bu sabitler için Java sınıfı ve kodları aşağıdaki gibi olmalıdır.

DynamicNodeConstants.java

```
package com.ardic.android.sampleiotigniteprojectwithdynamicconfigurations.constants;


public class DynamicNodeConstants {
    public static final String TYPE = "DYNAMIC NODE - DHT11 SENSOR";
    public static final String TEMPERATURE_SENSOR = "DHT11 Temperature Sensor";
    public static final String HUMIDITY_SENSOR = "DHT11 Humidity Sensor";
    public static final String ACTUATOR_BLUE_LED = "Blue LED Actuator";
    public static final String LED_ON_ACTION = "{\"status\":1.0}";
    public static final String LED_OFF_ACTION = "{\"status\":0.0}";

    private DynamicNodeConstants() {
    }

}
```

Sabitler için kodlarımızı yazdıktan sonra uygulamanın **MainActivity** kodlarını yazabiliriz.

MainActivity.java

```
1    package com.ardic.android.sampleiotigniteprojectwithdynamicconfigurations.activity;
2
3    import android.content.ActivityNotFoundException;
4    import android.content.DialogInterface;
5    import android.content.Intent;
6    import android.net.Uri;
7    import android.os.Bundle;
8    import android.support.annotation.NonNull;
9    import android.support.v7.app.AlertDialog;
10   import android.support.v7.app.AppCompatActivity;
11   import android.text.TextUtils;
12   import android.util.Log;
13   import android.view.Menu;
14   import android.view.MenuItem;
15   import android.view.View;
16   import android.widget.ImageView;
17   import android.widget.TextView;
18   import android.widget.Toast;
19
20   import com.afollestad.materialdialogs.DialogAction;
21   import com.afollestad.materialdialogs.MaterialDialog;
22   import com.ardic.android.iot.hwnodeapptemplate.base.BaseWifiNodeDevice;
23   import com.ardic.android.iot.hwnodeapptemplate.listener.CompatibilityListener;
24   import com.ardic.android.iot.hwnodeapptemplate.listener.ThingEventListener;
25   import com.ardic.android.iot.hwnodeapptemplate.listener.WifiNodeManagerListener;
26   import com.ardic.android.iot.hwnodeapptemplate.manager.GenericWifiNodeManager;
27   import com.ardic.android.iot.hwnodeapptemplate.node.GenericWifiNodeDevice;
28   import com.ardic.android.iot.hwnodeapptemplate.service.WifiNodeService;
29   import com.ardic.android.iotignite.exceptions.UnsupportedVersionExceptionType;
30   import com.ardic.android.sampleiotigniteprojectwithdynamicconfigurations.R;
31   import
com.ardic.android.sampleiotigniteprojectwithdynamicconfigurations.constants.DynamicNodeC
onstants;
32
33   import org.json.JSONException;
34   import org.json.JSONObject;
35
36   import java.util.List;
37   import java.util.concurrent.CopyOnWriteArrayList;
38
39   public class HomeActivity extends AppCompatActivity implements
View.OnClickListener, WifiNodeManagerListener, CompatibilityListener {
40
41       private static final String TAG = "Dynamic Node App";
42       private static final String LED_TEXT_OFF = "OFF";
43       private static final String LED_TEXT_ON = "ON";
44       private static final String DEGREE = "\u00b0" + "C";
45       private static boolean versionError = false;
46       private boolean isActiveEspConnected = false;
47       private TextView tempText, humText, ledText, nodeIDText;
48       private ImageView ledImageView, socketStateImageView, deleteNodeImageView;
49
50       private GenericWifiNodeManager espManager;
51       private List<BaseWifiNodeDevice> espNodeList = new CopyOnWriteArrayList<>();
52       private BaseWifiNodeDevice activeEsp;
53       private ThingEventListener espEventListener = new ThingEventListener() {
54
55           // update connection state in all callbacks. Sometimes connection callback
could be triggered after other messages.
56           @Override
57           public void onDataReceived(final String s, final String s1, final
com.ardic.android.iotignite.things.ThingData thingData) {
58
59               Log.i(TAG, "onDataReceived [" + s + "][" + s1 + "][" +
thingData.getDataList() + "]");
60
61               if (activeEsp != null) {
62                   Log.i(TAG, " Socket : " +
activeEsp.getWifiNodeDevice().getNodeSocket());
63
64                   setConnectionState(true);
65                   runOnUiThread(new Runnable() {
66                       @Override
67                       public void run() {
68
```

```
setUINodeId(activeEsp.getWifiNodeDevice().getHolder().getNodeId());
                            float data = Float.valueOf(thingData.getDataList().get(0));
                            int data_int = (int) data;
                            if (DynamicNodeConstants.TEMPERATURE_SENSOR.equals(s1)) {
                                tempText.setText(data_int + DEGREE);

                            } else if (DynamicNodeConstants.HUMIDITY_SENSOR.equals(s1))
{
                                humText.setText(data_int + "%");
                            }

                        }

                    });
                }

            }

            @Override
            public void onConnectionStateChanged(final String s, final boolean b) {
                Log.i(TAG, "onConnectionStateChanged [" + s + "][" + b + "]");
                if (activeEsp != null) {
                    setConnectionState(b);
                }
            }

            @Override
            public void onActionReceived(String s, String s1, String s2) {
                Log.i(TAG, "onActionReceived [" + s + "][" + s1 + "][" + s2 + "]");
                if (activeEsp != null) {
                    setConnectionState(true);
                }
            }

            @Override
            public void onConfigReceived(String s, String s1,
com.ardic.android.iotignite.things.ThingConfiguration thingConfiguration) {
                Log.i(TAG, "onConfigReceived [" + s + "][" + s1 + "][" +
thingConfiguration.getDataReadingFrequency() + "]");

                if (activeEsp != null) {
                    setConnectionState(true);
                }
            }

            @Override
            public void onUnknownMessageReceived(String s, String s1) {
                Log.i(TAG, "onUnknownMessageReceived [" + s + "][" + s1 + "]");

                if (activeEsp != null) {
                    setConnectionState(true);
                    // receive syncronization message here. Message received here
becasue we set a custom message.

                    try {
                        JSONObject ledStateJson = new JSONObject(s1);
                        if (ledStateJson.has("ledState")) {
                            int ledState = ledStateJson.getInt("ledState");
                            setLedUI(ledState == 0 ? false : true);
                        }
                    } catch (JSONException e) {
                        Log.i(TAG, "JSONException on onUnknownMessageReceived() : " +
e);

                    }
                }
            }

            @Override
            public void onNodeUnregistered(String s) {
                Log.i(TAG, "onNodeUnregistered [" + s + "]");
                if (activeEsp != null) {
                    // sendResetMessage();
                    activeEsp.removeThingEventListener(espEventListener);

                    for (BaseWifiNodeDevice dvc : espNodeList) {
                        if (dvc.getNode().getNodeID() != null &&
s.equals(dvc.getNode().getNodeID())) {
```

```
140                             Log.i(TAG, "Removing [" + s + "]");
141
142                             if (espNodeList.indexOf(dvc) != -1) {
143                                 espNodeList.remove(espNodeList.indexOf(dvc));
144                                 // remove from list. Update UI.
145                                 if (s.equals(activeEsp.getNode().getNodeID())) {
146                                     Log.i(TAG, "Updating... [" + s + "]");
147
148                                     updateActiveEsp();
149                                 }
150
151                                 break;
152                             } else {
153                                 Log.i(TAG, "Fail to remove : [" + s + "]");
154                             }
155                         }
156                     }
157                 }
158             }
159
160             @Override
161             public void onThingUnregistered(String s, String s1) {
162                 Log.i(TAG, "onThingUnregistered [" + s + "][" + s1 + "]");
163             }
164         };
165
166
167         @Override
168         protected void onCreate(Bundle savedInstanceState) {
169             super.onCreate(savedInstanceState);
170             setContentView(R.layout.activity_main);
171             // Servis HwNodeAppTemplate içinde başlatılır.
172             startService(new Intent(this, WifiNodeService.class));
173             // Uyumluluk olay dinleyicisi tanımlanır.
174             WifiNodeService.setCompatibilityListener(this);
175             Log.i(TAG, "Dynamic Node Application started...");
176             // Kullanıcı arayüzü başlatılır.
177             initUIComponents();
178             initSensorDatas();
179             /* İletişimi sağlamak için Genric Device ve Generic Wifi Node yöneticisi
başlatılır.*/
180             initEspDeviceAndNodeManager();
181         }
182
183         @Override
184         protected void onDestroy() {
185             stopService(new Intent(this, WifiNodeService.class));
186             super.onDestroy();
187             versionError = false;
188         }
189
190         private void initUIComponents() {
191
192             // textviews
193             tempText = (TextView) findViewById(R.id.temperatureTextView);
194             humText = (TextView) findViewById(R.id.humidityTextView);
195             ledText = (TextView) findViewById(R.id.ledTextView);
196             nodeIDText = (TextView) findViewById(R.id.nodeIDTextView);
197
198             // image view of led for on/off
199             ledImageView = (ImageView) findViewById(R.id.ledImageView);
200             socketStateImageView = (ImageView) findViewById(R.id.igniteStatusImgView);
201             deleteNodeImageView = (ImageView) findViewById(R.id.deleteNodeImgView);
202             deleteNodeImageView.setOnClickListener(this);
203             ledImageView.setOnClickListener(this);
204         }
205
206         /**
207          * Called when a view has been clicked.
208          *
209          * @param v The view that was clicked.
210          */
211         @Override
212         public void onClick(View v) {
213
214
215             if (v.equals(ledImageView)) {
216                 //final Esp esp = getActiveEsp();
```

```
217
218             if (activeEsp != null && isActiveEspConnected) {
219                 new Thread(new Runnable() {
220                     @Override
221                     public void run() {
222
223                         // change the background
224                         if (LED_TEXT_ON.equals(ledText.getText().toString())) {
225                             // send led_off message here.
226                             Log.i(TAG, "Sending LED_OFF message..");
227
228                             if
(activeEsp.sendActionMessage(DynamicNodeConstants.ACTUATOR_BLUE_LED,
DynamicNodeConstants.LED_OFF_ACTION)) {
229                                 setLedUI(false);
230                             }
231                         } else if
(LED_TEXT_OFF.equals(ledText.getText().toString())) {
232                             // send led_on message here.
233                             Log.i(TAG, "Sending LED_ON message..");
234                             if
(activeEsp.sendActionMessage(DynamicNodeConstants.ACTUATOR_BLUE_LED,
DynamicNodeConstants.LED_ON_ACTION)) {
235                                 setLedUI(true);
236                             }
237                         }
238                     }
239
240
241                 }).start();
242             } else {
243                 if (activeEsp == null) {
244                     Log.i(TAG, "Active Esp is NULL ");
245                 } else {
246                     Toast.makeText(getApplicationContext(), "Esp -> [" +
activeEsp.getWifiNodeDevice().getHolder().getNodeId() + "] <- is disconnected!",
Toast.LENGTH_LONG).show();
247                 }
248             }
249
250         } else if (v.equals(deleteNodeImageView)) {
251             //delete active node here.
252             new AlertDialog.Builder(HomeActivity.this)
253                     .setTitle("Delete Node")
254                     .setMessage("Are you sure you want to delete this node?")
255                     .setPositiveButton(android.R.string.yes, new
DialogInterface.OnClickListener() {
256                         public void onClick(DialogInterface dialog, int which) {
257                             // continue with delete
258                             if (activeEsp != null && activeEsp.getNode() != null) {
259                                 sendResetMessage();
260                             } else {
261                                 Log.i(TAG, "There is no active esp!!! ");
262                                 showDeleteNodeErrorToast();
263                             }
264                         }
265                     })
266                     .setNegativeButton(android.R.string.no, new
DialogInterface.OnClickListener() {
267                         public void onClick(DialogInterface dialog, int which) {
268                             // do nothing
269                         }
270                     })
271                     .setIcon(android.R.drawable.ic_dialog_alert)
272                     .show();
273
274         }
275
276
277     }
278
279     public void setLedUI(final boolean state) {
280
281         runOnUiThread(new Runnable() {
282             @Override
283             public void run() {
284                 if (state) {
285                     ledImageView.setImageResource(R.mipmap.led2_on);
```

```
286                     ledText.setText(LED_TEXT_ON);
287                 } else {
288                     ledImageView.setImageResource(R.mipmap.led2_off);
289                     ledText.setText(LED_TEXT_OFF);
290                 }
291             }
292         });
293 
294     }
295 
296     public void setConnectionState(final boolean state) {
297 
298         runOnUiThread(new Runnable() {
299             @Override
300             public void run() {
301                 if (state) {
302                     socketStateImageView.setImageResource(R.drawable.connected);
303                     isActiveEspConnected = true;
304                 } else {
305                     socketStateImageView.setImageResource(R.drawable.disconnected);
306                     isActiveEspConnected = false;
307                 }
308             }
309         });
310 
311     }
312 
313     @Override
314     public boolean onCreateOptionsMenu(Menu menu) {
315         getMenuInflater().inflate(R.menu.main_menu, menu);
316         return true;
317     }
318 
319     @Override
320     public boolean onPrepareOptionsMenu(Menu menu) {
321 
322         menu.clear();
323         getMenuInflater().inflate(R.menu.main_menu, menu);
324 
325         for (GenericWifiNodeDevice dvc : espManager.getWifiNodeDeviceList()) {
326             checkAndUpdateDeviceList(dvc);
327         }
328 
329         if (!espNodeList.isEmpty()) {
330             Log.i(TAG, "EspNode List Size : " + espNodeList.size());
331             for (BaseWifiNodeDevice esp : espNodeList) {
332                 Log.i(TAG, "Esp : " +
esp.getWifiNodeDevice().getHolder().getNodeId());
333                 if (esp.getWifiNodeDevice().getHolder().getNodeId() != null && !
TextUtils.isEmpty(esp.getWifiNodeDevice().getHolder().getNodeId())) {
334                     menu.add(Menu.NONE, Menu.NONE, Menu.NONE,
esp.getWifiNodeDevice().getHolder().getNodeId());
335                 }
336             }
337         } else {
338             Log.i(TAG, "EspNode List EMPTY");
339 
340         }
341         return super.onPrepareOptionsMenu(menu);
342     }
343 
344     @Override
345     public boolean onOptionsItemSelected(MenuItem item) {
346 
347         for (BaseWifiNodeDevice e : espNodeList) {
348             if (!TextUtils.isEmpty(item.getTitle()) && e.getNode() != null && !
TextUtils.isEmpty(e.getNode().getNodeID()) &&
item.getTitle().equals(e.getNode().getNodeID())) {
349                 activeEsp.removeThingEventListener(espEventListener);
350                 activeEsp = e;
351                 updateActiveEsp();
352                 break;
353             }
354         }
355         return super.onOptionsItemSelected(item);
356     }
357 
358 
```

```
359       private void initSensorDatas() {
360           runOnUiThread(new Runnable() {
361               @Override
362               public void run() {
363                   tempText.setText(DEGREE);
364                   humText.setText("%");
365                   setLedUI(false);
366                   setConnectionState(false);
367                   deleteNodeImageView.setVisibility(View.INVISIBLE);
368                   nodeIDText.setText(R.string.no_available_node);
369               }
370           });
371       }
372
373       @Override
374       public void onWifiNodeDeviceAdded(BaseWifiNodeDevice baseWifiNodeDevice) {
375           Log.i(TAG, ">>>>>>>>>> DEVICE ADDED <<<<<<<<<<");
376           checkAndUpdateDeviceList(baseWifiNodeDevice);
377       }
378
379       @Override
380       public void
onUnsupportedVersionExceptionReceived(com.ardic.android.iotignite.exceptions.Unsupported
VersionException e) {
381           Log.i(TAG, "Ignite onUnsupportedVersionExceptionReceived :  " + e);
382           if (!versionError) {
383               versionError = true;
384               if
(UnsupportedVersionExceptionType.UNSUPPORTED_IOTIGNITE_AGENT_VERSION.toString().equals(e
.getMessage())) {
385                   Log.e(TAG, "UNSUPPORTED_IOTIGNITE_AGENT_VERSION");
386                   showAgentInstallationDialog();
387               } else {
388                   Log.e(TAG, "UNSUPPORTED_IOTIGNITE_SDK_VERSION");
389                   showAppInstallationDialog();
390               }
391
392           }
393       }
394
395       @Override
396       public void onIgniteConnectionChanged(boolean b) {
397           Log.i(TAG, "Ignite Connection State Changed To -> " + b);
398       }
399
400
401       private void initEspDeviceAndNodeManager() {
402           espManager = GenericWifiNodeManager.getInstance(getApplicationContext());
403           // Ağda tespit edilen cihaz alınır ve işlemler yapılır.
404           espManager.addWifiNodeManagerListener(this);
405
406           for (GenericWifiNodeDevice dvc : espManager.getWifiNodeDeviceList()) {
407               checkAndUpdateDeviceList(dvc);
408           }
409
410       }
411
412       private void checkAndUpdateDeviceList(BaseWifiNodeDevice device) {
413           if
(DynamicNodeConstants.TYPE.equals(device.getWifiNodeDevice().getNodeType())) {
414
415               if (!espNodeList.contains(device)) {
416                   Log.i(TAG, "New node found adding to list.");
417                   espNodeList.add(device);
418               } else {
419                   Log.i(TAG, "New node already in list.Updating...");
420                   if (espNodeList.indexOf(device) != -1) {
421                       Log.i(TAG, "New node already in list.Removing...");
422                       espNodeList.remove(espNodeList.indexOf(device));
423                   }
424                   espNodeList.add(device);
425               }
426           }
427
428           updateActiveEsp();
429       }
430
431
```

```
432       private void showAgentInstallationDialog() {
433           runOnUiThread(new Runnable() {
434               @Override
435               public void run() {
436                   new MaterialDialog.Builder(HomeActivity.this)
437                           .title("Confirm")
438                           .content("Your IoT Ignite Agent version is out of date!
Install the latest version?")
439                           .positiveText("Agree")
440                           .negativeText("Disagree")
441                           .onPositive(new MaterialDialog.SingleButtonCallback() {
442                               @Override
443                               public void onClick(@NonNull MaterialDialog dialog,
@NonNull DialogAction which) {
444                                   openUrl("http://iotapp.link/");
445                               }
446                           })
447                           .show();
448               }
449           });
450       }
451
452       private void showAppInstallationDialog() {
453           runOnUiThread(new Runnable() {
454               @Override
455               public void run() {
456                   new MaterialDialog.Builder(HomeActivity.this)
457                           .title("Confirm")
458                           .content("Your Demo App is out of date! Install the latest
version?")
459                           .positiveText("Agree")
460                           .negativeText("Disagree")
461                           .onPositive(new MaterialDialog.SingleButtonCallback() {
462                               @Override
463                               public void onClick(@NonNull MaterialDialog dialog,
@NonNull DialogAction which) {
464                                   openUrl("https://download.iot-
ignite.com/DynamicNodeExample/");
465                               }
466                           })
467                           .show();
468               }
469           });
470       }
471
472       private void openUrl(String url) {
473           Intent i = new Intent(Intent.ACTION_VIEW);
474           i.setData(Uri.parse(url));
475           try {
476               startActivity(i);
477           } catch (ActivityNotFoundException e) {
478               showDialog("Browser could not opened!");
479           }
480       }
481
482       private void showDialog(String message) {
483           new MaterialDialog.Builder(HomeActivity.this)
484                   .content(message)
485                   .neutralText("Ok")
486                   .show();
487       }
488
489       private void setUINodeId(final String nodeId) {
490           runOnUiThread(new Runnable() {
491               @Override
492               public void run() {
493                   nodeIDText.setText(nodeId);
494                   deleteNodeImageView.setVisibility(View.VISIBLE);
495               }
496           });
497       }
498
499       private void sendResetMessage() {
500           runOnUiThread(new Runnable() {
501               @Override
502               public void run() {
503                   if (activeEsp != null) {
504                       updateActiveEsp();
```

```
505                     }
506                 }
507             });
508         }
509
510         private void showDeleteNodeErrorToast() {
511             runOnUiThread(new Runnable() {
512                 @Override
513                 public void run() {
514                     Toast.makeText(getApplicationContext(), "There is no active esp to
delete.", Toast.LENGTH_LONG).show();
515                 }
516             });
517         }
518
519         private void updateActiveEsp() {
520             // if only one device found set this device active.
521             if (espNodeList.size() > 0) {
522                 Log.i(TAG, " NODE LIST SIZE :" + espNodeList.size());
523
524
525                 if (espNodeList.size() == 1) {
526                     activeEsp = espNodeList.get(0);
527                 }
528
529                 if (activeEsp.getNode() != null) {
530                     activeEsp.addThingEventListener(espEventListener);
531                     setUINodeId(activeEsp.getNode().getNodeID());
532                     isActiveEspConnected = activeEsp.getNode().isConnected();
533                     setConnectionState(isActiveEspConnected);
534                 } else {
535                     Log.i(TAG, "ACTIVE NODE IS NULL");
536                 }
537             } else {
538                 initSensorDatas();
539                 activeEsp = null;
540             }
541         }
542     }
543
```

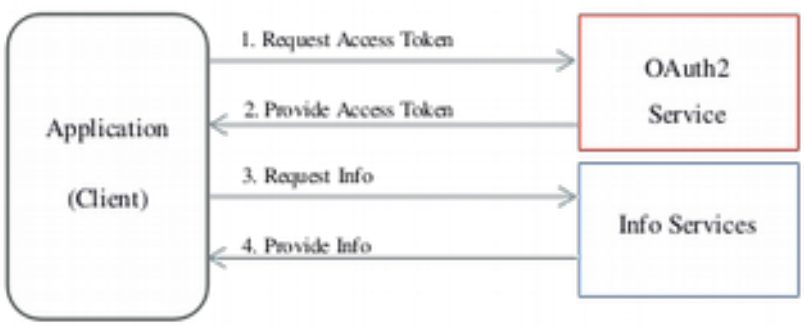
BÖLÜM 8

Ignite Cloud Katmanı

IoT-Ignite Cloud API

IoT-Ignite Servisleri'nde, web hizmetleri ve istemciler arasında kimlik doğrulama sağlamak için **OAuth2** kullanılmaktadır. Web hizmetleri kimlik doğrulama için erişim izni gerektirir. OAuth2 yetkilendirme Framework'u, bir üçüncü taraf uygulamanın, bir kaynak sahibi adına kaynak sahibi ve HTTP hizmeti arasındaki onay etkileşimini orkestrasyon yoluyla veya bir üçüncü kişinin uygulamaya izin vererek bir HTTP hizmetine sınırlı erişim elde etmesini sağlar.

IoT-Ignite API'lerini kullanabilmek için OAuth2 ile yetki alınması gerekmektedir. Yetkilendirmek akışı, aşağıdaki şekilde gösterilmiştir.



Akışta numaralandırılmış işlemlerde;

1. Uygulama, kullanıcının kimliği (e-posta ve parola) sağlayarak OAuth2 hizmetinden erişim izni talep eder.

2. Verilen kimlik geçerli ise, OAuth2 Server belirli bir geçerlilik süresi olan bir uygulamaya access token (erişim anahtarı) gönderir.

3. Uygulama, son kullanma tarihine kadar OAuth2 tarafından verilen erişim anahtarı ile IoT-Ignite Servisleri'nden API'ler ile etkileşime girer (GET, POST, PUT, DELETE).

4. Eğer erişim anahtarı doğru ve geçerlilik süresi dolmamışsa API'ler istenilen veriyi geri döndürecektir.

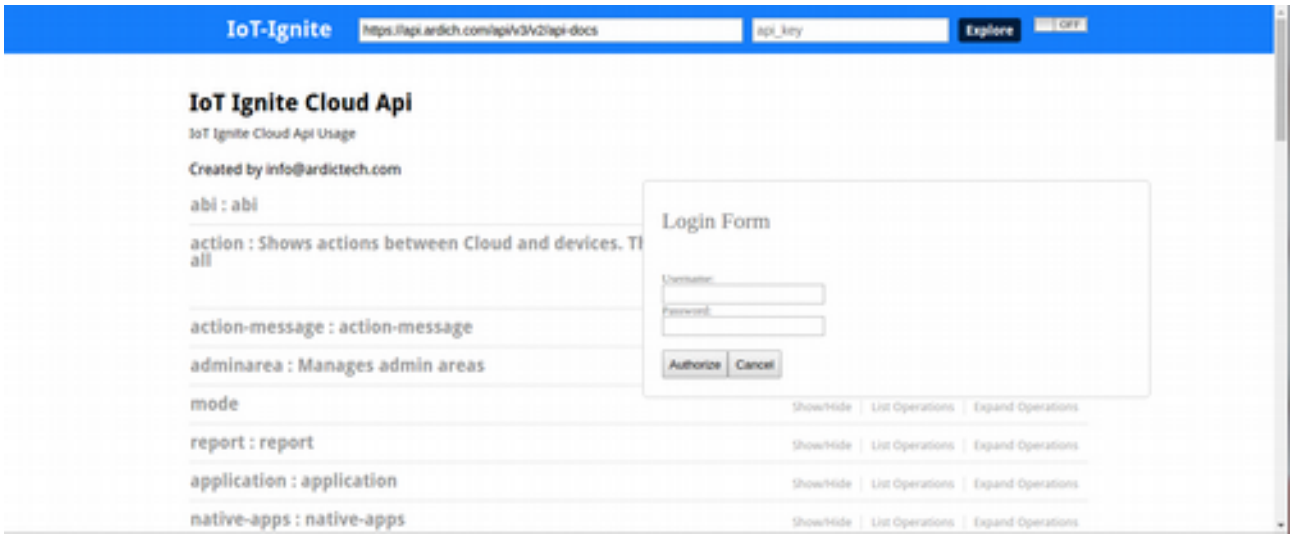
Swagger ile IoT-Ignite API'lerinin Kullanımı

Swagger, REST API'leri tek bir çatı altında toplayan ve geliştiricilerin kolaylıkla kullanabilmelerini sağlayan arayüzleri içeren bir araçtır. IoT-Ignite API'leri Swagger uyumludur. Swagger'da IoT-Ignite API'lerini çeşitli gruplar altında inceleyebilir ve bu API'lerin hangi parametreler ile **GET**, **POST**, **DELETE** ve **PUT** metotları ile çalıştığını görebilirsiniz.

> Swagger, Rest API geliştirmek için gerekli bir sözleşme standardı ve bu çerçevede işlev gören yardımcı araçlar sunan bir teknolojidir. Swagger sunduğu standart ve araçlarla API tasarım, geliştirme, dokümantasyon ve test aşamasında kolaylık sağlamaktadır.

IoT-Ignite Swagger sayfasına girmek için aşağıdaki adrese giriş yapın.

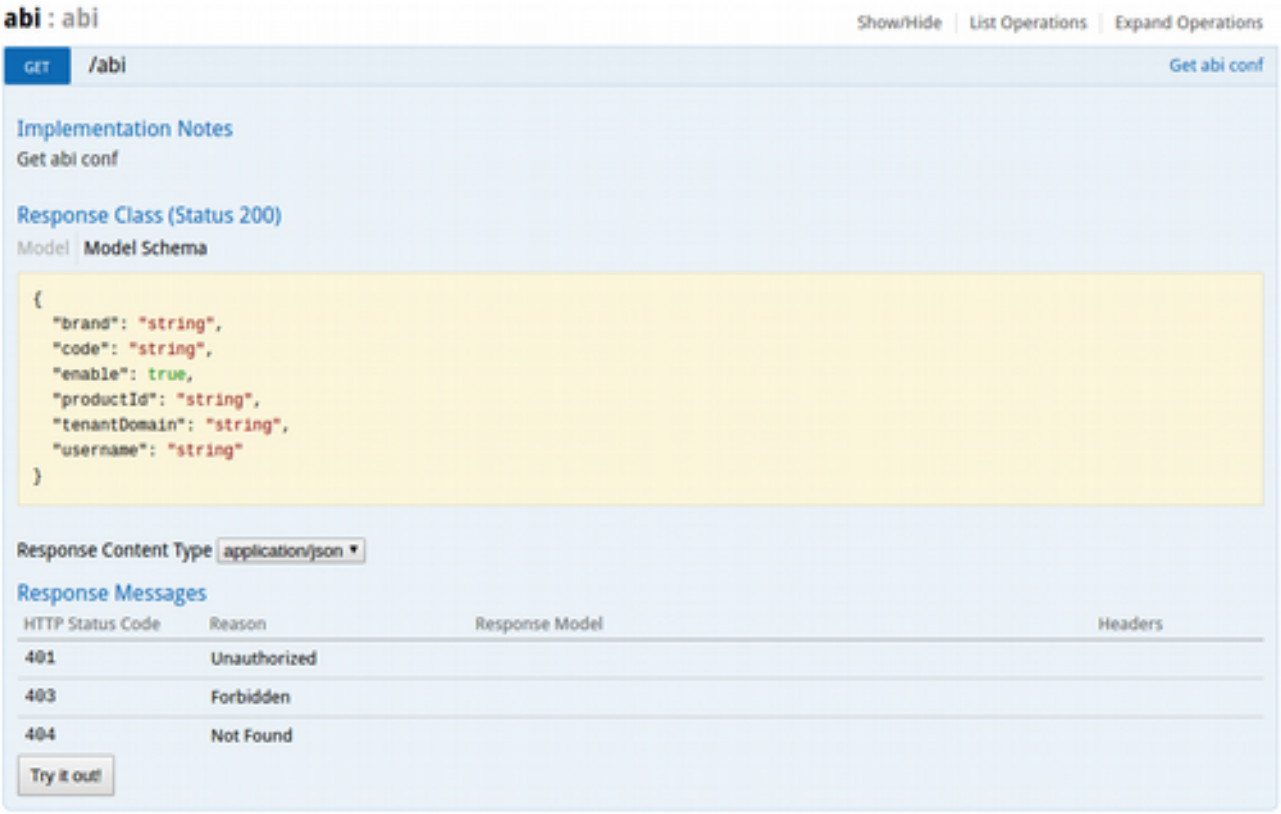
https://api.ardich.com/api/v3/apidoc/



Sayfaya girdiğinizde sağ üst köşede bir **ON/OFF** butonu göreceksiniz. Bu butona tıklayın ve açılan arayüzden Devzone'da kullandığınız e-posta ve parola bilgilerinizi girip **Authorize** (Yetki Ver) butonuna tıklayın. Giriş işlemi başarılı bir şekilde gerçekleştiğinde, sağ üstteki ON/OFF butonu mavi renk ile ON konumuna gelecektir. Bundan sonra her bir API'yı yetkilendirilmiş olarak kullanabileceksiniz.

Şimdi birkaç API seçerek nasıl kullanacağımıza bakalım...

Örneğin **abi** grubuna tıklayıp **GET** metodu ile **/abi** API'sini inceleyelim...



Model Schema (Şema) bakıldığında bu API'den dönen verinin **JSON** olduğu, içinde de **brand**, **code**, **enable**, **productId**, **tenantDomain** ve **username** bilgilerinin döndüğü; dönen bu bilgilerin de

veri formatları gösterilmektedir. Olası response'ların **HTTP** kodlarının da **401** (Yetkilendirilmemiş), **403** (İzin yok) ve **404** (Bulunamadı) olabileceği belirtilmiş.

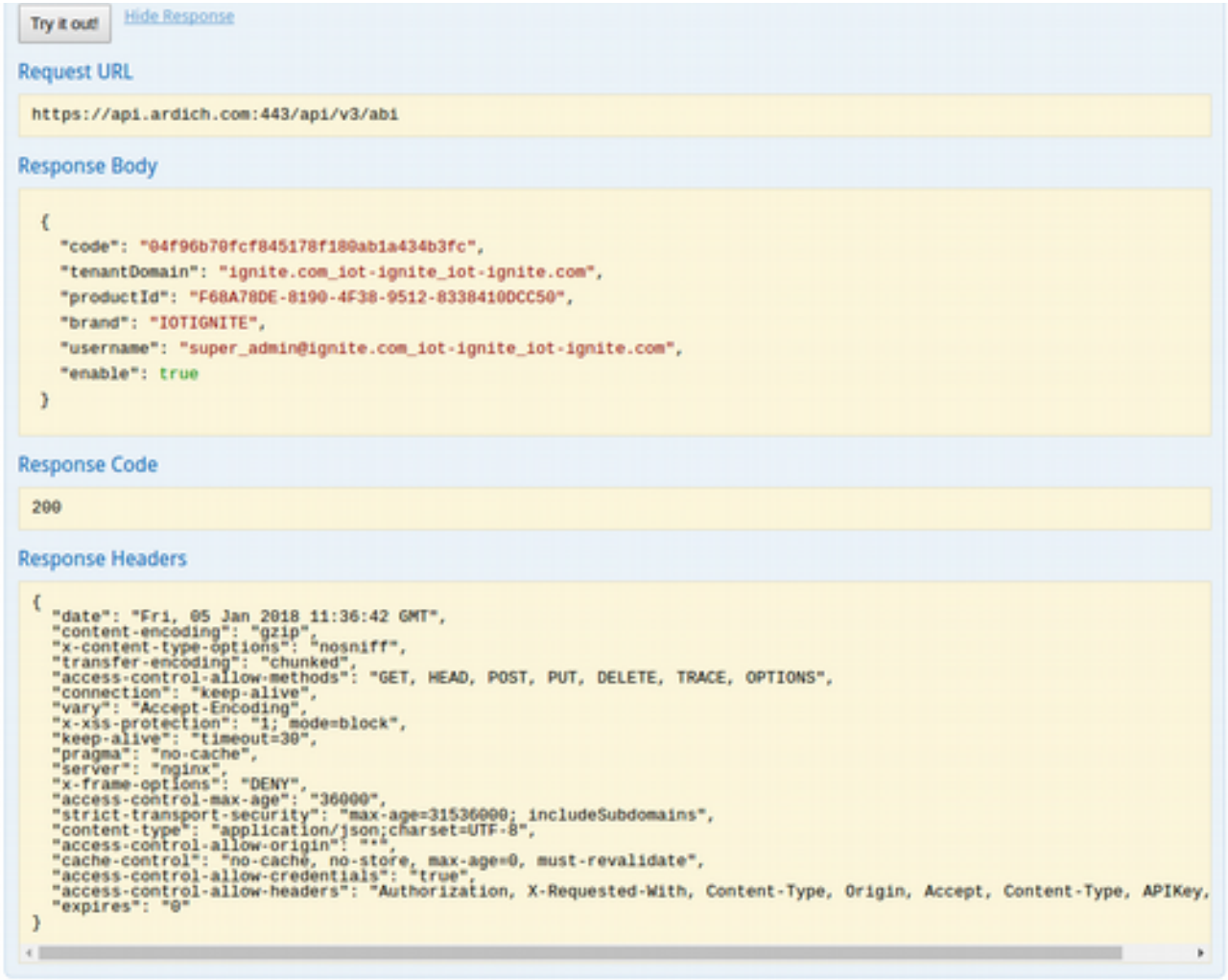
Try it out! (Dene) butonuna tıklayın.

Şekilde görüldüğü üzere bir takım bilgiler getirildi. İlk sırada **Request URL** ile hangi adrese istekte yapıldığı gösteriliyor.

İkinci satırda da **Response Body**'de gelen veri gösteriliyor. Model şemasında gösterildiği formatta bilgiler getirildi.

Bir sonraki satırda ise **Response Code** (HTTP dönüş kodu) **200** (Success, Başarılı) olarak geldiği gösterilmektedir.

Son olarak da **Response Header** alanında diğer bilgiler yer almaktadır.



Şimdi de yine **/abi** API'sini **PUT** metodu ile inceleyelim.

PUT metodu, var olan bir değeri değiştirmek (override, update) için kullanılır. PUT olarak kullanmak için **abi/brand** API'sine istek gönderebileceğimiz belirtilmiş. Model şemasına baktığımızda yine GET'te olduğu gibi bir takım bilgiler JSON içerisinde isteniyor. Bu bilgileri de **resource** (kaynak) alanına girip PUT edebiliriz. **Resource** alanının sağ tarafında **body** kısmında örnek bir kullanım yer almaktadır.

Örneğe göre brand, yani servis ismini güncelleyelim. Şu an brand **IOTIGNITE** olarak görülmektedir.

Source alanına JSON içinde brand ismini yazıp Try it out! Butonuna tıklayın.

> **body**'de yer alan sarı kutuya tıkladığınızda, parametre değerleri doğrudan **resource** alanına kopyalanacaktır. Kopyalanan şema üzerinde düzenleme yapmak daha kolaydır.

Görüldüğü üzere **IOTIGNITE** olan brand ismini **IOT-IGNITE** olarak güncellemek istedik. Response dönüşü olduğunda aşağıdaki gibi güncel bilgiler gelecektir.

Request URL

```
https://api.ardich.com:443/api/v3/abi/brand
```

Response Body


```
{
  "code": "04f96b70fcf845178f180ab1a434b3fc",
  "tenantDomain": "ignite.com_iot-ignite_iot-ignite.com",
  "productId": "F68A78DE-8190-4F38-9512-8338410DCC50",
  "brand": "IOT-IGNITE",
  "username": "super_admin@ignite.com_iot-ignite_iot-ignite.com",
  "enable": true
}
```


Response Code

```
200
```

DELETE ve **POST** metotları için bu defa cep grubunda yer alan metotları kullanalım. Örnek olması amacı ile hesabımızda var olan Rule'lardan (Rule'lar, bir diğer ismi olarak **Flow** olarak da kullanılmaktadır) birini pasif hale getirip, sonrasında da aynı Rule'u silelim.

Bir Flow'u, yani Rule'u silebilmek için o Rule'un id'sini bilmemiz gerekiyor. Rule'ları ilk olarak **GET** metodu ile listeleyelim...

cep/flow API'sini **GET** metodunu kullanarak çağıralım. Aşağıdaki şekilde görüldüğü gibi bir response geldi. JSON'da **list** key'i altında birer **ARRAY** içinde her bir flow, **JSON** elemanı olarak sıralanmış. Örneğimizde **flowName**'i "**Cloud, Temperature, More Than, 80, Message**" isimli Rule'u kullanacağız. Bu Rule'un da id'si **e0dec3dd-6ba8-4625-8118-5e4983cc5d44** olarak tanımlanmış, referans olarak bunu kullanacağız.

Request URL

```
https://api.ardich.com:443/api/v3/cep/flow
```

Response Body


```
  "error": null,
  "extras": {
    "list": [
      {
        "createDate": 1514374524000,
        "enabled": true,
        "flowName": "Twitter Mesajı",
        "id": "d0ffda49-1066-43df-b773-466e1e505839",
        "modifyDate": 1514374524000,
        "tenantDomain": "ignite.com_iot-ignite_iot-ignite.com",
        "createdBy": "3763",
        "modifiedBy": null,
        "flowDescription": "Humidity 50'den büyükse..."
      },
      {
        "createDate": 1513593292000,
        "enabled": true,
        "flowName": "Cloud, Temperature, More Than, 80, Message",
        "id": "e0dec3dd-6ba8-4625-8118-5e4983cc5d44",
        "modifyDate": 1513593292000,
```


Id'sini aldığımı bu Rule'u pasif hale getirmek için **/cep/flow/{flowId}/disable** API'sini kullanacağız. Yani API URL'İ **/cep/flow/e0dec3dd-6ba8-4625-8118-5e4983cc5d44/disable** olacaktır.

API kullanım arayüzünde flowId alanına id değerini yazıp **Try it out!** butonuna tıklayalım. İşlem sonucu olarak aşağıdaki gibi bir response dönecektir.

Acaba gerçekten de bu Rule pasif oldu mu diye kontrol etmek için **Devzone**'dan **Rules** sayfasına girip kontrol edelim.

Son olarak **DELETE** ile bu pasif hale getirdiğimiz Rule'u silelim. Bunun için **/cep/flow/{flowId}** API'ını kullanacağız.

Görüldüğü üzere **Parameters** (Parametreler) alanında **flowId** bulunmaktadır. Bu alana, daha önce edinmiş olduğumuz Rule'un id'sini yapıştırıp **Try it out!** butonuna tıklayın.

İşlem sonunda Rule silinecektir.

IoT-Ignite Cloud API Referans (Swagger) Listesi ve Postman ile Kullanımı

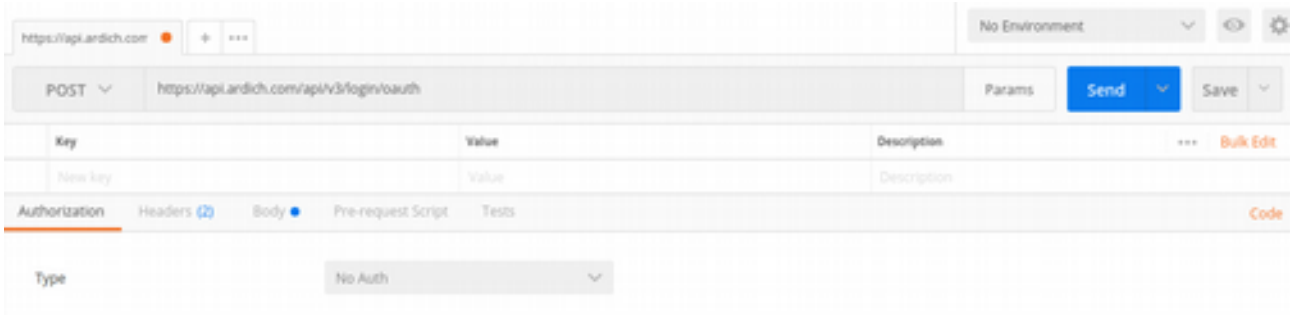
IoT-Ignite API'larını Swagger arayüzünden kullanmak yerine isterseniz Postman ile de kullanabilirsiniz. Postman, Chrome uzantısı olarak çalışan bir API Test yazılımıdır.

Örnek olması için **Login** işlemini Postman ile yapalım.

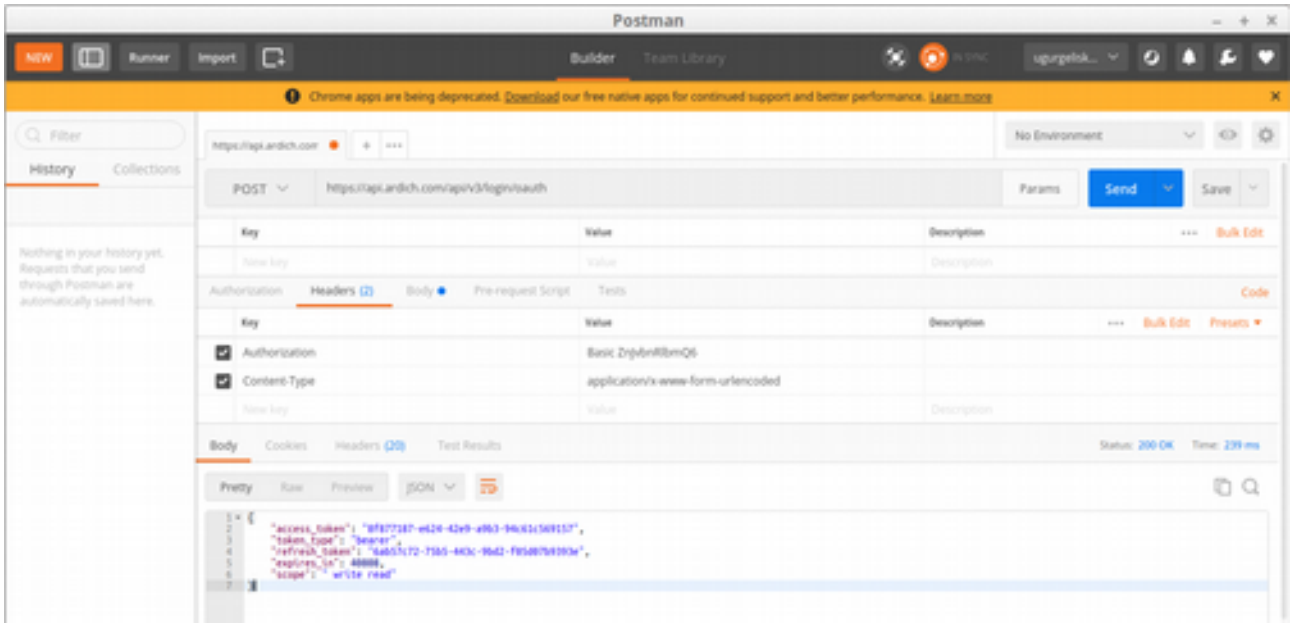
API Yetkilendirme

Postman'de **POST** metodunu kullanarak **https://api.ardich.com/api/v3/login/oauth** API URL'ine kullanıcı adı ve parolamızı göndererek Access Token alalım.

Authorization (Yetki) sekmesinde **Type**'ı (Tür) **No Auth** (Kimliksiz) olarak bırakabilirsiniz.

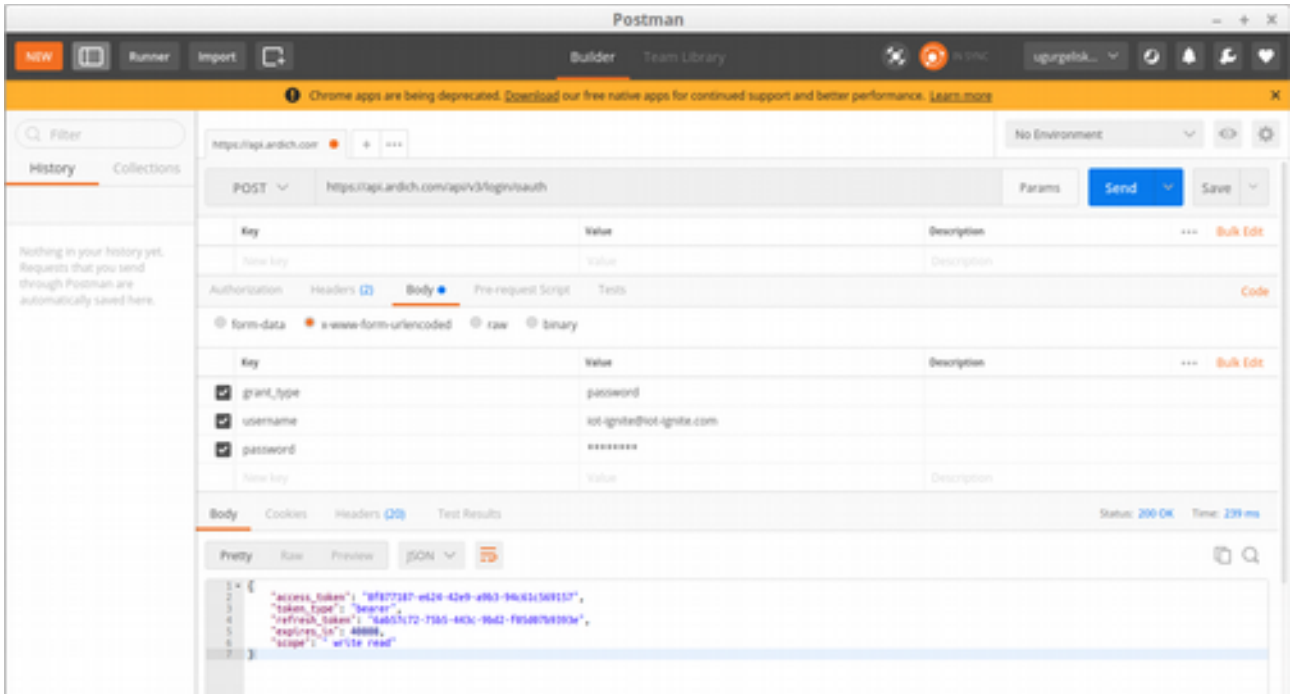


Headers (Başlıklar) sekmesinde API'nin header'ına 2 adet parametre eklenmesi gerekmektedir. Bunlar aşağıdaki şekilde görüldüğü üzere **Authorization** (Basic Og==) (Og== base64 ile : karakterinden gelmektedir) ve **Content-Type** (application/x-www-form-urlencoded) key'leridir.



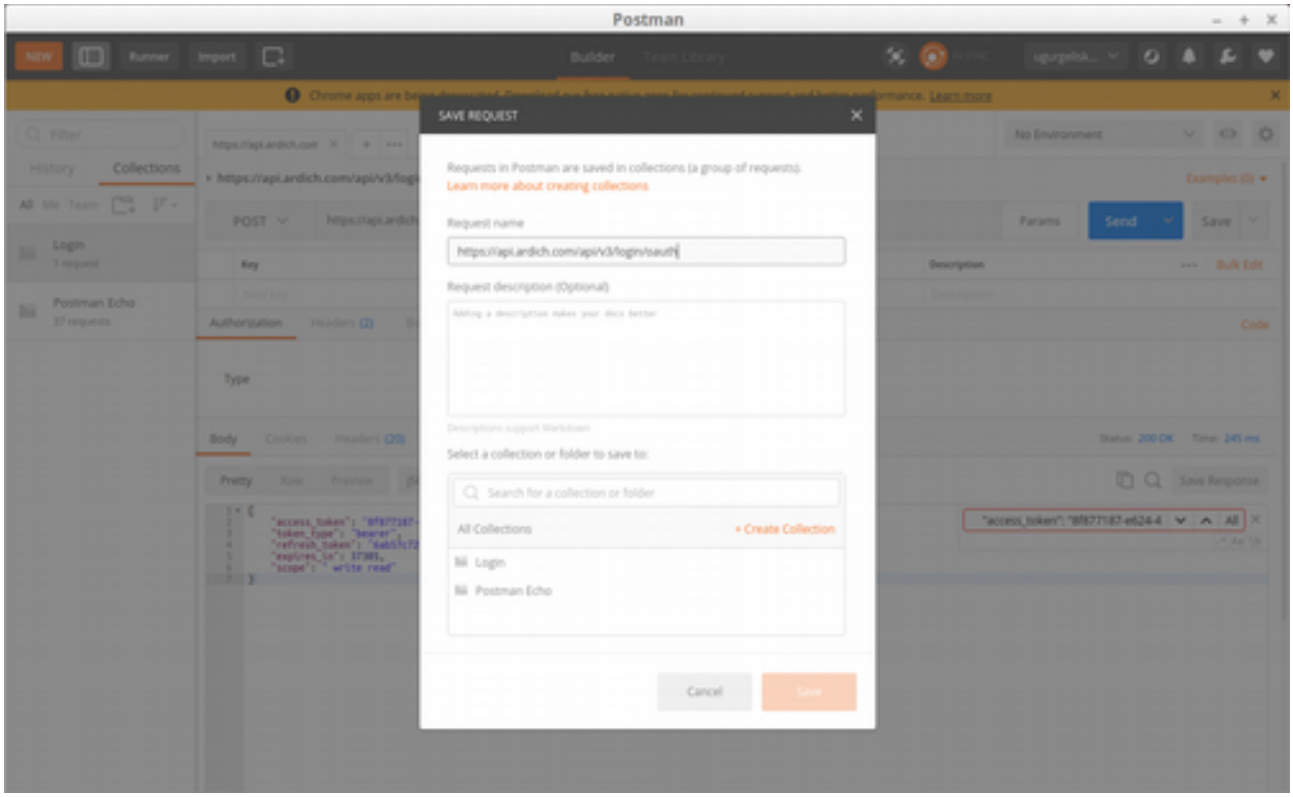
Body (Gövde) sekmesinde de 3 adet daha parametre eklenecek. Bunlar **POST** metodunda **data** içinde **JSON** olarak yer alacak. **grant_type: password**, **username**: e-posta adresiniz ve **password**: parola olacak şekilde **Key-Value** değerlerini tanımlayın. Türünü de **x-www-form-urlencoded** olarak seçin.

Son olarak **Send** (Gönder) butonuna tıklayın. İşlem sonucunda aşağıda görüldüğü üzere **JSON** içinde **access_token**, **token_type**, **refresh_token**, **expire_in** ve **scope** değerleri geldi.



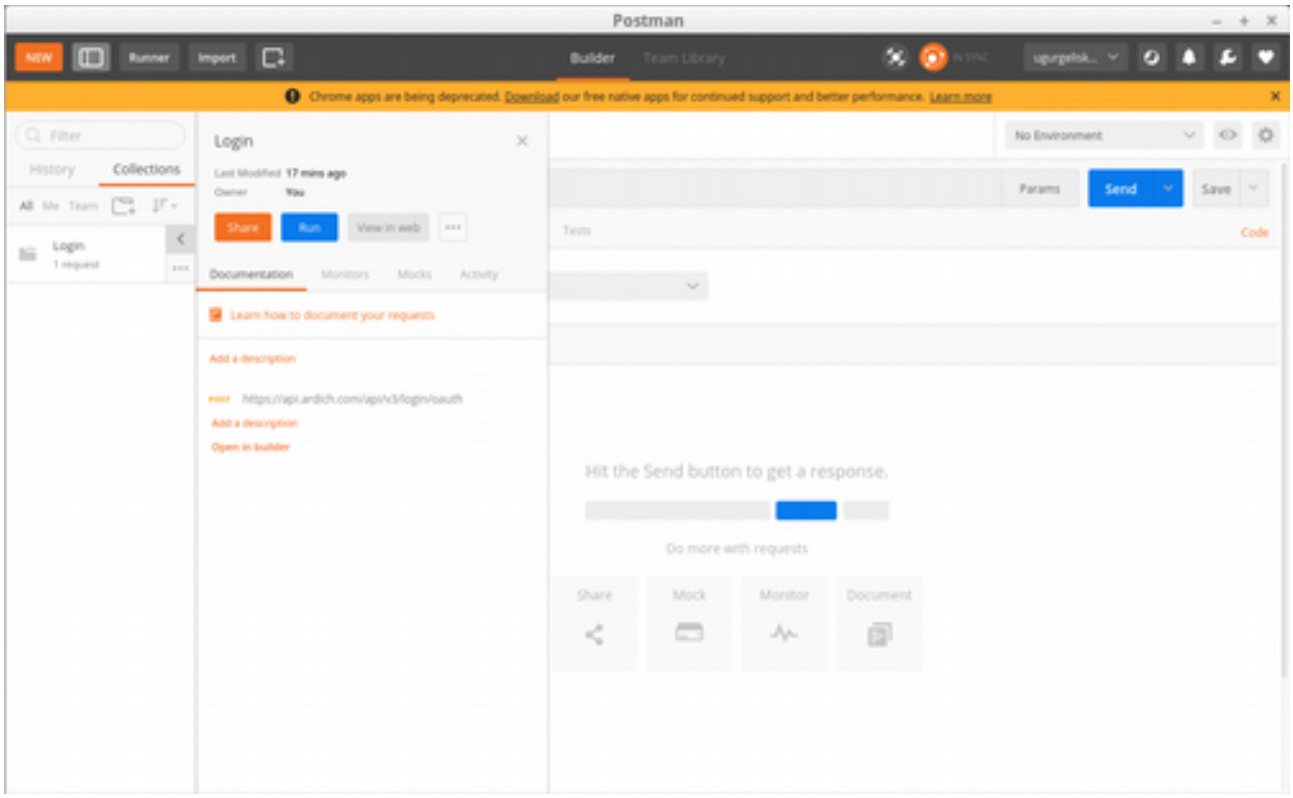
Send butonun sağ tarafında yer alan **Save** (Kaydet) butonuna tıklayarak **Login** yapılandırmanızı kaydedebilirsiniz. Böylece daha sonra tekrar bu ayarları yapmanıza gerek kalmaz ve tek tıklama ile yetkilendirme işlemini başlatabilirsiniz.

Save butonuna tıkladıktan sonra açılan **SAVE REQUEST** (İSTEĞİ KAYDET) penceresinde alt kısımda **Create Collection** (Koleksiyon Oluştur) butonuna tıklayıp, yapılandırma için bir isim tanımlayıp (Login ismini verdik) **Save**'e tıklayın.



İşlem sonunda Postman'ın sol tarafında yer alan **Collections** (Koleksiyonlar) sekmesinde kayıt

işlemini yaparken verdiğimiz isim ile API çağrımız görünecektir. O kutucuğun sağ tarafındaki ok simgesine tıklayıp açılan arayüzden de **Open in builder (Yapıcıda Aç)** butonuna tıklayın.



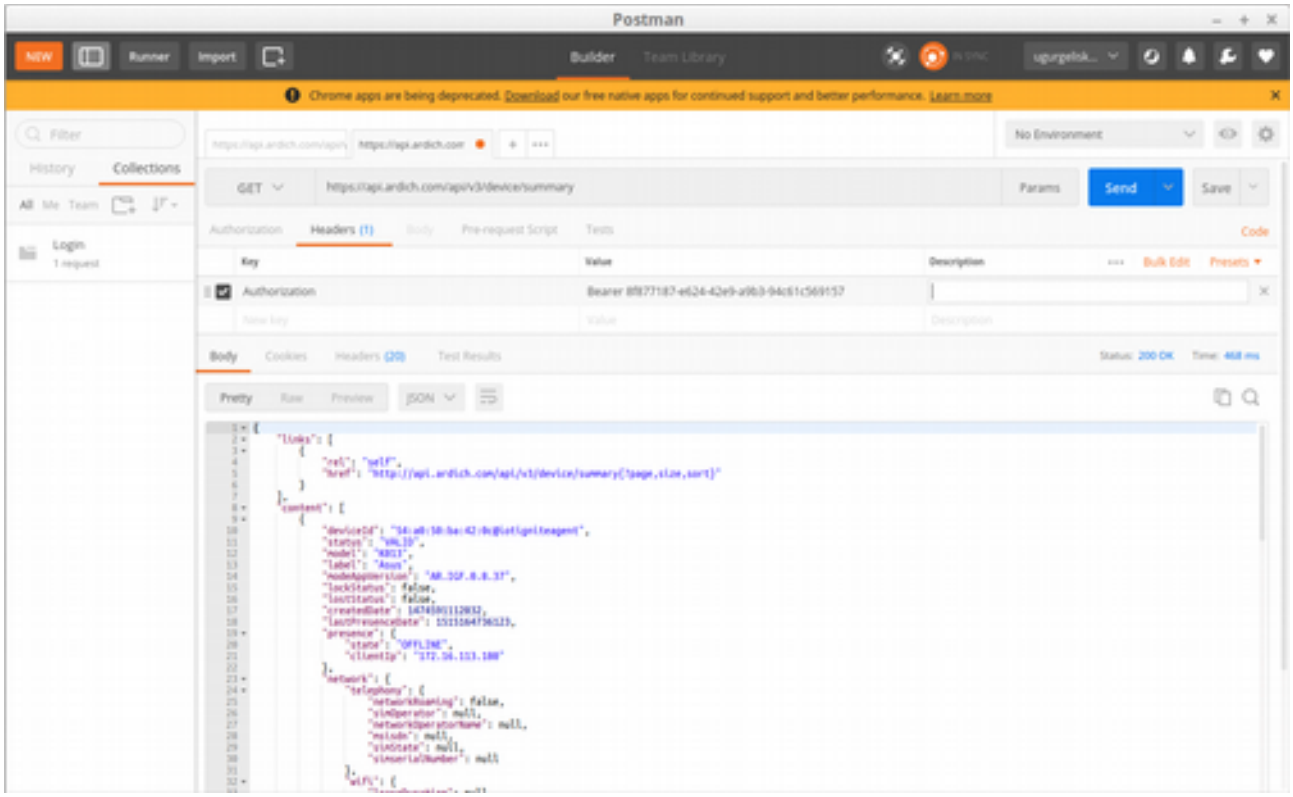
Bu işlem sonucunda yapılandırma ayarlarımız hazır olacak karşımıza çıkacak ve tekrar **Send** butonuna tıklayarak istekte bulunabileceğiz.

IoT-Ignite API'ları **OAuth 2.0** çalışmaktadır. Bu nedenle yetkilendirme için **access_token** gerekmektedir. Login işlemi ile aldığımız access_token'ı kullanarak yetki sağlayıp diğer API'ları kullanabiliriz.

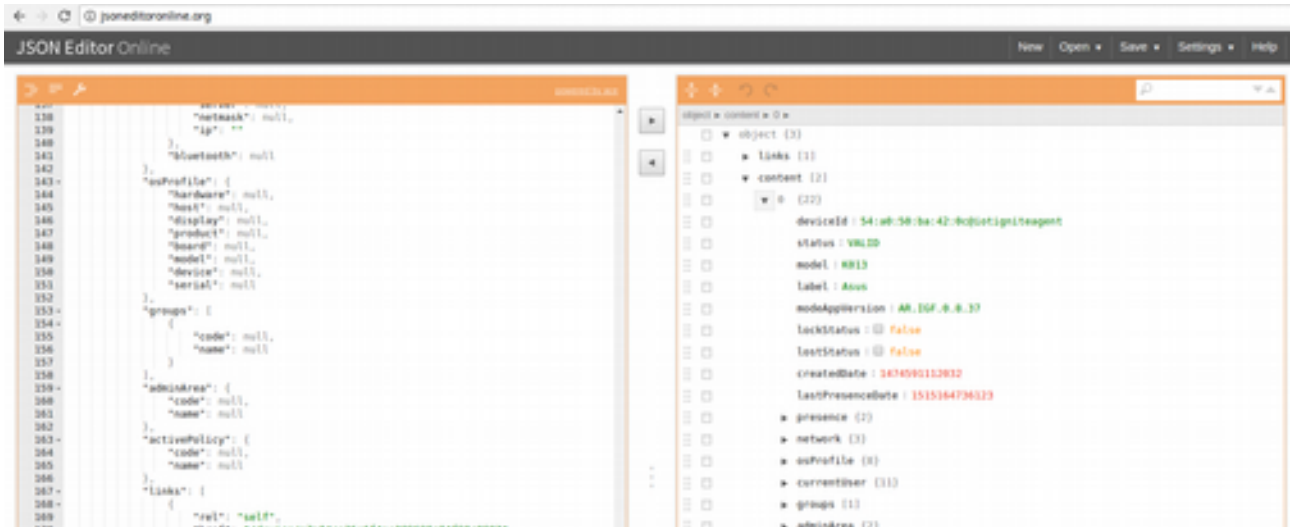
Gateway Bilgilerini Çekmek

Örnek olması amacıyla servisimize ait Gateway'lerin özet bilgisini çekelim. Kullanacağımız API URL'i /**device/summary**.

Üst kısımda yer alan **[+]** butonuna tıklayarak yeni bir sekme daha açalım. **GET** metodu ile `https://api.ardich.com/api/v3/device/summary` API URL'ini çağıralım. **Headers** sekmesinde **Authorization** Key'i ve **Bearer {access_token}** olacak şekilde Value'yu tanımlayın. Ardından **Send** butonu ile istekte bulunun. İşlem sonucunda Gateway'ler ile ilgili özet bilgisini veren uzunca bir JSON dönecektir. İsterseniz bu isteği yine yeni bir koleksiyon olarak kaydedebilirsiniz. Ancak Access Token'ın geçerlilik süresi dolduğu zaman tekrar Value'yu düzenlemeniz gerekecektir.



Uzun JSON'ları daha rahat görebilmek için *jsoneditoronline.org*'tan faydalanabilirsiniz. Sol tarafa JSON'u yapıştırıp, sağ tarafta da tree (ağaç) yapısı ile daha rahat bir şekilde görüntüleme sağlayabilirsiniz.



CURL ile IoT-Ignite API'larını Kullanma

Swagger ve Postman dışında IoT-Ignite API'larını CURL (Client URL) ile de kullanabilirsiniz. CURL kullanabilmek için Command Prompt (Windows) / Terminal (Linux) üzerinde sorguları yazarak yanıt olarak geleni anında görebilirsiniz.

CURL Kurulumu

CURL kullanabilmek için ilk olarak CURL'ın sisteminize tanıtılmış olması gerekmektedir.

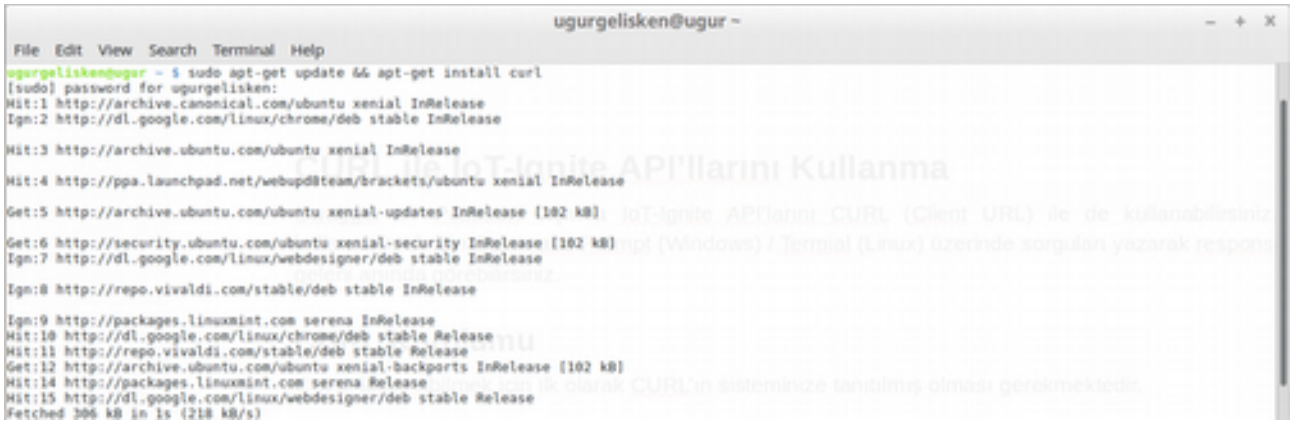
Windows için; **https://curl.haxx.se/download.html** adresine girip Windows için geçerli olan paketi indirip kurabilirsiniz.

Veya **curl.exe**'yi indirip (x86 veya x64) Windows>System32\ dizinine atmanız da yeterlidir.

Linux için ise aşağıdaki komutu Terminal'e yazıp yükleme yapabilirsiniz.

```
sudo apt-get update && apt-get install curl
```

Örneklerimizi **Linux** (Mint) üzerinden yaparak devam edeceğiz.



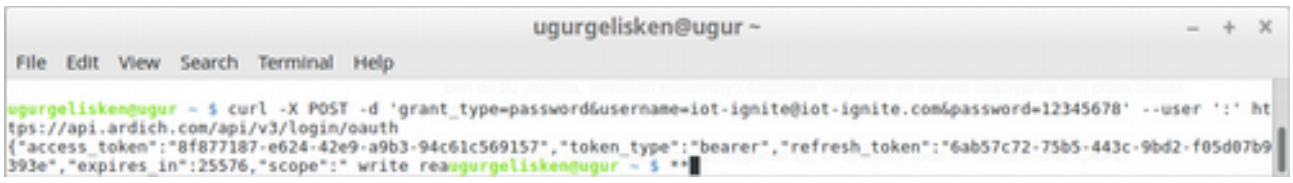
CURL artık sistem tarafından tanındığına göre birkaç sorgu yazalım...

Erişim Anahtarı Alma

İlk olarak CURL ile Access Token alalım.

```
curl -X POST -d 'grant_type=password&username={e-posta}&password={parola}'
--user ':' https://api.ardich.com/api/v3/login/oauth
```

{e-posta} yerine Devzone hesabınızdaki e-posta adresiniz, {parola} kısmına da giriş için kullandığınız parola yazılacaktır.



Görüldüğü üzere aşağıdaki gibi JSON yanıtı alındı.

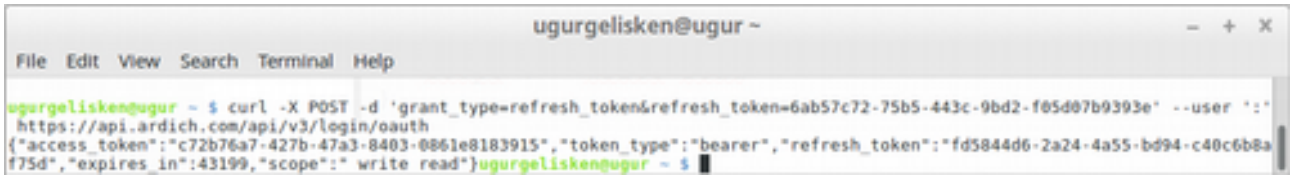
```
{
  "access_token": "8f877187-e624-42e9-a9b3-94c61c569157",
  "token_type": "bearer",
  "refresh_token": "6ab57c72-75b5-443c-9bd2-f05d07b9393e",
  "expires_in": 25466,
  "scope": "write read"
}
```

Erişim Anahtarını Yenileme

Access Token'ı yenilemek istediğimizde aşağıdaki gibi bir CURL yapısı kullanabiliriz.

```
curl -X POST -d 'grant_type=refresh_token&refresh_token={refresh_token}'
--user ':' https://api.ardich.com/api/v3/login/oauth
```

{refresh_token} yerine az önce Acces Token içeren JSON yanıtında yer alan refresh_token değerini yazıyoruz.



Görüldüğü üzere erişim bilgilerimiz yenilendi.


```
{
  "access_token": "c72b76a7-427b-47a3-8403-0861e8183915",
  "token_type": "bearer",
  "refresh_token": "fd5844d6-2a24-4a55-bd94-c40c6b8af75d",
  "expires_in": 43199,
  "scope": " write read"
}
```

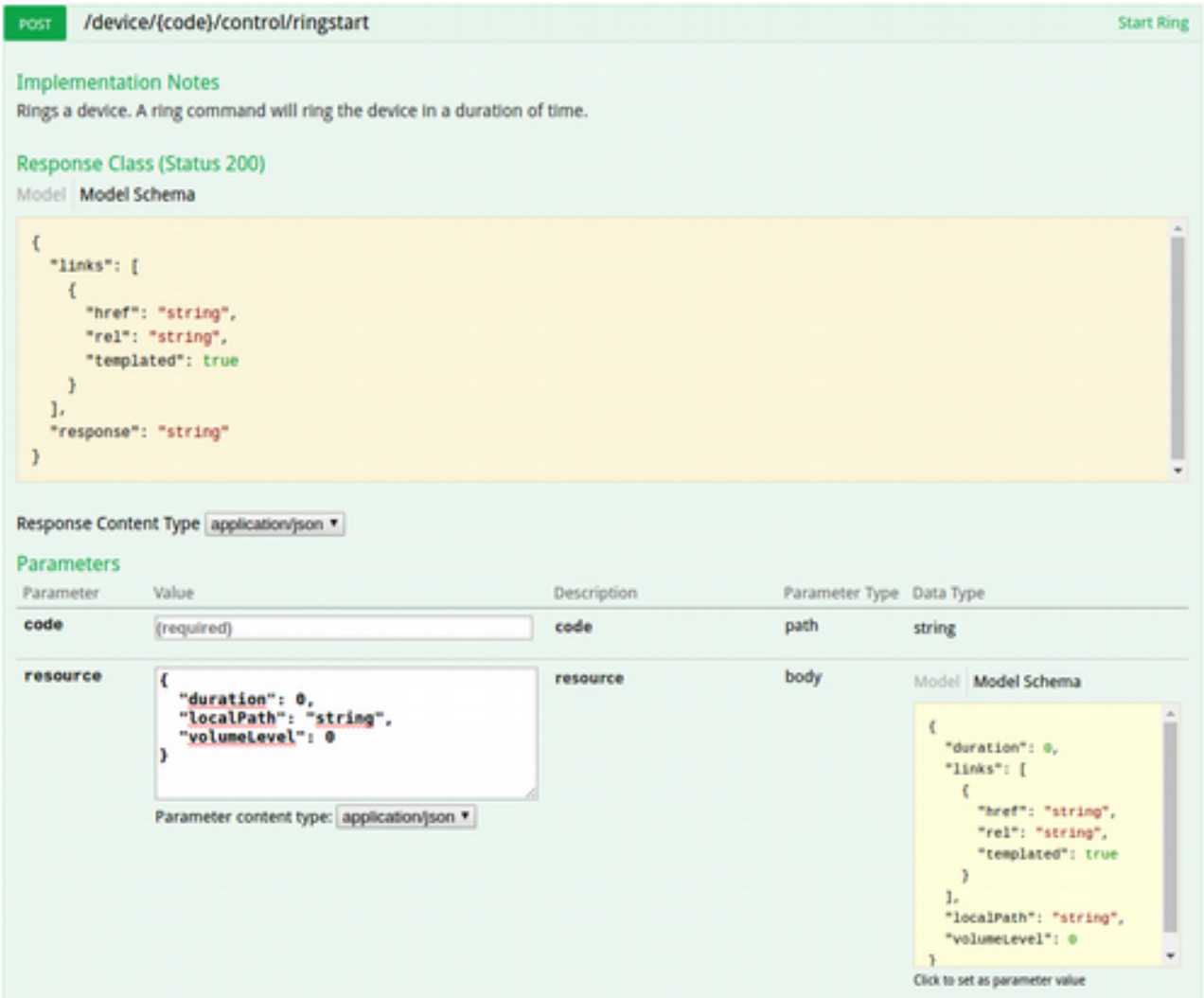

Erişim Anahtarını Kullanarak GET ve POST Yapma

Örneğimizde bir Gateway'imizde Ring aksiyonu ile zil çaldıralım. Bunu yapmak için iki farklı CURL kullanacağız. İlkinde, lisanslı olan Gateway'leri listeleyecek, ikincisinde de listelediğimiz Gateway'lerden birinin **code**'unu kullanarak Ring aksiyonu gerçekleştireceğiz.

İlk olarak Swagger'dan Ring çaldırmak için kullanılacak API'yi inceleyelim...

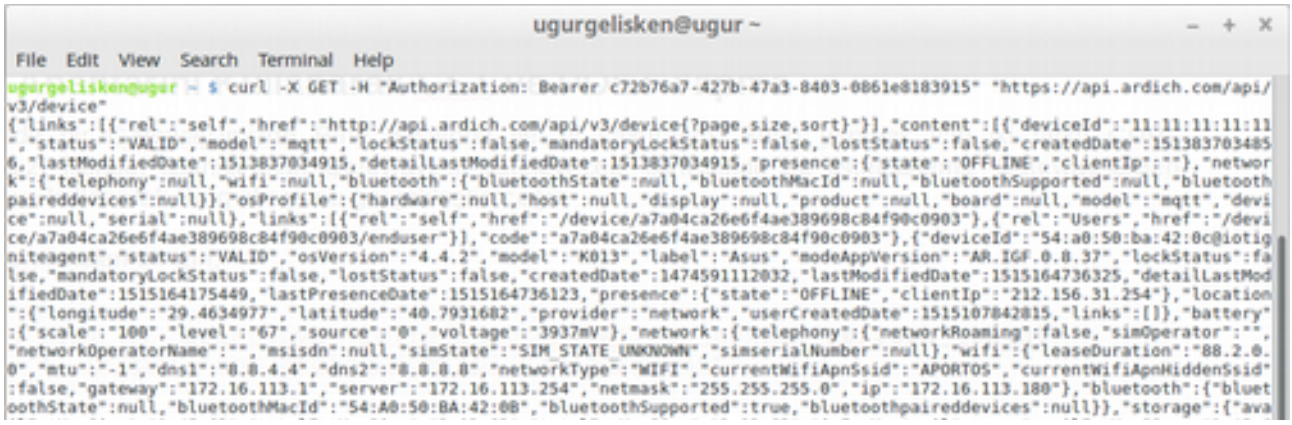
/device/{code}/control/ringstart

API'ye baktığımızda **POST** metodu içinde birtakım parameterleri JSON olarak bekliyor. Bu parametrelerden **duration** (saniye cinsinden zil çalma süresi), **localPath** (zil sesinin yolu) ve **volumeLevel** (ses seviyesi) zorunludur. Bu sebeple sadece bu parametreleri **JSON** olarak oluşturup göndereceğiz.



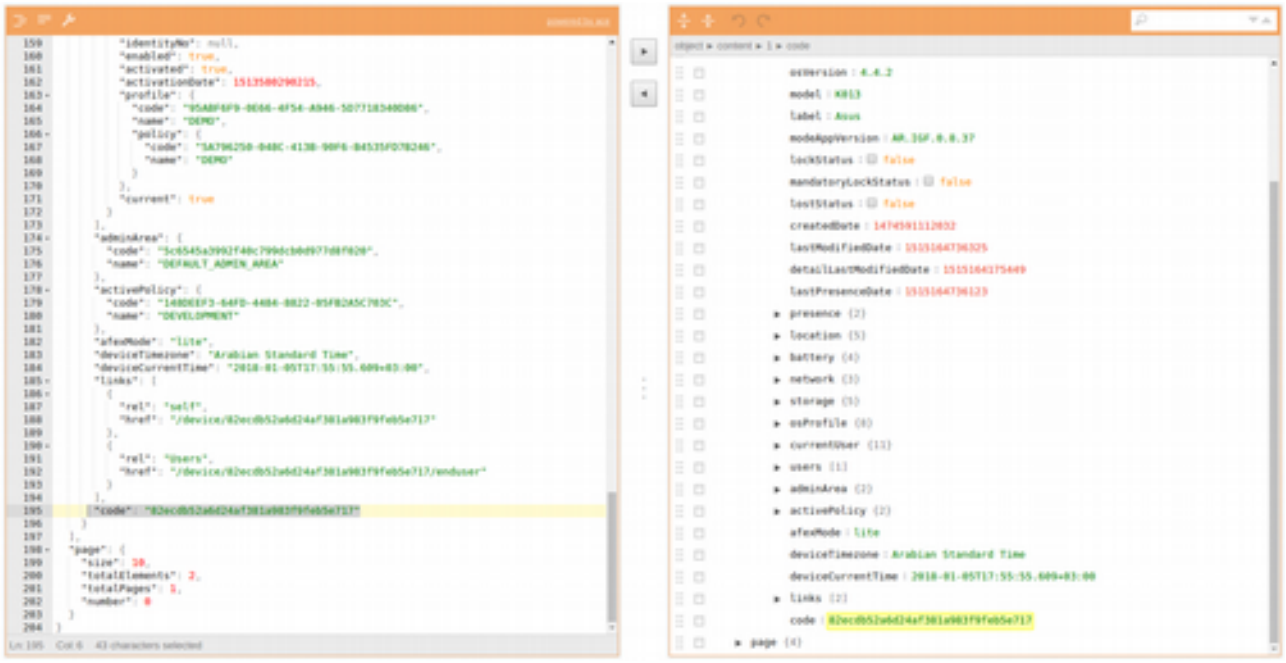
Gateway'leri listelemek için;

```
curl -X GET -H "Authorization: Bearer c72b76a7-427b-47a3-8403-0861e8183915"
"https://api.ardich.com/api/v3/device"
```



Dönen JSON uzun ve tek satırda yazdırıldığı için anlaşılması zor. Bu nedenle dönen JSON'u kopyalayıp jsoneditoronline.org'da düzenleyip inceleyelim.

Lisanslı Android Gateway'inimizin code'u **"code":"82ecdb52a6d24af381a983f9feb5e717"** olarak görünüyor.



Artık Gateway'in code'unu bildiğimize göre bir sonraki işleme geçelim...

```
curl -X POST -H 'Authorization: Bearer c72b76a7-427b-47a3-8403-0861e8183915'
-H "Content-Type: application/json" -d
'{"duration":1,"volumeLevel":5,"localPath":""}' --user ':'
https://api.ardich.com/api/v3/device/82ecdb52a6d24af381a983f9feb5e717/control/
ringstart
```

İşlem sonunda Android Gateway'imizde 1 saniye boyunca 5 ses seviyesinde varsayılan zil sesi çalacaktır. Eğer başka bir zil sesi çaldırmak isterseniz, localPath'te ses dosyasının yolunu da tanımlayabilirsiniz.

> CURL kullanırken **-H** > Header, **-d** Data demektir.

JavaScript ile Sensör Verilerini Gerçek Zamanlı Olarak Gösterme Örneği

Daha önce CURL ve Swagger üzerinden IoT-Ignite API'lerini nasıl kullanacağımızı görmüştük. Şimdi de JavaScript AJAX ile kullanımını adım adım bir örnek uygulama geliştirerek görelim.

Yapacağımız örneğimiz, en son haline geldiğinde aşağıdaki gibi görünecektir.

Uygulamamızda, ilk olarak e-posta ve parola bilgileri girilerek Login butonuna tıklanması ile Access Token alınacak, Brand Name (Servis İsmi) de ekranda gösterilecek. Sonrasında da Gateway'ler listelenecek. Gateway'lerden biri seçildiğinde, eğer ona bağlı Node varsa onlar da hemen yanında listelenecek. Node seçimi ile de eğer seçili Node'a bağlı Sensor'ler varsa, bunlar da hemen sağ tarafında yeniden listelenecek. Böylece her bir Gateway'in her bir Senor'üne erişebileceğiz.

Sensor seçimi ile de CanvasJS ile LineChart grafik türünü kullanarak Integer (Her Sensor Integer göndermeyebilir. Örneğimizde yine VirtualDemoNode altında yer alan Temperature, Humidity ve

Lamp'i kullanacağız) olarak aldığımız Sensor geçmiş verisi (Son 24 saatlik veriler) gösterilecek. Bununla birlikte bir WebSocket de açılacak ve anlık olarak gelen son veriler de grafiğe eklenecek. WebSocket bağlantısı kurabilmek için de öncelikle IoT-Ignite Cloud'tan bir WebSocket erişim izni isteyecek, gelen erişim anahtarı ve url'ini kullanarak seçtiğimiz Gateway id'sini parametre olarak WebSocket erişimi gerçekleştireceğiz. WebSocket'in de bize hangi verileri aktaracağını belirtmek için seçili olan Node ve Sensor id'lerini de parametre olarak WebSocket ile göndereceğiz. Böylece sadece o Sensor'e ait veriler WebSocket üzerinden gelecek (Sadece tek bir WebSocket değil, birden fazla da WebSocket açılıp farklı farklı Sensor'ler dinlenebilir. Veya parametre olarak herhangi bir Sensor belirtilmezse, bütün Sensor verileri WebSocket'ten gelir. Bu durumda da sizin gelen verilerden filtreleme yapmanız gerekecektir).

> jQuery, "az kod, çok iş" sloganı ile geliştirilmiş, DOM manipulasyonunda kullanılan, kullanımı kolay bir JavaScript Library'dir. CanvasJS ise verileri çeşitli grafik metotlarla görüntülemeyi sağlayan başka bir JavaScript Library'dir.

Adım 1: Arayüzün Hazırlanması

İlk olarak bir .html dosyası oluşturup arayüzümüzü hazırlayalım. Arayüzde form elemanları, form elemanlarını gruplayacak ve 'lar, gerekli olan etiket ve form elemanlarının id'lerinin tanımlanması ve jQuery ile CanvasJS'nin eklenmesi yapılacaktır.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title></title>
</head>
<body>
    <div id="f_login">
        <input type="text" id="i_email" placeholder="e-posta" />
        <input type="text" id="i_password" placeholder="password" />
        <button id="b_login">Login</button>
    </div>
    <div id="main_area" style="display:none;">
        <p id="brand_name"></p>
        <label>Gateway</label>
        <select id="gateways"></select>
        <span id="node_area" style="display: none">
        <label> &#9658; Nodes</label>
        <select id="nodes"></select>
        </span>
        <span id="sensor_area" style="display: none">
```

```
        <label> &#9658; Sensors</label>
        <select id="sensors"></select>
        </span>
    </div>
    <div id="chartContainer" style="height: 200px; width: 100%; margin-top: 10px">
    </div>
</body>
<script src="https://code.jquery.com/jquery-3.2.1.js"></script>
<script src="https://canvasjs.com/assets/script/canvasjs.min.js"></script>
<script>
...
</script>
</html>
```

Yazdığımız kodları inceleyelim...

Basit bir HTML5 syntax'ı oluşturup 3 farklı alanı tanımladık. Her birine sırası ile **f_login**, **main_area** ve **chart-container** isimlerini verdik.

- **f_login**: IoT-Ignite API'lerini kullanabilmek için **Access Token** alınması gerekmektedir. Bu içinde yer alan form elemanları ile Access Token alınması için kullanılacaktır. Access Token alındığında **f_login** 'i jQuery ile görünmez, **main_area** 'i de görünür hale getirilecek.
- **main_area**: Başlangıçta **display:none** CSS özelliği ile görünmez olan bir 'dir. Sebebi; ilk olarak üst kısımda yer alan form ile Access Token alınması gerektiğidir. Bu içinde de **brand_name** isimli bir **<p>** etiketi, 3 adet de **Select Box** bulunmaktadır. <p> etiketinde, **Login** işlemi yapıldığında **Servis İsmi** yazdırılacak. Select Box'lar ise sırası ile görünür hale gelecektir. Yani ilk olarak **gateways** isimli Select Box görünür olarak gelmektedir. Buradan bir Gateway seçildiğinde, hemen yanındaki **nodes** id isimli Select Box da görünür olacak ve seçilmiş olan Gateway'e bağlı olan Node'lar listelenecek. Aynı şekilde **nodes** id'li Select Box'tan da bir Node seçildiğinde, o Node'a bağlı olan Sensör'ler bir sonraki **sensors** id isimli Select Box'ta görüntülenecek. Access Token alındığında ise bu ** ** içinde ilk olarak jQuery ile görünür olacak. Sonrasında da az önce bahsetmiş olduğumuz Select Box'lardan seçimler yapıldıkça sırasıyla diğer Select Box'lar da görünür olacak.
- **chart-container**: Bu alanında ise **CanvasJS** ile oluşturulacak olan **Line Chart** yer alacak. Bir üstteki alanından seçilen Sensör'e göre chart yeniden render edilecek ve o Sensör'e ait **son 1 gün** içindeki **100 veri** gösterilecek. Sonrasında da anlık olarak veri geldiğinde Chart'a eklenmeye devam edecek. Başlangıçta herhangi bir veri olmadığından, yani render edilmediğinden bu boş görünecektir. Bu yüzden görünmez yapmaya gerek yoktur.

Son olarak <body></body> altında **<script></script>** ile sırasıyla **jQuery** ve **CanvasJS** kütüphaneleri ekleniyor. <script>...</script> alanında da JavaScript kodlarımızı yazacağız.

Adım 2: Erişim Anahtarı Alma, Gateway'leri Listeleme ve Brand Name'i Gösterme

Daha önce HTML5 arayüzümüzü oluşturup gerekli elementler için **id**'ler tanımlamıştık. jQuery ile bu id'leri referans alarak DOM erişimi sağlayacak ve yine jQuery AJAX metotlarını kullanarak **Access Token** alacağız. Aldığımız Access Token'ı **Local Storage**'da kaydedecek, daha sonra da diğer API'lerde kullanacağız. Access Token aldıktan sonra **main_area** id'li 'i görünür yapıp, **f_login** id'li 'i de görünmez yapacağız. f_login id'li içinde **e-posta** ve **parola** bilgilerinin girileceği 2 adet **<input>** ve 1 adet de **Login** isimli **<button>** bulunacak. Butona

tıklanmasıyla <input>'lara girilen bilgiler ile **/login/auth** API'sine **POST** yapılacak ve Access Token alınacak. Ardından **getBrandName** ve **getGatewayList** fonksiyonları da tetiklenecek.

getBrandName fonksiyonu, Access Token'ı Local Storage'den alarak **/abi** API'sini **GET** yapacak. Gelen response'dan **brand** Key'i alınıp **brand_name** id'li **<p>** etiketi içinde jQuery'nin **.html()** metodu ile yazdırılacak.

GetGatewayList fonksiyonunda da **/device** API'sine **GET** yapılacak. Gelen response'da lisanslı olan Gateway'ler yer alacak. jQuery'nin **.each()** metodunu (for each) kullanarak **gateways** isimli Select Box'ın Option'larını tek tek oluşturup **.append()** metodu ekleyeceğiz. Select Box'ların içi doldurulurken **<option>**'ların value ve text'leri döngü içinde tanımlanacak. Listenin en başına da **.prepend()** metodu ile **selected="select" disabled** özelliklerine sahip bir **<option>** ekleyip, "Please select..." gibi pasif bir eleman ekliyoruz. Böylece listeden bir seçim yapılması gerektiği gösterilmiş oluyor.

Kodlar içinde yer alan açıklama satırlarını da inceleyin...

```
/* jQuery yüklendiğinde ve hazır olduğunda ( .ready() ) kod blogu içindeki kodlar
çalıştırılıyor. */

$(document).ready(function() {

    //b_login id'li butona tıklandığında...
    $("#b_login").click(function() {

        // Access Token alınıyor...
        $.ajax({
        url: 'https://api.ardich.com/api/v3/login/oauth',
        type: 'POST',
        data: {
            "grant_type": "password",
            "username": $("#i_email").val(),
            "password": $("#i_password").val()
        },
        dataType: 'json',
        headers: {
            'Authorization': 'Basic ' + window.btoa(":"),
            'Content-Type': 'application/x-www-form-urlencoded'
        }
        }).done(loginSuccess).fail(loginError);

        // Access Token alındığında...
        function loginSuccess(data) {
        /* Dönen değer data parametresine aktarılıyor. Eğer data içinde acces_token
isimli bir Key varsa API 200 dönmüş demektir. */
        if (data.hasOwnProperty('access_token')) {

            // Local Storage'a access_token ismi ile kaydediliyor.
            window.localStorage.setItem("access_token", data.access_token);

            // Login form <div>'i gizleniyor.
            $("#f_login").hide();

            // Select Box'ların olduğu <div> görünür yapılıyor.
            $("#main_area").show();

            // Brand Name'i getirecek fonksiyon çağrılıyor.
            getBrandName();

            // Gateway'leri listeleyecek fonksiyon çağrılıyor.
            getGatewayList();

        }
        }
```

```
        function loginError(error) {
        // Eğer Login başarısız olursa uyarı verdirebilirsiniz...
        }

        // Brand Name'i getirecek API çağrılıyor...
        function getBrandName() {

        $.ajax({
            url: 'https://api.ardich.com/api/v3/abi',
            type: 'GET',
            dataType: 'json',
            contentType: "application/json",
            headers: {
             'Authorization': 'Bearer ' + window.localStorage.getItem("access_token"),
             'Content-Type': 'application/json',
            },
            success: function(data) {
             // <p> içinde yazdırılıyor.
             $("#brand_name").html(data.brand);
            }
        });
        }
    });

    // Gateway'leri getirecek API çağrılıyor.
    function getGatewayList() {

        $.ajax({
        url: 'https://api.ardich.com/api/v3/device',
        type: 'GET',
        dataType: 'json',
        contentType: "application/json",
        headers: {
         'Authorization': 'Bearer ' + window.localStorage.getItem("access_token"),
         'Content-Type': 'application/json',
        },
        success: function(data) {
        /* Gelen response data parametresine aktarılıyor. İlk olarak gateways id'li
select Box'a tıklanamayan pasif ve seçili olan Please select gateway text'li <option>
ekleniyor. Sonrasında da .each() metodu ile gelen data'da content içinde Array halinde
listelenmiş Gateway'ler sırası ile <option> olarak Select Box'a ekleniyor.
<option>'ların value'suna ve text'ine deviceId Key'leri veriliyor. */
            $('#gateways').prepend("<option selected='selected' disabled>Please select
gateway</option>");
            $.each(data.content, function(i, gateway) {
                $('#gateways').append($('<option>', {
                value: gateway.deviceId,
                text: gateway.deviceId
                }));
            });
        }
        });
    }


  /* Buradan sonra diğer kodlar yazılacak... */

});
```

Adım 3: Seçilen Gateway'in Node'larını Listeleme

Bir önceki aşamada Login işlemi yapılmasını, login formunun gizlenmesini, seçimlerin yapılabileceği alanın görüntülenmesini ve Gateway'lerin de listelenmesini sağladık. Şimdi de Gateway'in seçilmesi ile birlikte Node'ların da gösterilmesini sağlayacağız.

Yapacağımız işlemde **gateways** isimli Select Box'tan bir Gateway seçildiğinde, o Gateway'in **id**'si (value) ile o Gateway'e ait Node'ları **/device/{gatewayId}/device-node-inventory** API'sini **GET** metodu ile çağırarak getireceğiz. Gelen listeyi **nodes** isimli Select Box'ta yine **.each()** metodu ile listeleyeceğiz. Fakat burada farklı bir işlem olarak Node'ların yer alacağı Select Box'ın her Gateway seçimi ile boşaltılması gerektiğinden jQuery'nin **.empty()** metodu ile temizleyeceğiz.

İlk olarak kodlarımızın başında aşağıdaki gibi bir değişken tanımlayalım...

```
$(document).ready(function() {

    var selectedGatewayId;
    var nodesAndSensors;

  ...
```

selectedGatewayId değişkenine gateways isimli Select Box'tan seçtiğimiz Gateway'in id'si atanacak. Bu değeri de daha sonra diğer API'lerde kullanacağız.

nodesAndSensors'e de seçili olan Geteway'e ait Node'lar geldiğinde Node listesi atanacak. Node listesi içinde, her bir Node'a ait **Things** (Sensörler) de alt Array olarak eklidir. Yani Sensör'leri listelemek için yeni bir API çağırmayacak, bu değişken içinden ilgili Node grubunu filtreleyerek Sensör listesini oluşturacağız.

Önceki kodlarımızda yer alan /* Buradan sonra diğer kodlar yazılacak */ alanına aşağıdaki kodları ekleyerek devam edelim...

```
...

    // gateways id'li Select Box'tan seçim yapıldığında...
    $("#gateways").change(function() {

    // node_area isimli <span> görünür yapılıyor.
    $("#node_area").show();

        /* selectedGatewayId değerişkenine seçilen <option>'un value'su atanıyor. */
        selectedGatewayId = this.value;

        // Seçilen Gateway id'sine ile Node'lar çağrılıyor...
        $.ajax({
        url: 'https://api.ardich.com/api/v3/device/' + selectedGatewayId
+ '/device-node-inventory',
        type: 'GET',
        dataType: 'json',
        contentType: "application/json",
        headers: {
            'Authorization': 'Bearer ' + window.localStorage.getItem("access_token"),
            'Content-Type': 'application/json',
        },
        success: function(data) {
            /* Node'lar data parametresinin içinde extras.nodes altında yer
almaktadır. Bunu da nodesAndSensors değişkenine atıyoruz. */
            nodesAndSensors = data.extras.nodes;
```

```
            /* node id'li Select Box'ın içi ilk olarak .empty() metodu boşaltılıyor.
Çünkü yeni bir Gateway seçildiğinde liste sıfırlanmalı. Sonrasında da .prepend()
metodu ile Please select... yazdırılıyor. */
            $('#nodes').empty().prepend("<option selected='selected' disabled>Please
select node</option>");

            /* Döngü ile her bir Node ismi nodes id'li Select Box'ta oluşturuluyor.
value ve text olarak nodeId veriliyor. */
            $.each(nodesAndSensors, function(i, node) {
                $('#nodes').append($('<option>', {
                value: node.nodeId,
                text: node.nodeId
                }));
            });
        }
        })
    });

...
```

Adım 4: Seçilen Node'un Sensör'lerini Listeleme

Daha önce yazdığımız kodların peşine kodlarımızı yazmaya devam ediyoruz...

Artık Gateway'leri ve seçilen her bir Gateway'in Node'larını listeleyebiliyoruz. Şimdi yapacağımız işlemde **nodes** id'li Select Box'tan Node seçilmesi ile birlikte, hemen yanında Sensor'leri listeleyen sensors id'li başka bir Select Box'ı görünür yapacağız. Daha **sensorsAndNodes** adlı değişkene yüklediğimiz **Nodes+Things** verilerinden filtreleme yaparak sensör'leri Select Box'ta listeleyeceğiz.

İlk olarak yine kodlarımızın başında bir değişken daha tanımlayalım.

```
 var selectedNodeId;
```

selectedNodeId değişkenine, **nodes** isimli Select Box'tan seçilmiş olan **Node**'un value'su atanacak. Bu değer ile **sensorsAndNodes** değişkeninden **if** koşulu ile karşılaştırma yapıp, ilgili Node altında yer alan **Things**'leri (Yani Sensor ve Actuator'ler) **sensors** id'li Select Box'ta listeleyeceğiz.

```
...

    // nodes id'li Select Box'tan herhangi bir Node seçildiğinde....
    $("#nodes").change(function() {

        // sensor_area id'li <span> görünür yapılıyor.
        $("#sensor_area").show();

        // Seçilen Node'un value'su selectedNodeId değişkenine atanıyor.
        selectedNodeId = this.value;

       /* .each() döngüsü ile nodesAndSensors Array'inde sırası ile nodeId Key'i ile
selectedNodeId değişken değeri if ile karşılaştırılıyor. Böylece seçili olan Node'a
ait Node+Things grubu bulunacak. */
        $.each(nodesAndSensors, function(i, node) {

    /* Eşleşme sağlandığında daha önceki işlemlerde olduğu gibi Sensör'leri
listeleyecek Select Box oluşturuluyor. */
        if (node.nodeId == selectedNodeId) {

            $('#sensors').empty().prepend("<option selected='selected' disabled>Please
select sensor</option>");
```

```
            $.each(node.things, function(i, thing) {
                $('#sensors').append($('<option>', {
                value: thing.id,
                text: thing.id
                }));
            });
        }
        });
    });

...
```

Adım 5: Seçilen Sensör'ün Geçmiş Verilerini Alma ve Line Chart ile Gösterme

Artık Select Box oluşturma, görüntüleme gibi işlemlerimiz bitti. Bundan sonrasında da **sensors** id'li Select Box'tan bir **Sensör** seçilmesi ile birlikte **Line Chart** oluşturup geçmiş verileri göstereceğiz.

Yine bir takım değişkenlere ihtiyacımız olacak. Bunları da kodlarımızın başına ekleyelim...

```
var selectedSensorId;
var sensorDataIntArray = [];
```

selectedSensorId değişkenine **sensors** id'li Select Box'tan seçilen Sensör'ün **valeu**'su atanacak. Bu değeri, **Gateway / Node / Sensor** ilişkisi ile sensör verilerini çekeceğimiz API'de kullanacağız.

sensorDataIntArray değişkeni de **Array** olarak tanımlanmış. Bu Array'e, Sensör'ün verilerini **value+date** olarak **JSON** formatında ekleyeceğiz. Yani Array'in her elemanı Sensör'ün verisinin değerini ve tarihini içeren bir JSON olacak. JSON'un formatı da CanvasJS'nin Line Chart'ı oluşturmak için kullanacağı **{ x: Date, y: Value }** formatında olacaktır.

Aşağıdaki kodlarımızı ekleyerek devam edelim...

```
...
    // Herhangi bir Sensör seçildiğinde...
    $("#sensors").change(function() {

        // Seçilen Sensör'ün id'si selectedSensorId'ye atanıyor.
        selectedSensorId = this.value;


        /* Sensör verileri çekilirken API'ye veri aralığı tarihini de belirtiyoruz.
Verinin bitiş tarihini anlık zamanı milisaniye olarak atıyoruz. Başlangıç tarihini de
1 gün öncesine getiriyoruz. */
        // Şimdi...
        var now = new Date().getTime();
        // Şimdi'den 24 saat öncesi...
        var yesterday = new Date().getTime() - 86400000;

        /* Seçilen Gateway'in seçilen Node'unun seçilen Sensor'ünün son 1 gün içindeki
son 100 verisi çağrılıyor. */
        // API'yi oluştururken kullanılan değişkenlere dikkat edin!
        $.ajax({
        url: 'https://api.ardich.com/api/v3/device/' + selectedGatewayId
+ '/sensor-data-history?nodeId=' + selectedNodeId + '&sensorId=' + selectedSensorId
+ '&startDate=' + yesterday + '&endDate=' + now + '&pageSize=100',
        type: 'GET',
        dataType: 'json',
        contentType: "application/json",
        headers: {
            'Authorization': 'Bearer ' + window.localStorage.getItem("access_token"),
            'Content-Type': 'application/json',
```

```
        },
        success: function(data) {
            /* Gelen sensör verileri data parametresine atanıyor. */

            /* sensorDataIntArray dizisi sıfırlanıyor. Çünkü her sensor seçimi
işleminde bu diziye yeniden seçilen sensör verileri atanacak. */
            sensorDataIntArray = [];

            // Döngü ile gelen sensör verileri okunuyor...
            $.each(data.list, function(i, sensorData) {
                /* Sensör verisi [“34”] gibi bir String olarak gelmektedir. Bu
String’te gerekli karakter temizleme yapılıp ham Integer veri alınıp sensorDataInt
değişkenine atanıyor. */
                var sensorDataInt = sensorData.data.replace(/\'/g, "").replace(/\]/g,
"").replace(/\[/g, "").replace(/\"/g, "");

                // Line Chart’ı oluşturmak için gerekli olan veri dizisi
oluşturuluyor. SensorDataIntArray dizisine JSON olarak x ve y değerleri ekleniyor. X
değeri Yatay eksende sensör değer oluşturulma tarihini, Y değeri de Dikey eksende
sensör değerini gösterecek. */
                sensorDataIntArray.push({
                y: parseInt(sensorDataInt),
                x: new Date(sensorData.createDate)
                });
            });

            /* drawChart fonksiyonu, oluşturduğumuz bu sensorDataIntArray dizisi
parametre olarak verilerek çağrılıyor. Bu fonksiyon ile Line Chart çizdirilecek. */
            drawChart(sensorDataIntArray);

            // Anlık verileri de gösterebilmek için WebSocket bağlantısı kurulması
gerekiyor. Bunun için de WebSocket için yine bir erişim anahtarı, bir id ve WebSocket
bağlantısı için URL gerekiyor. Bu yetkileri alabilmek için Access Token ile
/subscribe/device/token API’sini, Gateway id’sini parametre olarak ekleyip GET ile
çağırıyoruz. Böylece sadece o Gateway için WebSocket kanalı açılacak. */
            $.ajax({
                url: 'https://api.ardich.com/api/v3/subscribe/device/token',
                type: 'GET',
                dataType: 'json',
                contentType: "application/json",
                data: {
                'device': selectedGatewayId
                },
                headers: {
                'Authorization': 'Bearer ' +
window.localStorage.getItem("access_token"),
                'Content-Type': 'application/json',
                },
                success: function(data) {
                /* connectRealTime fonksiyonu, API’den gelen veriler ile data
parametresine eklenerek tetikleniyor. WebSocket ile gerçek zamanlı veriler de alınıp
Line Chart'a eklenecek. Şu an bu fonksiyonu yazmadık, bir sonraki adımda yazacağız. */
                connectRealTime(data);
                }
            })

        }
        })

    });


  // Line Chart’ı çizecek olan fonksiyon.
  /* Sensör verileri döndüğünde sensorDataIntArray dizisine JSON olarak x-y değerleri
eklemiştik. Bu dizi de drawChart fonksiyonunda parametre olarak kullanılıyor. Gelen
parametreler chartDatas değişkenine aktarılıyor. */
    function drawChart(chartDatas) {

        /* ChartJS’nin Chart oluşturma şablonunu kullanıyoruz ve verilerimizi
dataPoints’e aktarıyoruz. Son olarak chart.render() ile de Chart’ı çizdiriyoruz. */
        var chart = new CanvasJS.Chart("chart-container", {
        axisY: {
            includeZero: false
        },
        data: [{
            type: "line",
            dataPoints: chartDatas
```

```
        }]
        });
        chart.render();
    }

...
```

Adım 6: WebSocket ile Sensör'ün Anlık Verisinin Almak ve Lince Chart'a Ekleme

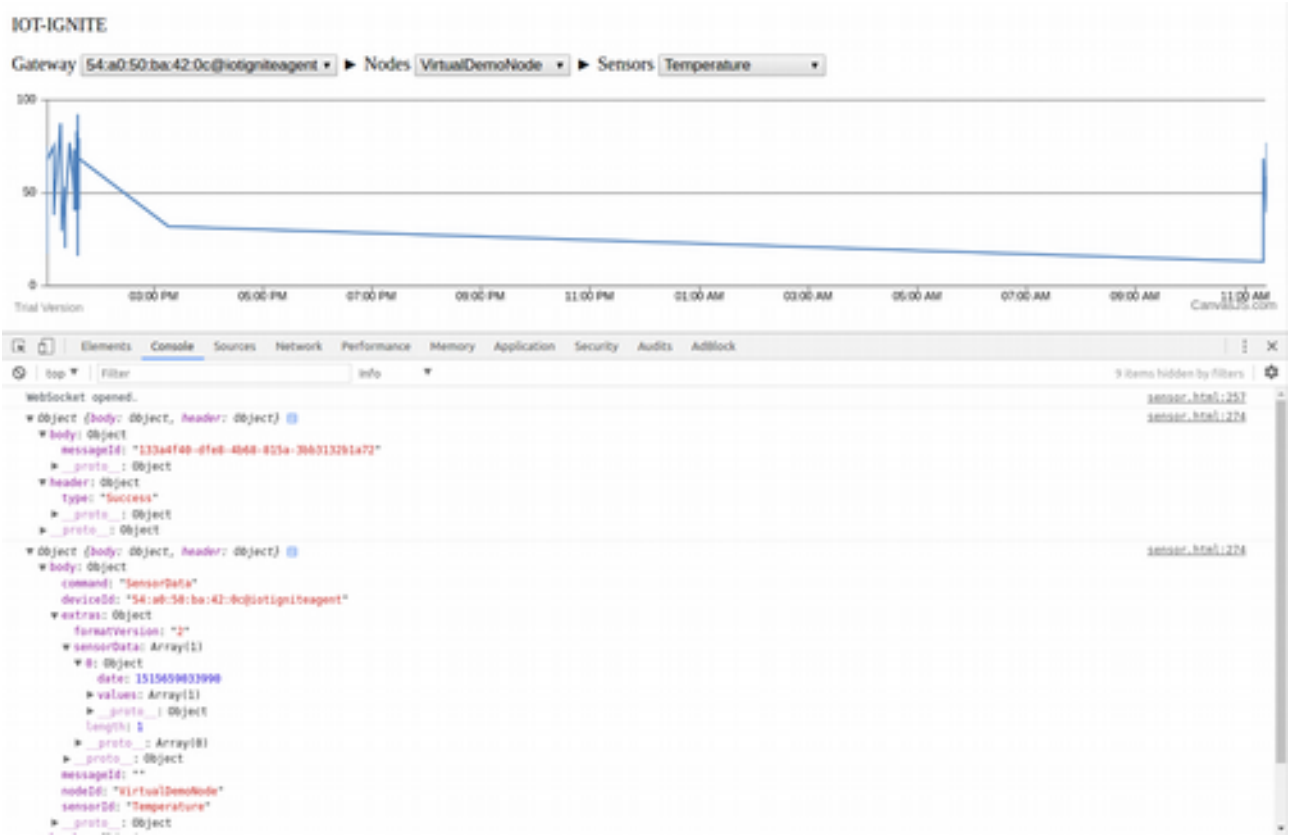
Bu aşamaya kadar seçtiğimiz Sensör verilerini Line Chart'ta çizdirebiliyoruz. Ancak yeni bir veri geldiğinde bunu göremeyeceğiz (Eğer listeden tekrar aynı sensörü seçersek görürüz). Anlık olarak yeni verileri de Line Chart üzerinde görebilmemiz için IoT-Ignite Cloud'u ile WebSocket üzerinden bağlantı kurup, o Gateway'den gelen verileri dinlememiz gerekir. Bir önceki aşamada WebSocket açabilmek için gerekli olan bilgileri edinmiş, connectRealTime fonksiyonuna parametre olarak aktarmıştık. Ancak o fonksiyonumuz olmadığı için hata verecektir. Şimdi bu fonksiyonu da hazırlayıp, son gelen verileri de Line Chart'ta gösterelim...

Kodlarımızın peşine ekleyerek devam edelim...

```
...

  /* Fonksiyona subscribe API'sinden gelen response parametre olarak verilerek success
içinde çağrılıyordu. */
    function connectRealTime(data) {

        /* ws ismi ile bir WebSocket açılıyor. URL olarak data.url veriliyor */
        ws = new WebSocket(data.url);

        // WebSocket açıldığında...
        ws.onopen = function(evt) {
        // Console'da mesaj yazdırılıyor...
        console.log("WebSocket opened.")

        /* subscribeMessage isminde bir Object oluşturuluyor. Bu Object içinde
deviceId, nodeId, sensorId tanımlanarak hedef belirtiliyor. Yetkilendirme için de id
ve token değerleri de data parametresinden alınarak ekleniyor. */
        var subscribeMessage = new Object();
        subscribeMessage.deviceId = selectedGatewayId
        subscribeMessage.nodeId = selectedNodeId;
        subscribeMessage.sensorId = selectedSensorId;
        subscribeMessage.id = data.id;
        subscribeMessage.token = data.token;
        subscribeMessage.version = "2.0";

        // subscribe ismi ile başka bir Object oluşturuluyor...
        /* Bu Object'in de type'ı sabit olarak subscribe, message'ı da az önce
oluşturduğumuz Object atanarak belirtiliyor. */
        var subscribe = new Object();
        subscribe.type = "subscribe";
        subscribe.message = subscribeMessage;

        // Açtığımız WebSocket'e subscribe isimli Object String'e çevrilerek mesaj
olarak gönderiliyor. Böylece hedef belirttiğimiz Sensör verilerini dinleyebilmek için
yetki istemiş olacağız. Eğer yetkilendirme izni verilirse, bir mesaj dönecektir. Bu
mesajı aşağıdaki şekilde görebilirsiniz. */
        ws.send(JSON.stringify(subscribe));
        };

        // WebSocket'ten bir mesaj geldiğinde...
        ws.onmessage = function(evt) {
        // Gele mesajı JSON'a çevirip response değişkenine atıyoruz.
        var response = JSON.parse(event.data);

        // Gelen mesajı görmek isteriz diye Console'da yazdırıyoruz.
        console.log(response);

        /* Gelen mesajlar farklı farklı formatlarda olabilir. Örneğin ilk gelen mesaj
```

```
yetki verildiği ile ilgiliydi. Bize sensör verisinin geldiği mesaj gerektiğinden,
response.body.command değeri SensorData olan mesajları dikkate alıyoruz. */
        if (response.body.command == "SensorData") {
            /* Gelen mesajın sensör verisi içerdiğini anladığımızdan, gelen veriyi
sensorDataIntArray dizisine unshift ile ekliyoruz. Ekleme yaparken yine Y ve X
değerlerini Integer ve Date olarak parse edip, JSON olarak eklediğimize dikkat edin.
*/
            sensorDataIntArray.unshift({
                y: parseInt(response.body.extras.sensorData[0].values[0]),
                x: new Date(response.body.extras.sensorData[0].date)
            });

            /* Yeni verinin de eklendiği sensorDAtaIntArray dizisi ile Line Chart'ı
yeniden çizdiriyoruz. */
            drawChart(sensorDataIntArray);
        }
        };

        // WebSocket ile ilgili bir hata olursa...
        ws.onerror = function(evt) {
           console.log("error > ", evt)
        };

    // WebSocket kapanırsa...
        ws.onclose = function(evt) {};
    }

...
```

Anlık olarak WebSocket'ten gelen mesajları **Console**'da görebilirsiniz.



BÖLÜM 9

Uygulama Örnekleri

IoT-Ignite ile MQTT Client Library Kullanımı

MQ Telemetry Transport veya MQTT, düşük güçlü cihazları bağlamak için geliştirilen ve her geçen gün popülerlik kazanan bir mesajlaşma ve iletişim protokolüdür. MQTT başlangıçta, pub/sub mimarisine dayalı iletişimler için tasarlanmasına rağmen, zaman içinde genel amaçlı bir kavram olan “machine-to-machine” (M2M) iletişim protokolü olarak kullanılmaya başlanmıştır. MQTT kavramının ne olduğu, temel kullanım amacı ve nasıl çalıştığıyla ilgili bilgileri birinci bölümünde incelemiştik. Burada bu protokolü kullanarak IoT-Ignite tabanlı IoT uygulamaları geliştirmek için bilmemiz gereken temel bilgilerden bahsedeceğiz.

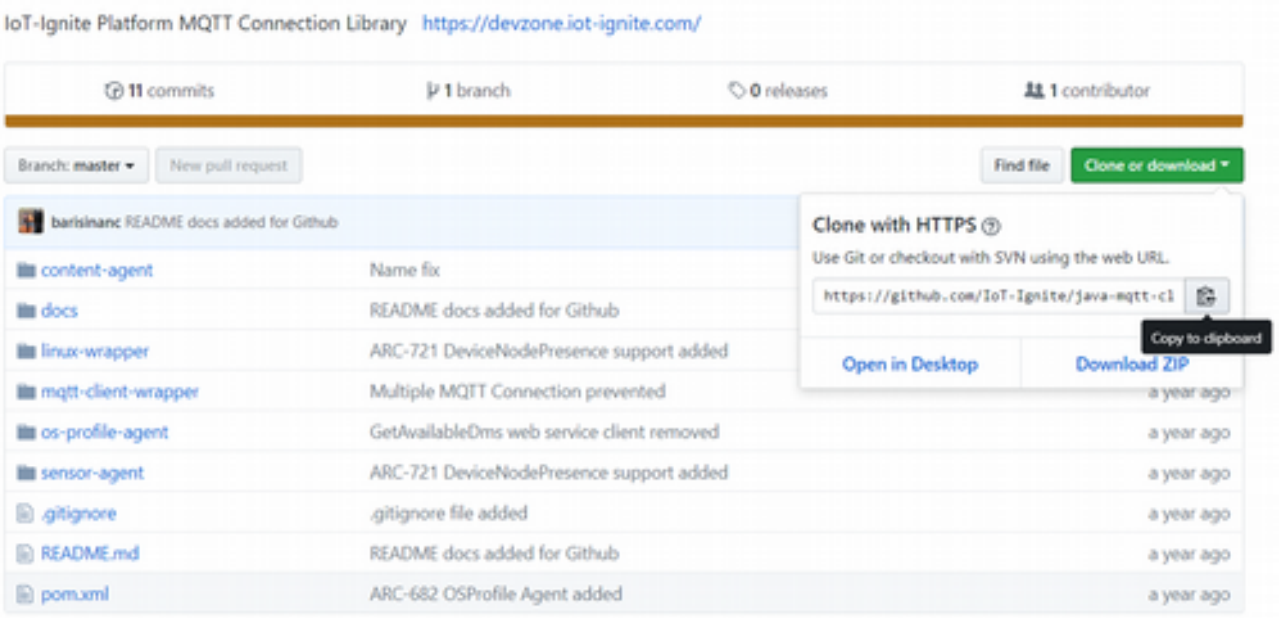
MQTT’nin kullanım kolaylığından dolayı IoT-Ignite platformunda, uygulamalarda kullanılmak üzere MQTT Client kütüphanesi geliştirilmiştir. Bu kütüphane açık kaynak olduğundan dolayı isterseniz kendiniz de bir kütüphane geliştirebilirsiniz. MQTT Client kütüphanesi ve örnek kullanım kaynak kodlarına Github üzerinden erişim sağlayabilirsiniz.

Github’tan MQTT Client Library’nin Klonlanması

MQTT Client kütüphanesinin kaynak kodları Github üzerinden paylaşılmaktadır. Bu kütüphaneyi bilgisayarınıza indirip çalışabilmek için aşağıda verilen linki kullanabilirsiniz.

https://github.com/IoT-Ignite/java-mqtt-client

Bu linki ziyaret ettiğinizde aşağıda verilen ekran görüntüsü bizi karşılar.



Clone or download seçeneğinden istemci kütüphanesini bilgisayarınıza indirebilir ve üzerinde değişiklik yapabilirsiniz. MQTT Client kütüphanesi, gateway cihazların MQTT prokolü üzerinden IoT-Ignite platformuna bağlanmasını sağlamak için kullanılır.

Bu kütüphane 4 temel bileşenden meydan gelir. Bunlar aşağıdaki gibidir:

- **mqtt-client-wrapper**: MQTT Core bağlantı kütüphanesidir. IoT-Ignite platformu ile bağlantı kurmayı sağlayan kütüphanedir.
- **sensor-agent**: Dahili ve harici bileşenlerden sensör verileri elde etmeyi sağlayan kütüphanedir.

- **content-agent**: IoT-Ignite tarafından sağlanan içerikleri indirmek için kullanılan başka bir MQTT Agent kütüphanesidir.
- **linux-wrapper**: Yukarıda verilen projeleri yürütmek için kullanılır. Kendi MQTT istemcinizi oluşturmak için bu projeden başlayarak değişiklik yapabilirsiniz.

mqtt-client-wrapper

Bu kütüphane IoT-Ignite platformu için geliştirilen MQTT client kütüphanesinin temel bileşenidir. Oturum başlatma, IoT-Ignite platformuna mesaj gönderme ve IoT-Ignite bulutundan yanıt almayı sağlamakla görevlidir. Bu kütüphane ile gelen toplamda üç adet hizmet vardır.

SessionService

Verilen kimlik bilgilerini içeren bir oturumu yönetmeyi sağlamak için kullanılır. Bu servis ile oturumları başlatabilir, kurtarabilir veya yok edebiliriz.

CloudMessageService

Mesajları yönlendirmek için kullanılan servistir. Bulutta meydan gelen tüm mesajları yakalar ve onları kayıtlı olan cihazlara yönelendirir. Kayıtlı olan cihaz buluttan belirli bir mesaj türünü dinlemek isterse, **addListener** metodu başladığı zaman cihazın bu servise kayıt olması gerekir. Eğer yayınlanan mesaj veya mesajların yanıtını dinlemek isterse bunun için **addResponseListener** ile bu servise kayıt olması gerekir.

MessagePublisherService

Client, yani istemci mesajlarını yayınlamayı sağlayan servistir. Platforma mesaj göndermek için bizlere 3 yöntem sunulmaktadır.

- **publishLevel2Response**: Mesajın uygun bir temsilciye yönlendirilip iletilmediğine bakılmaksızın, platformu bilgilendirmek için bir yanıt mesajı yayınlamayı sağlar. Bu yöntem, **CloudMessageService** tarafından uygun bir temsilciye yönlendirilen iletiden sonra otomatik olarak çağrılır.
- **publishLevel3Message**: İstenen işlem başarıyla tamamlanıp tamamlanmadığını platforma bildirmek için bir mesaj yayınlar. Bu yöntem, sonuç ile işlemden sonra bir temsilci tarafından çağrılmalıdır. İsteğe bağlı olarak ayrıntılı yanıt gönderilebilir.
- **publishMessage**: Platform hizmetlerine veri göndermek için kullanılır. Örneğin, sensör verilerini göndermek için bu yöntem kullanılır.

Bölüm Yapılandırması

IoT-Ignite platformu MQTT Broker, istemci ve broker arasındaki tüm mesajlaşma için 2 konu yapısını kullanır ve tüm konular (topic) tek bir istemci için gizlidir. Bölüm yapılandırması dediğimiz bu yapıları aşağıdaki gibidir:

- {deviceId}/subscribe/EXAMPLE_TOPIC
- {deviceId}/publish/EXAMPLE_TOPIC

Her başlık tek yönlü iletişim için kullanılır. Subscribe, yani abonelikte, IoT-Ignite platformundan istemciye veri alma sağlanır. Publish işleminde ise istemciden IoT-Ignite platformuna veri gönderme işlemi sağlanır. Oturum başlatıldığında, IoT-Ignite platformunun MQTT Broker'ı otomatik olarak istemcilerin tümüne abone olur. Böylece her konuyu istemci tarafında abone

etmeye gerek yoktur. Burada bölüm yapılandırması yapılırken bölüm adlarının daima **{deviceId}/subscribe** ve **{deviceId}/publish** başladığından emin olmanız gerekiyor.

Sensör Envanterinin Sağlanması

Her bir client, IoT-Ignite platformuna en az bir kez kendi sensör envanterini vermelidir. Envanter sağlamak, platformun algılayıcı verilerini gösterge tablosunda görüntülemesine olanak tanır.

Sensör envanteri ileti biçimi aşağıda verilmiştir:

```
{
  "data": [
        {
        "nodeId": {"NodeId"},
        "things": [
        {
        "id": {"SensorId"},
        "dataType": {"INTEGER"|FLOAT|STRING},
        "vendor": VENDOR_NAME,
        "actuator": {true|false},
        "type": SENSOR_TYPE
        }
      ]
    }
  ]
}
```

Node aygıtı, üzerinde sensörler ve actuator'ler bulunan fiziksel veya sanal bir aygıttır. Bir client, sensörleri ve actuator'leri olan birden fazla node sahip olabilir. Node'lar, **data** içinde **nodeId** ve **things** dizisi ile belirtilmelidir.

Yukarıda verilen yapı içinde gelen bazı kavramlar var. Bunların açıklaması aşağıdaki gibidir:

things; bir node aygıtına bağlı olan tüm sensör ve actuator'leri içeren bir dizidir. Sensörler fiziksel veya sanal veri toplayan aygıtlar iken, aktüatörler, platformdan veya istemciden gelen girdilere göre işlem yapabilen donanımlardır.

id; node ve sensor için benzersiz olan kimlik verisini tanımlar.

dataType; sensörlerden elde edilen verilerin tipini belirtmek için kullanılır.

vendor; kullanıcılara yönelik bilgilendirici etikettir. Buraya bir sensör üreticisinin adı verilebilir.

actuator; node bağlı bileşenin bir sensör veya actuator olup olmadığını belirten boolean bir değerdir. Sensör için false, actuator için true olur.

type; sensör veya actuator için ortak bir tipi tanımlar.

Burada verdiğimiz bilgileri baz aldığımızda örnek bir node ve tek sensörlü cihaz için envanter mesajı aşağıdaki gibi olur.

```
{
  "data":[
        {
        "nodeId": "Built-in Sensors",
        "things": [
        {
        "id": "RoomTemperature",
        "dataType": "INTEGER",
        "vendor": "ARDIC",
        "actuator": false,
        "type": "Temperature"
        }
      ]
```

```
        }
      ]
    }
```

Yukarıdaki örnekte aygıtımız, dahili sensör adı verilen tek bir node aygıtına sahiptir. Node'da, **ARDIC** tarafından **INTEGER** verilerini gönderen **RoomTemperature** isimli bir sıcaklık sensörü bulunmaktadır. Burada kullanılan thing bir sensör olduğundan actuator değeri false olarak belirlenmiştir.

Envanter mesajının konuyu yayınladığından emin olmamız gerekiyor:

"{deviceId}/publish/DeviceProfile/Status/DeviceNodeInventory"

Sensör, envanter mesajını yayınladığında, önceki sensörlerinin tümünü (varsa) siler ve tüm envanterleri yeniden yazar. Bu durumda sensörlerin geçmiş verileri silinmez. Ancak bu verilere API'ler veya Dashboard üzerinden erişim sağlanamaz. Agent'ın Sensör envanterini göndermesi gerekiyorsa, göndermeden önce tüm sensörlerin Sensör Envanter Mesajı'nda olduğundan emin olmamız gerekiyor.

Node ve Sensör'lerin Bağlantı Durumlarını Yayınlama

Node ve sensörlerin bağlantı durumları cihaz üzerinden sağlanmalıdır. Sensör bağlantı durumları, bağlantı durumu mesajı verilinceye kadar "Bilinmiyor" olarak gösterilir. Bu mesaj "DeviceNodePresence" olarak isimlendirilir. Device Node Presence mesajı formatı aşağıdaki gibidir:

```
{
  "data":[
        {
        "nodeId": {NodeId},
        "descrtiption":NODE_SENSOR_DESCRIPTION,
        "connected":{1|0}
        },
        {
        "nodeId": {NodeId},
        "thingId": {SensorId},
        "descrtiption":NODE_SENSOR_DESCRIPTION,
        "connected":{1|0}
    }
  ]
}
```

Presence Message, yani durum mesajında, her node ve sensör bağlantı durumu ayrı ayrı verilmelidir. "DeviceNodePresence" mesaj parametreleri şu şekilde tanımlanır:

- **nodeId:** Biricik Node ID bilgisi.
- **thingId:** Biricik sensör ID bilgisi.
- **description:** İnsanlar tarafından okunabilen açıklama, düğüm veya sensör açıklaması tanımlamak için kullanılır.
- **connected:** Node veya sensörün bağlantı durumunu belirtir. 1: online, 0 : offline.

Burada verilen bilgilere göre örnek bir DeviceNodePresence uygulaması aşağıdaki gibidir:

```
{
  "data":[
```

```
            {
            "nodeId": "Built-in Sensors",
            "descrtiption": "OnBoard Sensors Node",
            "connected":1
            },
            {
            "nodeId": "Built-in Sensors",
            "thingId": "RoomTemperature",
            "descrtiption":"Room Temperature Sensor"
            "connected":1
          }
      ]
}
```

Burada verilen örnek uygulama bir önceki başlıkta verilen envanter mesajı uygulamasıyla uyumludur. Bu mesajın şu konuyla yayınlanması gerekiyor:

"{deviceId}/publish/DeviceProfile/Status/DeviceNodePresence"

Bağlantı durumlarını yayınladıktan sonra, Devzone Dashboard ortamında sensör ve node durumlarının "online" olduğunu görebilirsiniz.

Sensör Verileriyle Çalışma

MQTT client sensör verilerini, {deviceId}/publish/DeviceProfile/**{nodeId}/{sensorId}** konusu ile göndermelidir. IoT-Ignite MQTT broker bu çağrıya {deviceId}/subscribe/DeviceProfile/**{nodeId}/{sensorId}** konusunda (topic) yanıt göndermelidir. MQTT client kütüphanesi otomatik olarak {deviceId}/publish and {deviceId}/subscribe parçaları ile ilgilienir. Bundan dolayı bu projede yalnızca **"DeviceProfile/{nodeId}/{sensorId}"** ile ilgileneceğiz.

nodeId ve sensorId sensörümüzü tanımlamak için benzersiz kimlik verileri sağlar. Bu kimlikler için herhangi bir değer kullanabilirsiniz, ancak her seferinde aynı sensörden veri gönderirken aynı kimlik değerlerini kullandığınızdan emin olmanız gerekiyor.

IoT-Ignite hizmetleri, JSON formatında çalışmaktadır. Sensörler için veri mesaj formatı aşağıdaki gibidir:

```
{
  "data":{
          "sensorData":[
          {
          "date":TIME_IN_EPOCH_TIME_FORMAT,
          "values":[value1,value2,...,valuen]
          },
          {
          "date":TIME_IN_EPOCH_TIME_FORMAT,
          "values":[value1,value2,...,valuen]
        }
    ]
  }
}
```

Bir mesaj ile birden fazla ölçüm değeri gönderilebilir. Örneğin, ölçüm hızı 2 saniye ise, bir mesaj için dakikada 30 değer gönderebilirsiniz. Bu durumda, bant genişliğini kayıt edebilirsiniz. IoT Ignite platformu, birden fazla sensör değerlerini de destekler. Örneğin GPS koordinatlarını [longitude, latitude, altitude] veya sıcaklık değerini [temperatureInCelcius].

IoT-Ignite MQTT Client’ın Kullanımı

MQTT, MQ Telemetry Transport ifadesinin kısaltması olarak kullanılmaktadır. MQTT, publish/subscribe (yayın/abone) mantığını benimseyen, son derece basit bir mesajlaşma protokolü olan, kısıtlı cihazlar ve düşük bant genişliği için tasarlanan, yüksek gecikme süresi ve güvenilmez ağlar için geliştirilen bir iletişim protokolüdür.

IoT-Ignite platformu bulut iletişimini pratik olarak gerçekleştiren MQTT Client kütüphanesi ile bize bu protokolü kullanma imkanı sağlamıştır. Bu kütüphane Java 8 ile geliştirildi. MQTT Client kütüphanesi, geliştiricilerin kendi IoT Uygulamaları için özel sensörler ve özel agent uygulamaları oluşturmalarını sağlar. Bu başlık altında, IoT-Ignite MQTT Client kütüphanesini kendi projelerinizde nasıl kolayca kullanacağınızı öğreneceksiniz.

MQTT Client oluşturmak için şunlara ihtiyacımız var:

- IoT-Ignite Devzone üyeliği,
- Java programlama dili bilgisi,
- IoT-Ignite MQTT Client kütüphanesi. Bunu şu linkten indirebilirsiniz.

https://github.com/IoT-Ignite/java-mqtt-client

Devzone Geliştirme Ortamından Ücretsiz Servis Başlatılması

MQTT ile çalışabilmek için Devzone ortamına ücretiz bir şekilde kayıt olmanız gerekiyor. Bunun için aşağıdaki linki kullanabilirsiniz.

https://devzone.iot-ignite.com/dpanel/signup.php

Üyelik işlemiyle birlikte ücretsiz bir şekilde servislere erişim sağlayabilirsiniz.

MQTT İstemcisi Kimlik Bilgilerinin Tanımlanması

MQTT İstemcisini IoT Ignite Platformuna bağlamak için bir istemci kimliği, kullanıcı adı ve şifre gerekir. Bu işlem her Client için sadece bir kez yapılır. Kayıt işlemi, IoT-Ignite Platform tarafından sağlanan REST API'sini kullanarak basit bir web sayfası ile yapılabilir.

Kimlik bilgilerini oluşturmak için aşağıda verilen API kullanılır.

https://api.ardich.com/api/v3/device-admin/register-device

JSON biçiminde kimlik bilgilerini aşağıdaki gibi göndermeniz gerekiyor:

```
{
      "deviceId": "{DEVICE_ID}",
      "password": "{PASSWORD}",
      "username": "{USERNAME}"
}
```

Yukarıda bazı parametrelerimiz bulunmaktadır. Bunları açıklamakta fayda var:

- **DEVICE_ID:** MQTT İstemcisi için benzersiz bir cihaz kimliği.
- **USERNAME:** Mqtt istemcisi için benzersiz kullanıcı adı.
- **PASSWORD:** Kullanıcı Şifresi.

Device MQTT kimlik bilgileri, Unix benzeri işletim sistemlerinden aşağıdaki komutu ile kaydedilebilir.

```
$ curl -X POST -d 'deviceId={DEVICE_ID}&username=USERNAME&password=PASSWORD&' \
                  -H “Authorization: Bearer {ACCESS-TOKEN}” \
         > https://api.ardich.com/api/v3/device-admin/register-device
```

ACCESS_TOKEN parametresi **IoT-Ignite Oauth2 Service Authorization**’a göre doldurulmalıdır.

MQTT İstemci Uygulanmasının Derlenmesi

MQTT Client kütüphanesi Maven ile kolayca derlenebilir. Burada Maven’i bağımlılık yönetimi (dependency management) ve derleme için kullanacağız.

Maven POM Dosyası ile Örnek Projenin Başlatılması

Maven ile çalışırken MQTT kütüphanesi bağımlılıklarını **pom.xml**'e ekleyerek örnek bir proje başlatabiliriz.

```
    4.0.0
            com.example
            MQTTExample
            0.0.1-SNAPSHOT
            jar

            MQTTExample

            org.apache.maven.plugins
            maven-compiler-plugin
            3.5.1

                1.8
                1.8

            maven-assembly-plugin

                com.example.MQTTExample.App


                jar-with-dependencies

                make-assembly
                package

                    single

            com.ardic.mqtt
```

```
29            mqtt-client-wrapper
30            1.0.0
31
32            com.google.code.gson
33            gson
34            2.8.0
35
36            repo.iotignite
37            https://repo.iot-ignite.com/content/groups/public/
38
39            package com.ardic.android.connectionwatcher;
```

Rastgele Sayı Üreten Servisin Oluşturulması

Rastgele sayı üreten bir servis oluşturmak için aşağıdakileri kullanmamız gerekiyor.

```
1     import java.util.Random;
2     import java.util.Date;
3
4     import com.ardic.mqtt.client.service.MessagePublisherService;
5     import com.google.gson.Gson;
6
7     public class RandomValueCollector {
8         public static RandomValueCollector service = new RandomValueCollector();
9
10        private RandomValueCollector() {
11            collectRandomValue();
12        }
13
14        public static RandomValueCollector getInstance() {
15            return service;
16        }
17
18        private void collectRandomValue() {
19            MessagePublisherService publisher = MessagePublisherService.getInstance();
20            while (true) {
21                Random randomGenerator = new Random();
22                int value = randomGenerator.nextInt(1000);
23                sendSensorValue(publisher, value);
24                try {
25                    Thread.sleep(30000);
26                } catch (InterruptedException e) {
27                    e.printStackTrace();
28                }
29            }
30        }
31
32        private void sendSensorValue(MessagePublisherService publisher, int value) {
33        }
34    }
```

Üretilen değeri IoT-Ignite'a göndermek için **sendSensorValue** metodunu kullanacağız.

Sensör Veri Modelinin Tanımlanması

Sensör verileri mesajı oluşturmak için Google Gson kütüphanesini kullanacağız. Google Gson ile Java nesneleri kolay bir şekilde JSON formatına dönüştürülür. Bu işlemi yapabilmek için 3 tane sınıf oluşturmamız gerekiyor.

SensorDataValue.java

```
1     public class SensorDataValue {
2
```

```
3         private long date;
4         private T[] values;
5
6         public SensorDataValue(long time, T[] values) {
7             this.date = time;
8             this.values = values;
9         }
10
11        public long getDate() {
12            return date;
13        }
14
15        public void setDate(long date) {
16            this.date = date;
17        }
18
19        public T[] getValue() {
20            return values;
21        }
22
23        public void setValue(T[] values) {
24            this.values = values;
25        }
26    }
```

SensorDataPackage.java

```
1     public class SensorDataPackage {
2
3         private SensorDataValue[] sensorData;
4
5         public SensorDataValue[] getSensorData() {
6             return sensorData;
7         }
8
9         public void setSensorData(SensorDataValue[] sensorData) {
10            this.sensorData = sensorData;
11        }
12
13    }
```

SensorMessage.java

```
1     public class SensorMessage {
2
3         private SensorDataPackage data;
4
5         public SensorDataPackage getData() {
6             return data;
7         }
8
9         public void setData(SensorDataPackage data) {
10            this.data = data;
11        }
12
13    }
```

Yayın için oluşturacağımız mesaj için yukarıda verilen üç modeli kullanalım. Bunun için **RandomValueCollector.java** içinde oluşturduğumuz **sendSensorValue** metodunu aşağıdaki gibi yazalım.

```
1     private void sendSensorValue(MessagePublisherService publisher,int value){
```

```
        Integer[]valueArray={value};
        SensorDataValue[]dataValue=new SensorDataValue[1];
        dataValue[0]=new SensorDataValue(new Date().getTime(),valueArray);
        SensorDataPackage dataPackage=new SensorDataPackage();
        dataPackage.setSensorData(dataValue);
        SensorMessage message=new SensorMessage();
        message.setData(dataPackage);

        publisher.publishMessage("DeviceProfile/Built-in Sensors/RandomData",
        new Gson().toJson(message));
    }
```

SendSensorValue metodu **DeviceProfile/Built-in Sensors/RandomData** konusuna sahip bir mesaj yayınlamayı sağlar. Ayrıca bu kullanım ile Node, **dahili sensör (Built-in-Sensors)** olarak, sensör ise **rasgele veri (RandomData)** olarak isimlendirilir.

Sensör Envanter Modelinin Tanımlanması

Sensör değerlerini IoT-Ignite Platformuna göndermeden önce Sensör eEnvanteri mesajında **RandomData** sensörü sağlamalıyız. Bunun için aşağıda verilen sınıflara ihtiyacımız var.

Sensor.java

```
    public class Sensor {
        private String id;
        private String dataType;
        private String vendor;
        private boolean actuator;
        private String type;

        public Sensor(String id, String dataType, String vendor, boolean actuator,
String type) {
            this.id = id;
            this.dataType = dataType;
            this.vendor = vendor;
            this.actuator = actuator;
            this.type = type;
        }

        public String getId() {
            return id;
        }

        public String getDataType() {
            return dataType;
        }

        public String getVendor() {
            return vendor;
        }

        public boolean isActuator() {
            return actuator;
        }

        public String getType() {
            return type;
        }
    }
```

Node.java

```
1    import java.util.List;
2
3    public class Node {
4
5        private String nodeId;
6        private List things;
7
8        public Node(String nodeId, List things) {
9            this.nodeId = nodeId;
10           this.things = things;
11       }
12
13       public String getNodeId() {
14           return nodeId;
15       }
16
17       public void setNodeId(String nodeId) {
18           this.nodeId = nodeId;
19       }
20
21       public List getThings() {
22           return things;
23       }
24
25       public void setThings(List things) {
26           this.things = things;
27       }
28   }
```

ListSensorData.java

```
1    import java.util.List;
2    public class ListSensorData {
3
4        private List data;
5        public List
6        getData() {
7            return data;
8        }
9
10       public void setData(List data) {
11           this.data = data;
12       }

13   }
```

Node ve Sensör'lerin Bağlantı Durumlarının Yayınlanması

IoT-Ignite kontrol paneli, sensörlerin ve node aygıtlatın bağlantı durumunu gösterebilir. Bağlantı durumunu sağlamak için IoT-Ignite bulutuna "Device Node Presence" mesajı göndermelisiniz. Bu mesajda aynı node'daki tüm node'ları ve sensörleri bağlantı durumlarıyla listelemeliyiz. Bu mesajda liste öğeleri oluşturmak için **Thing.java** kullanacağız.

```
1    public class InventoryItem {
2
3        private String nodeId;
4        private String sensorId;
5        private String description;
6        private int connected;
```

```
7
8         public InventoryItem(String nodeId, String sensorId, String description,
int connected) {
9             this.nodeId = nodeId;
10            this.sensorId = sensorId;
11            this.description = description;
12            this.connected = connected;
13        }
14
15        public String getNodeId() {
16            return nodeId;
17        }
18
19        public void setNodeId(String nodeId) {
20            this.nodeId = nodeId;
21        }
22
23        public String getSensorId() {
24            return sensorId;
25        }
26
27        public void setSensorId(String sensorId) {
28            this.sensorId = sensorId;
29        }
30
31
32        public String getDescription() {
33            return description;
34        }
35
36        public void setDescription(String description) {
37            this.description = description;
38        }
39
40        public int getConnected() {
41            return connected;
42        }
43
44        public void setConnected(int connected) {
45            this.connected = connected;
46        }
47
48    }
```

Yukarıdaki sınıf içinde kullandığımız parametreler şunlardır:

- **nodeId:** Benzersiz Node kimliği.
- **sensorId:** Benzersiz sensör kimliği.
- **description:** Node veya sensör açıklaması. Opsiyoneldir.
- **connected:** Node veya sensör için bağlantı durumu. 1:online, 0: offline.

Main Class'ı Oluşturmak

Yukarıda anlattıklarımızdan sonra son olarak yapmanız gereken son işlem, MQTT için bir **main** class oluşturmaktır. Bu sınıfın amacı; MQTT oturumunu başlatmaktır. Sonrasında **sensorInventory** bilgileri gönderir ve Iot-Ignite platformuna sensör değeri göndermek için **RandomValueGenerator** başlatılır.

App.java

```
1     import java.util.ArrayList;
2     import java.util.List;
3
```

```
4     import com.ardic.mqtt.client.service.MessagePublisherService;
5     import com.ardic.mqtt.client.service.SessionService;
6     import com.google.gson.Gson;
7
8     public class App {
9         static boolean connectionClose = false;
10
11        public static void main(String[] args) {
12            SessionService session = SessionService.getInstance();
13            if (session.connect()) {
14                sendSensorInventory();
15                initiateAgents();
16
17                while (!connectionClose) {
18                    try {
19                        Thread.sleep(1000);
20                    } catch (InterruptedException e) {
21                        e.printStackTrace();
22                    }
23                }
24                session.disconnect();
25                System.exit(0);
26            } else {
27                System.exit(1);
28            }
29        }
30
31        private static void sendSensorInventory() {
32            // Initialize Thing list for DeviceNodePresence message
33            ListSensorData thingList = new ListSensorData();
34            thingList.setData(new ArrayList());
35
36            // Initialize sensor list
37            List sensorList = new ArrayList();
38
39            Sensor randomSensor = new Sensor("RandomSensor", "INTEGER", "ARDIC", false,
"Random Sensor");
40            sensorList.add(randomSensor);
41
42            List nodeList = new ArrayList();
43            Node builtinNode = new Node("Built-in Sensors", sensorList);
44            nodeList.add(builtinNode);
45
46            /* Add "Built-In Sensors" node and "RandomSensor" as online to thingList */
47            thingList.getData().add(new InventoryItem(builtinNode.getNodeId(), null,
"", 1));
48            thingList.getData()
49                    .add(new InventoryItem(builtinNode.getNodeId(),
randomSensor.getId(),
50                     randomSensor.getType(), 1));
51
52            ListSensorData inventory = new ListSensorData();
53            inventory.setData(nodeList);
54
55            // Publish DeviceNodeInventory
56            MessagePublisherService.getInstance().publishMessage(
57                    "DeviceProfile/Status/DeviceNodeInventory",
58                    new Gson().toJson(inventory));
59
60            // Publish DeviceNodePresence
61            MessagePublisherService.getInstance().publishMessage(
62                    "DeviceProfile/Status/DeviceNodePresence",
63                    new Gson().toJson(thingList));
64        }
65
66        private static void initiateAgents() {
67            RandomValueCollector.getInstance();
68        }
69    }
```

MQTT İstemciyi Çalıştırmak

Tüm işlemleri yaptıktan ve build işlemini gerçekleştirdikten sonra maven, hedef klasörde **MQTTExample-1.0.0-with-dependencies.jar** doyasını oluşturacaktır. Bu .jar uzantılı dosyayı çalıştırmak için MQTT kimlik bilgileri sağlanmalıdır. Bunun için **mqtt.properties** dosyasını aşağıdaki gibi oluşturunuz.

```
MQTT_USERNAME = {mqtt-username}
MQTT_PASSWORD = {mqtt-password}
MQTT_CLIENT_ID = {mqtt-client-id}
```

Yapılandırma dosyasını oluşturduktan sonra jar dosyasını aşağıdaki komutu kullanarak çalıştırın.

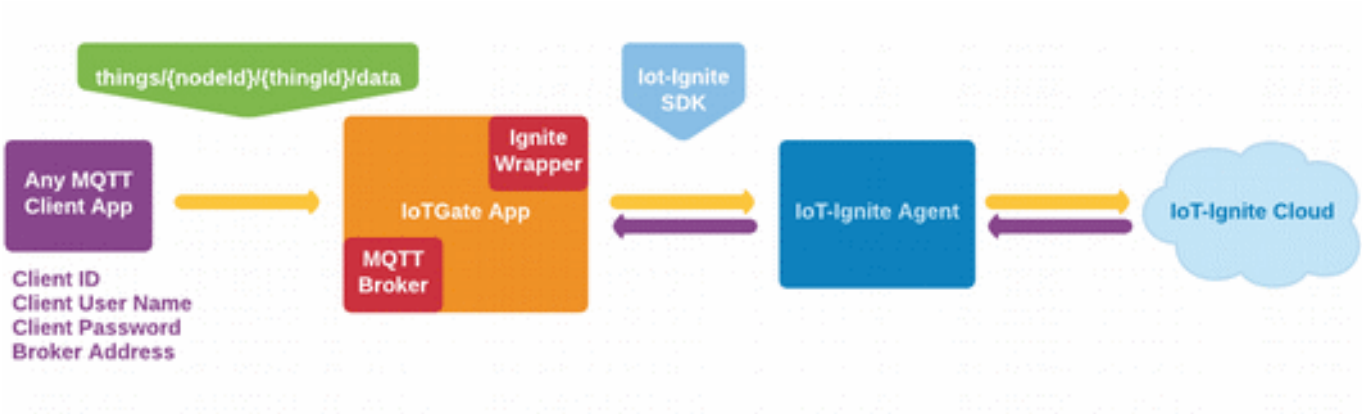
```
$ java -jar MQTTExample-1.0.0-jar-with-dependencies.jar
-Dmqtt.properties="./mqtt.configurations"
```

IoTGate Uygulaması

IoTGate uygulaması, herhangi bir platformdaki MQTT Client'ınız ile iletişim kuran, IoT-Ignite platformu bağlantısı kurmanıza yardımcı olacak bir **MQTT Broker** uygulamasıdır.

MQTT ile ilgili detaylı bilgiyi 1. bölümde **Veri Protokolleri** başlığı altında bulabilirsiniz. Bu bölümde IoTGate uygulaması anlatılacaktır.

IoTGate uygulaması çalışma şekli aşağıdaki şekilde gösterilmiştir.



Uygulamayı Kullanmaya Başlamadan Önceki Gereksinimler

1. Android Gateway (telefon veya tablet): OS Versiyonu, KitKat, Lollipop, Marshmallow, Nougat olabilir.
2. Android Gateway'inize kuracağınız IoT-Ignite Agent uygulaması.

   https://play.google.com/store/apps/details?id=com.ardic.android.iotigniteagent

3. IoT-Ignite platform hesabı. Henüz bir hesabınız yoksa aşağıdaki link'ten oluşturabilirsiniz:

   https://devzone.iot-ignite.com/dpanel/signup.php?page=development

4. Android Gateway'iniz, IoT-Ignite hesabınıza lisanslanmış olmalıdır. Bu işlem 6. bölümde "User" başlığı altında anlatılmıştır.

5. Örnek MQTT Client. Biz bu dokümanda **MqttBox Google Chrome** plug'inini kullanacağız. Linux, Windows ve Mac OS uyumlu versiyonları mevcut.

   https://chrome.google.com/webstore/detail/mqttbox/kaajoficamnjijhkeomgfljpicifbkaf

Başlayalım...

IoTGate uygulamasını Android Gateway'inize yükleyin.

IoTGate.apk uygulama dosyasını IoT-Ignite hesabınızda **App Store** kısmına ekleyin ve T**rusted App** olarak işaretleyin. Nasıl trusted yapacağınızı öğrenmek için aşağıdaki link'e atabilirsiniz:

https://devzone.iot-ignite.com/knowledge-base/making-your-app-trusted/

Cihazınızı kablosuz bir ağa bağlayın ve IoTGate uygulamasını açın. Wireless, IoT-Ignite ve MQTT Broker ikonları renkli olmalıdır. MQTT Broker'ın IP'si App Bar üzerinde ve MQTT ikonunun altında yazacaktır. Her şey hazır olduğunda uygulama aşağıdaki gibi görünecektir.

IoTGate uygulaması, bağlantı kuracak client'lar için hazır.

MQTT Client Uygulamasını Trusted Yapmak

IoTGate uygulamasını kullanmak için, IoT-Ignite Cloud hesabımızdan (EHUB'dan) hangi Client'ın bağlanacağını belirlememiz gerekir. Bu bir güvenlik önlemidir.

Sensor Type Configuration ve ardından da **Ardic Generic Authenticator** aşağıdaki gibi seçilir.

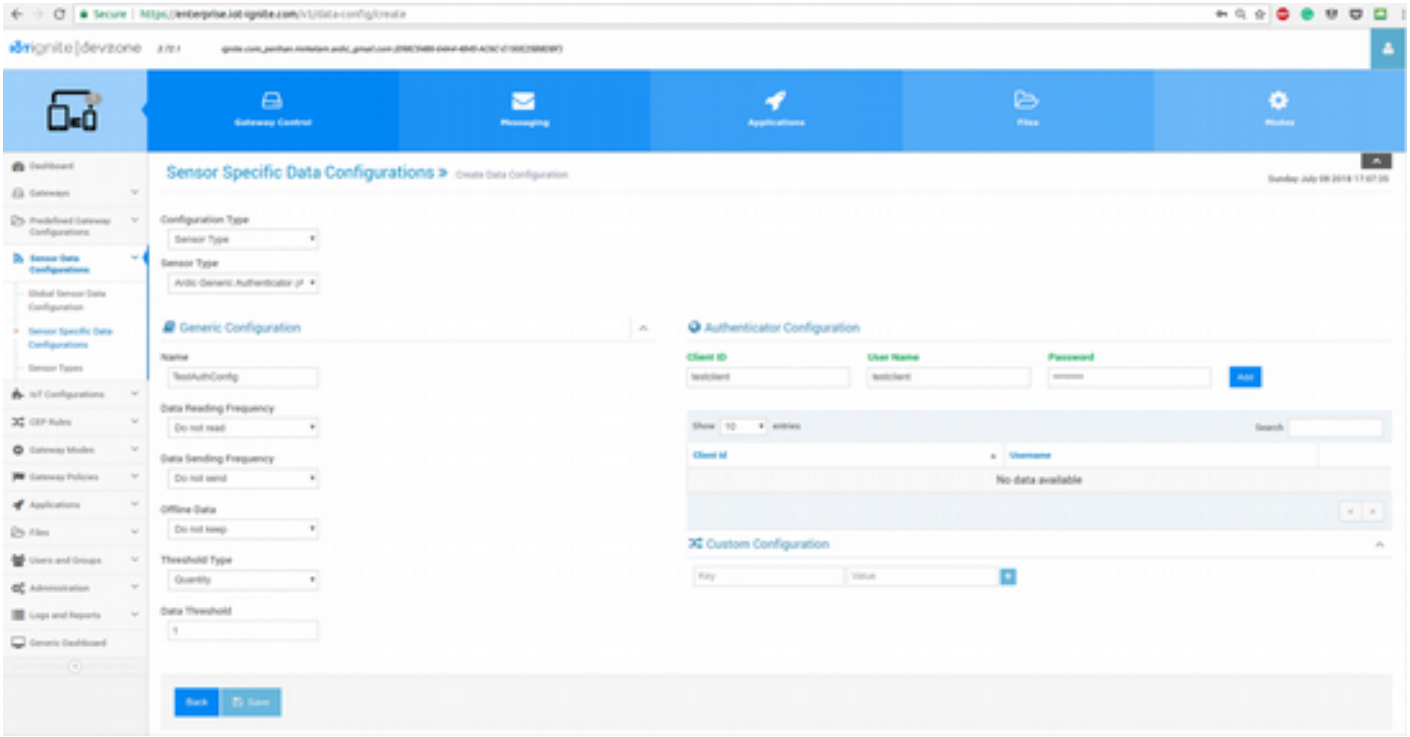
Bilgileri aşağıdaki gibi tanımlayabiliriz:

clientid: testclient

username: testclient

password: test12345

Örnek MQTT Client kimlik doğrulama konfigürasyonu aşağıdaki şekilde görülmektedir.



Konfigürasyona bir isim verin ve kaydedin. Yapmanız gereken son şey, bu konfigürasyonu Mode'a ekleyip cihazınıza göndermek.

Tüm ayarlar bu kadardı, şimdi bu bilgilere sahip herhangi bir MQTT Client, MQTT Broker'ınıza bağlanabilir. Bu son işlemlerle ilgili bir sorun yaşadıysanız, örnek kimlik doğrulama işlemleri için bu videoya göz atabilirsiniz:

https://www.youtube.com/watch?v=NLWnYMhHo98

Topic Yazım Şekli

Kimlik bilgilerini MQTT Client'ınıza uygulayıp Broker'a bağlanmalısınız. IoTGate için Topic yazım standartları aşağıdaki gibidir:

1. IoT-Ignite Cloud'a veri göndermek için Topic:

   things/{nodeId}/{thingId}/data

2. IoT-Ignite Cloud'tan konfigürasyon mesajı almak için Topic:

   things/{nodeId}/{thingId}/config

Bu yazım şekli ile bir kere **subscribe** ve **publish** data yaptığınızda IoTGate uygulaması;

- **{nodeId}** kısmında belirlediğiniz Node ID'nizi node olarak,
- **{thingId}** kısmında belirlediğiniz Thing IDnizi thing olarak IoT-Ignite Cloud hesabınıza kaydeder.

Bundan sonra kayıtlı thing'leriniz için konfigürasyon eklemeyi unutmayın! Aksi halde Client'ınız Broker'a bağlanabilir, ancak IoT-Ignite Cloud'a veri gönderemez.

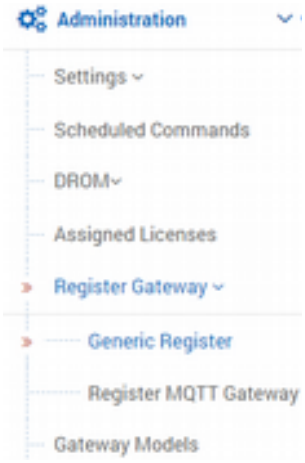
Bu anlatımı kendi MQTT Client'ınıza uygulayabileceğiniz gibi, MqttBox Google Chrome plug'inini kullanarak örnek bir kullanım için bu videoya göz atabilirsiniz:

https://www.youtube.com/watch?v=PX5usk96-zI

IoTGate-MQTT Uygulama Kullanımı

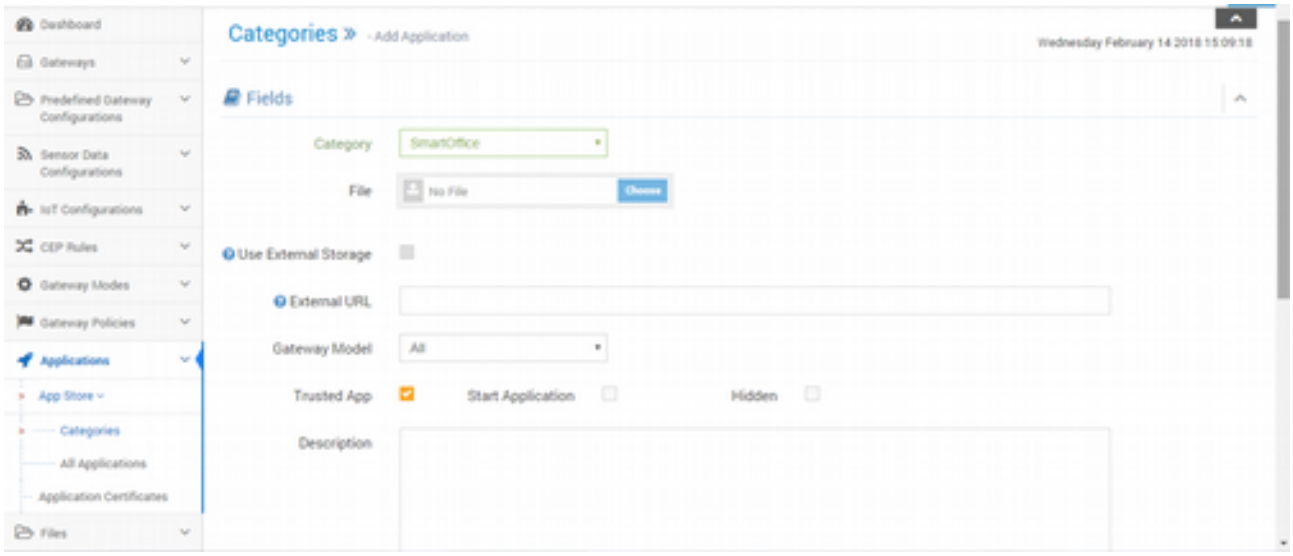
IoTGate-MQTT, Android bir cihazın IoT-Ignite bulut sistemi ve Node cihazları arasında bir Gateway olarak çalışmasını sağlamak için geliştirilen bir Android uygulamasıdır. IoTGate-MQTT uygulaması, Node'lar arasındaki iletişim için MQ Telemetri Aktarımı (MQTT) protokolünü kullanır.

Android yüklü bir telefon veya tablete IoTGate-MQTT uygulamasını kurduktan sonra, IoT-Ignite platformunda cihazınızı kaydetmeniz gerekir. Bunun için https://enterprise.iot-ignite.com/v3/ adresine gidip giriş yaptıktan sonra, aşağıda verilen yolu (Administration -> Register Gateway-> Generic Register) takip ediniz.

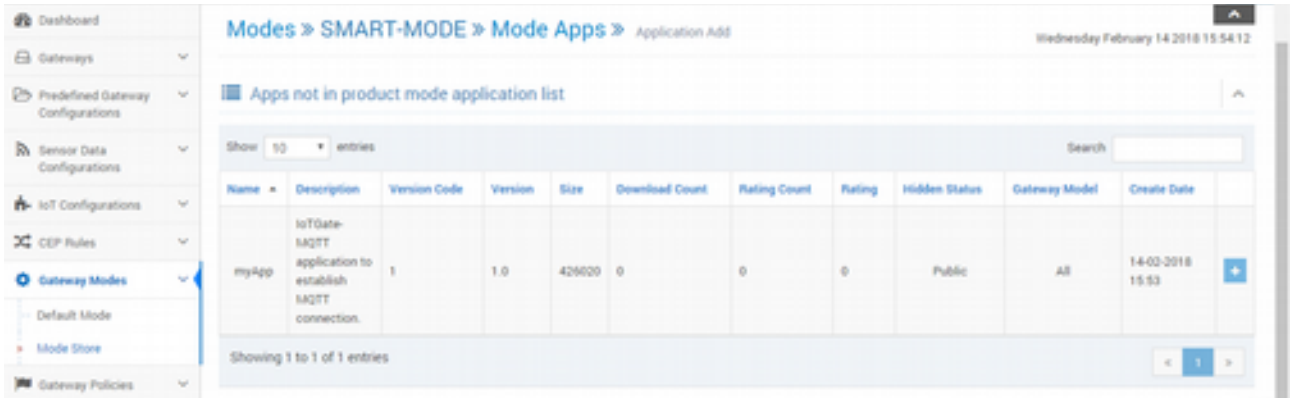


Açılan pencereden kullanıcı adınızı seçip, size gösterilen QR kodunu IoT-Ignite Agent ile taratınız.

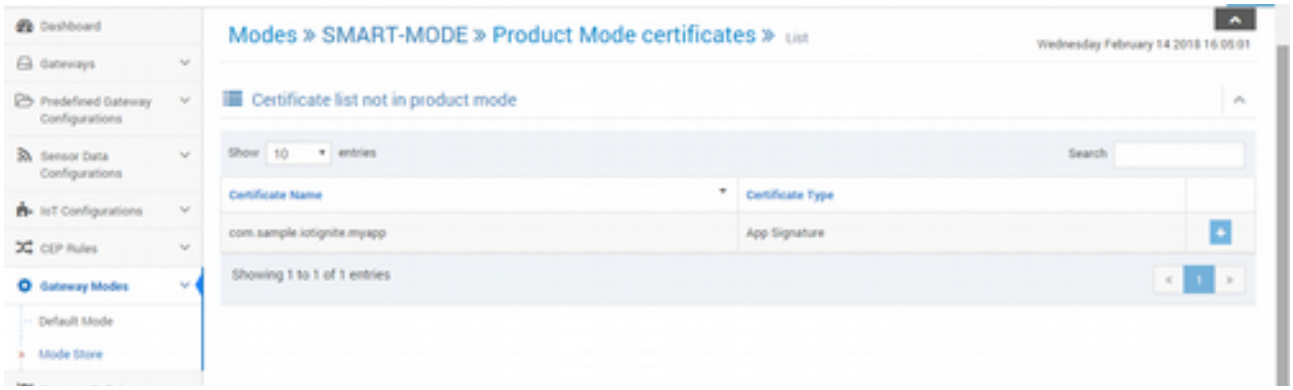
Bunu tamamladıktan sonra, bu uygulamayı IoT-Ignite'da doğrulamanız gerekir. Bunun için **Application > App Store > Categories** yolunu takip ediniz, **Add Application** seçeneği ile uygulamayı ekleyebilir veya eklemeden önce **Create Category** ile yeni bir kategori oluşturabilirsiniz. **Category** açılır listesinden bir kategori seçin, **.apk**'inizin yolunu seçin, **Trusted App** onay kutusunu işaretleyin ve **Description** metin kutusuna açıklama ekleyin, ardından sağ alt köşedeki **Upload** butonuna tıklayıp işlemi tamamlayınız.



Son adımda mod ekleyip bunu Gateway'e göndereceğiz**. Gateway Modes > Default > Applications > Add Application** yolunu takip ederek mod ekranına geçelim. Uygulamanın yanında bulunan + simgesine tıklayınız.

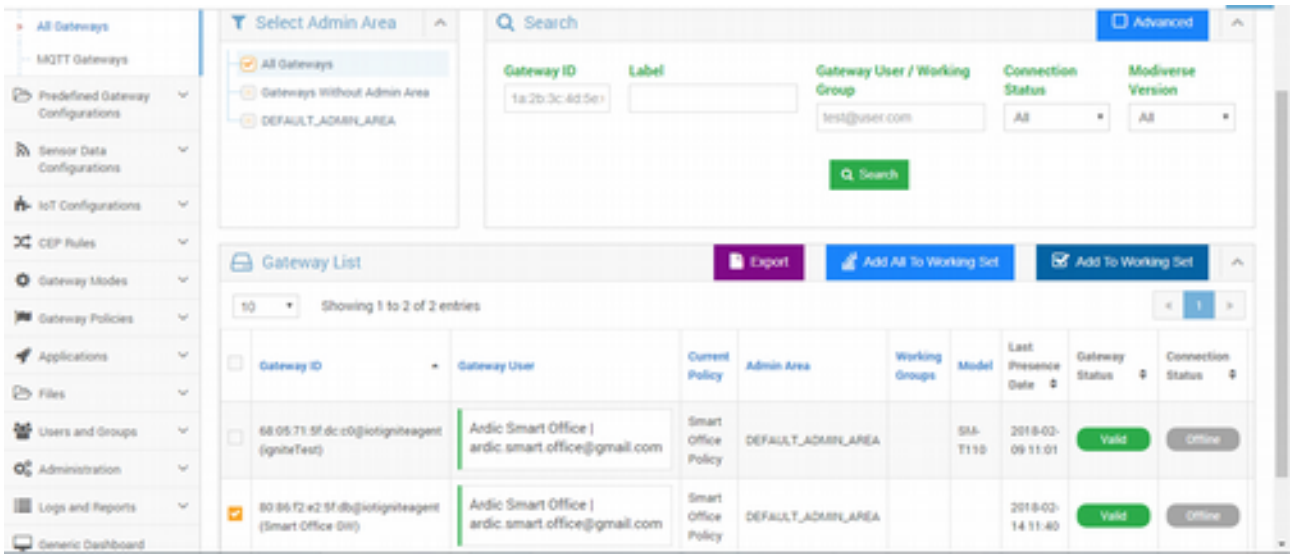


Sonrasında **Modes > Default** yolunu takip ediniz, **Application Certificates** bölümünü seçiniz. **Add Certificate** seçeneğine tıklayınız. Uygulamanın sol tarafında bulunan + simgesine tıklayıp **Yes** deyiniz.

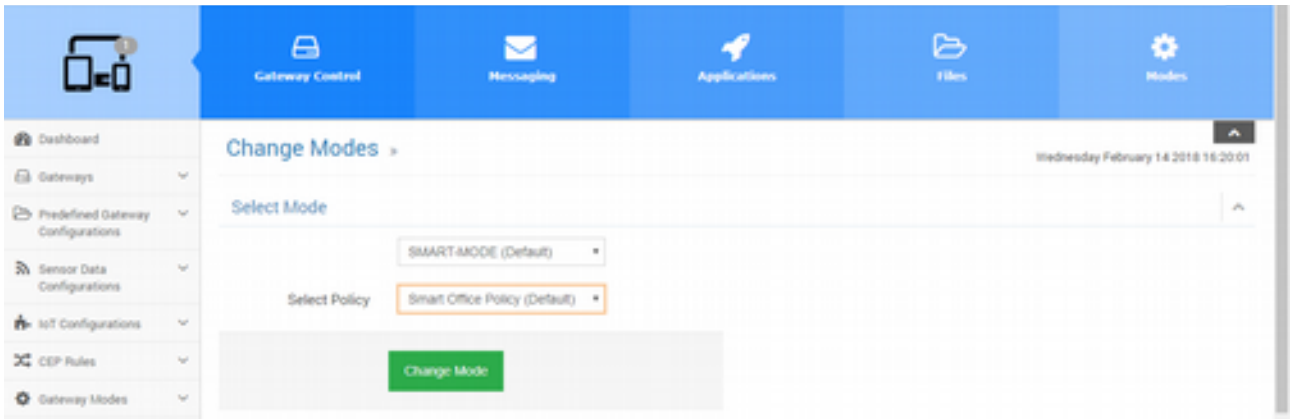


Bunları yaptıktan sonra oluşturduğumuz modu Gateway'e göndermemiz gerekiyor. Bunun için **Gateways-> All Gateways** yolunu takip ediniz. Listeden cihazınızı seçip **Add To Working Set**

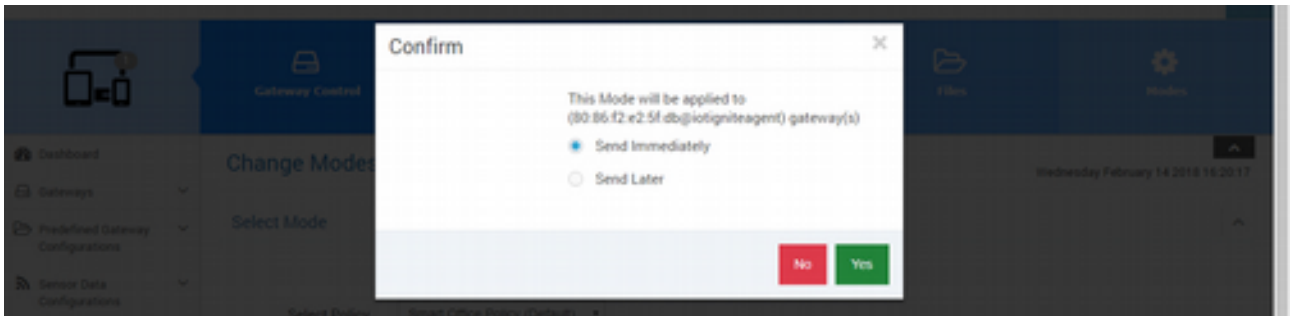
seçeneğine tıklayınız. Açılan pencereden **Clear and Add** seçeneğine tıklayınız.



Ekranın sağ üst köşesinde bulunan **Modes** seçeneğine tıklayınız. Ardından modunuzu ve politikanızı seçip **Change Mode** seçeneğine tıklayınız.



Açılan yeni pencereden **Send Immediately** seçeneğini seçip **Yes** deyiniz.



Bunları yaptıktan sonra cihazınız kullanıma hazır hale gelir.

Sıcaklık Durumunu Tweet & IFTTT ile Google Drive WeMos ile Göndermek

Bu ilk projemizde sıcaklık durumunu ölçüp, değeri nasıl Tweet atacağımızı göstereceğiz.

Donanım Gereksinimleri

Sıcaklık bilgisini ölçen bu proje için ihtiyacımız olan donanımlar aşağıdaki gibidir:

- 1 adet WeMos D1 Mini Pro Board
- 1 adet WeMos DHT 11 Shield

Yazılım ve Servis Gereksinimleri

Projenin geliştirilmesinde ihtiyacımız olan diğer bir gereksinim şüphesiz yazılım ve servislerdir. Bu uygulama için aşağıda verilen yazılım ve servisleri öncelikle temin etmemiz gerekiyor.

- Arduino IDE
- IoT-Ignite Servis Kaydı

Proje Konusu Hakkında

Bu proje, DHT 11 sensörü tarafından üretilen sıcaklık ve nem değerlerinin toplanmasını ve gerçek zamanlı olarak web panosunda görselleştirmeyi sağlamaktadır. Sensörden toplanan veriler MQTT üzerinden IoT-Ignite platformuna gönderilir. IoT-Ignite platformu gelen verilerin görselleştirmesini sağlamak için kullanılır.

IoT-Ignite, IoT cihazlarının izlemesini ve kontrol edilmesini sağlayan açık bir platformdur. Hem kişisel kullanım hem de IoT uygulamalarına ilk defa başlayanlar için ücretsizdir.

Burada verilen işlemleri tamamladıktan sonra, sıcaklık veya nem sensör değerleri belirlenmiş bir eşiği aşarsa bir bildirim alabileceksiniz. Ayrıca gelen bir bildirim ile alarmları etkinleştirebilir veya devre dışı bırakabilir, mevcut sıcaklığı veya nemi sınır değerler ile karşılaştırabilir ve belirli bir değer aşıldığında IoT-Ignite'de bir olay yayınlayabilirsiniz.

Bu proje hızlı, basit bir sıcaklık ve nem monitörüdür. Özellikle bebek odası, sunucu odası vb. için yararlı olabilir.

Veri Akış Modeli

DHT11 sensörü Wemos veya NodeMCU kartına bağlanır. Wemos, ESP8266 ile WiFi ağına bağlanır. Daha sonra verileri MQTT protokolü aracılığıyla IoT-Ignite sunucusuna iletir. Veriler, yerleşik özelleştirilebilir Dashboard kullanılarak görselleştirilir. Wemos üzerinde çalışan uygulama oldukça basit olup Arduino SDK kullanılarak yazılmıştır.

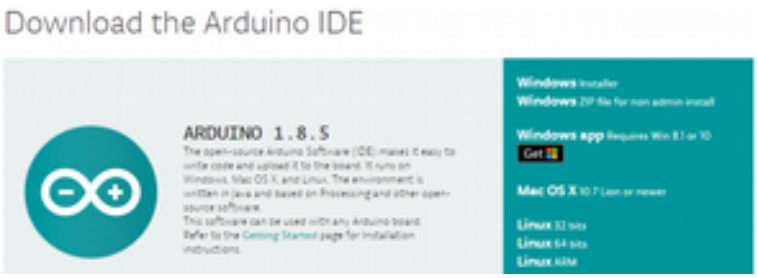
Yapılandırmalar ve İşlem Adımları

Arduino IDE kullanarak Arduino programlamanın temellerini biliyorsanız, ilk adımı atlayabilir ve 2. adımla projeye devam edebilirsiniz.

Adım 1: Arduino IDE

WeMos programlamaya başlamak için Arduino IDE'yi ve ilgili tüm yazılımı kurmalısınız. Uygulamayı indirmek için aşağıdaki linki kullanabilirsiniz.

https://www.arduino.cc/en/Main/Software



Kurulum dosyasını yukarıdaki sayfadan indirebilirsiniz. Amacımız Windows ortamında kurulum işlemi yapmak olduğundan dolayı, Windows için geliştirilen setup dosyasını seçmemiz gerekiyor.

Adım 2: Arduino Kütüphanelerinin Kurulumu

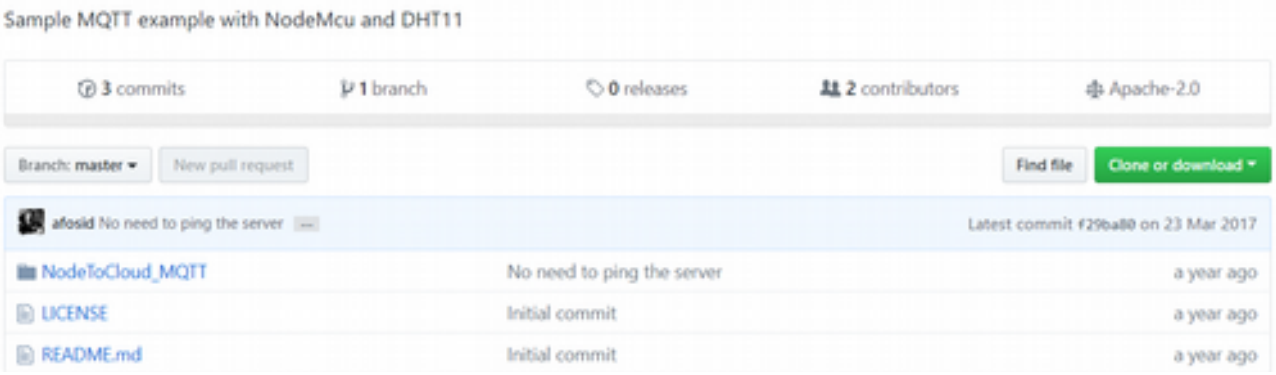
Arduino IDE kurulumunu yaptıktan sonra uygulamayı çalıştırınız ve **Sketch** > **Include Library** > **Manage Libraries** yolunu takip ediniz. Açılan sayfada aşağıdaki kütüphaneleri bulup kurulum işlemini başlatınız:

- Time kütüphanesi için:
  https://github.com/PaulStoffregen/Time
- ESP8266WiFi kütüphanesi için:
  https://github.com/ekstrand/ESP8266wifi
- Adafruit_MQTT kütüphanesi için:
  https://github.com/adafruit/Adafruit_MQTT_Library
- ArduinoJson kütüphanesi için:
  https://github.com/bblanchon/ArduinoJson
- DHT kütüphanesi için:
  https://github.com/adafruit/DHT-sensor-library
- WiFiUdp kütüphanesi için:
  https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/WiFiUdp.h

Adım 3: Taslakların Hazırlanması

Öncelikle aşağıda verilen linkten taslağı indirip Arduino IDE ile açınız.

https://github.com/IoT-Ignite/sample-mqtt-NodeToCloud



Bu taslak, uygulama için ihtiyacımız olan kodları temin etmektedir. Ancak bazı ayarları kendimizin yapması gerekiyor. Taslak üstünde aşağıdaki sabitler ve değişkenleri düzenlememiz gerekiyor.

- **DHTPIN D4:** Sensörün bağlı olduğu WeMos DHT pini.

```
#define DHTPIN D4  // Bağlı olduğumuz dijital pin
```

- **WLAN_SSID:** Kablosuz erişim noktanısın adı.
- **WLAN_PASS:** Erişim noktası şifresi.

```
#define WLAN_SSID "GUEST"
#define WLAN_PASS "12345"
```

Bir diğer ayar da MQTT Server ayarlarıdır. MQTT için şu ayarları yapmamız gerekiyor.

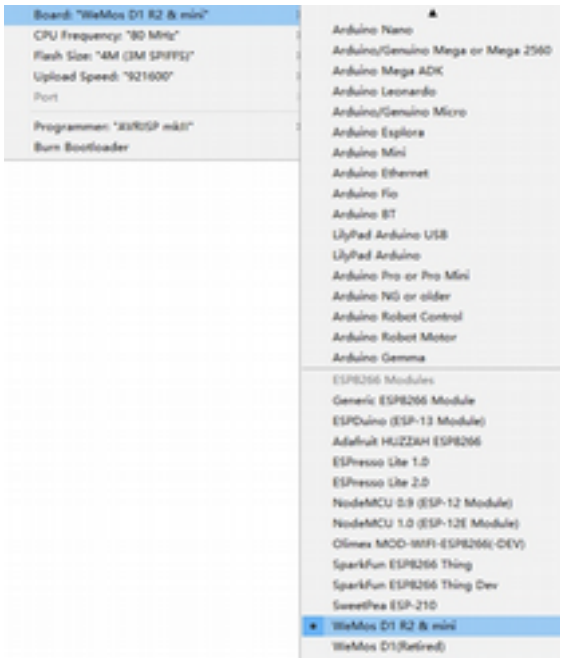
- **ARB_USERNAME:** MQTT hesabının kullanıcı adı.
- **ARB_PW:** MQTT hesabının şifresi.
- **DEVICE_ID:** WeMos için benzersiz ID bilgisi.
- **NODE_ID:** Node adı.

```
#define ARB_SERVER "mqtt.ardich.com"
#define ARB_SERVERPORT 8883 // SSL için 8883 kullanın
#define ARB_USERNAME "mqttuser"
#define ARB_PW "a12345678"
```

- **delayTime:** Sunucuya veri göndermeden önce ne kadar bekleneceği belirlenir. Buradaki sürenin birimi milisaniyedir.

```
//IoT-Ignite'a veri göndermeden önce 20 saniye beklenir.
int delayTime = 20000;
```

Yukarıdaki işlemleri yaptıktan sonra WeMos cihazınızı USB kablosuyla bilgisayara bağlayın ve Arduino IDE'de **WeMos** portunu seçin. **Upload** butonunu kullanarak kodunuz derleyin ve cihaza yükleyin.



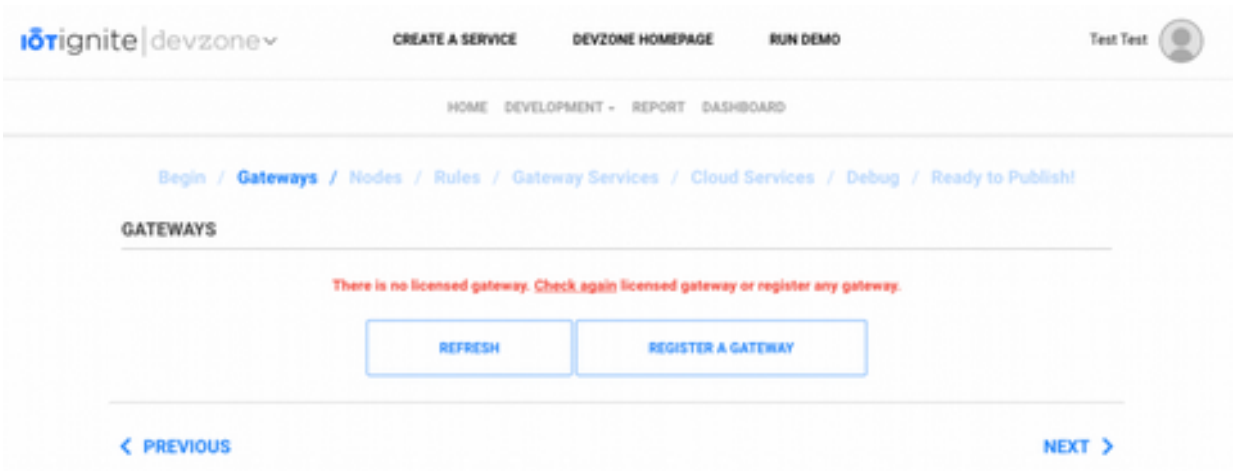
Uygulama yüklendikten ve cihaz başlatıldıktan sonra IoT-Ignite, MQTT istemcisini kullanarak Node'a bağlanmayı ve "sıcaklık" ve "nem" verilerini almayı deneyecektir.

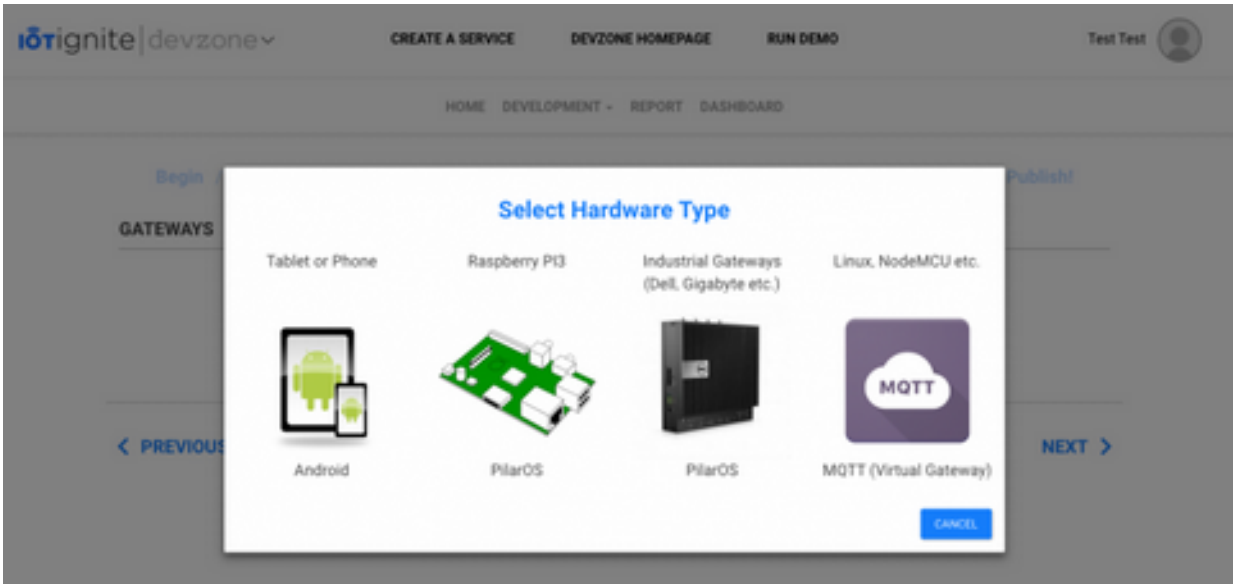
Adım 4: IoT-Ignite Devzone Hesap Açma ve MQTT Lisanslama

Hesap açmak ve MQTT lisanlaması yapmak için aşağıdaki adresten kayıt olun ve ücretsiz olarak geliştirici hesabına giriş yapınız.

https://devzone.iot-ignite.com/dpanel/login.php?page=development

Kodunuzu yükledikten sonra, sensör tiplerini yapılandırmanız gerekiyor. Bunun için **Enterprise** panelinize giriş yapın ve aşağıda gösterilen işlemleri sırasıyla gerçekleştiriniz.


IOTignite | devzone
CREATE A SERVICE
DEVZONE HOMEPAGE
RUN DEMO
Test Test
HOME DEVELOPMENT REPORT DASHBOARD
Begin / Gateways / Nodes / Rules / Gateway Services / Cloud Services / Debug / Ready to Publish!
GATEWAYS
There is no licensed gateway. Check again licensed gateway or register any gateway.
REFRESH
REGISTER A GATEWAY
PREVIOUS
NEXT

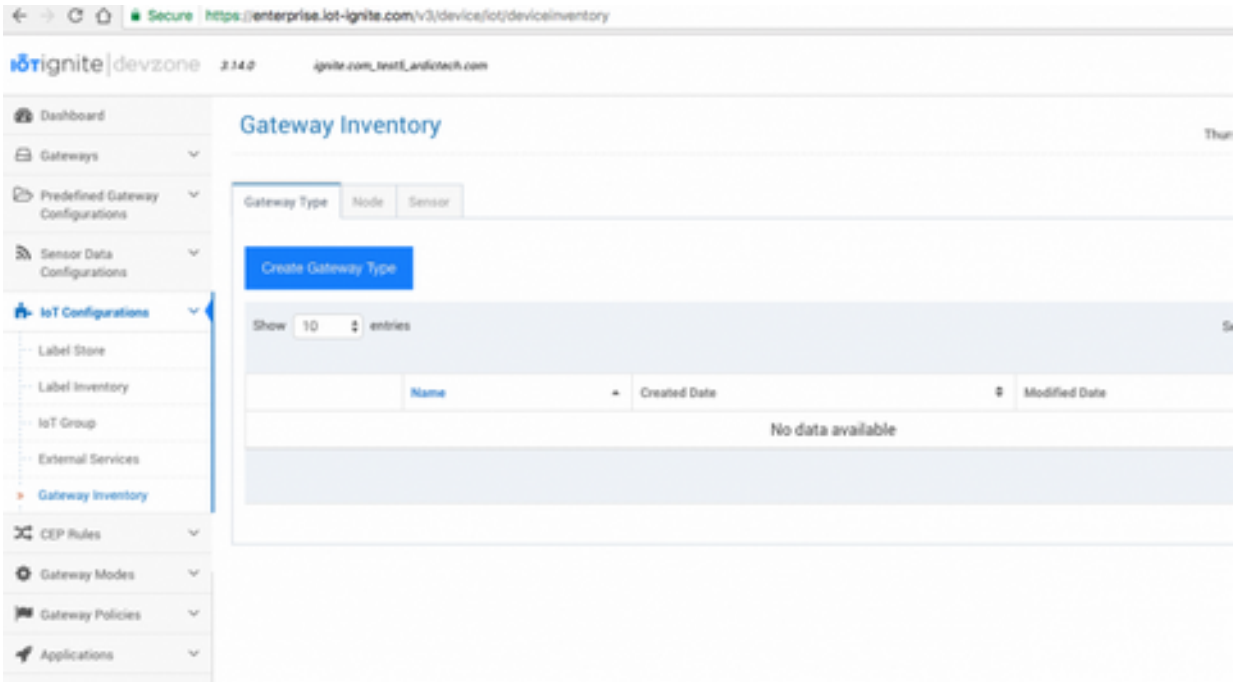

IOTignite | devzone
CREATE A SERVICE
DEVZONE HOMEPAGE
RUN DEMO
Test Test
HOME DEVELOPMENT REPORT DASHBOARD
GATEWAYS
Select Hardware Type
Tablet or Phone
Raspberry PI3
Industrial Gateways (Dell, Gigabyte etc.)
Linux, NodeMCU etc.
MQTT
Android
PilarOS
PilarOS
MQTT (Virtual Gateway)
CANCEL
PREVIOUS
NEXT


GATEWAYS
Register Your Gateway
DEVICE ID *
10:68:3f:4b:b8:58
USERNAME *
MqttUsername
PASSWORD *
CONFIRM PASSWORD *
REGISTER
CANCEL
PREVIOUS
NEXT
Powered by ARDIC | All rights reserved
Version: 1.26.1

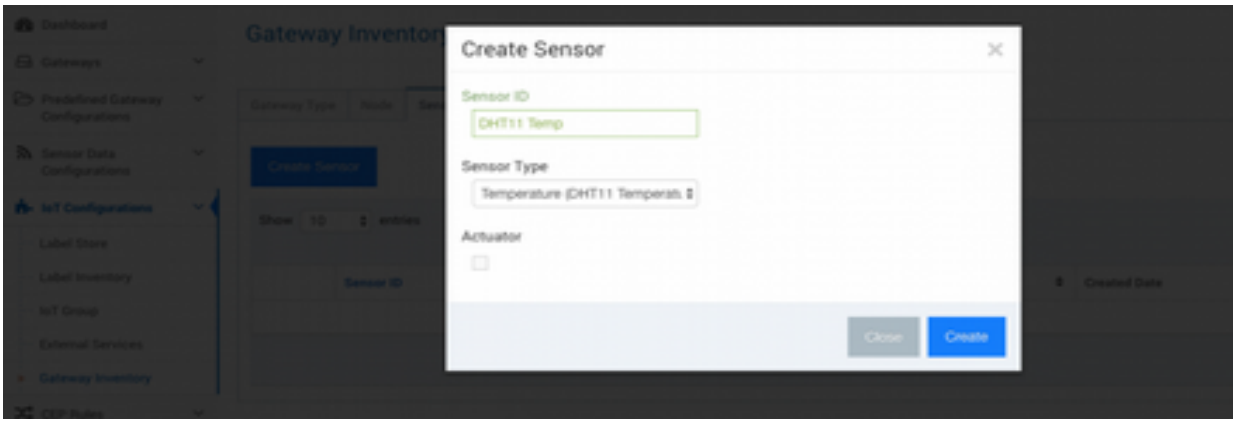
Bu son işlemle birlikte MQTT lisanslaması başarıyla gerçekleşmiş olur.

Adım 5: Sensör Yapılandırmaları

IoT-Ignite'a giriş yaptıktan sonra sensörleri yapılandırmamız gerekiyor. Bunun için **IoT Configuration > Gateway Inventory and Sensor Tab** yolunu takip ediniz.



Yukarıdaki pencereden **Sıcaklık** sensörünü yapılandırmak için aşağıdaki işlemi yapınız. **Sensör ID** alanına **DHT11 Temp** yazın ve **Sıcaklık** listesinden **sensör tipini** seçiniz.



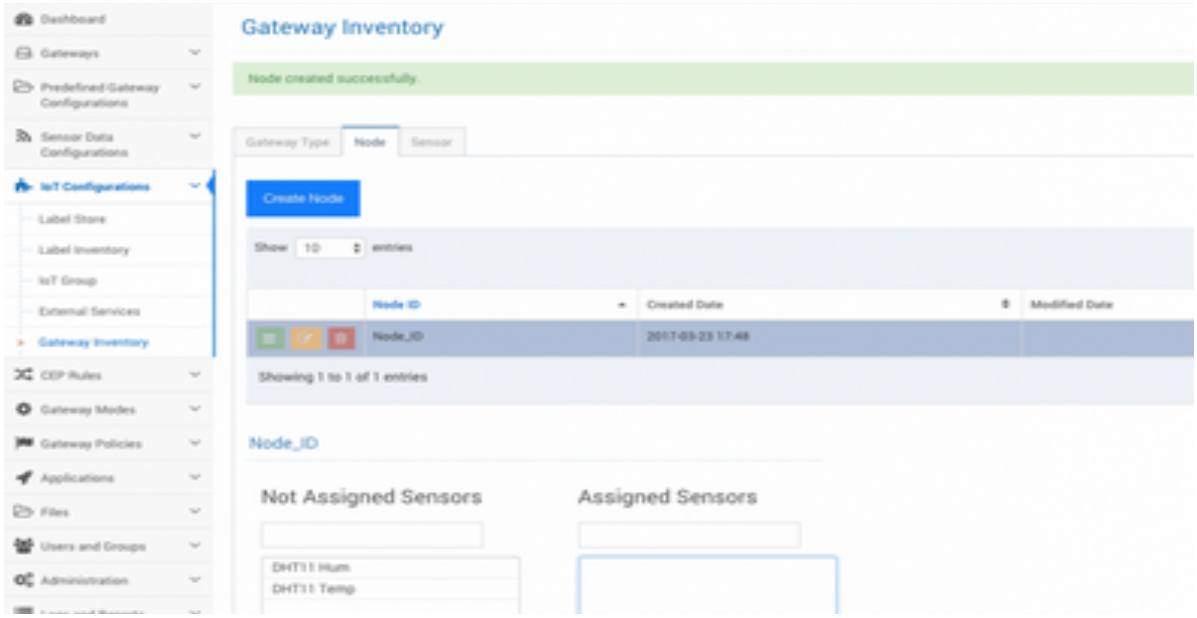
Nem sensörü içinde **Sensör ID** alanına **DHT11 Hum** yazın ve sensör tipi listesinden **Humiditiy**'yi seçin.

Tüm bu işlemlerden sonra sensörlerin yapılandırma işlemi tamamlanmış olur. Sensör tanıtımı yapılmadan verileri alamayız.

Adım 6: Node Yapılandırmaları

IoT-Ignite'ta sensör yapılandırmasını yaptıktan sonra **Node** yapılandırması yapmamız gerekiyor. Bunun için **IoT Configuration > Gateway Inventory and Node Tab** yolunu takip ediniz.

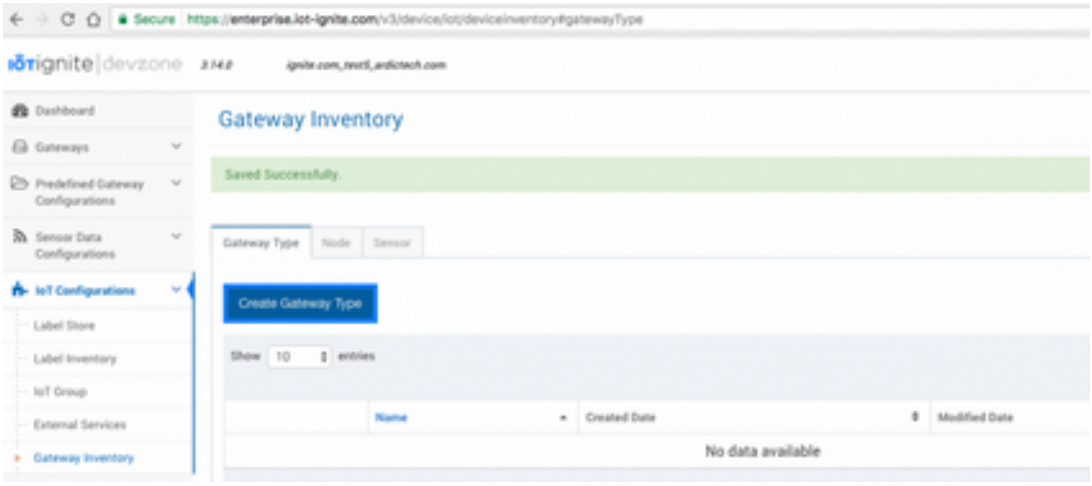
Node ID alanına **Node_ID** yazın ve **Create** butonuna basınız.



Node’a sensörleri atamak için sarı renkli **Edit** butonuna tıklayın. **DHT11 Hum** ve **DHT11 Temp** sensörlerini seçin ve kaydedin. Bu işlemler birlikte Node ve Sensor ataması IoT-Ignite tarafında tamamlanmış olur.

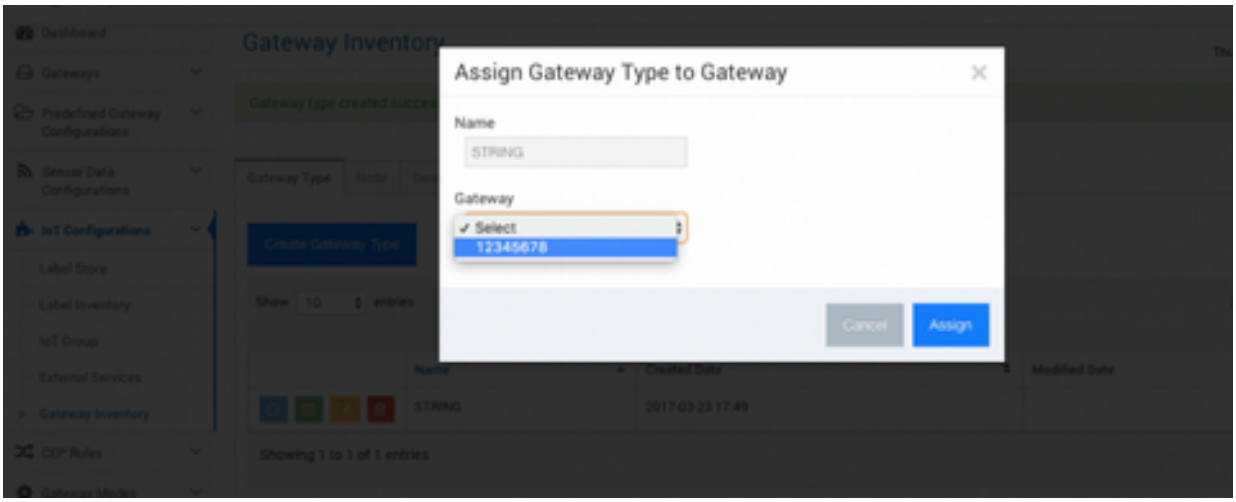
Adım 7: Gateway Envanteri

IoT-Ignite’de Node yapılandırmasını yaptıktan sonra Gateway ayarlarınız yapmamız gerekiyor. Bunun için **IoT Configuration > Gateway Inventory** yolunu takip ediniz.



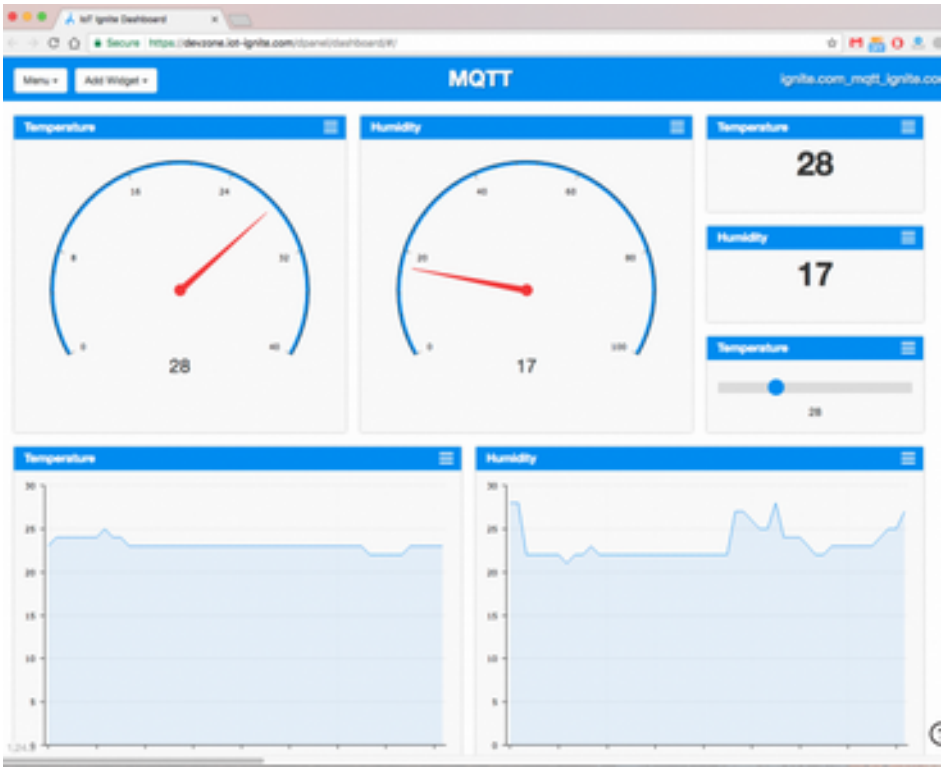
Yukarıda açılan pencerede görülen **Create Gateway Type** butonuna tıklayınız. Aşağıda açılan sayfada **Name** alanına **STRING** ifadesini giriniz.

Son olarak oluşturulan Gateway’i seçin ve atama düğmesine tıklayın.



Dashboard’ta Sensörleri Tanımlamak ve Grafiklerle Verileri Anlık İzleme

Devzone'un iki görsel raporlama aracı vardır. Rapor veya gösterge tablosu aracını kullanabilirsiniz. **Rapor** sekmesi basit veri listesi ve görselleştirme aracıdır.

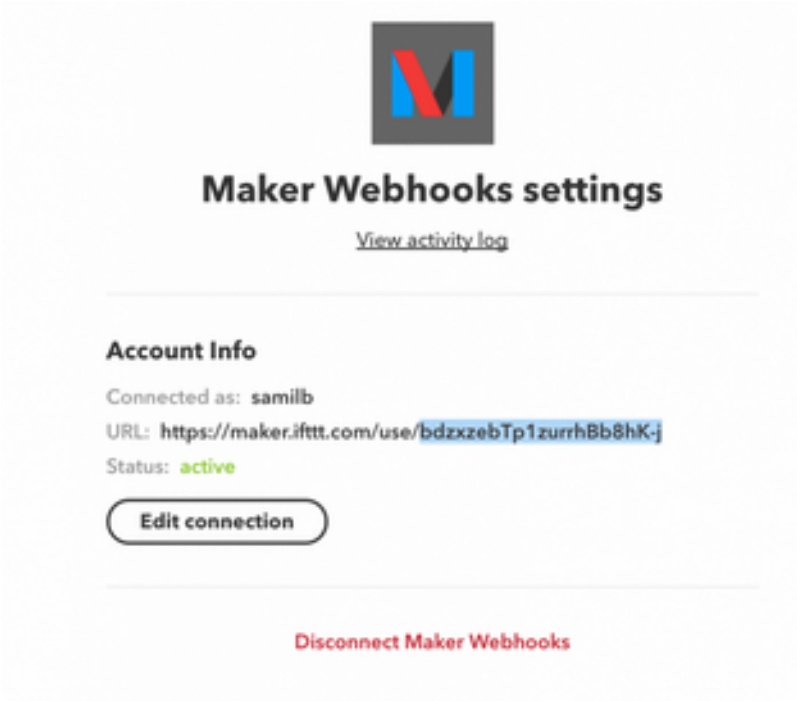


Dashboard, sensör verilerini neredeyse gerçek zamanlı ve etkileşimli olarak alır. Ayrıca sezgisel sürükle ve bırak arayüzünü kullanarak verilerinizi dakikalar içinde görselleştirebilirsiniz. Dashboard'ı açtığınızda çeşitli araç türlerini kullanabilir ve konfigürasyon seçeneklerini değiştirebilirsiniz.

IFTT Bağlantısı ve Google Drive Entegrasyonu

IoT-Ignite platformunda bu işlemi yapabilmek için **IoT Configurations > External Services** yolunu takip ediniz ve aşağıdaki adımları gerçekleştiriniz:

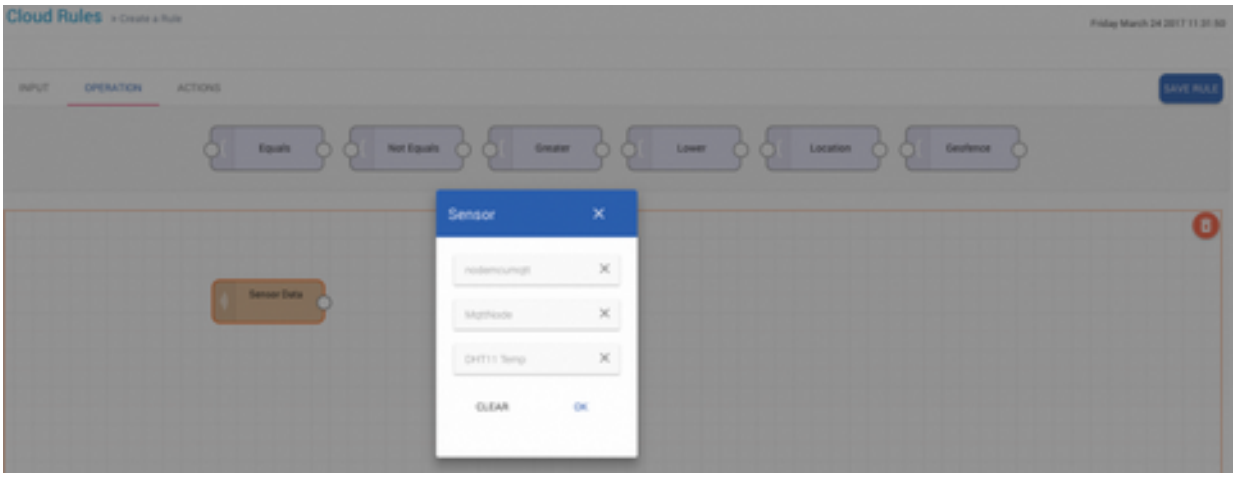
- IFTTT hesabınızı giriniz, **Maker Channel API**'sı oluşturup anahtarı kopyalayın.
- **API** anahtarı bilgilerini yapıştırın ve **CEP Rules > Cloud Rules** menüsüne gidiniz.
- **New Cloud Rule** butonuna tıklayınız ve veri değişimi için kurulum kuralı yapılandırmasını yapınız.
- IFTTT hesabına gidin ve yapımcı kanalı ve Google sürücü kanalıyla uygulama oluşturun.

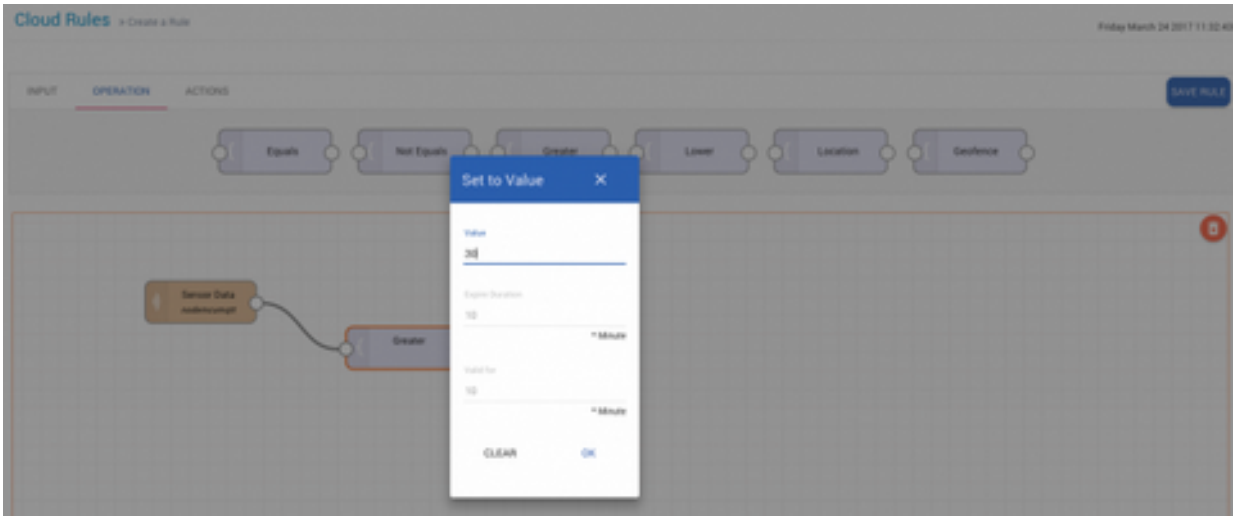


Twitter Bağlantısı Sağlamak ve Sensör Verilerini Twitter'a Göndermek

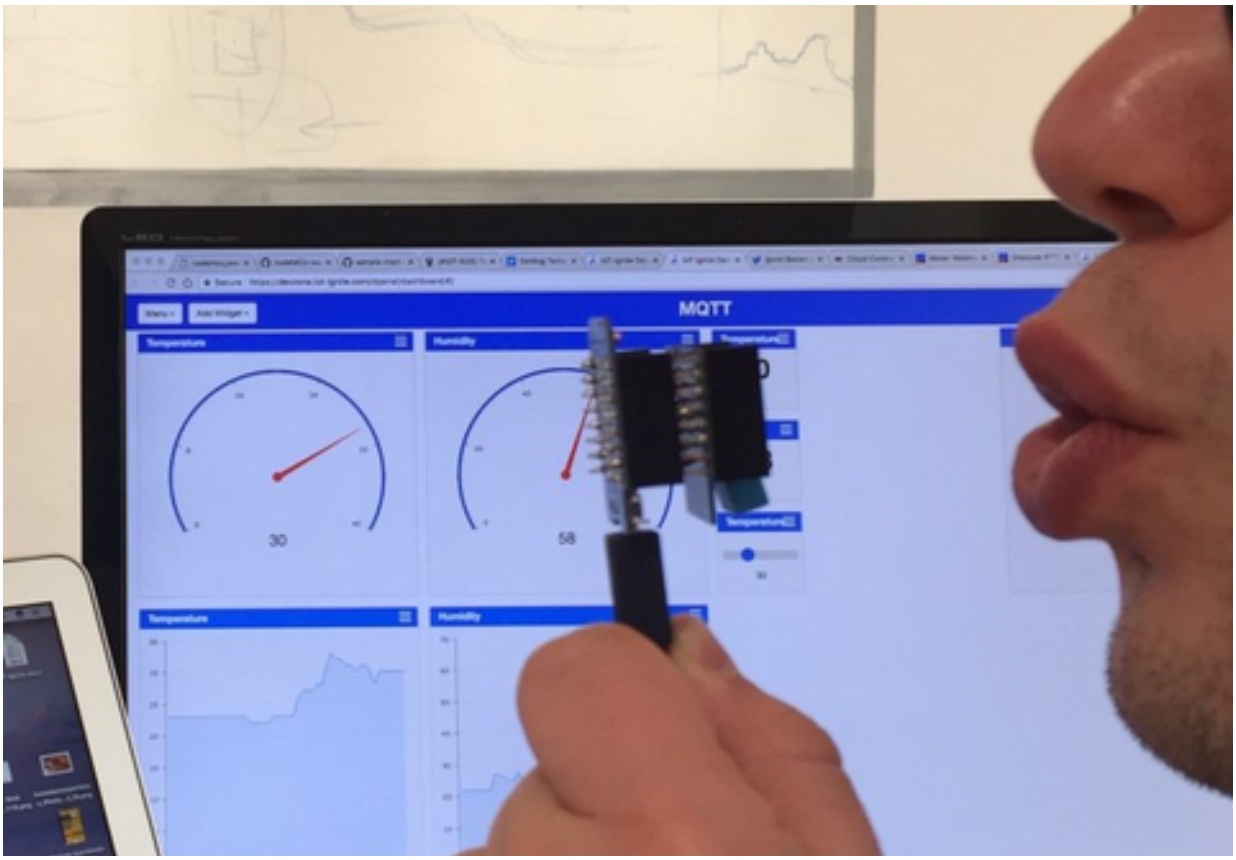
Twitter bağlantısını IoT-Ignite'dan sağlamak için **IoT Configurations > External Services** yolunu takip ediniz ve **Twitter** tabına tıklayınız. Twitter sekmesi altında şu işlemleri sırasıyla yapınız.

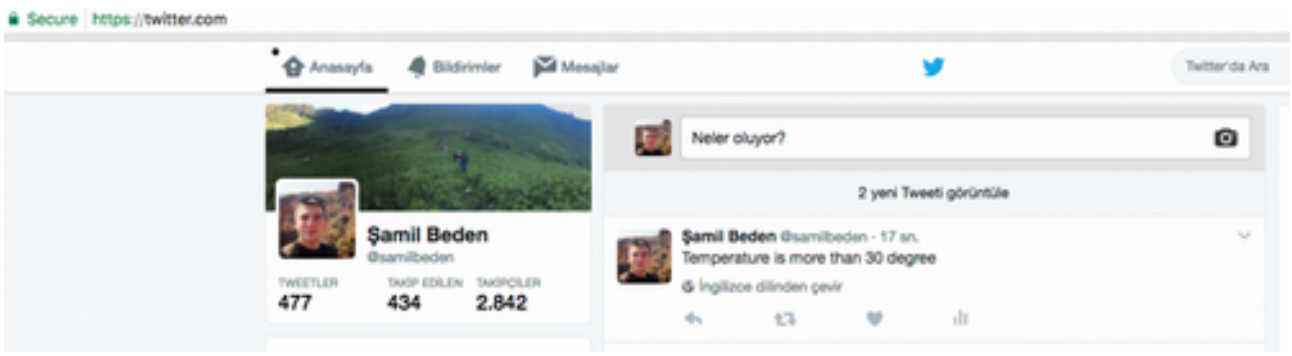
- Twitter hesabınıza giriş yapmak için **connect** butonuna tıklayın ve IoT-Ignite uygulamasına izin verin.
- **CEP Rules > Cloud Rules** menüsüne gidiniz.
- **New Cloud Rule** butonuna tıklayınız ve tweet için kural yapılandırmasını oluşturunuz.
- **Sıcaklık** için bir kural oluşturup test ediniz.





Yukarıdaki işlemleri yaptıktan sonra test aşamasına geçebiliriz.





Yukarıda görüleceğiz üzere sıcaklık değiştiğinde otomatik olarak tweet atılması sağlanır.

Android Cihaz ile Çevresel Sensör Bilgilerini Edinme (Sıcaklık ve Nem Sensörleri)

Bu projemizde Android bir cihazı gateway olarak kullanıp sıcaklık ve nem verilerini edinmeye çalışacağız.

Donanım Gereksinimleri

Bu proje için ihtiyacımız olan donanımlar şunlardır:

- NodeMCU ESP8266 Breakout Board x1 adet
- DHT11 Temperature & Humidity Sensor x1 adet

- LED (generic) x 1 adet
- Resistor 221 ohm
- Android tablet ya da telefon x 1 adet
- Breadboard x 1 adet
- Button (generic) x 1 adet

Yazılım ve Servis Gereksinimleri

İhtiyacımız olan yazılım ve servsileri de aşağıdaki gibidir:

- Arduino IDE
- IoT-Ignite Service Registration
- IoT-Ignite Agent APP
- IoT-Ignite Service APP

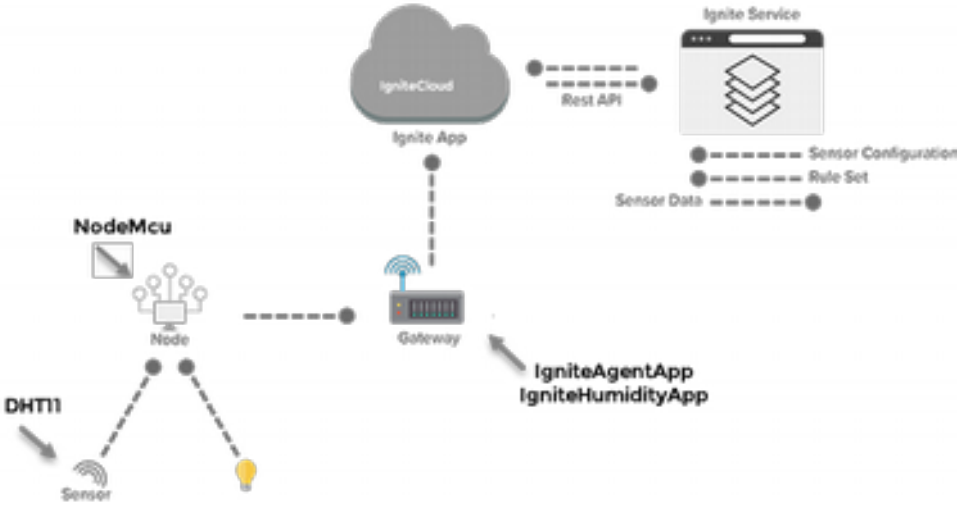
Proje Konusu Hakkında

Projemizin konusu NodeMCU üzerinden kablosuz olarak DHT11 sensöründen elde edilen verilerin aktarımını hedeflemektedir. Bu aktarım işleminde Android cihazı ağ geçidi olarak kullanılacaktır. Ayrıca tanımlanan bir kural ile LED'in kontrol edilmesi sağlanacaktır.

Veri Akış Modeli

Uygulamanın daha iyi anlaşılması için projenin veri akış modeli aşağıdaki gibi olacaktır. Bu akış modelinden aşağıdaki çıkarımlarda bulunabiliriz:

- NodeMCU bir erişim noktası olacaktır.
- AP modunda cihazı sisteme kaydedin ve kablosuz bağlantı kurulacaktır.
- İstemci bağlantısı ayarlandığında, sıcaklık ile nem iletiminin frekansı okunur ve verileri bu frekansta müşteriye iletmeye başlar.
- Bağlantı kesilirse ağa yeniden bağlanmaya çalışılır.
- LED için tanımlanan olay meydana geldiğinde LED’in çalışması sağlanır.



Yapılandırmalar ve İşlem Adımları

Android cihaz aracılığıyla sıcaklık ve nem verilerini alabilmek için şu yapılandırma ayarlarınız sırasıyla yapmamız gerekiyor.

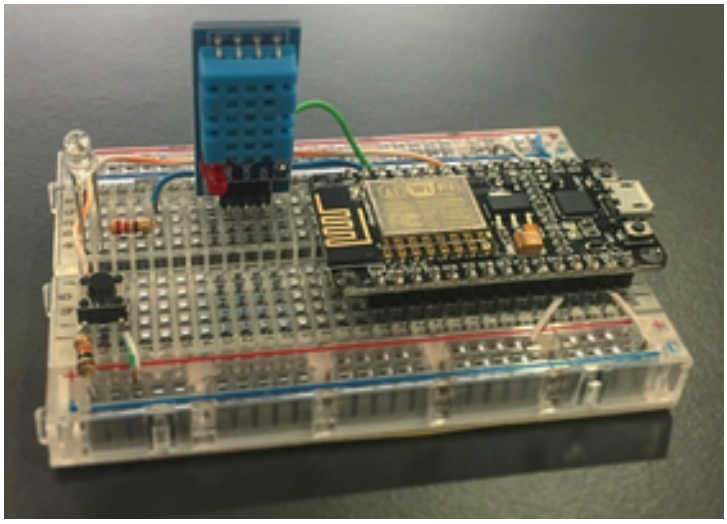
Adım 1: Android Cihazı IoT Gateway Olarak Tanımlamak

Android Tabletimizi Gateway olarak kayıt etmek için şu adımları takip etmeliyiz.

- Şu bağlantıdan http://iotapp.link/ Agent uygulamasını indirip yükleyin.
- https://devzone.iot-ignite.com/dpanel/login.php?page=development linkini kullanarak Devzone ortamına kayıt işlemini yapıp sisteme giriş yapın.
- Bir servis oluşturun.
- **Register a Gateway**'e tıklayarak açılan sayfadan aşağıda görüleceği üzere Android Phone seçeneğine tıklayın.
- QR kodunu tara butonuna tıklayın ve Android tablet ile QR kodunu tarayın. Bunu yaptıktan sonra birkaç saniye içinde cihazınız bir Gateway olarak sisteme kayıt edilir.

Adım 2: Gateway Kaydı için NodeMCU

NodeMCU için kayıt işlemine başlamadan önce aşağıda görüleceği üzere devremizi oluşturalım.



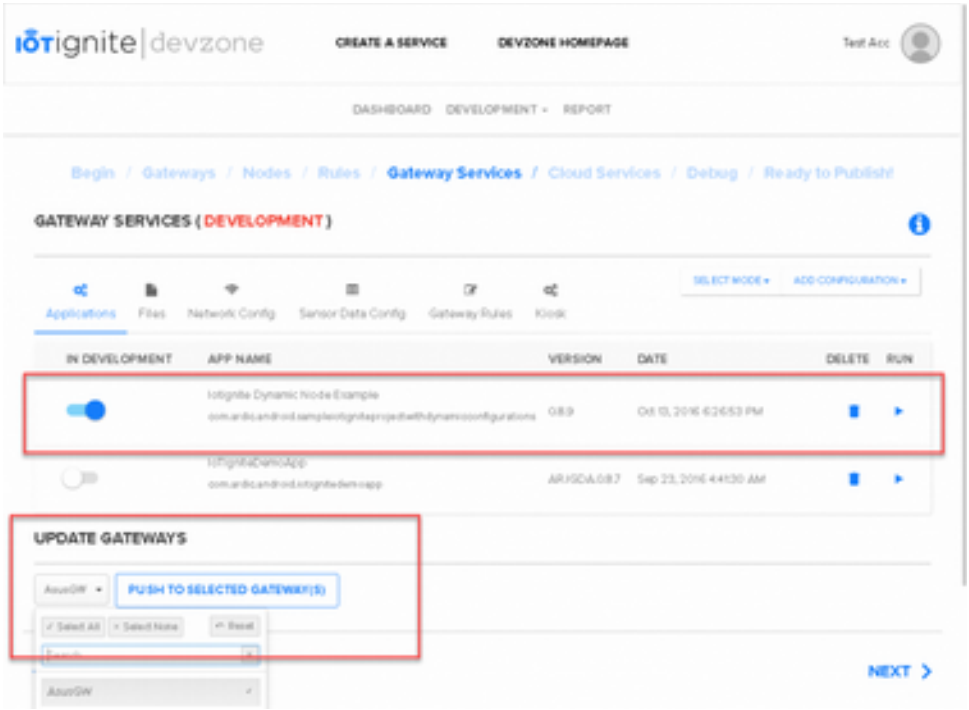
Yukarıda görüleceği üzere fiziksel bağlantılar sağlandıktan sonra uygulama kodunu NodeMCU'ya yükleyebiliriz.Kodların nasıl yüklendiğini "NodeMCU Kodlarının Yüklenmesi" başlığı altında ele aldığımız için buraya almadık. Lütfen bundan sonraki adıma geçmeden önce başlığı inceleyiniz.

Adım 3: SPA ve NodeMCU'nın Gateway için Kaydı ve Yapılandırmalar

NodeMCU uygulama kodunu yükledik ve aygıtımızı erişim noktası olarak başlattık. Bu adımda NodeMCU'yu SPA ile kaydetmeyi göstereceğiz. SPA ile bir aygıtı Gateway olarak kayıt edebiliriz.

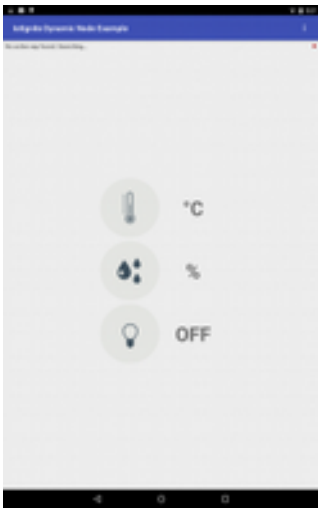
Bu işlemi "SPA ile NodeMCU Register (Kayıt) İşlemi" başlığından anlatmıştık. Bu başlığı inceleyip işlemleri yaptıktan sonra aşağıdan devam edebilirsiniz.

DHT 11 (NodeMCU) sensöründen alınan verileri IoT-Ignite platformuna iletecek olan kullanıcı uygulamasını aşağıda verilen şekilde yükleyiniz.

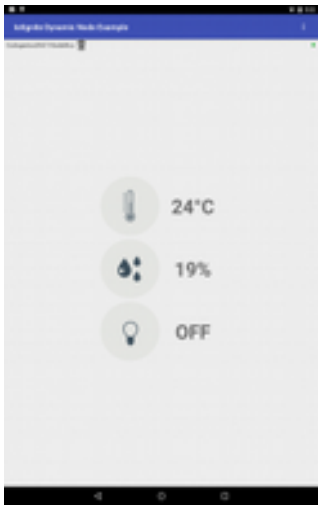


Data View ile verileri aşağıdaki gibi görselleştirebilirziniz.

NodeMCU'nuzu kaydetmek için SPA'yı çalıştırın. Node bağlanmadığında ekran görüntüsü şöyledir:

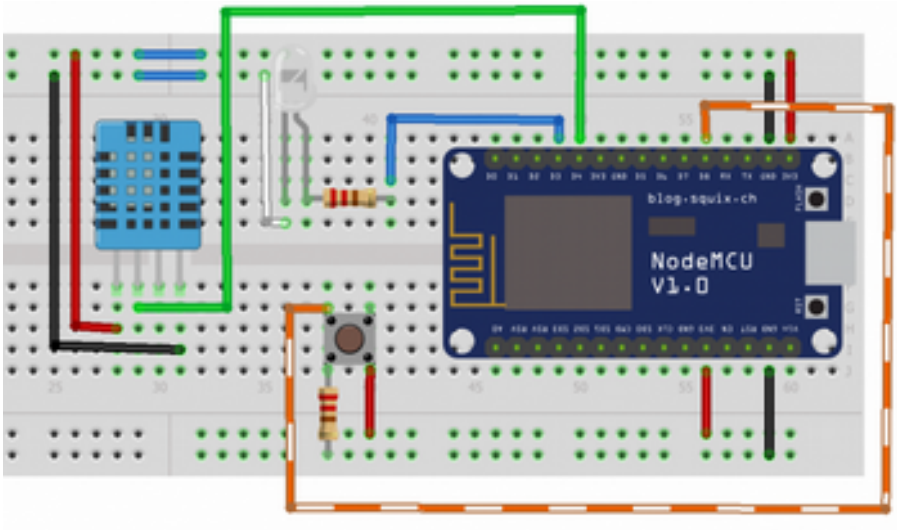


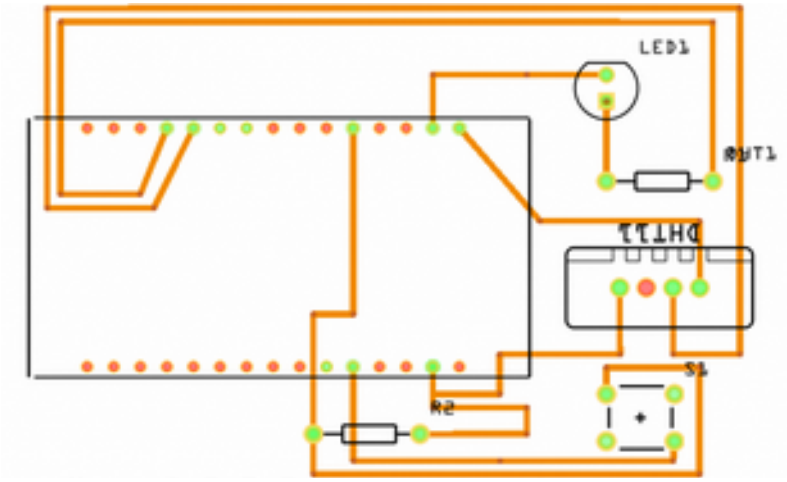
Node bağlantısı sağlandığı zaman ekran çıktısı aşağıdaki gibi olur.

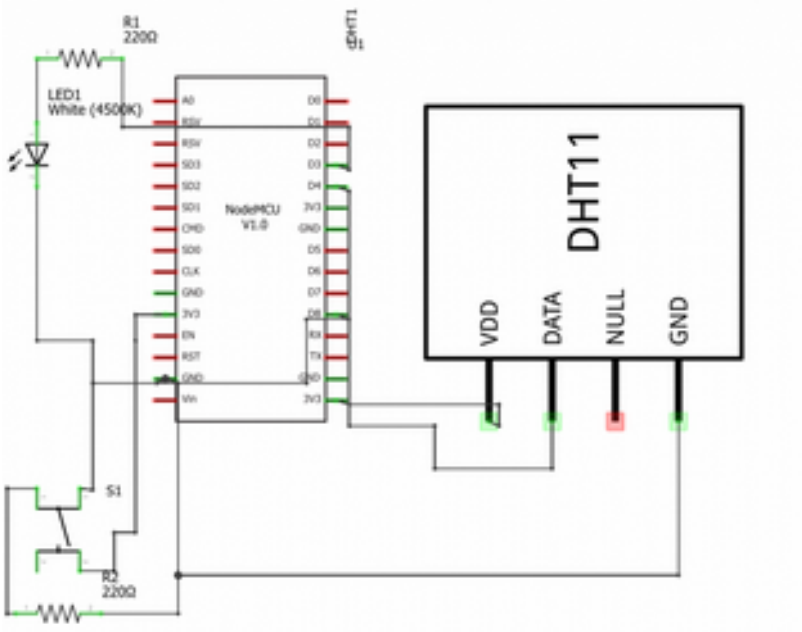


Adım 4: Devre Şemalarının Bağlanması

Uygulamanın devre şemaları aşağıdaki gibidir.







Dashboard’ta Sensörleri Tanımlamak ve Grafiklerle Verileri Anlık İzleme

Yukarıda yapılan tüm işlemlerden sonra IoT-Ignite ortamında sensörlerden gelen verileri anlık olarak aşağıdaki gibi takip edebiliriz.

Raspberry Pi 3 ile Çevresel Sensör Bilgilerini Edinme (Sıcaklık ve Nem Sensörleri)

Bu uygulamada PilarOS yüklü Raspberry Pi 3 kartını kullanarak sıcaklık ve nem bilgilerini almaya göstereceğiz.

Donanım Gereksinimleri

İhtiyacımız olan donanımlar aşağıdaki gibidir:

- NodeMCU ESP8266 Breakout Board x1
- DHT11 Temperature & Humidity Sensor x1
- LED (generic) x1
- Resistor 221 ohm
- Android Phone x1
- Breadboard x1

Yazılım ve Servis Gereksinimleri

Uygulamayı geliştirmek için ihtiyacımız olan donanımlarda aşağıdaki gibidir:

- Arduino IDE
- IoTIgnite Service Registration
- IoTIgnite Agent APP
- IoTIgnite Service APP

Proje Konusu Hakkında

Temel olarak, Proje NodeMCU üzerinden kablosuz olarak DHT11 sensöründen veri aktarımı yapacaktır. Bu transfer işleminde ağ geçidi olarak Raspberry Pi 3 kartı kullanacağız. Belirlenen kurallara göre sistemde bulunan LED'in yanması sağlanacaktır.

Veri Akış Modeli

Projenin veri akış modeli şu şekilde gerçekleşecektir.

- NodeMCU bir erişim noktası olacaktır.
- AP modunda cihazı sisteme kaydedin ve kablosuz bağlantı kurulacaktır.
- İstemci bağlantısı ayarlandığında, sıcaklık ve nem iletiminin frekansı okunur ve verileri bu frekansta müşteriye iletmeye başlar.
- Bağlantı kesilirse ağa yeniden bağlanmaya çalışılır.
- LED için tanımlanan olay meydana geldiğinde LED'in çalışması sağlanır.

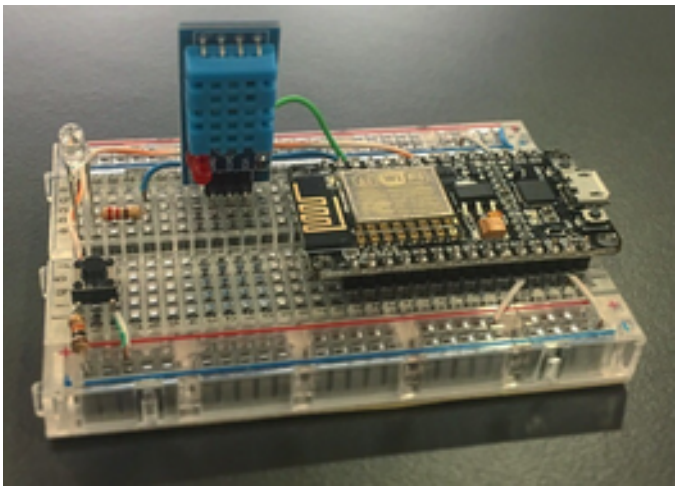
Raspberry Pi 3 kartını Gateway olarak kullanmak için öncelikle PilarOS işletim sistemini kurmanız gerekiyor.

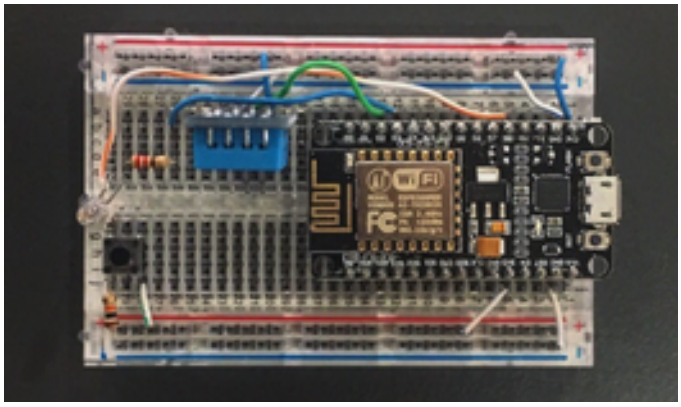
Yapılandırmalar ve İşlem Adımları

Raspberry Pi 3 aracılığıyla sıcaklık ve nem verilerini alabilmek için şu yapılandırma ayarlarınız sırasıyla yapmamız gerekiyor.

Adım 1: NodeMCU'nun Hazırlanması

NodeMCU için kayıt işlemine başlamadan önce aşağıda görüleceği üzere devremizi oluşturalım.





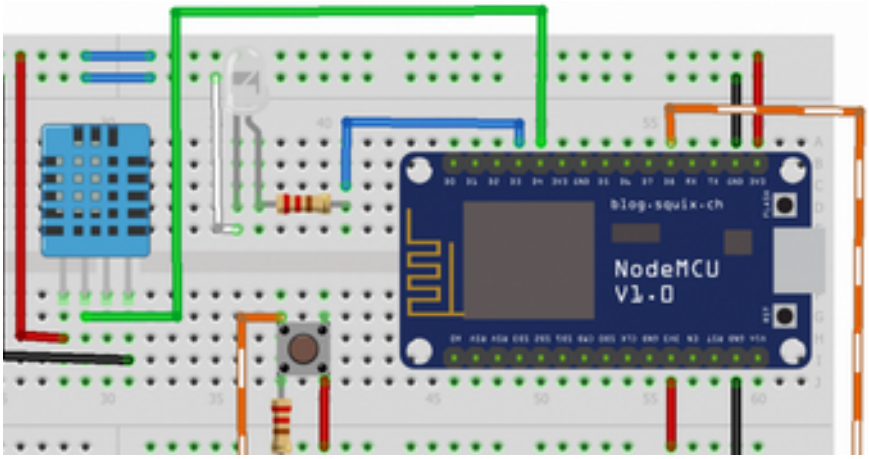
Yukarıda görüleceği üzere fiziksel bağlantılar sağlandıktan sonra uygulama kodunu NodeMCU'ya yükleyebiliriz. Kodların nasıl yüklendiğini "NodeMCU Kodlarının Yüklenmesi" başlığı altında ele aldığımız için buraya almadık. Lütfen bundan sonraki adıma geçmeden önce başlığı inceleyiniz.

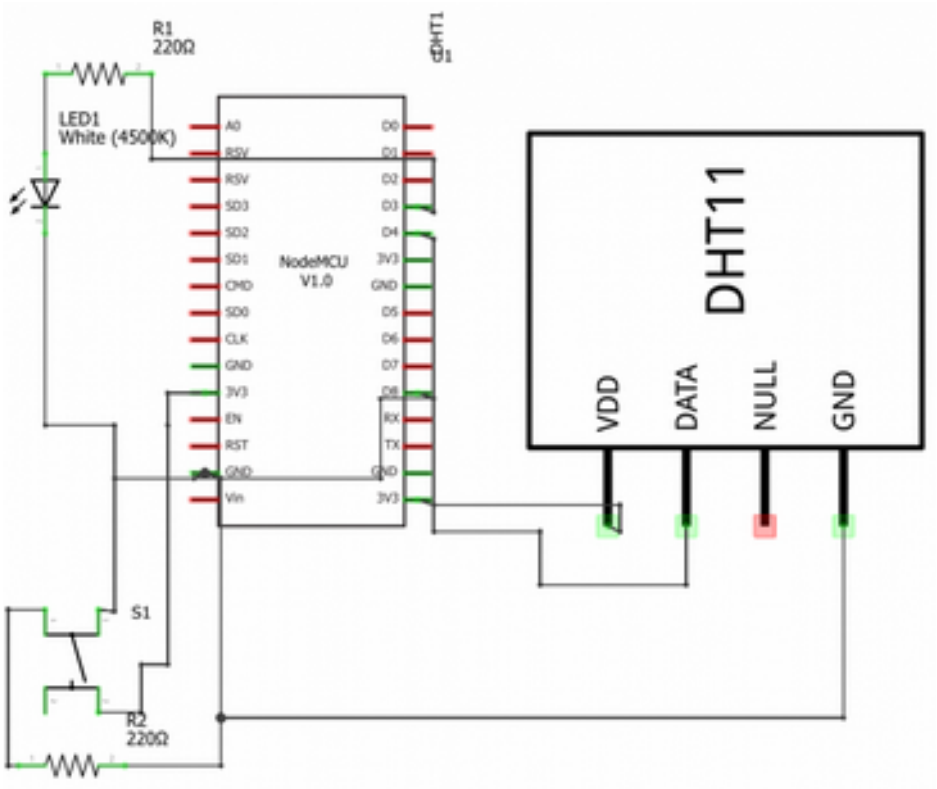
Adım 2: SPA ile NodeMCU Kaydı

SPA ile NodeMCU kaydını gerçekleştirmek için "SPA ile NodeMCU Register (Kayıt) İşlemi" başlığını incelemenizi tavsiye ediyoruz. Bu adımı tamamalamadan bir sonraki adıma geçmeyiniz.

Adım 3: Devre Şemalarının Bağlanması

Uygulamada kullandığımız devrenin şemaları aşağıdaki gibidir.





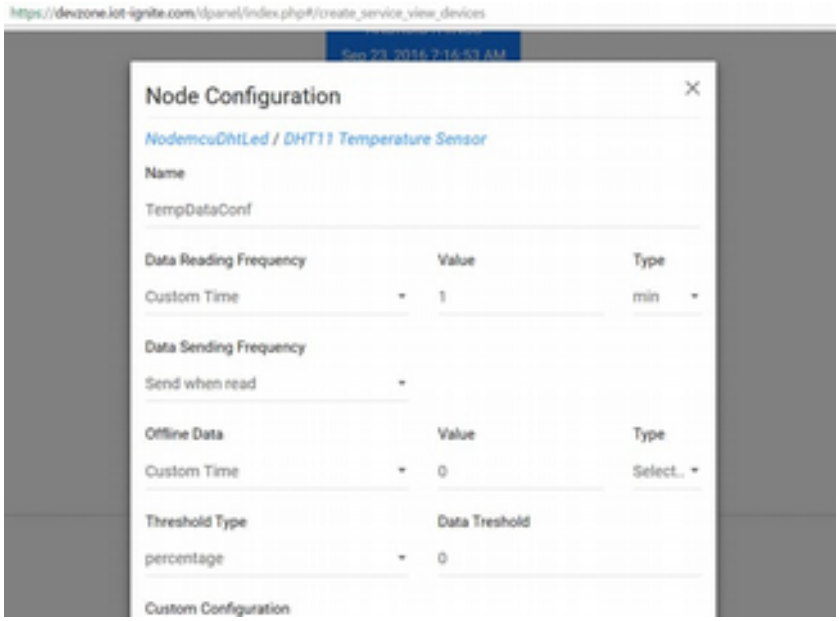
Dashboard’ta Sensörleri Tanımlamak ve Grafiklerle Verileri Anlık İzleme

NodeMCU'nuz kayıt edildiğinde, veri yapılandırma işlemlerini yapabilirsiniz. Bunu yapmak için Devzone’a giriş yapmanız gerekiyor. Devzone’da bulunan **Development** menü öğesini tıklayın ve ardından Nodes öğesini seçin. Burada NodeMCU için şu veri yapılandırmalarını yapabilirsiniz.

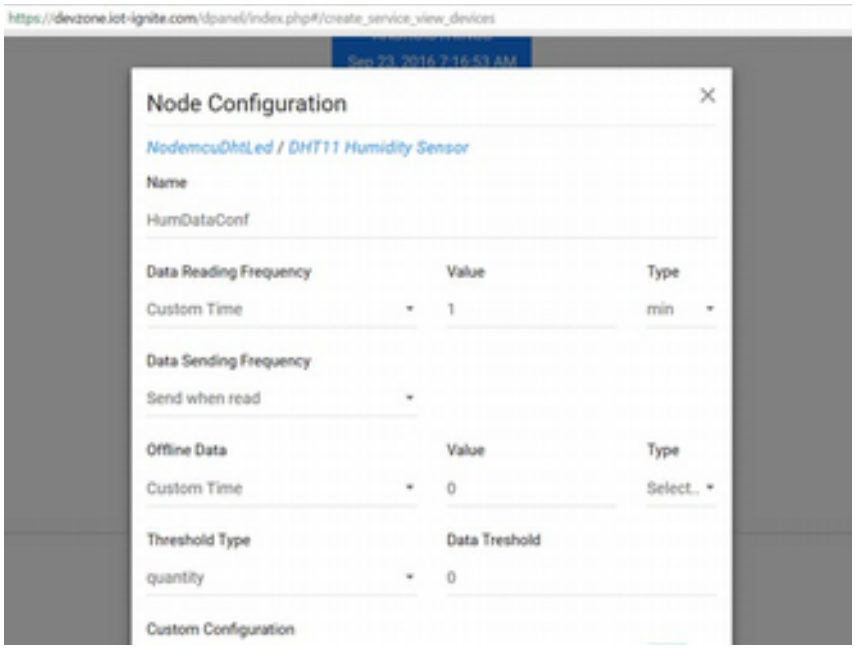
- Veri Okuma Frekansı
- Veri Gönderme Frekansı
- Eşik tipi
- Çevrimdışı Veriler
- Özel yapılandırma

Örnek veri okuma ve gönderme frekansının tipi özel ve minimum değer 1 dakika olmalıdır. Değerleri ayarladıktan hemen sonra konfigürasyon ayarları cihaza aktarılmalıdır.

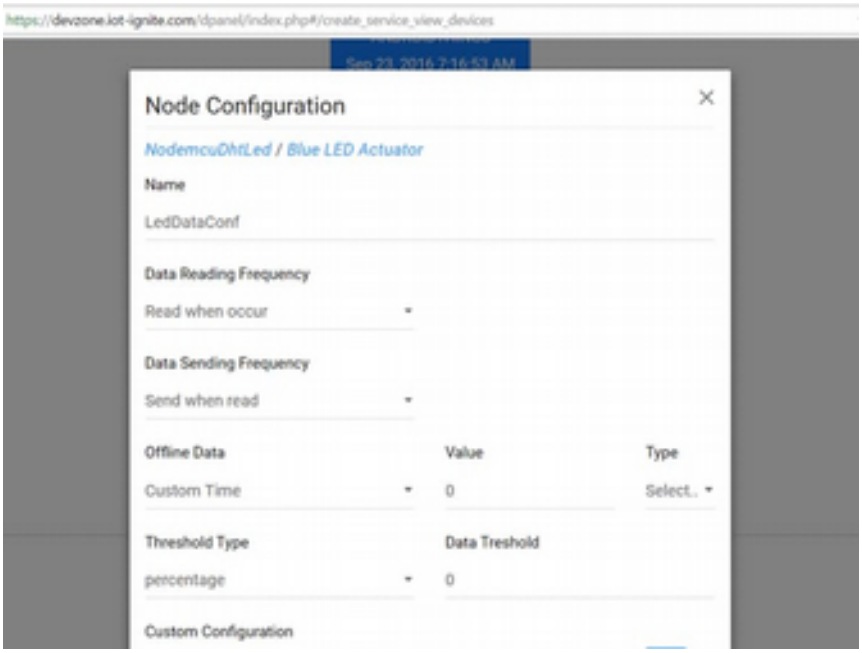
DHT11 Sıcaklık sensörü için yapılandırma ayarları aşağıdaki gibidir.



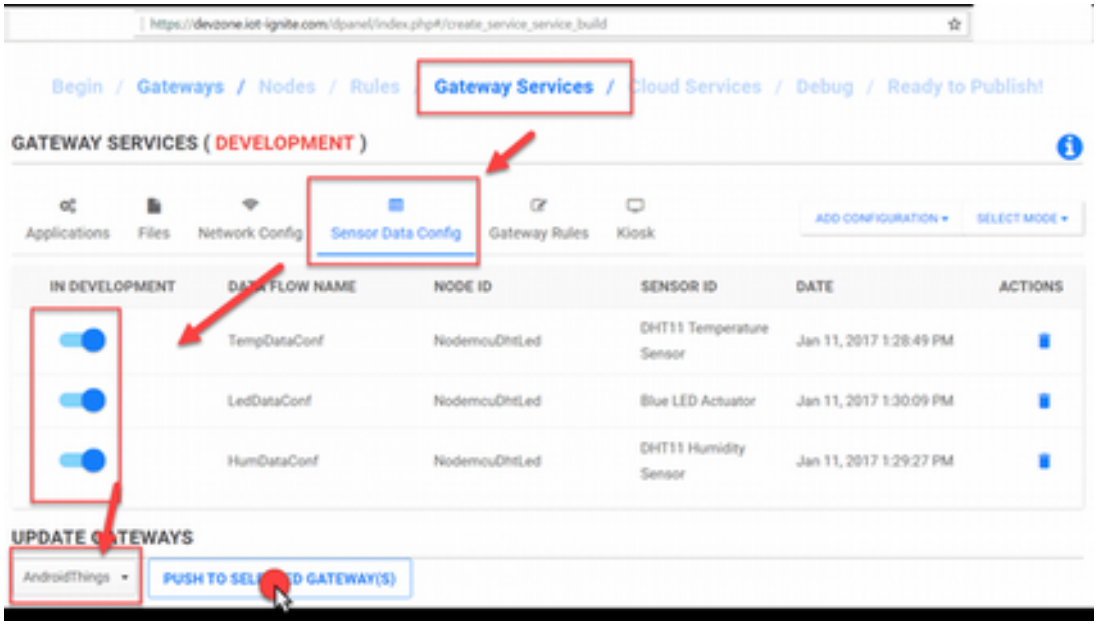
DHT11 nem sensörü için yapılandırma ayarları aşağıdaki gibidir.



LED donanımı için yapılandırma ayarı aşağıdaki gibidir.

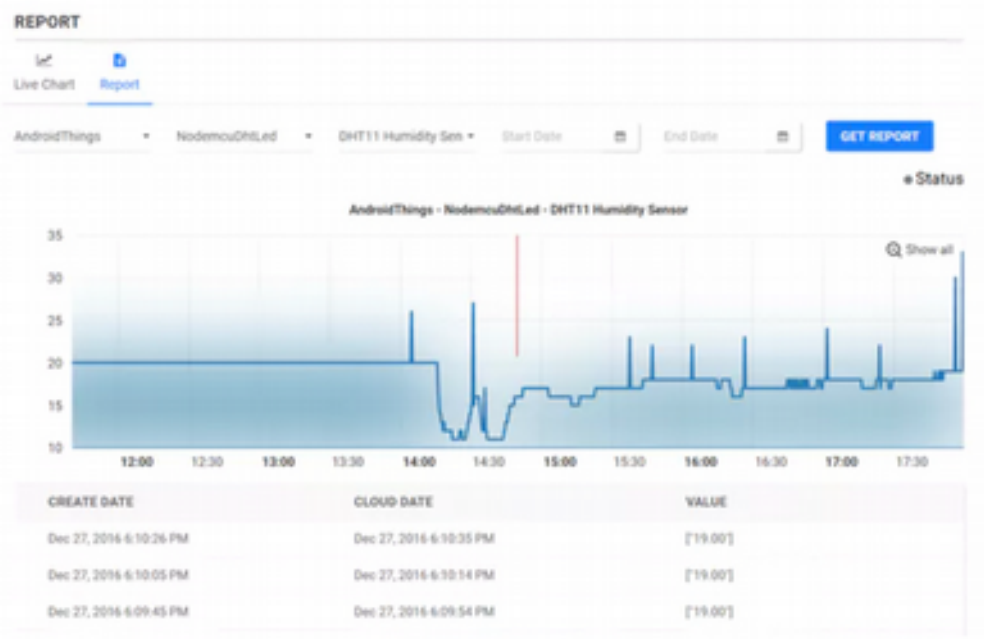


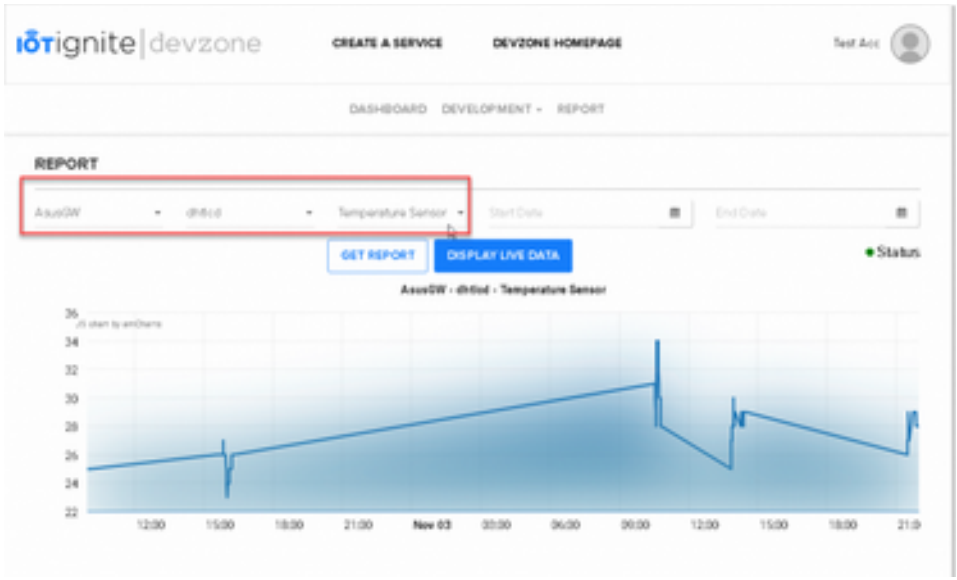
Yukarıda görüldüğü gibi değerleri ayarlayınız. Yapılandırma işleminden sonra, bu ayarların aşağıdaki gibi cihaz gönderilmesi gerekiyor.

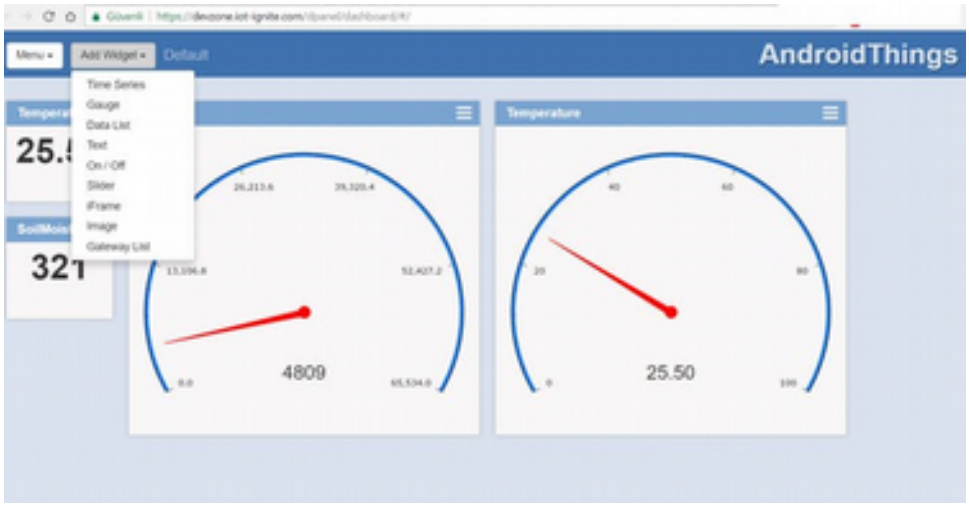


Yukarıdaki son işlemle birlikte sensörlerden verileri alabiliriz.

Sensör verilerini okumak ve görselleştirmek için Devzone ortamı bizlere iki görsel raporlama aracı sunmaktadır. Bunlar; **Report** sekmesi ve **Generic Dashboard** sayfasıdır. Rapor sekmesi basit veri listesi ve görselleştirme aracı olarak kullanılmaktadır. Tüm işlemlerden sonra verileri aşağıdaki gibi görselleştirebiliriz.


REPORT
Live Chart
Report
AndroidThings
NodemcuDhtLed
DHT11 Humidity Sen
Start Date
End Date
GET REPORT
Status
AndroidThings - NodemcuDhtLed - DHT11 Humidity Sensor
Show all
35
30
25
20
15
10
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00
16:30
17:00
17:30
CREATE DATE
CLOUD DATE
VALUE
Dec 27, 2016 6:10:26 PM
Dec 27, 2016 6:10:35 PM
["19.00"]
Dec 27, 2016 6:10:05 PM
Dec 27, 2016 6:10:14 PM
["19.00"]
Dec 27, 2016 6:09:45 PM
Dec 27, 2016 6:09:54 PM
["19.00"]


IOTignite | devzone
CREATE A SERVICE
DEVZONE HOMEPAGE
Test Acc
DASHBOARD
DEVELOPMENT
REPORT
REPORT
AsusGW
dht11d
Temperature Sensor
Start Date
End Date
GET REPORT
DISPLAY LIVE DATA
Status
AsusGW - dht11d - Temperature Sensor
36
34
32
30
28
26
24
22
12:00
15:00
18:00
21:00
Nov 03
03:00
06:00
09:00
12:00
15:00
18:00
21:0


Menu
Add Widget
Default
AndroidThings
Time Series
Gauge
Data List
Text
On / Off
Slider
iFrame
Image
Gateway List
Temperature
4809
25.50

BÖLÜM 10

Adım Adım Basit Müşteri Uygulaması

3. bölümde Demo uygulamasını kullanarak IoT-Ignite platformunun nasıl çalıştığını göstermeye çalıştık. Demo uygulamasında Android ortamında sanal sensörler ile çalıştık. Sanal sensör olarak şunları kullanmıştık:

- Sıcaklık (Sensor)
- Nem (Sensor)
- Lamba (Actuator)

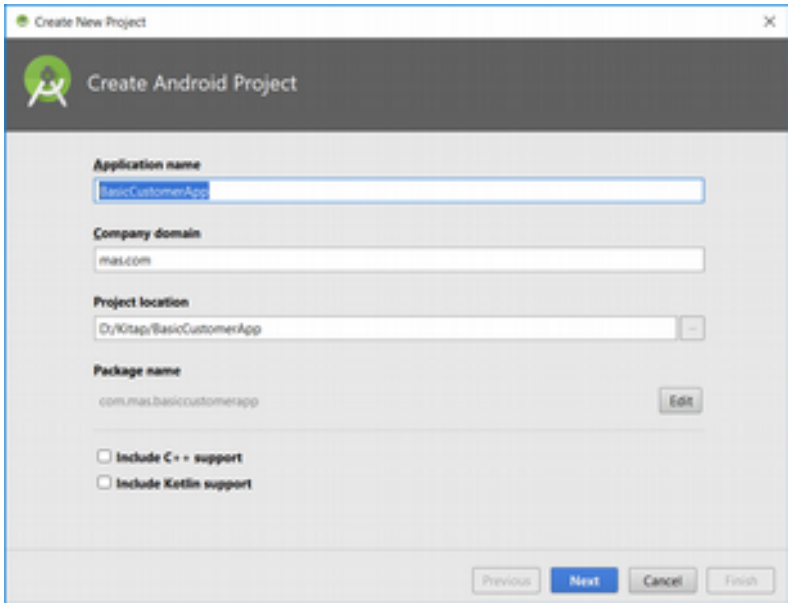
Sanal sensörler Demo uygulamasıyla birlikte gelmekte olup IoT-Ignite ortamında otomatik olarak oluşturulmaktadır. Bu başlık altında Android Studio ortamında sanal sensör oluşturmayı ve bunları IoT-Igniteortamında tanımlayı göstereceğiz. Uygulamanın basit ve anlaşılır olması için bir tane sanal sensör ve bir tane actuator ile çalışacağız.

Android Studio ile Basit Müşteri Uygulaması

Android Studio ortamında IoT-Ignite destekli basit bir müşteri uygulaması geliştirmek için takip edilmesi gereken adımlar şu şekildedir.

Yeni Proje Oluşturmak

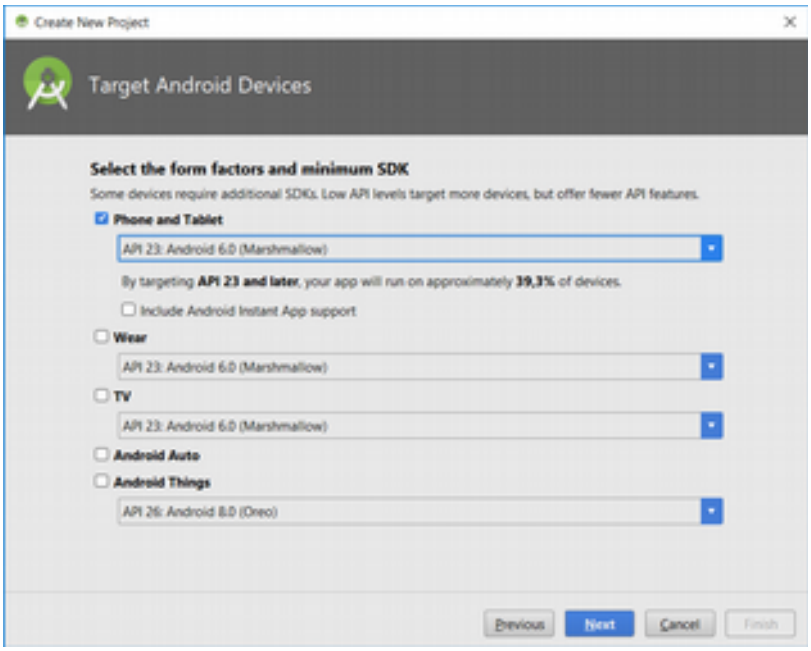
Android Studio geliştirme ortamını açınız ve yeni bir proje oluşturunuz.



Projeye **BasicCustomerApp** ismini verdikten sonra **Next** butonuna tıklayalım.

Hedef Aygıtları Ayarlamak

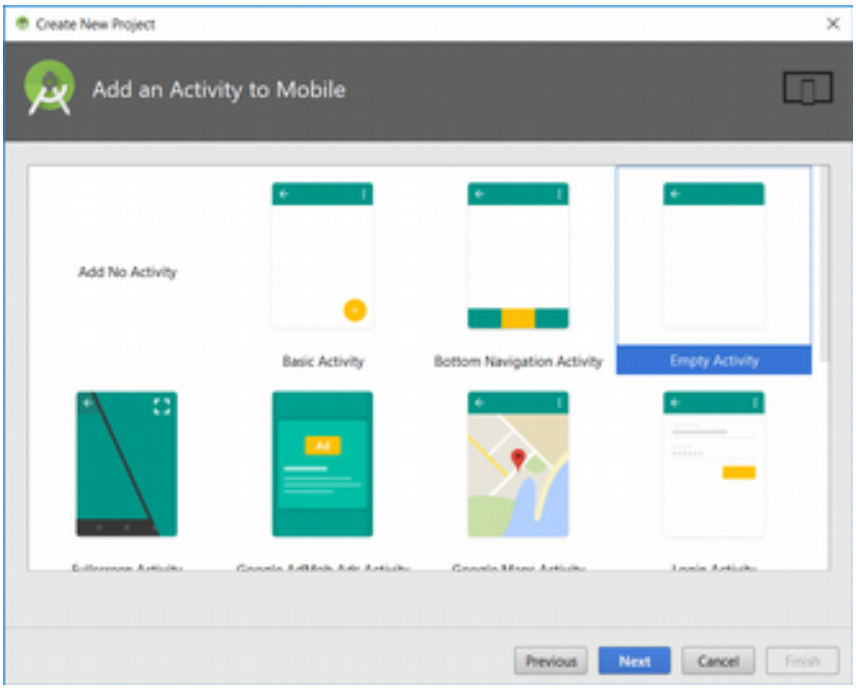
Uygulamanın çalışacağı hedef aygıtlar aşağıda açılan yeni pencerede belirlenir. Amacımız uygulamayı akıllı telefonlarda çalışacak şekilde kullanmaktır. Ayrıca minimum SDK veya API değeri olarak **API 23**'ü seçtik.



Hedef aygıtı belirledikten sonra **Next** butonuna tıklayıp devam edelim.

Main Activity Eklemek

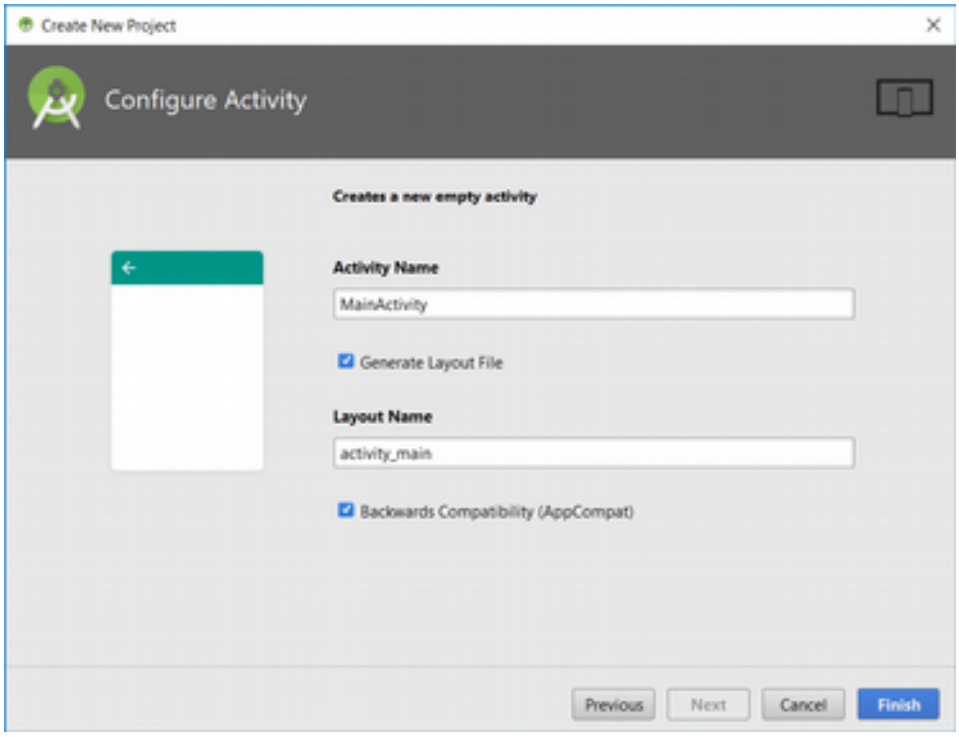
Açılan yeni pencerede yeni bir **Activity** eklememiz gerekiyor.



Empty Activity'yi seçtikten sonra **Next** butonuna tıklayıp devam edelim.

Main Activity'i Düzenlemek

Activity'yi seçtikten sonra aşağıda açılan pencerede activity ismini değiştirebilir veya varsayılan isim ile devam edebilirsiniz.

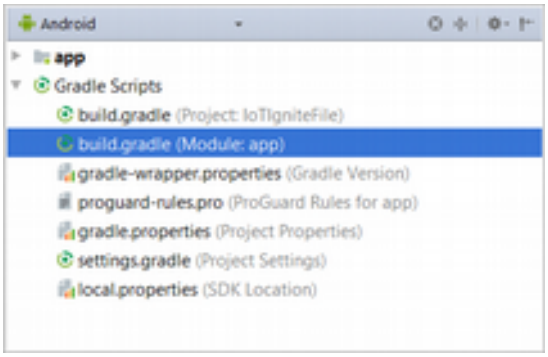


Finish butonuna tıklayarak projemizi oluşturabiliriz.

Kütüphane ve Bağımlılıkları Eklemek

IoT-Ignite için oluşturduğumuz bu yeni projede IoT-Ignite ile uygulama geliştirmek için bazı ayarlamalar yapmamız gerekiyor. Bunları yapmadan IoT-Ignite için geliştirilen kütüphaneleri kullanamayız.

Öncelikle aşağıda verilen dosyayı açalım. Dosyayı kolay bir şekilde bulabilmek için proje görünümünü Android olarak değiştirmenizi tavsiye ediyoruz.



Dosya açıldıktan sonra aşağıda koyu renkli gösterilen satırları dosyanıza ekleyiniz.

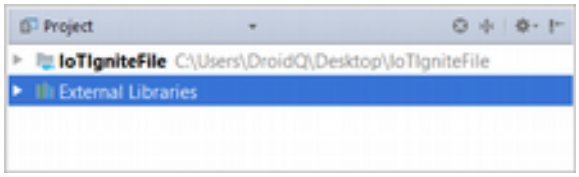
```
20   repositories {
21       mavenCentral()
22       maven {
23           url "http://repo.maven.apache.org/maven2"
24       }
25       maven {
26           url "https://repo.iot-ignite.com/content/repositories/releases"
27       }
28   }
29   dependencies {
         ...
```

```
36        compile 'com.google.code.gson:gson:2.7'
37        compile 'com.ardic.android:IoTIgnite:0.8.2'
38    }
```

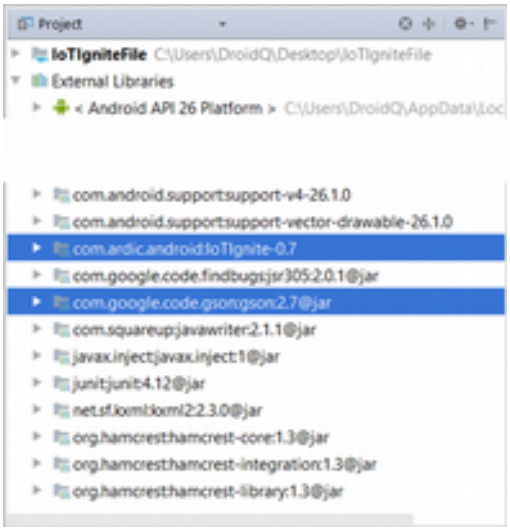
Yukarıda verilen **repositories** blogu olduğu gibi eklenirken, **dependencies** içerisine ise sadece koyu renkli iki satır eklenmelidir. Dependencies içine eklenen ilk satır, Google tarafından geliştirilen **Gson** kütüphanesini eklemek için kullanılır. Bunu ekleme sebebimiz IoT-Ignite platformunda veri transferi için **JSON** yapısının kullanılmasıdır. Gson kütüphanesi, JSON verilerini ayrıştırmak için kullanılır. Bundan dolayı bunu mutlaka eklemeniz gerekiyor. Diğer satır ise **ARDIC** tarafından geliştirilen IoT-Ignite kütüphanesini projeye eklemek için kullanılmaktadır.

Bunları ekledikten hemen sonra **Build > Rebuild** yolunu takip ederek ilgili kütüphaneleri projeye ekleyebiliriz.

Kütüphanelerin projeye eklenip eklenmediğini kontrol etmek adına öncelikle Android görünümünden **Project** görünümüne geçiş yapalım.



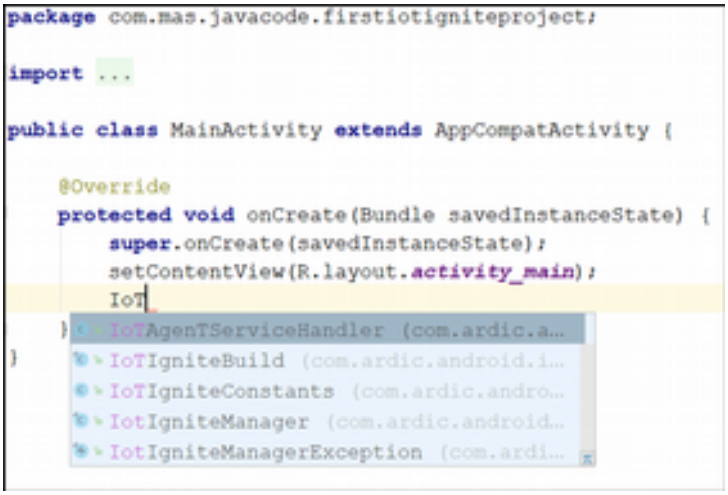
Project görünümü altında yer alan **External Libraries** açılır listesini genişlettiğimiz zaman, harici olarak eklenen kütüphaneleri görebiliriz. Aşağıda verilen listede eğer işaretli kısımları görüyorsanız kütüphanelerin projeye eklendiğinden emin olabilirsiniz.



Kütüphaneleri Test Etmek

Kütüphaneleri ekledikten sonra ilk test aşamasını gerçekleştirebiliriz. Buradaki amaç editör alanında IoT-Ignite ile eklenen kütüphanelere erişim sağlayıp sağlamadığımızı kontrol edebilmektir.

MainActivity içinde bulunan **onCreate()** metodunda **IoTIgnite** yazdığınızda aşağıda verilen sonuç ile karşılaşırsanız, bu durumda işlemlerin başarıyla gerçekleştiğini söyleyebiliriz.



Arayüz Tasarımı

Uygulamanın arayüzü için aşağıda verilen kodları kullanacağız.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="@dimen/activity_vertical_margin">

    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <!--IoTIgnite bağlantı durumu -->
        <TableRow
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="right">

            <TextView
                android:id="@+id/textConnection"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="right|center"
                android:layout_marginRight="5dp"
                android:layout_span="2"
                android:text="Disconnected"
                android:textAppearance="?android:attr/textAppearanceSmall" />

            <ImageView
                android:id="@+id/imageViewConnection"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="right|center"
                android:src="@drawable/disconnected" />

        </TableRow>

        <!--Sıcaklık sensörü -->
        <TableRow
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <TextView
                android:id="@+id/textTemperature"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
```

```
                    android:layout_margin="5dp"
                    android:text="Temperature"
                    android:textAlignment="center"
                    android:textAppearance="?android:attr/textAppearanceMedium" />
            </TableRow>

            <View
                android:layout_height="1dp"
                android:layout_gravity="center_vertical"
                android:layout_margin="5dp"
                android:background="@color/colorLightGrey" />

            <TableRow
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:background="@color/colorLightGrey">

                <TextView
                    android:id="@+id/textTemperatureValue"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_marginTop="15dp"
                    android:text="0 &#x2103;"
                    android:textAlignment="center"
                    android:textAppearance="?android:attr/textAppearanceLarge" />

                <SeekBar
                    android:id="@+id/seekBarTemperature"
                    android:layout_height="wrap_content"
                    android:layout_margin="20dp"
                    android:layout_weight="1" />
            </TableRow>


            <!--Lamba aktuatörü -->
            <TableRow
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_marginTop="5dp"
                android:gravity="center_horizontal">

                <TextView
                    android:id="@+id/textViewLamp"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_margin="5dp"
                    android:layout_span="2"
                    android:text="Lamp"
                    android:textAlignment="center"
                    android:textAppearance="?android:attr/textAppearanceMedium" />
            </TableRow>

            <View
                android:layout_height="1dp"
                android:layout_gravity="center_vertical"
                android:layout_margin="5dp"
                android:background="@color/colorLightGrey" />

            <TableRow
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:gravity="center_horizontal">

                <ImageView
                    android:id="@+id/imageLamp"
                    android:layout_width="100dp"
                    android:layout_height="100dp"
                    android:layout_gravity="center_horizontal"
                    android:layout_marginTop="10dp"
                    android:src="@drawable/lamp_off" />

                <ToggleButton
                    android:id="@+id/toggleLamp"
                    android:layout_gravity="center_vertical" />

            </TableRow>
        </TableLayout>

```

```
125    </LinearLayout>
```

Bu kodları yazdıktan sonra uygulama arayüzü aşağıdaki gibi olur.

Uygulama Kodları

Arayüzü tasarladıktan hemen sonra şimdi asıl işlemi yapan java kodlarımızı yazabiliriz. Kodlarımızı sınıf sınıf aşağıdaki gibi hazırlayalım.

Constants Sınıfı

Uygulamanın genelinde kullanacağımız sabitleri tanımladığımız sınıftır. Bu sınıf içinde şu kodları yazalım.

Constants.java

```
1    package com.mas.basiccustomerapp;
2
3    /*Uygulamada kullanacağımız sabitleri tanımladığımı sınıfımız*/
4    public class Constants {
5
6        /*Kurucu metodumuz*/
7        private Constants() {
8
9        }
10
11       /*Sanal sensörler için VENDOR_INFO bilgisi*/
12       public static final String VENDOR_INFO = "Basic Customer App";
13
14       /*Sanal node ismi*/
15       public static final String NODE_ID = "VirtualCustomerNode1";
16
17       /*Sanal sensör ismi*/
18       public static final String TEMP_NAME = "Temperature";
19
20       /*Sanal actuator ismi*/
21       public static final String LAMP_NAME = "Lamp";
22
23       /*Temperature sensörü için ilk değer*/
24       public static final int FIRST_VALUE_FOR_TEMP = 21;
25
26       /*Lamp actuatoru için ilk değer*/
27       public static final int FIRST_VALUE_FOR_LAMP = 1;
28
```

```
29    }
```

VirtualCustomerNodeHandler Sınıfı

IoT-Ignite ortamında sanal sensörler oluşturabiliriz. Burada sensörleri EHUB'da değil, Android kodları ile oluşturacağız. Bunun için VirtualCustomerNodeHandler sınıfını geliştirdik. Bu sınıf özetle aşağıda verilen işlemleri yapmaktadır.

- IoT-Ignite platformuna bağlantı sağlamak ve bağlantı durumunu kontrol etmek,
- Thing oluşturup bunları IoT-Ignite ortamına kayıt etmeyi sağlamak,
- Arayüzde bulunan sanal sensörlerden gelen verileri okumak ve bu verileri IoT-Ignite platformuna göndermek,

Bunlar genel olarak yapılan işlemler. Ancak kodumuzu incelerseniz daha birçok işlemin gerçekleştirdiğini görebilirsiniz.

Bu sınıfın kodları aşağıdaki gibi olup her kodun açıklamasını beraberinde verdik. Bu sayede uygulamanın çalışma mantığını daha iyi kavrayabilirsiniz.

VirtualCustomerNodeHandler.java

```
1     package com.mas.basiccustomerapp;
2
3     import android.app.Activity;
4     import android.content.Context;
5     import android.util.Log;
6     import android.widget.ImageView;
7     import android.widget.SeekBar;
8     import android.widget.TextView;
9     import android.widget.ToggleButton;
10
11
12    import com.ardic.android.iotignite.callbacks.ConnectionCallback;
13    import com.ardic.android.iotignite.enumerations.NodeType;
14    import com.ardic.android.iotignite.enumerations.ThingCategory;
15    import com.ardic.android.iotignite.enumerations.ThingDataType;
16    import com.ardic.android.iotignite.exceptions.AuthenticationException;
17    import com.ardic.android.iotignite.exceptions.UnsupportedVersionException;
18    import com.ardic.android.iotignite.listeners.NodeListener;
19    import com.ardic.android.iotignite.listeners.ThingListener;
20    import com.ardic.android.iotignite.nodes.IotIgniteManager;
21    import com.ardic.android.iotignite.nodes.Node;
22    import com.ardic.android.iotignite.things.Thing;
23    import com.ardic.android.iotignite.things.ThingActionData;
24    import com.ardic.android.iotignite.things.ThingConfiguration;
25    import com.ardic.android.iotignite.things.ThingData;
26    import com.ardic.android.iotignite.things.ThingType;
27
28    import java.util.Hashtable;
29    import java.util.Timer;
30    import java.util.TimerTask;
31    import java.util.concurrent.Executors;
32    import java.util.concurrent.ScheduledExecutorService;
33    import java.util.concurrent.ScheduledFuture;
34    import java.util.concurrent.TimeUnit;
35
36    /*VirtualCustomerNodeHandler bu sınıf özetle
37    IOTIGNITE PLATFORMUNA BAĞLANTI,
38    THING OLUŞTURMA VE IGNITE PLATFORMUNA KAYIT ETMEK,
39    MAINACTIVITY ARAYÜZÜNDE BULUNAN SENSÖR VERİLERİNİ OKUMAK VEYA OKUMAYI İPTAL ETMEK,
40    SENSÖR VERİLERİNİ IGNITE ORTAMINA GÖNDERMEK ve daha birçok  işlemi yapmak için
41    kullanılmaktadır.*/
42
43    /*onConnected() ve onDisconnected() metotlarını eklemek için sınıfa
44    ConnectionCallback arayüzü uygulanmalıdır.*/
45    public class VirtualCustomerNodeHandler implements ConnectionCallback {
```

```
46
47        /********DEĞİŞKENLER********/
48
49        /*Uygulama içinde kullanacağımız değişkenlerimiz*/
50        private static final String TAG = "Basic Customer App1";
51
52        /*ScheduledExecutorService: Komutları belirli bir gecikmeden sonra çalıştırmayı
veya periyodik olarak yürütmeyi sağlar.*/
53        private static volatile ScheduledExecutorService mExecutor;
54
55        /*İş parçacığı havuzunda tutulacak iş parçacığı sayısı 2 tane olacaktır.*/
56        private static final int NUMBER_OF_THREADS_IN_EXECUTOR = 2;
57
58        /*EXECUTOR_START_DELAY: iş parçası için gecikme süresi tanımladık*/
59        private static final long EXECUTOR_START_DELAY = 100L;
60
61        /*Thing bileşenlerden veri okumayı sağlayan bir görev tanımladık.
62        Hashtable: Bu sınıf, key-value çiftlerinden oluşan bir karma tablo oluşturmayı
sağlar.*/
63        private Hashtable<String, ScheduledFuture<?>> tasks = new Hashtable<>();
64
65        /*IotIgniteManager: IoTIgnite pltaformuna bağlanmayı sağlayan temel sınıftır.*/
66        private static IotIgniteManager mIotIgniteManager;
67
68        /*Sanal node oluşturmak için Node sınıfı kullanılır.*/
69        private Node mNode;
70
71        /*Sanal Thing oluşturmak için Thing sınıfı kullanılır.*/
72        private Thing mTempThing,  mLampThing;
73
74        /*ThingType: Sensörün türünü belirtir. Sensör tipi ve üretici hakkında bilgi
depolamayı sağlar.*/
75        private ThingType mTempThingType,  mLampThingType;
76
77        /*ThingDataHandler: Thing verilerini IoT-Igniteortamına göndermeyi sağlayan iş
parçacığı.*/
78        private ThingDataHandler mThingDataHandler;
79
80        /*IoT-Ignite ortamına bağlantı durumunu tutan bir değişken tanımladık.*/
81        private boolean igniteConnected = false;
82
83        /*Bağlantı işlemini kontrol etmek için oluşturulan görevin bekleme süresi.*/
84        private static final long IGNITE_TIMER_PERIOD = 5000L;
85
86        /*Timer: Bir iş parçacığını gelecekte yürütmek için kullanılır.*/
87        private Timer igniteTimer = new Timer();
88
89        /*IgniteWatchDog İş parçacığı ile IoT-Ignitebağlantısı kontrol edilir.*/
90        private IgniteWatchDog igniteWatchDog = new IgniteWatchDog();
91
92        private Context ctx;
93        private Activity activity;
94
95        /********KURUCU METODUMUZ********/
96        public VirtualCustomerNodeHandler(Activity activity) {
97            this.ctx = activity.getApplicationContext();
98            this.activity = activity;
99        }
100
101
102       /********IOTIGNITE PLATFORMUNA BAĞLANTI ********/
103       /*IgniteWatchDog İş parçacığı ile IoT-Ignite bağlantısı kontrol edilir.
104       Bunun için sınıfın TimerTask sınıfından türetilmesi gerekiyor.
105       TimerTask sınıfı bir defalık veya tekrarlanan görevlerin yürütülmesi
sağlanır.*/
106       private class IgniteWatchDog extends TimerTask {
107           @Override
108           public void run() {
109               /*IoT-Ignite platformuna bağlantı yoksa bağlantı işlemi yapılmaya
çalışılır.*/
110               if(!igniteConnected) {
111                   Log.i(TAG, "Ignite bağlantısı kuruluyor...");
112                   /*start metodu ile bağlantı sağlanır.*/
113                   start();
114               }
115           }
116       }
117
```

```
118       /*IoT-Ignite platformuna bağlanmayı sağlayan metodumuz.*/
119       public void start() {
120
121           try {
122               /*IotIgniteManager.Builder: IgniteManger oluşturulur.
123               Bu sınıf IoTIgnite platformuna bağlanmayı sağlar.
124               setContext(): Bağlantı işleminin hangi uygulama için yapılacağı
belirtilir.
125               Buraya parametre olarak Context verilir.
126               setConnectionListener: Parametre olarak ConnectionCallback arayüzü
alır.
127               Bağlantı sağlandığı zaman bu arayüz ile gelen onConnected()  metodu
çağrılır.
128               Bu arayüzü VirtualCustomerNodeHandler sınıfına yukarıda uygulamıştık.
129               build(): bu metot ile bağlantı işlemi sağlanır.*/
130               mIotIgniteManager = new IotIgniteManager.Builder()
131                       .setContext(ctx)
132                       .setConnectionListener(this)
133                       .build();
134
135           } catch (UnsupportedVersionException e) {
136               Log.e(TAG, e.toString());
137
138           }
139           /*Bağlantı sağlandıktan sonra timer durdurulur.*/
140           cancelAndScheduleIgniteTimer();
141       }
142
143       /*cancelAndScheduleIgniteTimer: IoT-Igniteplatformuna bağlantı sağlandıktan
sonra Timer sonlandırılır.*/
144       private void cancelAndScheduleIgniteTimer() {
145           /*cancel: Planlanmış görevleri iptal ederek Timer nesnesini sonlandırır.*/
146           igniteTimer.cancel();
147           igniteWatchDog.cancel();
148
149           /*Timer nesneleri yeniden oluşturulur.*/
150           igniteWatchDog = new IgniteWatchDog();
151           igniteTimer = new Timer();
152
153           /*schedule(): igniteWatchDog iş parçası 500 milisaniye sonra tekrar
başlatılır.*/
154           igniteTimer.schedule(igniteWatchDog, IGNITE_TIMER_PERIOD);
155       }
156
157       public void stop() {
158           if (igniteConnected) {
159               /*Ignite ile bağlantı kesildiği zaman Node ve Thing bileşenleri
160               çevrimdışı yapmak için setConnected() metodunu kullanırız.
161               Bu metot bağlantı durumunu ayarlamayı sağlar. Metodun ikinci
162               parametresine istediğiniz değeri verebilirsiniz.*/
163               mLampThing.setConnected(false, "Application Destroyed");
164               mTempThing.setConnected(false, "Application Destroyed");
165               mNode.setConnected(false, "ApplicationDestroyed");
166           }
167           /*mExecutor null ise sonlandırılır.*/
168           if(mExecutor != null) {
169               mExecutor.shutdown();
170           }
171       }
172
173       /*onConnected() ve onDisconnected() metotları ConnectionCallback arayüzü ile
174       gelen metotlardır.
175       IoT-Ignite platformuna bağlantı sağlanırsa onConnected() metodu çağrılır.*/
176       @Override
177       public void onConnected() {
178
179           Log.i(TAG, "Ignite Bağlantısı Kuruldu!");
180           igniteConnected = true;
181
182           /*Bağlantı durumu aşağıdaki metot ile kullanıcı arayüzüne uygulanır.*/
183           updateConnectionStatus(true);
184
185           /*Ignite platformuna bağlantı sağlanırsa aşağıdaki metotlar icra edilir.*/
186           /*Node ve Thing oluşturulup IoT-Ignite platformuna kayıt edilir. Ayrıca
187           Thing bileşenlere ilk değerleri atanır.*/
188           initIgniteVariables();
189           sendInitialData();
190           cancelAndScheduleIgniteTimer();
```

```
191       }
192
193       /*Ignite platformuna bağlantı kurulamaz ise onDisconnected() metodu çağrılır.*/
194       @Override
195       public void onDisconnected() {
196           Log.i(TAG, "Ignite Bağlantısı Kesildi!");
197           igniteConnected = false;
198           /*Bağlantı durumu aşağıdaki metot ile kullanıcı arayüzüne uygulanır.*/
199           updateConnectionStatus(false);
200           cancelAndScheduleIgniteTimer();
201       }
202
203       /*Ignite platformu ile bağlantının olup olmadığını
204       MainActivity'de göstermek için bu metot kullanılır.
205       Amaç kullanıcıya bağlantı durumu hakkında bilgi vermektir.*/
206       private void updateConnectionStatus(final boolean connected) {
207           if(activity != null) {
208               /*runOnUiThread(): Main Thread üzerinde değişiklik yapmak için bir
209               Runnable tanımlar. Runnable ile arayüz güncellenir.*/
210               activity.runOnUiThread(new Runnable() {
211                   @Override
212                   public void run() {
213                       /*MainActivity arayüzündeki iki kontrole erişim sağlanır.*/
214                       ImageView imageViewConnection =
activity.findViewById(R.id.imageViewConnection);
215                       TextView textViewConnection =
activity.findViewById(R.id.textConnection);
216
217                       /*connected true ise kullanıcı arayüzünde bağlantının kurulduğu
218                       false ise bağlantının kurulamadığı bilgisi verilir.*/
219                       if (connected) {
220
imageViewConnection.setImageDrawable(activity.getResources().getDrawable(R.drawable.conn
ected));
221                           textViewConnection.setText("Bağlandı");
222                       } else {
223
imageViewConnection.setImageDrawable(activity.getResources().getDrawable(R.drawable.disc
onnected));
224                           textViewConnection.setText("Bağlanamadı");
225                       }
226                   }
227               });
228           }
229       }
230
231       /********THING OLUŞTURMA VE IGNITE PLATFORMUNA KAYIT ETMEK********/
232       /*initIgniteVariables(): Bu metot ile Node ve Thing oluşturulup IoT-Ignite
platformuna kayıt edilir.*/
233       private void initIgniteVariables() {
234           /*ThingType: Sensörün türünü belirtir. Sensör tipi ve üretici hakkında
bilgi depolamayı sağlar.
235           Bu sınıfın kurucu metodu için üç parametre alır:
236           THING_TYPE: Thing tipi bilgisi tanımlar.
237           VENDOR: Thing hakkında üretici bilgisi tanımlar.
238           THING_DATA_TYPE: Thing için veri tipi tanımlar.*/
239           mTempThingType = new ThingType(Constants.TEMP_NAME, Constants.VENDOR_INFO,
ThingDataType.FLOAT);
240           mLampThingType = new ThingType(Constants.LAMP_NAME, Constants.VENDOR_INFO,
ThingDataType.INTEGER);
241
242           /*NodeFactory(): Node nesnesi oluşturulur.
243           createNode() Bu metot oluşturulacak Node'un bilgilerini tanımlamayı sağlar.
244           Parametre olarak şu bilgileri kullanır.
245           NODE_ID: Node için benzersiz bir ID bilgisi.
246           NODE_TYPE: Desteklenen Node türlerini tanımlar. GENERIC, RASPBERRY_PI ve
ARDUINO_YUN parametrelerini alabilir.
247           Burada genel amaçlı sanal bir sensör tanımladığımız için NodeType.GENERIC
parametresini kullandık.
248           4. parametreye bu gibi durumlarda null verilir.
249           NodeListener: Kayıt işlemini dinlemek için tanımlanır.*/
250           mNode = IotIgniteManager.NodeFactory.createNode(Constants.NODE_ID,
251                                                           Constants.NODE_ID,
252                                                           NodeType.GENERIC,
253                                                           null,
254                                                           new NodeListener() {
255               @Override
256               public void onNodeUnregistered(String s) {
```

```
257                     Log.i(TAG, Constants.NODE_ID + " kayıt edilmedi!!");
258                 }
259             });
260 
261             /*Node kayıtlı değilse kayıt edilir ve bağlantı sağlanır.*/
262             if (!mNode.isRegistered() && mNode.register()) {
263                 mNode.setConnected(true, Constants.NODE_ID + " online");
264                 Log.i(TAG, mNode.getNodeID() + " kayıt edildi!");
265             } else {
266                 mNode.setConnected(true, Constants.NODE_ID + " online");
267                 Log.i(TAG, mNode.getNodeID() + " zaten kayıtlı!");
268             }
269 
270             /*Node daha önce kayıtlı ise bu durumda bu Node'a bağlı olan Thing, yani
sensör ve actuator oluşturabiliriz.*/
271             if (mNode.isRegistered()) {
272 
273                 /*createThing(): Thing oluşturmak için kullanılır. Şu parametreleri
alır.
274                 THING_ID: Thing için ID bilgisi. Bunun eşsiz olması zorunludur.
275                 Thing Type: Thing tipini tanımlar. EXTERNAL, BUILTIN veya UNDEFINED
değerlerini alır.
276                 Burada EXTERNAL kullandık. Çünkü sensörlerimiz Gateway dışında
tanımlıdır.
277                 FLAG_STATE: Thing'in bir actuator gibi hareket edip etmeyeceğini
tanımlar.
278                 mTempThing için bu değer false. Çünkü bunu sıcaklık sensörü olarak
kullanacağız.
279                 mLampThing için bu değer true. Çünkü bunu actuator olarak kullanacağız.
True olduğu zaman Cloud ortamından Action Message alabilir. Aksi durumda mesaj alamaz.
280 
281                Lambayı Cloud Rule ile yakmak için bunu kullanacağız.
282                Thing Listener: Olay dinleyici tanımlamayı sağlar. Bunun için
tempThingListener olay dinleyicisini  inceleyiniz.
283                 Son parametre null olacaktır.*/
284                 mTempThing = mNode.createThing(Constants.TEMP_NAME,
285                                                mTempThingType,
286                                                ThingCategory.EXTERNAL,
287                                                false,
288                                                tempThingListener,
289                                                null);
290                 mLampThing = mNode.createThing(Constants.LAMP_NAME,
291                                                mLampThingType,
292                                                ThingCategory.EXTERNAL,
293                                                true,
294                                                lampThingListener,
295                                                null);
296 
297                 /*registerThingIfNotRegistered(): Bu metot ile sensörlerin IoT-Ignite
298                 platformuna kayıt edilmesi sağlanır.*/
299                 registerThingIfNotRegistered(mTempThing);
300                 registerThingIfNotRegistered(mLampThing);
301             }
302         }
303 
304 
305         /*Tanımladığımız sıcaklık sensörünü dinlemek için ThingListener tanımladık.
306         Bu olay dinleyicide 3 metot bulunmaktadır. Konfigürasyon ve action message
verileri buradan alınır.*/
307 
308         private ThingListener tempThingListener = new ThingListener() {
309 
310             /*onConfigurationReceived(): Yapılandırma IoT-Ignite tarafından
ayarlandığında bu metot icra edilir.*/
311             @Override
312             public void onConfigurationReceived(Thing thing) {
313                 Log.i(TAG, "Konfigürasyon alındı " + thing.getThingID());
314                 /*applyConfiguration: Thing bileşenlerden veri okumayı sağlayan metot
*/
315                 applyConfiguration(thing);
316             }
317 
318             /*onActionReceived(): Thing bir aktüatör olarak ayarlanmışsa, gönderilen
eylem mesajları burada ele alınır.
319             Sıcaklık sensörü bir actuator olmadığından burada bir işlem yapılmayacak.*/
320             @Override
321             public void onActionReceived(String nodeId, String sensorId,
ThingActionData thingActionData) {
```

```
322
323           }
324
325           /*onThingUnregistered(): Thing, IoT-Ignite ortamına kayıt edilmediğinde bu
metot çağrılır.*/
326           @Override
327           public void onThingUnregistered(String nodeId, String sensorId) {
328               Log.i(TAG, "Sıcaklık sensörü kayıt edilmedi!");
329               /*Thing kayıt edilemediği zaman veri okumayı sağlayan task'ın
durdurulması gerekir.*/
330               stopReadDataTask(nodeId, sensorId);
331           }
332       };
333
334       /*Tanımladığımız lamba aktüatörünü dinlemek için ThingListener tanımladık.
335       Bu olay dinleyicide 3 metot bulunmaktadır. Konfigürasyon ve action message
verileri buradan alınır.*/
336
337       private ThingListener lampThingListener = new ThingListener() {
338           /*onConfigurationReceived(): Yapılandırma IoT-Ignite tarafından
ayarlandığında bu metot icra edilir.*/
339           @Override
340           public void onConfigurationReceived(Thing thing) {
341               Log.i(TAG, "Konfigürasyon alındı " + thing.getThingID());
342               /*applyConfiguration: Thing bileşenlerden veri okumayı sağlayan metot
*/
343               applyConfiguration(thing);
344           }
345
346           /*onActionReceived(): Thing bir aktüatör olarak ayarlanmışsa, gönderilen
eylem mesajları burada ele alınır.
347           Lamb bir actuator olduğundan burada aşağıdaki işlemler yapılacaktır.*/
348           @Override
349           public void onActionReceived(String nodeId, String sensorId, final
ThingActionData thingActionData) {
350
351               Log.i(TAG, "Action Message alındı" + nodeId + " " + sensorId);
352
353               /*Lambanın açık ve kapalı durumu ayarlanır.*/
354               if(activity != null) {
355                   /*Main Thread yani arayüzde güncelleme yapmak için runOnUiThread
kullanılır. Arayüz güncellemelerini doğrudan yaparsanız hata verir.*/
356
357                   activity.runOnUiThread(new Runnable() {
358                       @Override
359                       public void run() {
360                           /*MainActivity arayüzünde bulunan Lamba kontrolüne erişim
sağlanır.*/
361                           ToggleButton toggleLamp =
activity.findViewById(R.id.toggleLamp);
362                           Log.i(TAG, thingActionData.getMessage());
363                           int message = 0;
364                           try {
365                               /*Action Message, getMessage() ile alınır.*/
366                               String s = thingActionData.getMessage();
367
368                               /*Mesaj null değilse*/
369                               if (s != null) {
370                                   /*Gelen mesaj String olduğu için bunu int tipine
cast ederiz.*/
371                                   message = Integer.parseInt(s.replace("\"", ""));
372                               }
373                           } catch (NumberFormatException e) {
374                               Log.i(TAG, "Mesaj Geçersiz");
375                           }
376                           /*Gelen mesaj 1 ise lamba yanar. Aksi durumda söner.*/
377                           toggleLamp.setChecked(message == 1);
378                       }
379                   });
380               }
381           }
382
383           /*onThingUnregistered(): Thing, IoT-Ignite ortamına kayıt edilmediğinde bu
metot çağrılır.*/
384           @Override
385           public void onThingUnregistered(String nodeId, String sensorId) {
386               Log.i(TAG, "Lamp kayıt edilmedi!");
387               /*Thing kayıt edilemediği zaman veri okumayı sağlayan task'ın
```

```
durdurulması gerekir.*/
388                 stopReadDataTask(nodeId, sensorId);
389             }
390         };
391 
392         /*registerThingIfNotRegistered(): Kod içinde tanımlanan Thing nesnelerini
393         IoT-Ignite ortamına kayıt etmeyi sağlar.*/
394         private void registerThingIfNotRegistered(Thing t) {
395             /*Thing bileşenlerinin kayıtlı olup olmadığı kontrol edilip,
396             kayıtlı olmayanların kayıt edilmesi sağlanır.*/
397             if (!t.isRegistered() && t.register()) {
398                 /*IoT-Ignite ile bağlantı kesildiği zaman Node ve Thing bileşenleri
399                 çevrimdışı yapmak için setConnected() metodunu kullanırız.
400                 Bu metot bağlantı durumunu ayarlamayı sağlar. Metodun ikinci
401                 parametresine istediğiniz değeri verebilirsiniz.*/
402                 t.setConnected(true, t.getThingID() + " bağlandı");
403                 Log.i(TAG, t.getThingID() + " kayıt edildi!");
404             } else {
405                 t.setConnected(true, t.getThingID() + " bağlandı");
406                 Log.i(TAG, t.getThingID() + " zaten kayıtlı!");
407             }
408             /*applyConfiguration: Thing bileşenlerden veri okumayı sağlayan metot */
409             applyConfiguration(t);
410         }
411 
412         /********MAINACTIVITY ARAYÜZÜNDE BULUNAN SENSÖR VERİLERİNİ OKUMAK VEYA OKUMAYI
İPTAL ETMEK********/
413         /*applyConfiguration: Thing bileşenlerden veri okumayı sağlayan metot*/
414         private void applyConfiguration(Thing thing) {
415 
416             if(thing != null) {
417 
418                 /*stopReadDataTask(): Konfigürasyon alındığı zaman öncelikle veri
okumayı sağlayan görevin durdurulması gerekir.*/
419 
420                 stopReadDataTask(thing.getNodeID(), thing.getThingID());
421 
422                 /*getThingConfiguration(): Thing için gelen konfigürasyonu almayı
sağlar.
423                 getDataReadingFrequency(): Veri okuma sıklığını milisaniye cinsinden
alır. Varsayılan olarak READING_DO_NOT_READ değerine sahiptir.
424                 Hiçbir yapılandırma yoksa bulut için veri gönderilmez.
425                 Eğer bu değer 0 ise, verinin geldiği anlaşılır.*/
426                 if (thing.getThingConfiguration().getDataReadingFrequency() > 0) {
427 
428                     /*ThingDataHandler: Thing verilerini IoT-Ignite ortamına göndermeyi
sağlayan iş parçacığı.*/
429                     mThingDataHandler = new ThingDataHandler(thing);
430 
431                     /*Executors: ScheduledExecutorService nesnesi oluşturmayı sağlar.
432                     newScheduledThreadPool: Belirli bir gecikme sonrasında komut
çalıştırmak veya düzenli olarak
433                     yürütmek için komutları zamanlayabilen bir iş parçacığı havuzu
oluşturur.*/
434                     mExecutor =
Executors.newScheduledThreadPool(NUMBER_OF_THREADS_IN_EXECUTOR);
435 
436                     /*ScheduledFuture: ScheduledExecutorService için planlı bir eylem
tanımlar.
437                     scheduleAtFixedRate(Runnable command, long initialDelay, long
period, TimeUnit unit):
438                     Verilen ilk gecikmeden sonra (EXECUTOR_START_DELAY), verilen
periyotta (thing.getThingConfiguration().getDataReadingFrequency())
439                     bir eylem oluşturmayı sağlar. Aynı zamanda bu eylemi yürütmeyi de
sağlar. Sonuç olarak Thing verisi IoT-Igniteortamına gönderilir.*/
440 
441                     ScheduledFuture<?> sf =
mExecutor.scheduleAtFixedRate(mThingDataHandler, EXECUTOR_START_DELAY,
thing.getThingConfiguration().getDataReadingFrequency(), TimeUnit.MILLISECONDS);
442 
443                     /*thing ve node için ID bilgileri alınıp tek bir key elde edilir.*/
444                     String key = thing.getNodeID() + "|" + thing.getThingID();
445 
446                     /*put: Hashtable'da yani task içine key-value eşleştirmesi ekler.*/
447                     tasks.put(key, sf);
448                 }
449             }
450         }
```

```
451
452       /*MainActivity'den gelen verilerin alınması sağlanır.
453       Daha sonra bu verilerin IoT-Ignite ortamına aktarılması sağlanır.
454       Bunu Runnable sınıfından türettik.*/
455       private class ThingDataHandler implements Runnable {
456
457           /*Gelen Thing bileşeni tutacak nesnemiz*/
458           Thing mThing;
459
460           /*Kurucu metodumuz Thign bileşeni alır.*/
461           ThingDataHandler(Thing thing) {
462               mThing = thing;
463           }
464
465           @Override
466           public void run() {
467
468               /*ThingData: IoT-Ignite ortamına Thing verilerini göndermek için
kullanılır.
469               Her Thing nesnesi veri göndermek veya aktüatörler gibi eylem mesajı
almak için bir ThingData nesnesine ihtiyaç duyar.*/
470
471               ThingData mThingData = new ThingData();
472
473               /*Gelen Thing, sıcaklık sensörü ise*/
474               if(mThing.equals(mTempThing)) {
475                   /*MainActivity arayüzünde bulunan sıcaklık değerini üreten SeekBar
kontrolüne erişim sağlanır.*/
476                   SeekBar seekBarTemperature =
activity.findViewById(R.id.seekBarTemperature);
477
478                   /*SeekBar'dan gelen veri ThingData nesnesine eklenir.*/
479                   mThingData.addData(seekBarTemperature.getProgress());
480               } else if(mThing.equals(mLampThing)) {
481                   /*Gelen Thing, lamba aktuatörü ise,*/
482                   /*MainActivity arayüzünde bulunan ve lambayı temsil eden Toggle
kontrolüne erişim sağlanır.*/
483                   ToggleButton toggleLamp =  activity.findViewById(R.id.toggleLamp);
484
485                   /*Toggle'dan gelen veri ThingData nesnesine eklenir.*/
486                   mThingData.addData(toggleLamp.isChecked() ? 1 : 0);
487               }
488
489               /*ThingData nesnesinde bulunan veri IoT-Ignite ortamına gönderilir.*/
490               if(mThing.sendData(mThingData)){
491                   Log.i(TAG, "VERİ GÖNDERİMİ BAŞARILI : " + mThingData);
492               }else{
493                   Log.i(TAG, "VERİ GÖNDERİMİ BAŞARISIZ");
494               }
495           }
496       }
497
498       /*Thing bileşenlerden veri okumayı sağlayan görevi durdurmayı sağlar.
499       Özellikle Thing bileşen IoT-Ignite ortama kayıtlı olmadığında veya yeni gelen
500       bir konfigürasyon bilgisini alırken  veri okumanın bu metot ile durdurulması
501       gerekiyor. Parametre olarak nodeId ve thingId bilgilerini vermemiz gerekiyor.*/
502       public void stopReadDataTask(String nodeId, String sensorId) {
503           /*nodeId ve thingId bilgileri birleştirilerek key bilgi elde edilir.*/
504           String key = nodeId + "|" + sensorId;
505
506           /*Task nesnesi gelen key bilgisini içeriyorsa if blogu çalışır.*/
507           if (tasks.containsKey(key)) {
508               try {
509                   /*Task içinde belirtilen key bilgisine sahip olan görev iptal
edilir.*/
510                   tasks.get(key).cancel(true);
511
512                   /*Task içinde belirtilen key bilgisine sahip olan görev silinir.*/
513                   tasks.remove(key);
514               } catch (Exception e) {
515                   Log.d(TAG, "Veri alma durdurulamadı" + e);
516               }
517           }
518       }
519
520       /********SENSÖR VERİLERİNİ IOT-IGNITE ORTAMINA GÖNDERMEK********/
521       /*IoT-Ignite ortamına bağlantı sağlandığı zaman aşağıdaki metot çağrılır.
522       Burada sensör ve aktüatör için ilk değer ataması gerçekleşir. */
```

```
523      private void sendInitialData() {
524          /*sendData(): Bu metot ID bilgisi verilen Thing bileşenlere değer atamak
için kullanılır.*/
525          sendData(Constants.TEMP_NAME, Constants.FIRST_VALUE_FOR_TEMP);
526          sendData(Constants.LAMP_NAME, Constants.FIRST_VALUE_FOR_LAMP);
527      }
528
529      /*Ignite ortamına veri göndermeyi sağlar. Bu işlemin yapılabilmesi için
getDataReadingFrequency() metodu READING_WHEN_ARRIVE değerine sahip olmalıdır.*/
530
531      public void sendData(String thingId, int value) {
532
533          /*Ignite bağlantısı varsa*/
534          if(igniteConnected) {
535              try {
536
537                  /*Node ID bilgisi IoT-Ignite ortamından alınır.*/
538                  Node mNode = mIotIgniteManager.getNodeByID(Constants.NODE_ID);
539
540                  /*Node null değilse işlem yapılır.*/
541                  if(mNode != null) {
542
543                      /*Node'a bağlı olan ve thingId ID bilgisine sahip olan Thing
nesnesi alınır.*/
544                      Thing mThing = mNode.getThingByID(thingId);
545
546                      /*Thing null değilse işlem yapılır.*/
547                      if (mThing != null) {
548
549                          /*ThingData: IoT-Ignite ortamına Thing verilerini göndermek
için kullanılır.
550                          Her Thing nesnesi veri göndermek veya aktüatörler gibi
eylem mesajı almak için bir ThingData nesnesine ihtiyaç duyar.*/
551
552                          ThingData mthingData = new ThingData();
553
554                          /*Gelen veri ThingData nesnesine eklenir.*/
555                          mthingData.addData(value);
556
557                          /*getDataReadingFrequency() metodu READING_WHEN_ARRIVE
değerini üretiyorsa IoT-Ignite platformuna veri gönderilebilir.
558                          Aksi durumda gönderilemez*/
559                          if (isConfigReadWhenArrive(mThing) &&
mThing.sendData(mthingData)) {
560                              Log.i(TAG, "VERİ GÖNDERİMİ BAŞARILI : " + mthingData);
561                          } else {
562                              Log.i(TAG, "VERİ GÖNDERİLEMEDİ!");
563                          }
564                      } else {
565                          Log.i(TAG, thingId + " kayıtlı değil!");
566                      }
567                  } else {
568                      Log.i(TAG, Constants.NODE_ID + " kayıtlı değil!");
569                  }
570              } catch (AuthenticationException e) {
571                  Log.i(TAG, "AuthenticationException!");
572              }
573          } else {
574              Log.i(TAG, "Ignite Bağlantısı Kesildi!");
575          }
576      }
577
578      /*getDataReadingFrequency() metodu READING_WHEN_ARRIVE değerini üretiyorsa IoT-
Ignite platformuna veri gönderilebilir. Aksi durumda gönderilemez*/
579
580      private boolean isConfigReadWhenArrive(Thing mThing) {
581          if (mThing.getThingConfiguration().getDataReadingFrequency() ==
ThingConfiguration.READING_WHEN_ARRIVE) {
582              return true;
583          }
584          return false;
585      }
586  }
```

MainActivity Sınıfı

MainActivity sınıfı uygulamanın ana sayfasıdır. Bu sınıfın amacı VirtualCustomerNodeHandler sınıfını kullanarak IoT-Ignite platformuna bağlanmayı sağlamak ve sanal sensörlerle arayüzde bulunan kontroller arasındaki iletişimi sağlamaktır. Sınıfın kodları aşağıdaki gibidir.

MainActivity.java

```
1    package com.mas.basiccustomerapp;
2
3    import android.os.Bundle;
4    import android.support.v4.content.ContextCompat;
5    import android.support.v7.app.AppCompatActivity;
6    import android.widget.CompoundButton;
7    import android.widget.ImageView;
8    import android.widget.SeekBar;
9    import android.widget.TextView;
10   import android.widget.ToggleButton;
11
12
13   public class MainActivity extends AppCompatActivity {
14
15       /*Arayüzde bulunan kontroller için değişken tanımlama.*/
16       private TextView temperatureValue;
17       private SeekBar seekBarTemperature;
18       private ImageView imageLamp;
19       private ToggleButton toggleLamp;
20       private String centigrade = " \u00b0 C";
21
22       /*VirtualCustomerNodeHandler
23       bu sınıf özetle IOT-IGNITE PLATFORMUNA BAĞLANTI,
24       THING OLUŞTURMA VE IGNITE PLATFORMUNA KAYIT ETMEK,
25       MAINACTIVITY ARAYÜZÜNDEKİ SENSÖR VERİLERİNİ OKUMAK VEYA OKUMAYI İPTAL ETMEK,
26       SENSÖR VERİLERİNİ IGNITE ORTAMINA GÖNDERMEK ve daha birçok işlemi yapmak
27       için kullanılmaktadır.*/
28       private VirtualCustomerNodeHandler customerNodeHandler;
29
30       @Override
31       protected void onCreate(Bundle savedInstanceState) {
32           super.onCreate(savedInstanceState);
33           setContentView(R.layout.activity_main);
34           /*Parametre olarak MainActivity activitysini gönderdik.*/
35           customerNodeHandler = new VirtualCustomerNodeHandler(this);
36
37           /*start metodu ile IoT-Ignite platformuna bağlantı sağlanır.*/
38           customerNodeHandler.start();
39
40           /*Arayüz için bu metot çağrılır.*/
41           initUIComponents();
42       }
43
44       /*MainActivity arayüzünde bulunan kontrollere erişim sağlanır.*/
45       private void initUIComponents() {
46
47           /*Sıcaklık sensörü için kontrol erişimi*/
48           seekBarTemperature = findViewById(R.id.seekBarTemperature);
49           temperatureValue = findViewById(R.id.textTemperatureValue);
50
51           /*Kontrollere ilk değerler atanır. Sıcaklık 21, lamba ise yanık olur.*/
52           seekBarTemperature.setProgress(Constants.FIRST_VALUE_FOR_TEMP);
53           temperatureValue.setText(String.format("%d" + centigrade,
Constants.FIRST_VALUE_FOR_TEMP));
54
55           /*sıcaklık 0 ile 100 arasında üretilir.*/
56           seekBarTemperature.setMax(100);
57
58           /*SeekBar değeri değiştiğinde algılamayı sağlayan bir listener eklenir.*/
59           seekBarTemperature.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
60
61               @Override
62               public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
63                   /*Yeni değer kullanıcının okuyabilmesi için textview'e yazılır.*/
64                   temperatureValue.setText(String.format("%d" + centigrade,i));
65               }
```

```
66
67                  @Override
68                  public void onStartTrackingTouch(SeekBar seekBar) {
69
70                  }
71
72                  @Override
73                  public void onStopTrackingTouch(SeekBar seekBar) {
74                      /*SeekBar üzerinde yapılan hareket durduğunda üretilen değer
75                      sendData metodu ile IoT-Ignite platformuna gönderilir.*/
76                      customerNodeHandler.sendData(Constants.TEMP_NAME,
seekBar.getProgress());
77                  }
78              });
79
80              /*Lamba için kontrol erişimi*/
81              imageLamp = findViewById(R.id.imageLamp);
82              toggleLamp = findViewById(R.id.toggleLamp);
83
84              /*Lambamız başlangıçta açık olarak ayarlanır.*/
85              toggleLamp.setChecked(true);
86
87              /*Açık lamba resmi kullanıcıya gösterilir.*/
88
imageLamp.setImageDrawable(ContextCompat.getDrawable(getApplicationContext(),
R.drawable.lamp_on));
89
90              /*Toggle'ın işaret durumu değiştiğinde algılamayı sağlayan bir listener
tanımlanır.*/
91              toggleLamp.setOnCheckedChangeListener(new
ToggleButton.OnCheckedChangeListener() {
92
93                  @Override
94                  public void onCheckedChanged(CompoundButton compoundButton, boolean
isChecked) {
95                      /*Toggle'ın işaret durumu yani true veya false durumu isChecked
değişkenindendir.
96                      İşarete göre arayüzdeki görsel değiştirilir. Her iki sonuç içinde
üretilen veri IoT-Ignite ortamına sendData ile gönderilir.*/
97
98                      if(isChecked) {
99
imageLamp.setImageDrawable(ContextCompat.getDrawable(getApplicationContext(),
R.drawable.lamp_on));
100                         customerNodeHandler.sendData(Constants.LAMP_NAME, 1);
101                     } else {
102
imageLamp.setImageDrawable(ContextCompat.getDrawable(getApplicationContext(),
R.drawable.lamp_off));
103                         customerNodeHandler.sendData(Constants.LAMP_NAME, 0);
104                     }
105                 }
106             });
107     }
108
109     @Override
110     protected void onDestroy() {
111         super.onDestroy();
112         /*Uygulama sonlandığı zaman IoT-Ignite bağlantısı kesilir.*/
113         customerNodeHandler.stop();
114     }
115 }
```

AndroidManifest Dosyası

Uygulama hakkında bilgilerin bulunduğu, uygulamada bulunan activity, service gibi bileşenlerin eklendiği ve ihtiyaç duyulan izinlerin tanımlandığı AndroidManifest dosyamız bu uygulamada aşağıdaki gibi olacaktır.

AndroidManifest.xml

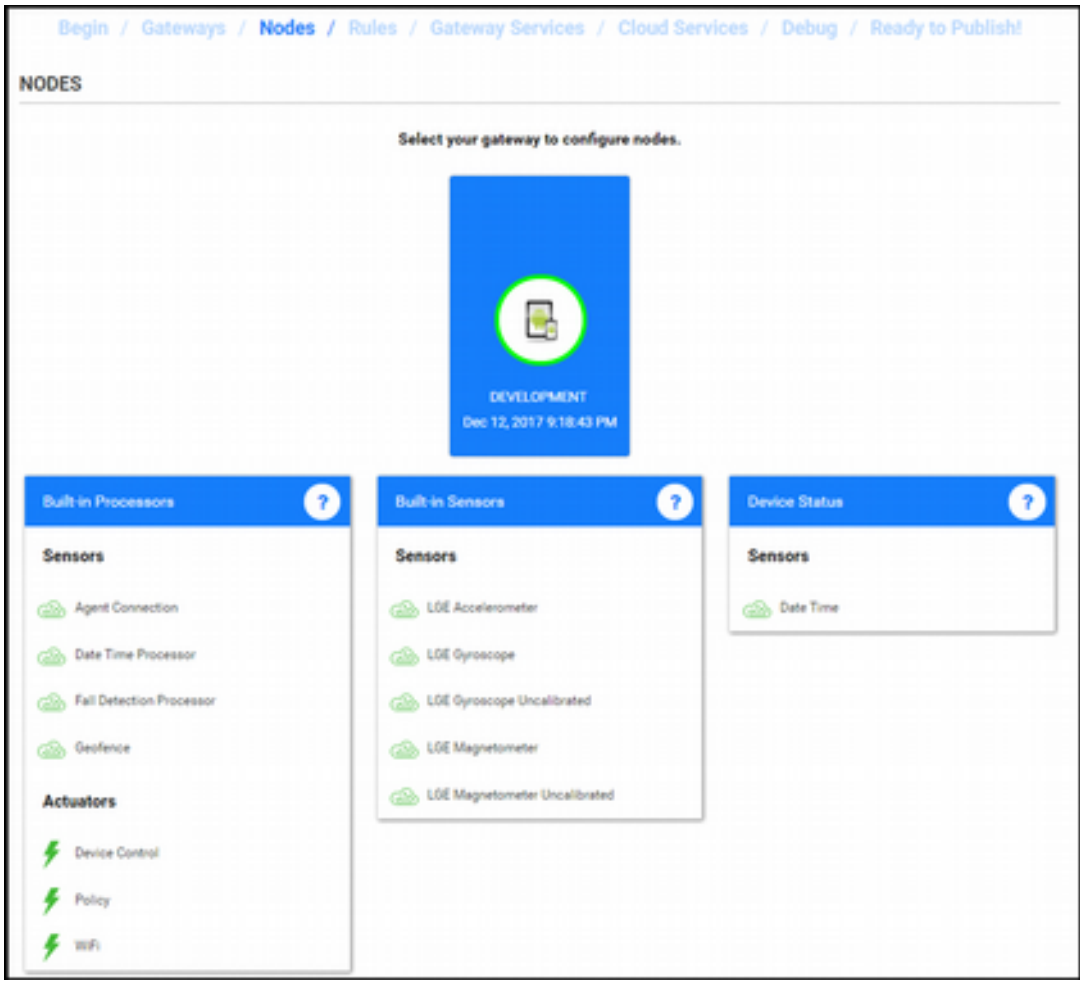
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mas.basiccustomerapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <!--
        Uygulamanın ana giriş noktası veya ana sayfası MainActivity isimli
etkinliğimiz olacak.
        -->
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Uygulama için ihtiyacımız olan kodların tamamını açıklamalarıyla birlikte sizlerle paylaştık. Bu uygulamanın özelliği, IoT-Ignite platformunda herhangi bir işlem yapmadan doğrudan Java kodlarıyla sanal olarak Node ve Thing oluşturmayı sağlamaktır. Aynı zamanda bu sensörlere Cloud ortamında kural tanımlayabilir hatta bu sensörlerden gelen verileri Cloud ortamına aktarabilirsiniz.

Uygulamayı AppStore'a Yüklemek

Müşteri uygulamasını oluşturduk. Şimdi bu uygulamayı EHUB ortamında bulunan AppStore'a nasıl yükleyebiliriz bundan bahsedelim. Bir uygulamanın IoT-Ignite platformu ile iletişime geçebilmesi için uygulamanın Trusted, yani güvenilir hale getirilmesi gerekir. Bu yapılmadığı zaman uygulamanız hiçbir şekilde IoT-Ignite platformu ile iletişime geçemez.

Uygulamayı AppStore'a yükleyebilmek için öncelikle Devzone ortamına üye olmanız ve bir Gateway cihazını sisteme kayıt etmeniz gerekiyor. Burada Gateway olarak Android bir telefonu kullandık. Bizim kullandığımız cihazın Devzone ortamındaki görüntüsü aşağıdaki gibidir.

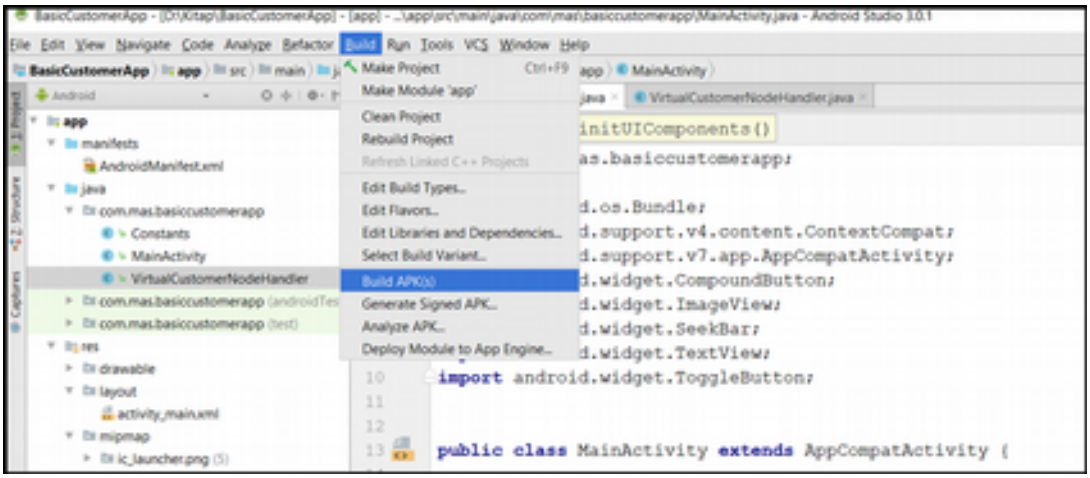


Yukarıda cihazda bulunan Node'ların listesi gösterilmektedir. Buradaki işlemleri yaptıktan sonra **VirtualCustomerNode** isimli bir Node daha listeye eklenecektir.

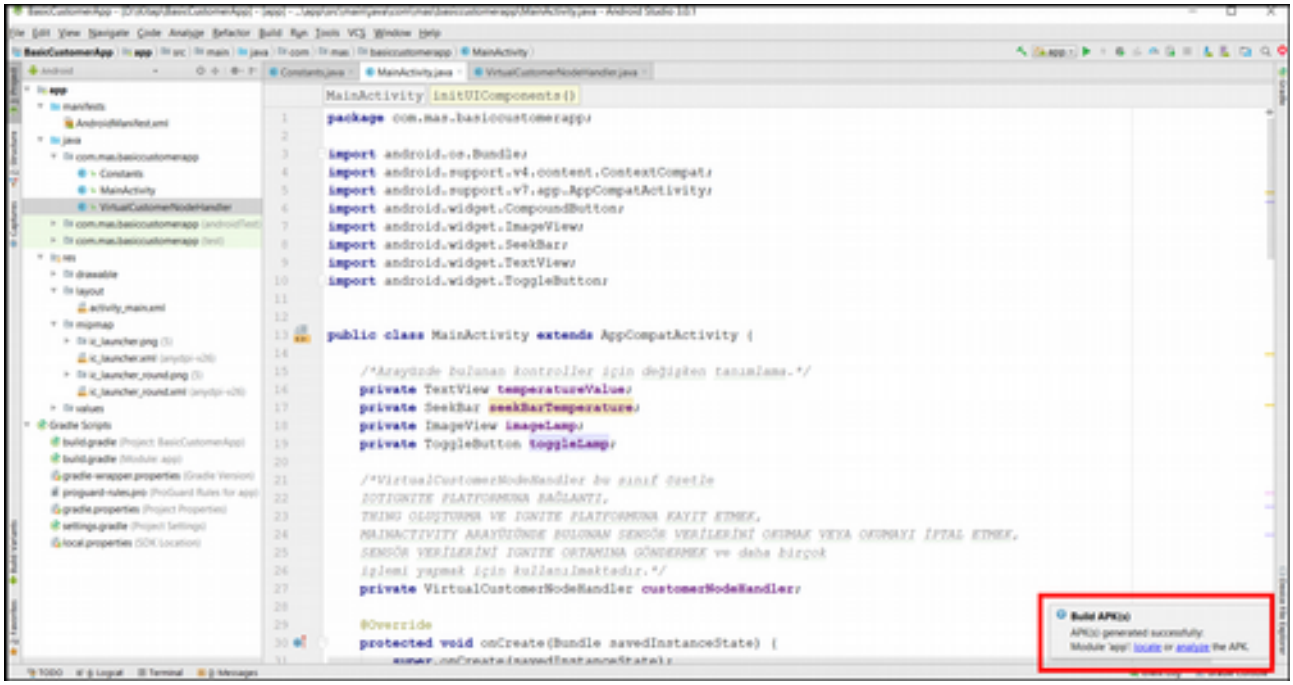
Uygulamayı güvenilir olarak kayıt etmek ve sertifika oluşturmak için sırasıyla şu başlıkları uygulamamız gerekiyor.

APK Oluşturmak

Yapılması gereken ilk işlem uygulamanın APK dosyasını oluşturmaktır. Bunun için Android Studio ortamında **Build > Build APK** yolunu takip ederek APK oluşturabilirsiniz.



Bu yolu takip ettikten sonra Android Studio ortamında sağ alt köşede APK dosyasının oluşturulduğu bize gösterilir. Buraya tıklayarak uygulamanın bulunduğu path bilgisini kopyalayınız.

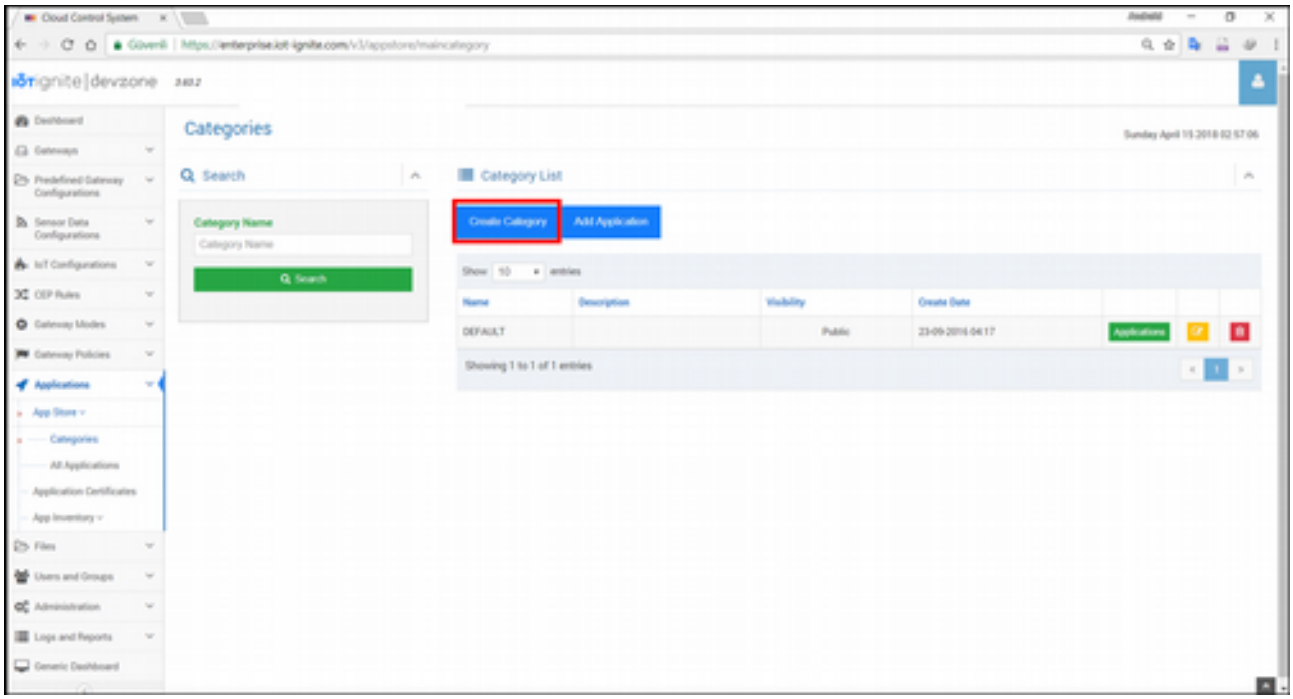


Uygulama Sertifikasını Cihaz Moduna Eklemek

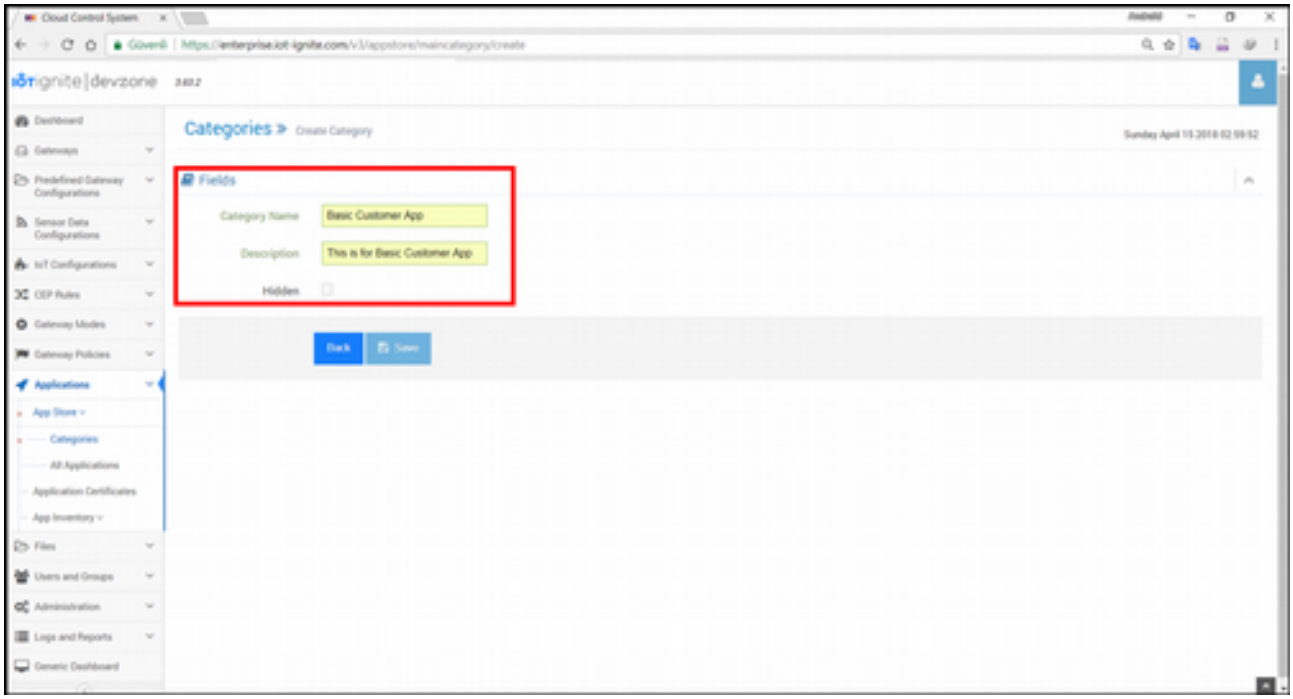
APK dosyamızı oluşturduktan sonra şimdi de bunu AppStore'a yükleyip nasıl sertifika oluşturacağımızı gösterelim. Öncelikle aşağıda verilen linkten EHUB ortamına giriş yapalım. Devzone için kullandığınız giriş bilgilerini burası içinde kullanabilirsiniz.

https://enterprise.iot-ignite.com/v3/access/login

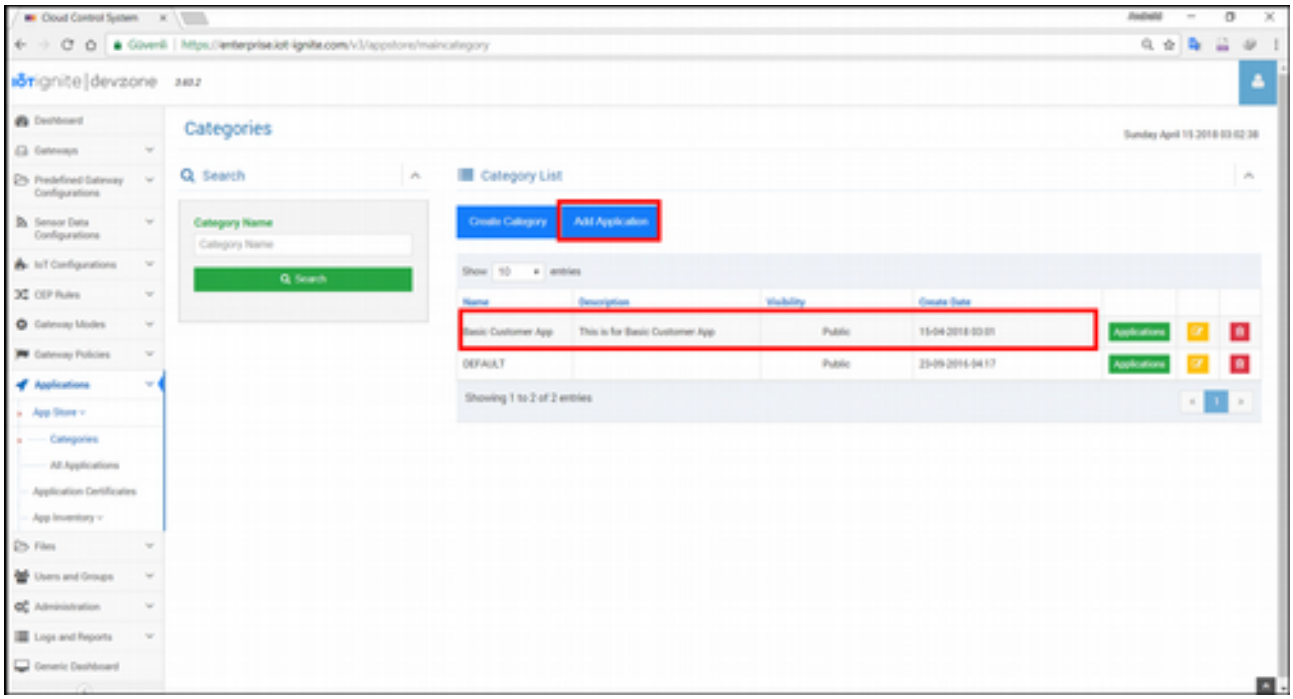
Giriş yaptıktan sonra menüden **Application > AppStore > Categories** yolunu takip ederek aşağıdaki sayfaya geçiniz.



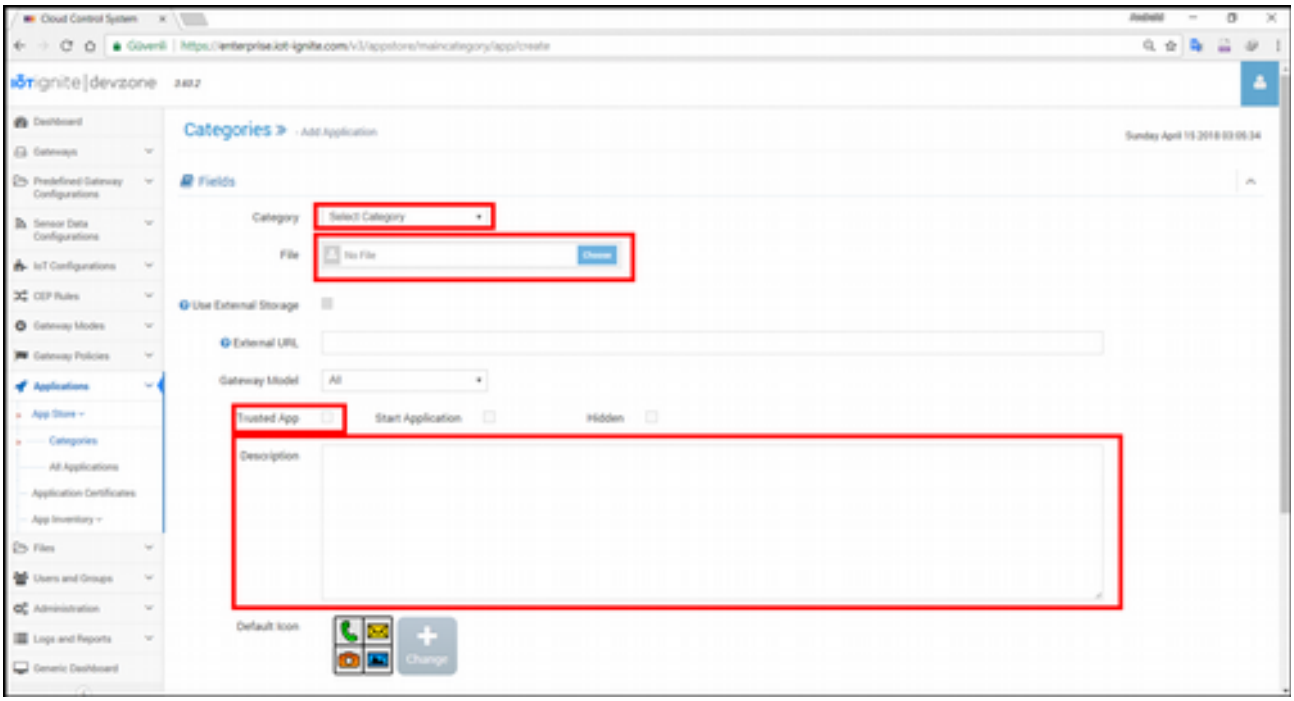
Açılan sayfada kırmızı alan içine aldığımız **Create Category** butonuna tıklayınız. Bunu kullanarak yeni bir kategori oluşturacağız. Açılan sayfaya aşağıdaki bilgileri giriniz.



Save butonuna tıklayarak yeni kategorinizi ekleyebilirsiniz. Kayıt işleminden sonra tekrar **Categories** seçeneğine tıklayınız. Açılan sayfada oluşturduğunuz yeni kategoriyi görebilirsiniz.



Bu işlemden sonra yukarıda yer alan **Add Application** butonuna tıklayalım.

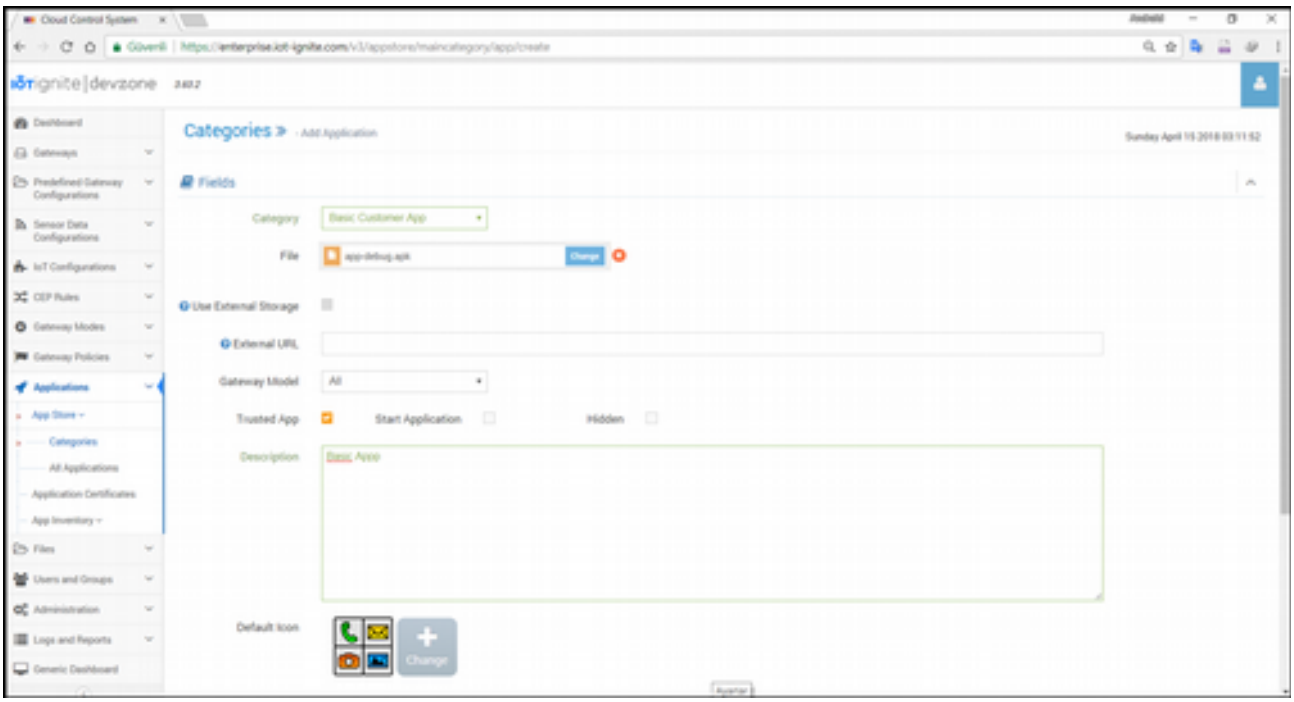


Açılan sayfada yukarıda özellikle işaretlenen alanların uygun şekilde doldurulması gerekiyor. Bunlar hakkında kısaca bilgi vermekte fayda var.

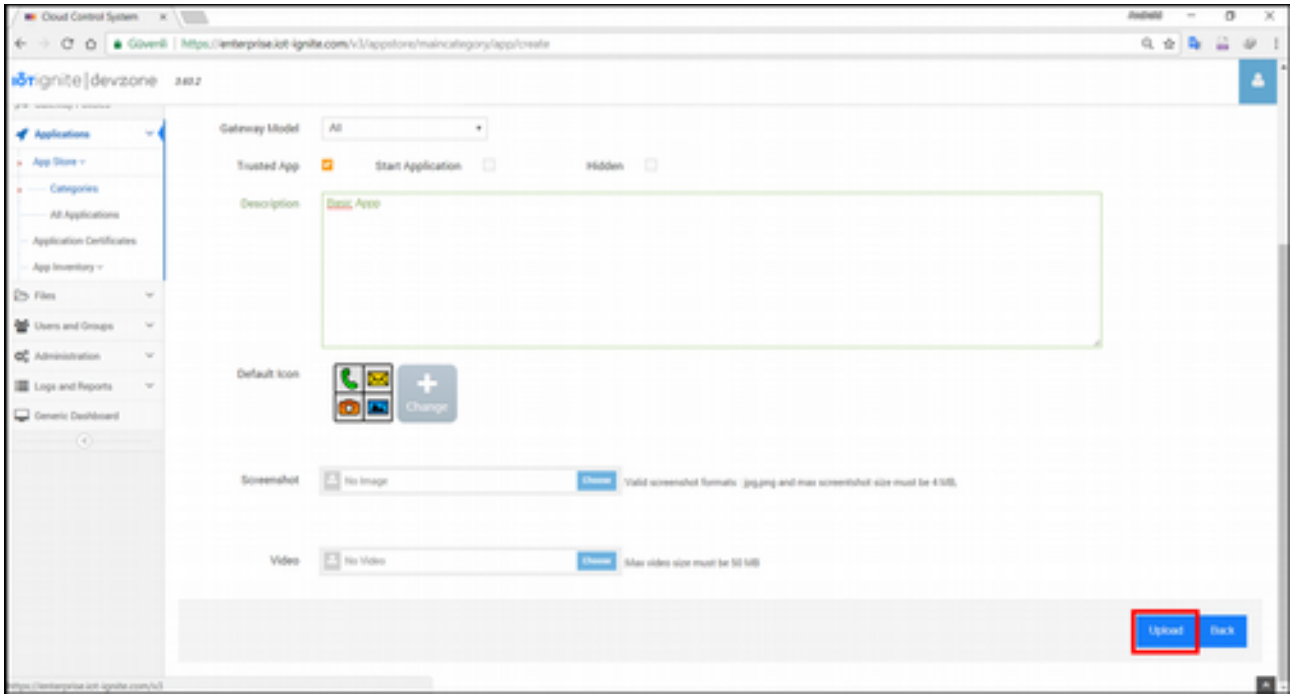
- **Category**: Kayıtlı olan bir kategori seçmek için kullanılır. Burada Basic Customer App kategorisini seçeceğiz.
- **File**: AppStore ortamında yüklemek istediğiniz APK dosyası buradan seçilir.
- **Trusted App**: Bu seçeneği özellikle check (aktif) etmeniz gerekiyor. Bunu yapmadığınızda uygulamanız IoT-Ignite ile iletişime geçemez.

- **Description**: Bu alana yüklediğiniz uygulama hakkında açıklama ekleyebilirsiniz.

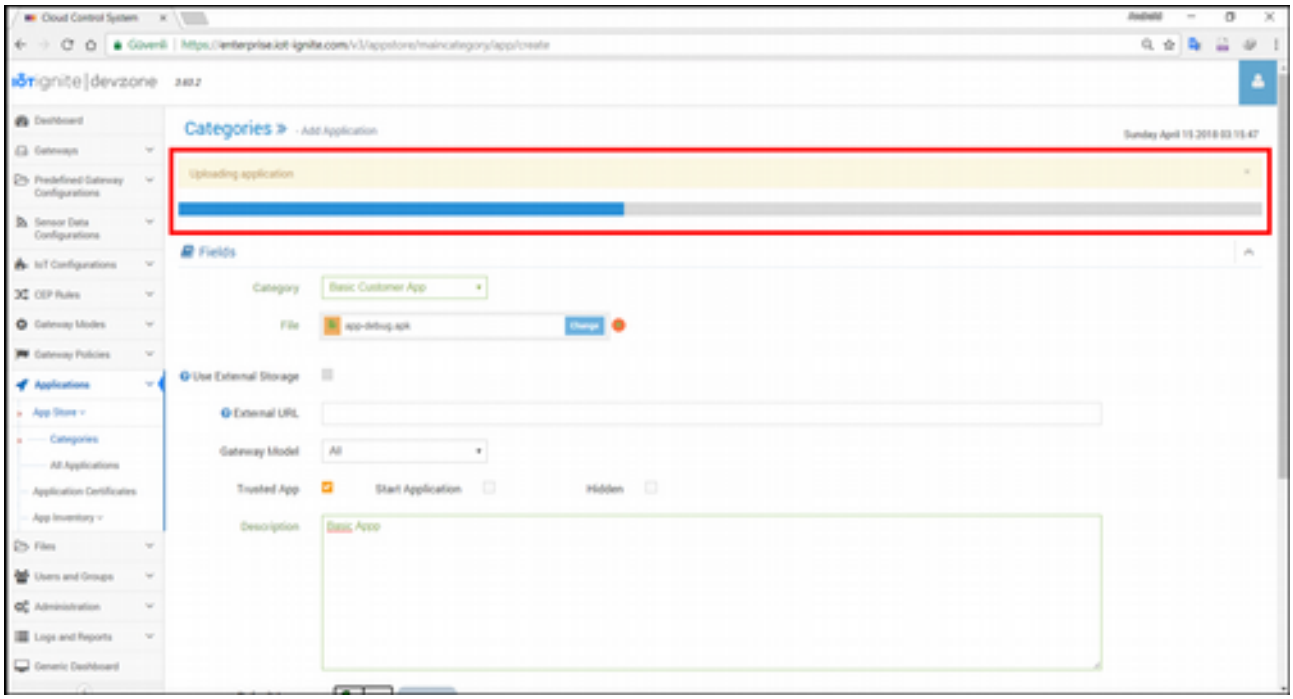
Yukarıda açıklamalarını yaptığımı alanları aşağıdaki gibi doldurunuz.



Bu işlemleri tamamladıktan sonra sayfayı aşağı kaydırıp **Upload** butona tıklayınız.

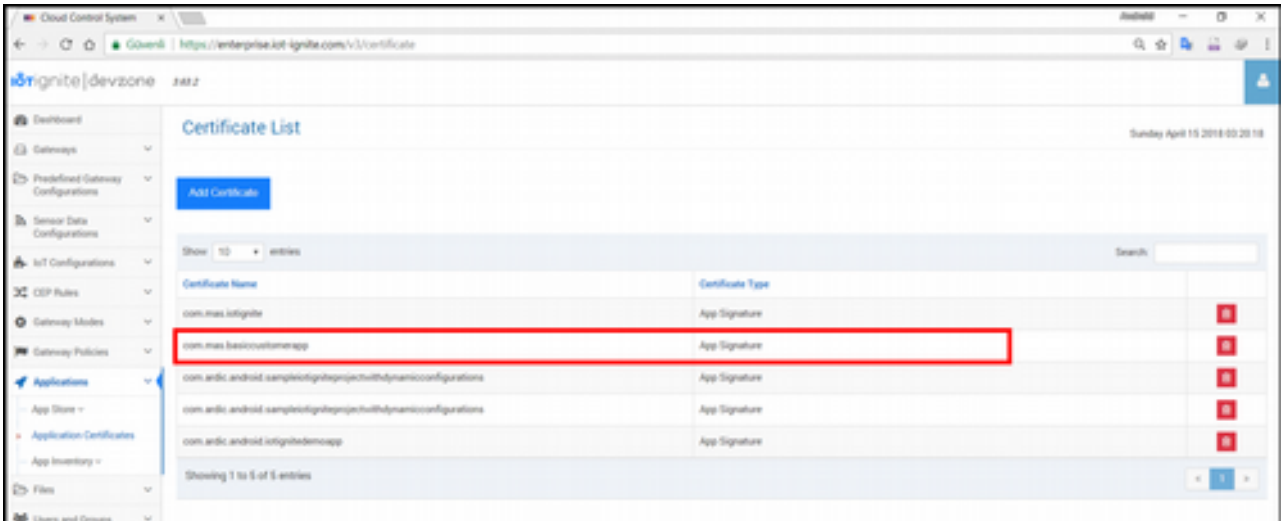


Bunu yaptıktan sonra aşağıdaki gösterildiği gibi uygulamanızın yüklenmesi bir iş çubuğu (progress bar) içinde gösterilir.



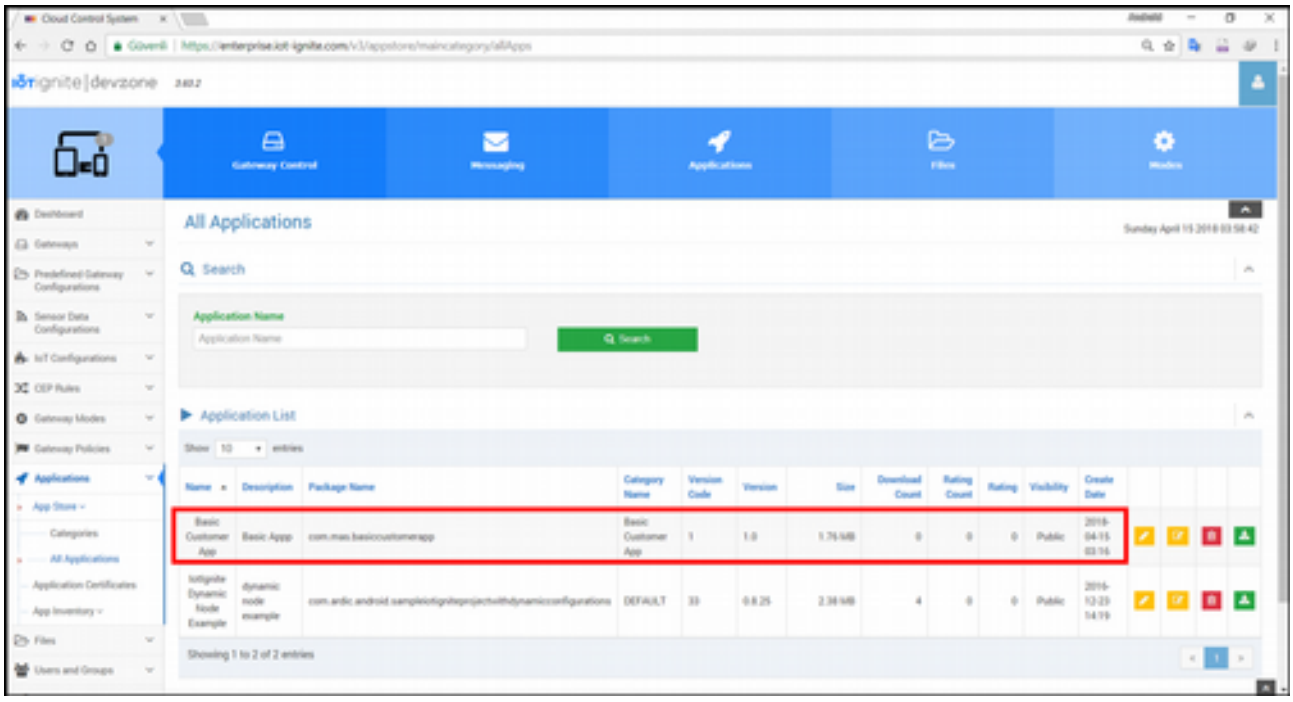
Yükleme tamamlandıktan sonra dikdörtgen alan için aşağıdaki mesajı okumanız gerekiyor. Eğer bunu görmezseniz yükleme işlemini yeniden denemeniz gerekebilir.

Uygulama başarılı bir şekilde yüklendiğine göre **Application > Application Certificates** yolunu takip ederek aşağıdaki sayfaya geçelim. Buradan yüklediğimiz uygulama için bir sertifika üretildiğini görebilirsiniz.



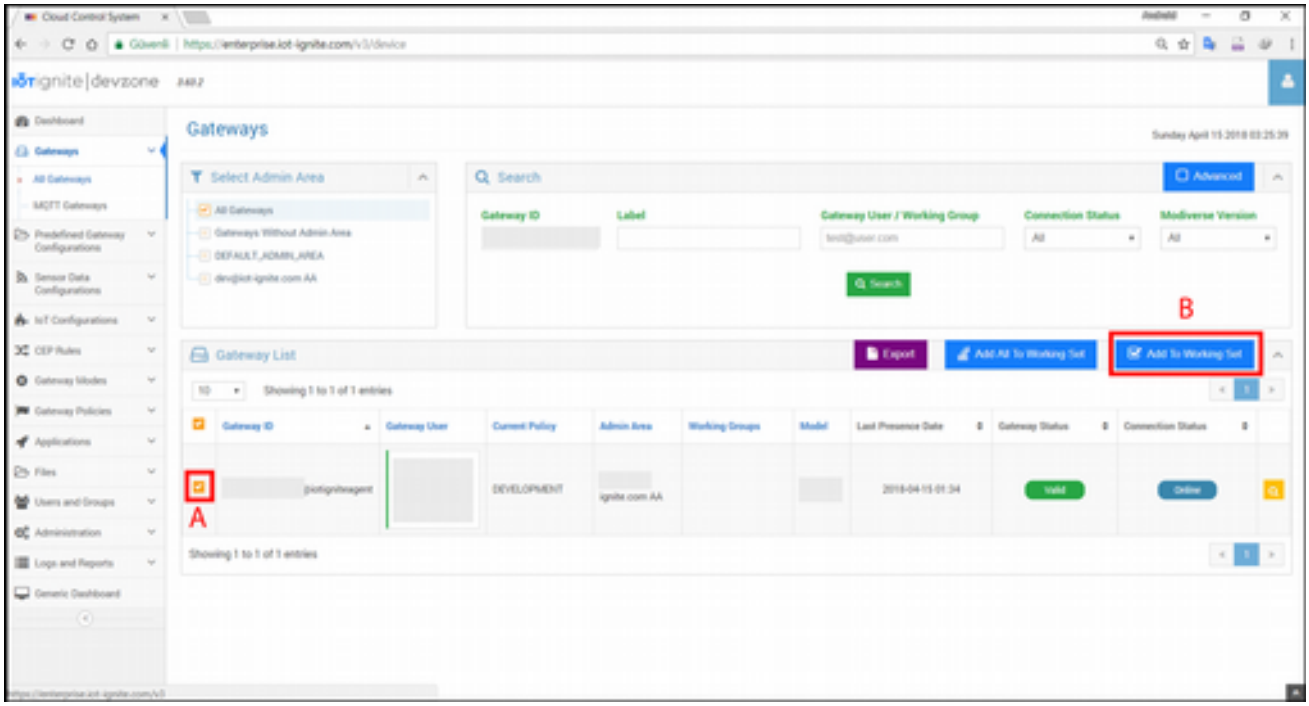
Burada sertifika isminin uygulamanın paket ismi olduğuna dikkat etmenizi istiyoruz. Bu şekilde kontrolünüzü yapabilirsiniz. Ayrıca **Application > AppStore > All Application** yolunu takip ederek

uygulamanızın yüklendiğini daha net görebilirsiniz.



Modu Cihaza Göndermek

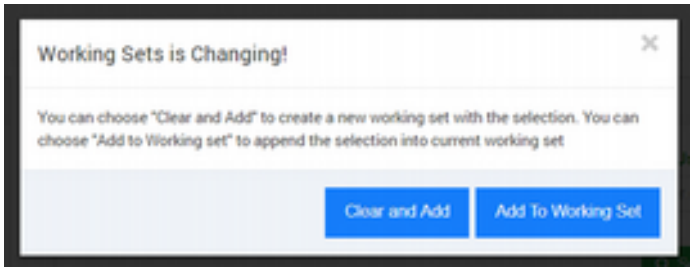
Uygulamayı AppStore'a yükleyip sertifikasını ürettikten sonra şimdi bu yapılanları Gateway moduna eklemeyi ve bunları cihaza göndermeyi gösterelim. Öncelikle **Gateways --> All Gateways** yolunu takip ederek aşağıdaki sayfaya geçelim.



Aşağıda verilen A ve B adımlarını uygulamanız gerekiyor.

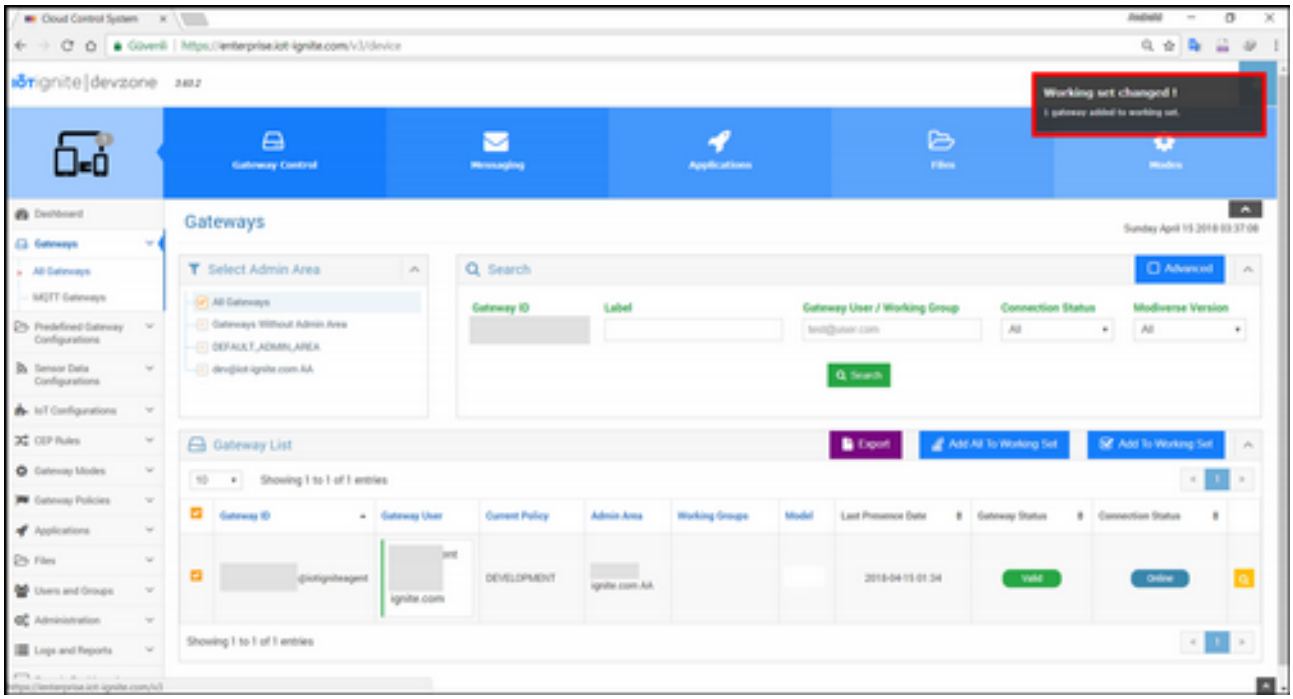
- **A:** Yapılan işlemlerin hangi Gateway cihazına gönderileceği belirlenir. Burada bir tane Gateway cihazımız bulunduğu için sadece onu seçtik.
- **B:** Bu adımda ise yapılan işlemlerin, seçili Gateway cihazının moduna eklenmesi sağlanır.

Add To Working Set butonuna tıkladıktan sonra aşağıdaki pencere açılır.

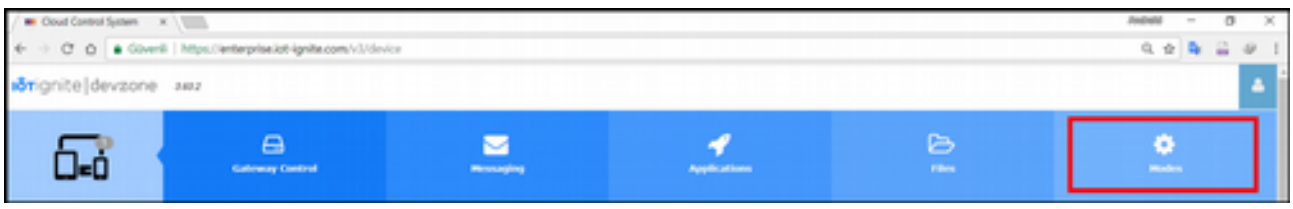


Tıklama işlemi ile birlikte karşınıza gelecek olan seçimden **Clear and Add** butonu ile Working Set'i temizleyip yeniden ekleme yapabilirsiniz. Veya Working Set'te daha önceden Gateway'ler varsa ve onların üzerine ekleme yapmak istiyorsanız **Add To Working Set** butonuna tıklayın. Şu an Working Set boş olduğu için hangisini seçerseniz seçin, fark etmez.

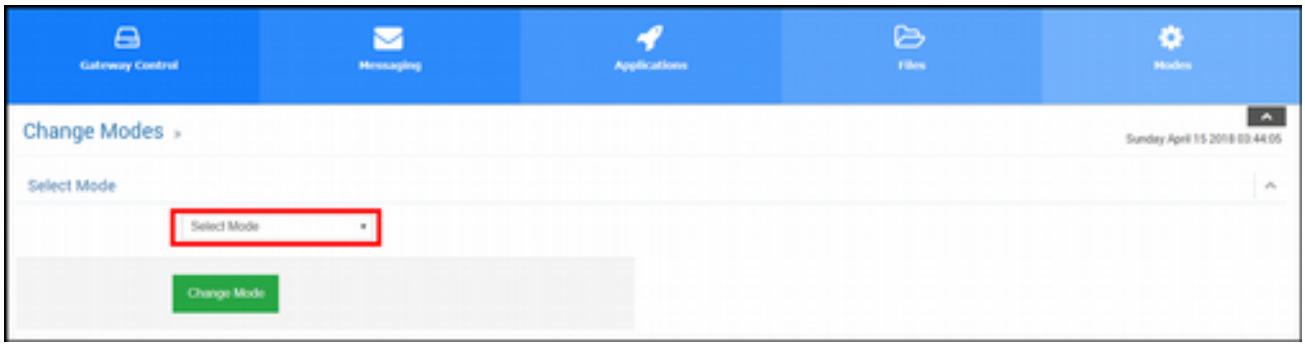
Herhangi bir seçeneğe tıkladıktan sonra sağ üst köşede kırmızı alanda yer alan mesaj gösterilir. Bu mesaj ile birlikte yapılan işlemler moda uygulanır.



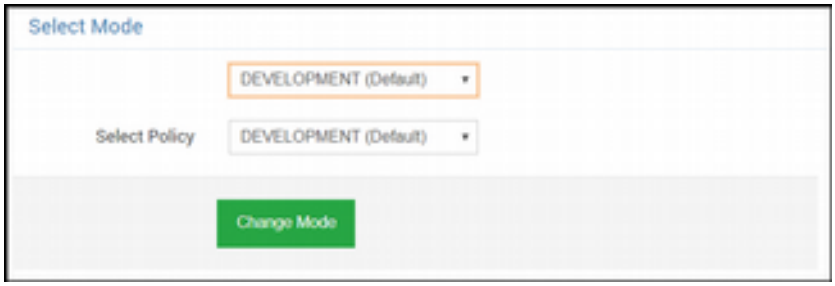
Yapılan işlemleri yukarıdaki gibi moda dahil ettikten sonra bunu cihazımıza gönderebiliriz. Bunun için yukarıdaki pencerede bulunan ve aşağıda verilen **Modes** seçeneğine tıklayınız.



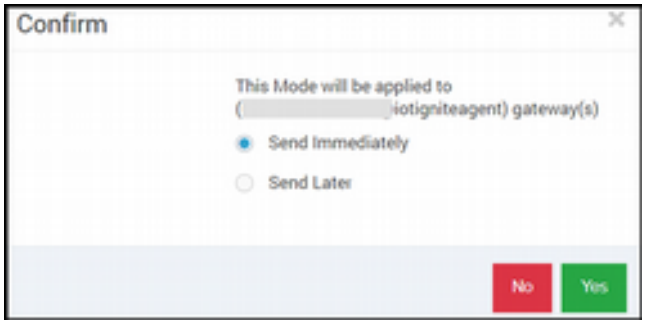
Modes seçeneğine tıkladıktan sonra aşağıdaki sayfa açılır.



Açılan pencerede kırmızı alan içinde olan açılır menüden aşağıdaki seçeneği seçiniz.

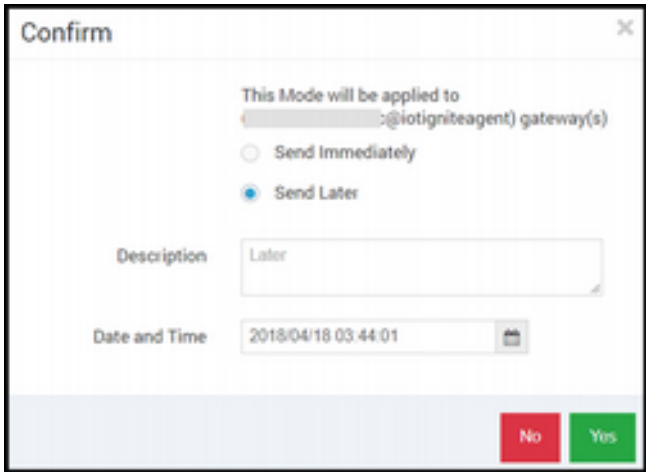


Burada kullandığımız cihaz şuanda **DEVELOPMENT** modunda olduğu için bunu kullanıyoruz. Bu seçeneği seçtikten sonra **Change Mode** butonuna tıklayalım.

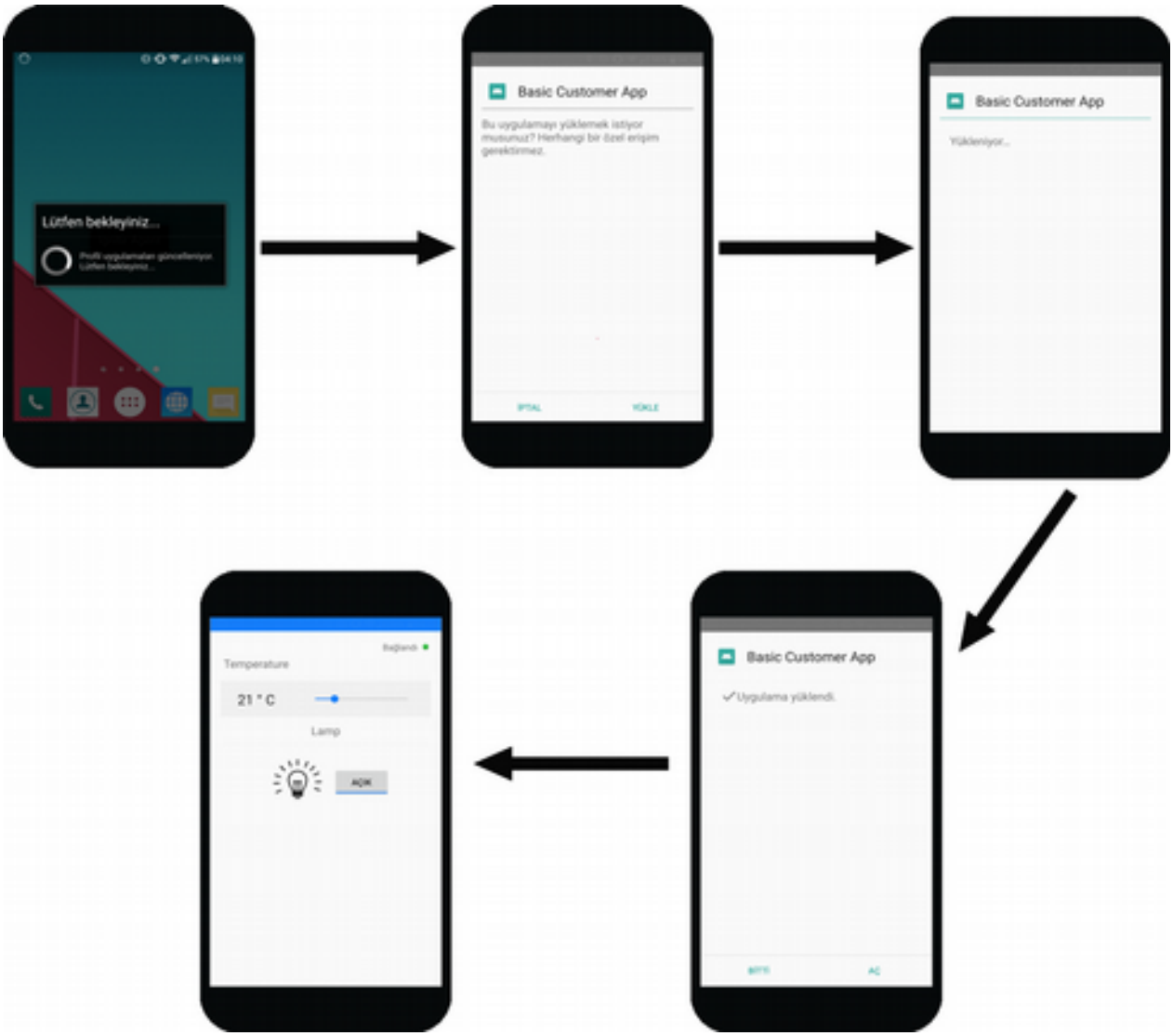


Açılan pencerede iki seçenek bunulur;

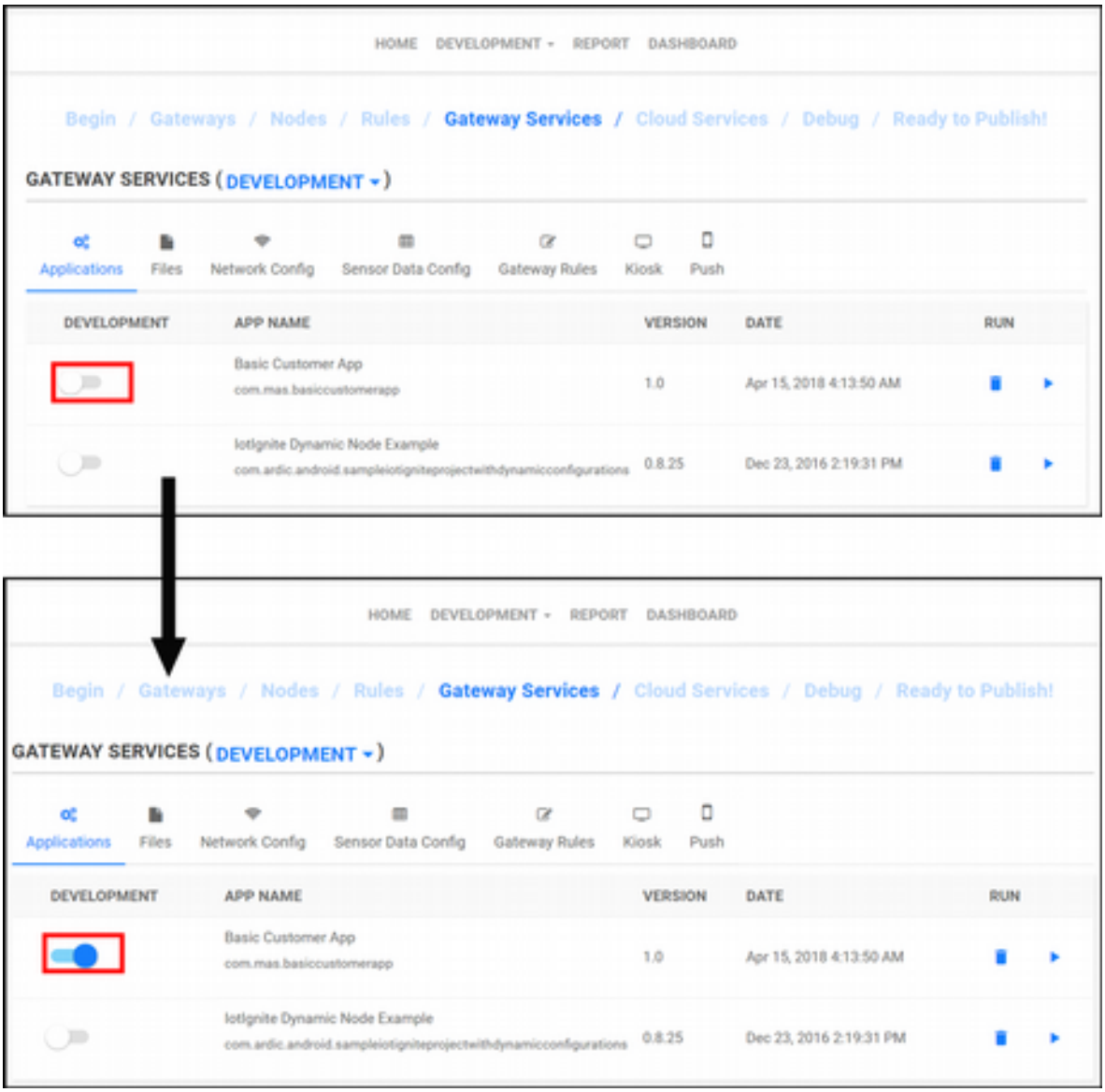
- **Send Immediately**: Bu seçenek ile mod, cihaza hemen gönderilir. Biz bunu seçeceğiz.
- **Send Later**: Bu seçenekte ise modun ileri bir tarih ve saatte yüklenmesi sağlanır.



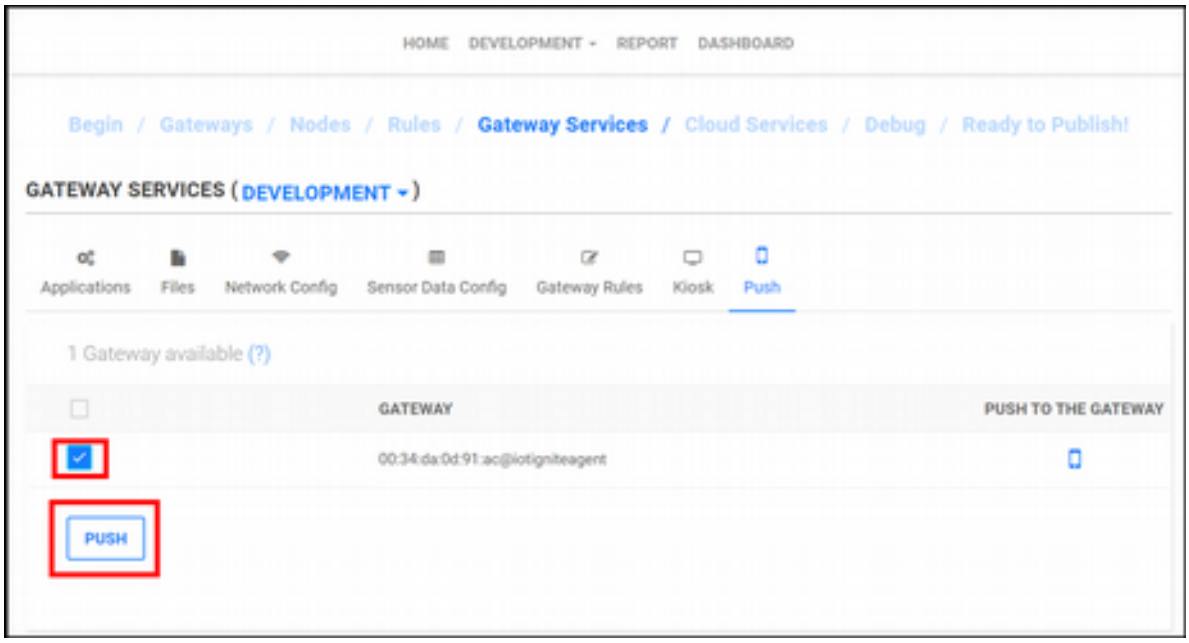
Send Immediately seçeneğini seçip **Yes** butonuna tıklayalım. Bu işlemle birlikte modumuz Gateway'e gönderilir. Tabi gönderme işlemi bittikten sonra müşteri uygulamamız akıllı telefona gönderilir. Aşağıda verilen adımları takip ederek uygulamayı yükleyip çalıştırabiliriz.



Eğer mod gönderildiği halde uygulamanızın APK'si akıllı telefona gelmezse (bağlantı kopması durumu vb.) aynı şekilde Devzone ortamından da uygulamanızı tekrar gönderebilirsiniz. Bunun için Devzone ortamında bulunan aşağıdaki sayfaya geçiniz ve yapılan işlemi sizlerde aynı şekilde uygulayınız.



Buradaki işlemi yaptıktan sonra aşağıdaki sayfaya geçelim.

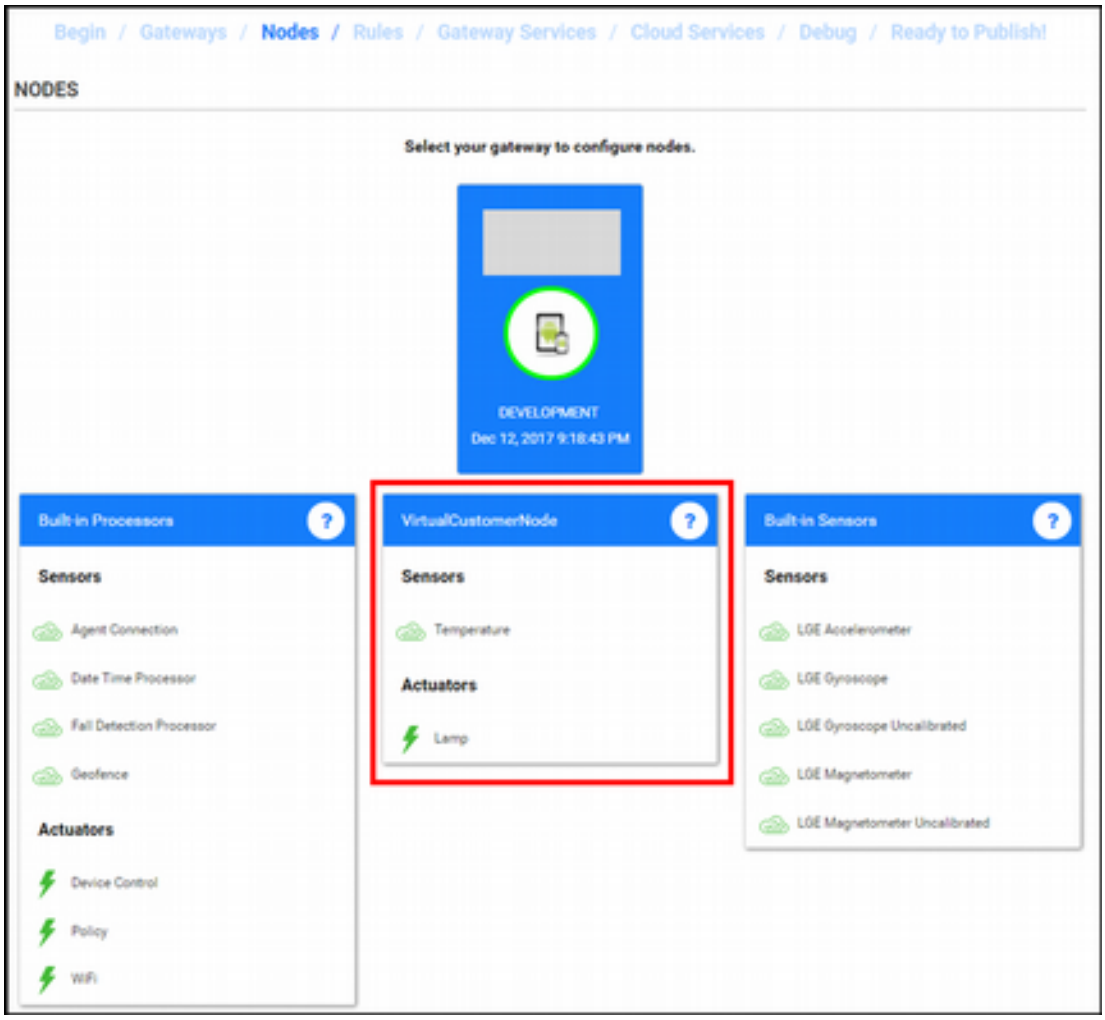


Açılan sayfada uygulamayı hangi Gateway'e göndermek istediğinizi seçip PUSH butonu ile APK dosyasını cihaza iletebilirsiniz. Bunu yaptıktan sonra daha önce gösterilen adımları takip ederek uygulamanızı yükleyebilirsiniz.

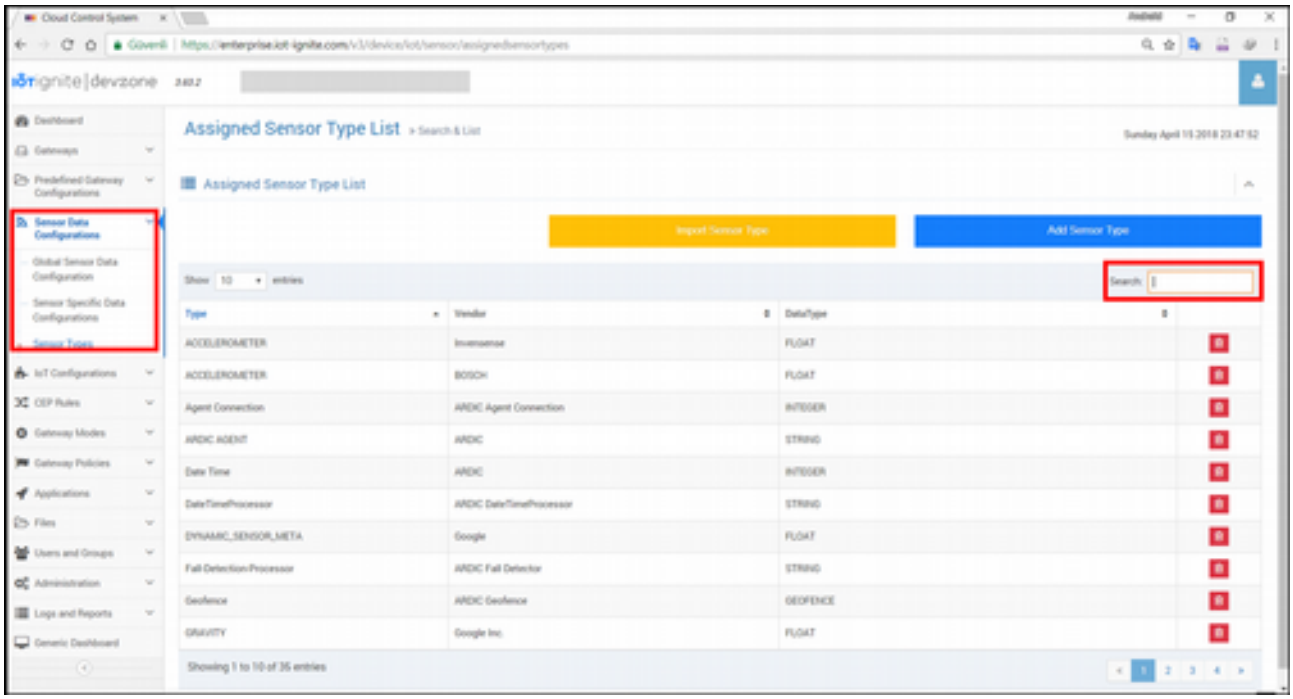
Node ve Thing Bileşenlerin Devzone ve EHUB'ta Oluşturulması

Android uygulamasında yazdığımız kodlar ile amaç, sanal sensör ve node bileşenlerin IoT-Ignite platformunda oluşturulmasını sağlamaktır. Sensör ve Node bileşeni IoT-Ignite ortamında oluşturmak için müşteri uygulamasını çalıştırmamız yeterlidir. Uygulama çalıştıktan hemen sonra Android uygulaması için yazdığımız kodlar icra edilerek EHUB ve Devzone ortamında sanal sensör ve sanal node'ların oluşturulması sağlanır.

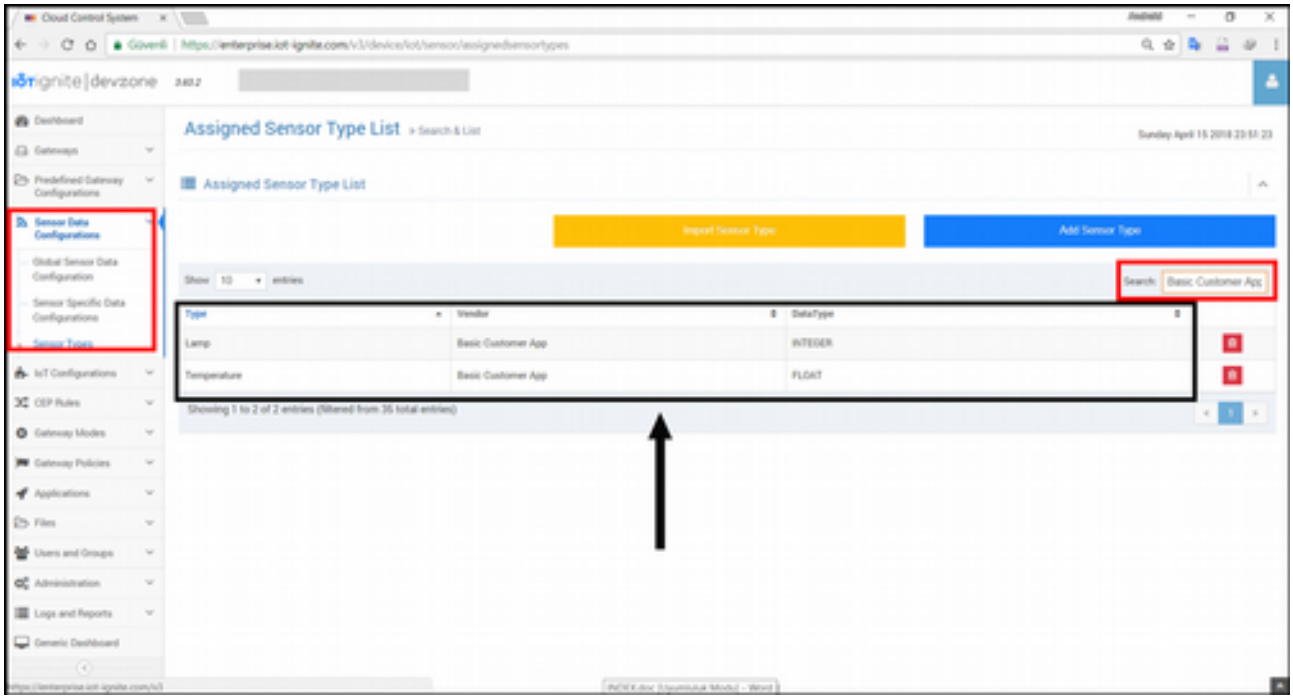
Müşteri uygulamasını çalıştırdıktan sonra Devzone ortamında, oluşturulan sanal sensör ve node bileşenlerini aşağıdaki gibi görmemiz gerekiyor.



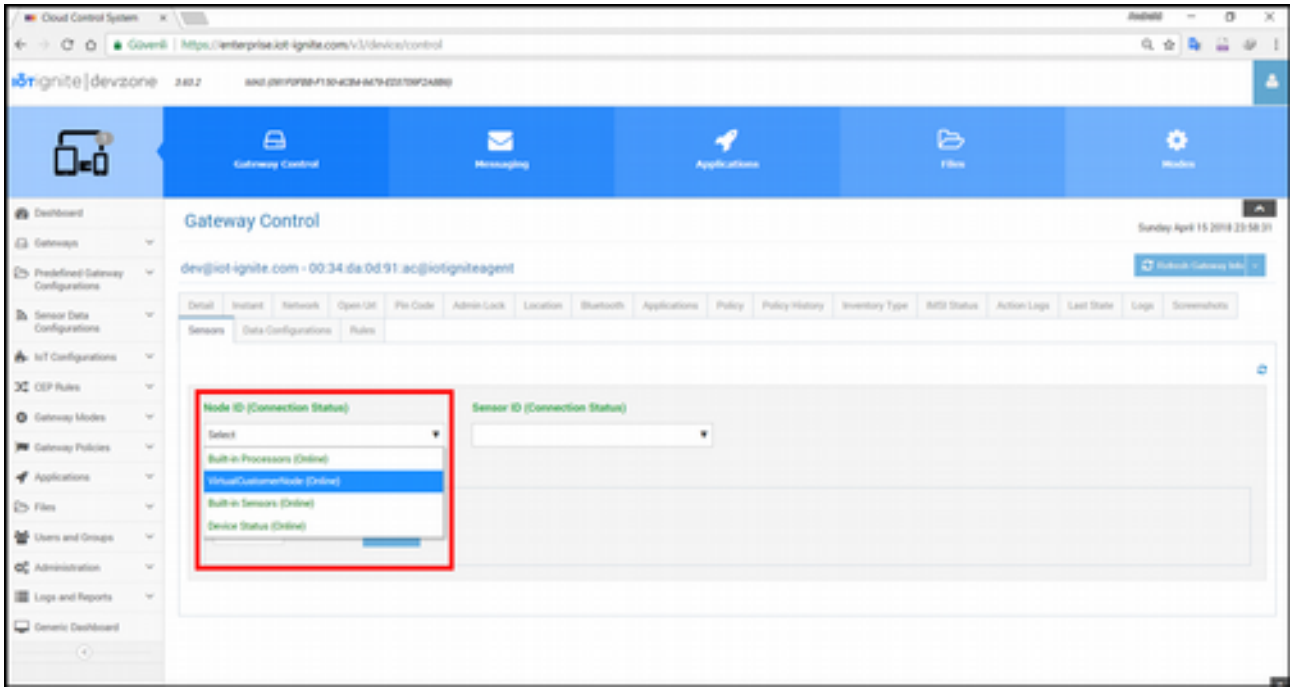
NODES sekmesi altına görüleceği üzere Android uygulaması ile oluşturduğumuz Node ve Thing bileşenlerin listelendiğini görebilirsiniz. Aynı durum EHUB ortamı içinde geçerlidir. EHUB platformuna geçerek **Sensor Data Configurations > Sensor Types** yolunu takip ederek aşağıda verilen sayfayı açınız.



Search yazılı olan arama kutusuna **Basic Customer App** ifadesini giriniz. Aşağıda görüleceği üzere bu **VENDOR** bilgisine sahip olan sensörlerin listelendiği görülür.



Burada sadece oluşturduğumuz Thing yani sensör ve actuator bileşenlerini görebiliriz. Uygulamada oluşturduğumuz Node bileşenini kontrol etmek için EHUB ortamında **Gateways > All Gateways** yolunu takip ederek **Gateway Control** arayüzüne ulaşmamız gerekiyor. Bu arayüzde aşağıda gösterilen **Sensors** sekmesi altında oluşturulan Node bileşeninin eklenip eklenmediğini kontrol edebiliriz.



Devzone’da Yapılması Gereken işlemler

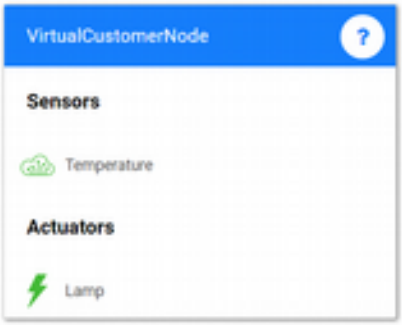
Amacımız müşteri uygulaması ile sanal sensörlerin oluşturulmasını sağlamakla birlikte ek olarak bu bölüme kadar yaptıklarımızın daha iyi anlaşılması için Devzone ortamında bazı işlemler gerçekleştireceğiz. Bu başlık altında sırasıyla şu işlemleri yapacağız:

- Sanal sensörlerin yapılandırılması,
- Cloud Rule ile bulut taraflı kural tanımlaması,
- Son olarak yukarıda yapılan işlemlerin Gateway’e gönderilmesini sağlamaktır.

Şimdi bunları nasıl yapacağımızı tek tek gösterelim.

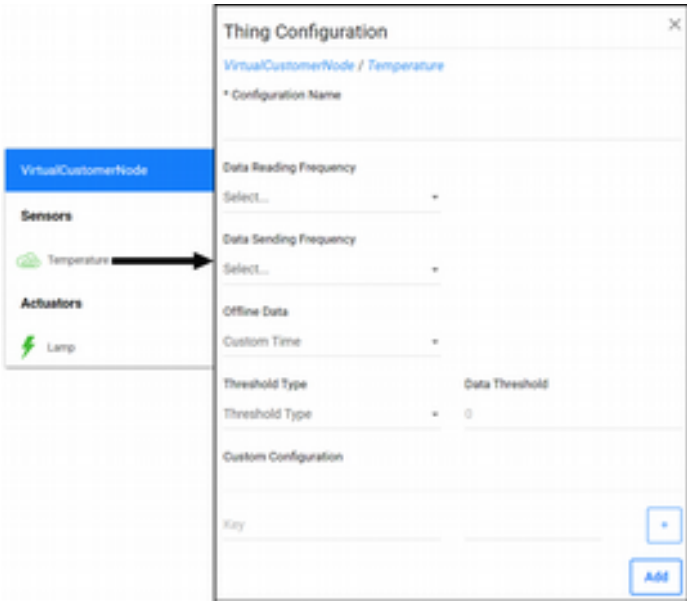
Sanal Sensörlerin Yapılandırılması

Sanal sensörleri Android kodları ile oluşturabiliriz. Ancak amacınız sensör verilerini IoT-Ignite platformuna aktarmak olduğunda sensörleri yapılandırmanız gerekiyor. Bunu yapmadığımız zaman sensörlerden gelen verileri okuyamayız. Burada kod ile oluşturduğumuz sıcaklık sensörü ve lamba actuator’u için yapılandırma ayarlarını yapmayı göstereceğiz. Daha önce Devzone ortamında **NODES** sekmesi altında oluşturduğumuz sensör ve node’ları sizlere göstermiştik. Bu sayfada bizi ilgilendiren alan aşağıdaki gibidir.

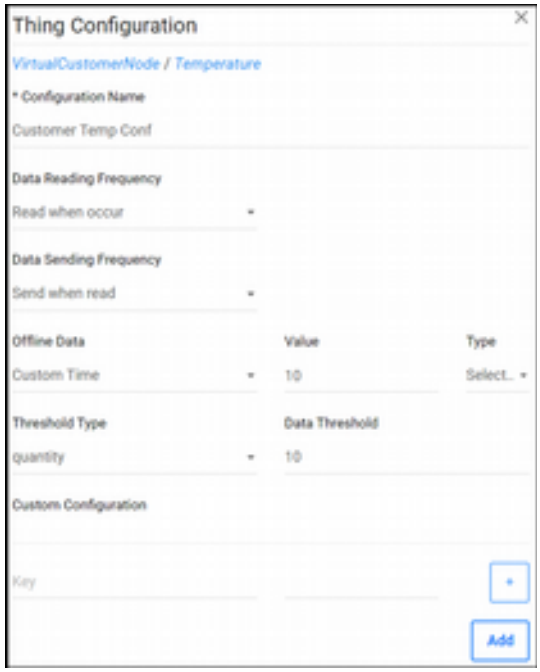


Sıcaklık Sensörünün Yapılandırılması

Yapacağımız ilk yapılandırma sıcaklık sensörü ile ilgili olacaktır. Öncelikle yukarıda listelenen sıcaklık sensörüne tıklayalım. Tıklama işleminden sonra aşağıdaki arayüz bizleri karşılar. Bu arayüzü kullanarak sensörümüzü yapılandırmayı sağlayacağız.



Yukarıda açılan pencerede bulunan seçenekler hakkında ayrıntılı bilgileri "**Genel Sensör ve Actuator'ler İçin Yapılandırma Yapmak ve Yapılandırmayı Gateway'e Push Etmek**" başlığı altında ayrıntılı olarak incelediğimiz için bunlarla zaman kaybetmeden seçeneklere aşağıdaki değerleri verelim.

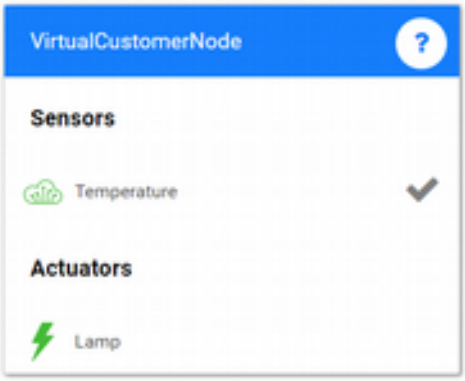


Özetle yapılan işlemleri aşağıdaki gibi sıralayabiliriz:

- **Configuration Name**: Sensör veri yapılandırması için bir isim (Customer Temp Conf)
- **Data Reading Frequency**: Veri okuma sıklığı (Read when occur: Sensörde yeni bir değer üretildiğinde veri okunur.)

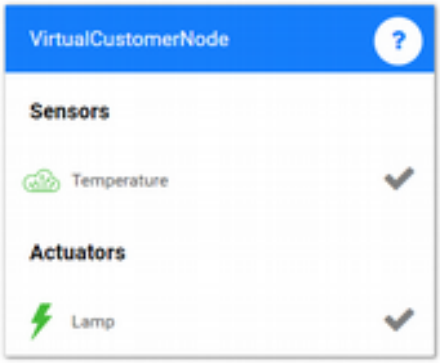
- **Data Sending Frequency**: Veri okuma sıklığı (Send when occur: Sensörde yeni bir değer üretildiğinde veriyi IoT-Ignite Cloud'a gönderir.)
- **Offline Data**: Çevrimdışı durumdayken verilerin ne yapılacağı (Custom time: Özel olarak bir zaman periyodu belirtip, sensör verileri saklanır. Gateway tekrar çevrimiçi olduğunda bu veriler topluca IoT-Ignite Cloud'a gönderilir.)
- **Threshold Type**: Veri eşiği ayarları (Quantity: Miktar. Veri eşiği miktar olarak değişirse, IoT-Ignite Cloud'a veri gönderilir.)

Bu yapılandırmaları eklemek için **Add** butonuna tıklamanız yeterlidir. **Add** (Ekle) butonuna tıklayıp eklediğimizde arayüz kapanacaktır. Sensörlerin listelendiği alanda ise **Temperature** sensörümüzün sağ tarafında bir ✓ (tanımı yapıldı) işareti belirecektir. Bunun anlamı; bu sensör için yeni bir sensör veri yapılandırması yapılmış demektir.



Lamba Actuator'ünün Yapılandırılması

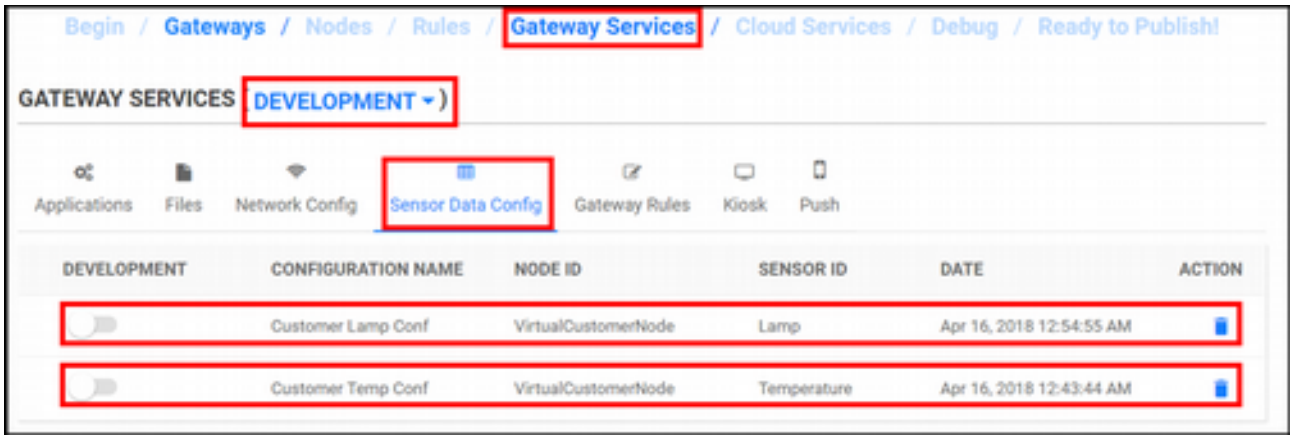
Sensörlerde olduğu gibi actuator bileşenlerinin kontrol edilebilmesi için yapılandırılması gerekiyor. Lamba actuator'ünü sıcaklık sensöründeki yapılandırmalar ile aynı olacak şekilde yapılandırmamız gerekiyor. Yapılandırma işleminden sonra sıcaklık sensöründe olduğu gibi Lamp'ın yanında bir ✓ işareti gösterilir.



Yapılandırmaların Gateway'e Push Edilmesi

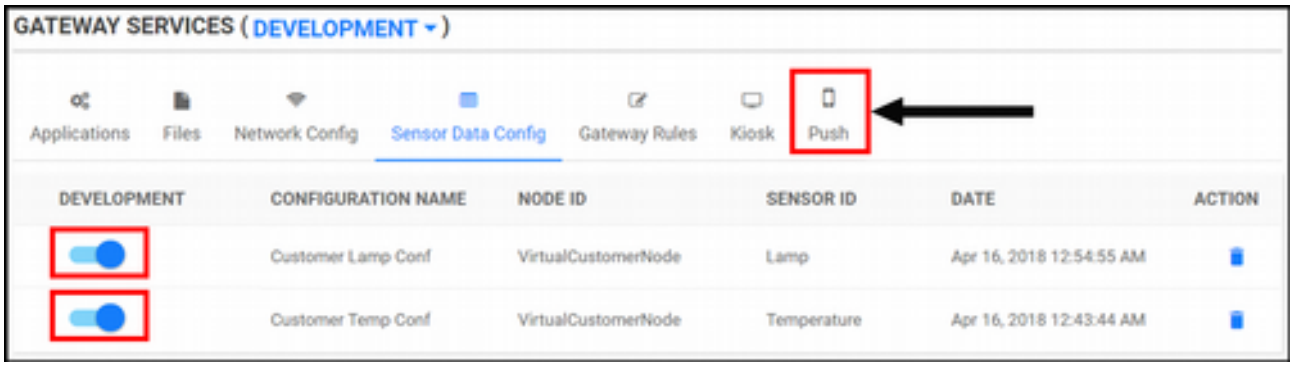
Sensörler için yapılan yapılandırmalar sadece envanterde tutulur. Bunların geçerli olabilmesi için gateway'e push edilmesi, yani gönderilmesi gerekir. Bunu yapmadığınızda, yapılan yapılandırmaların hiçbir işlevi olmayacaktır.

Bunları göndermek için Devzone ortamında bulunan **Gateway Services** sayfasını açalım.

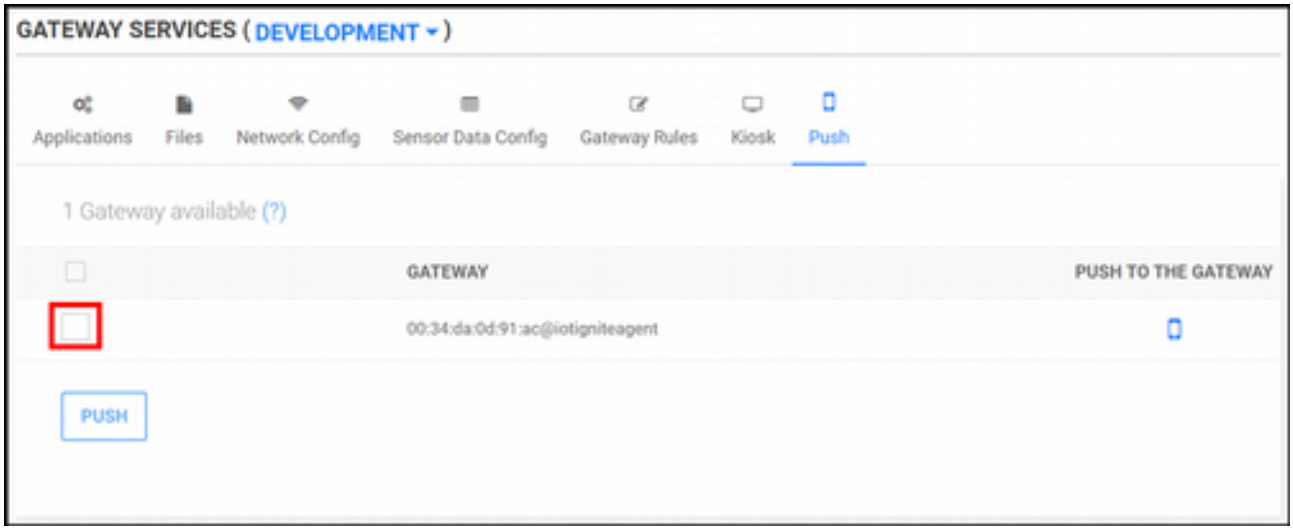


Açılan sayfayı inceleyip şunlara dikkat etmemiz gerekiyor:

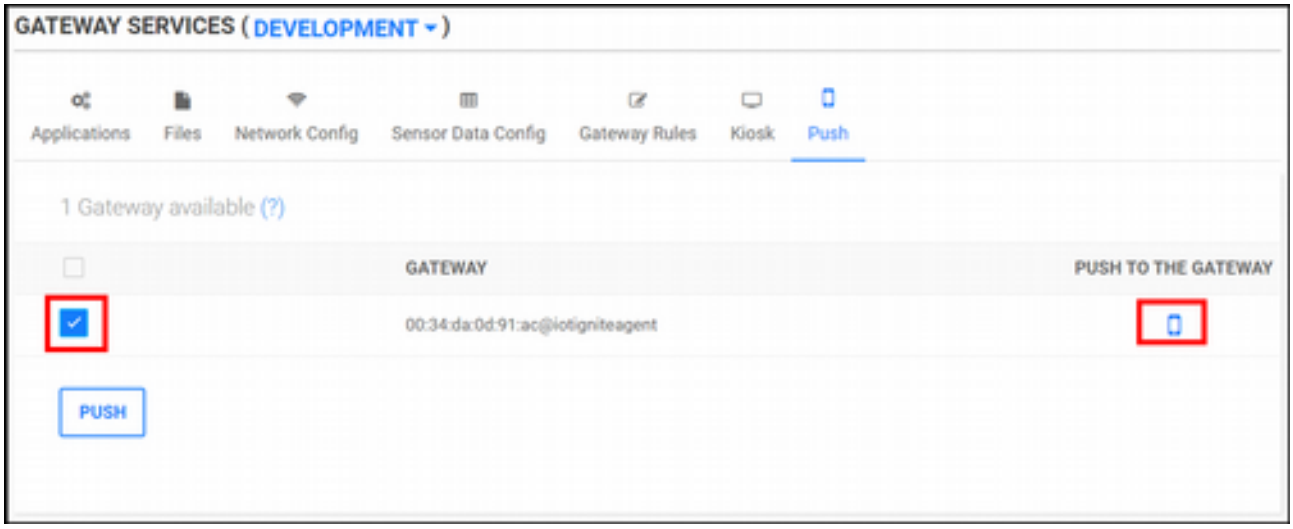
- Öncelikle çalışmak istediğimiz modu seçmemiz gerekiyor. Devzone ortamında varsayılan olarak **DEVELOPMENT** modu kullanılır. Biz bu mod ile uygulama geliştirdiğimiz için bu şekilde bırakıyoruz. (Eğer cihazınızı Demo altında Gateway olarak kayıt etmişseniz cihaz DEMO modunda olacaktır, Devzone ortamında kayıt etmişseniz DEVELOPMENT modunda olacaktır. IoTIgniteAgent uygulamasını açarak cihazınızın hangi modda olduğunu öğrenebilirsiniz.)
- Listede, yukarıda oluşturduğumuz **Temperature** ve **Lamp** yapılandırması yer almaktadır. Switch butonuna tıklayıp her iki yapılandırmayı aktif ediniz. İsterseniz bu sayfada yapılandırmayı çöp kutusu ikonuna tıklayarak silebilirsiniz.



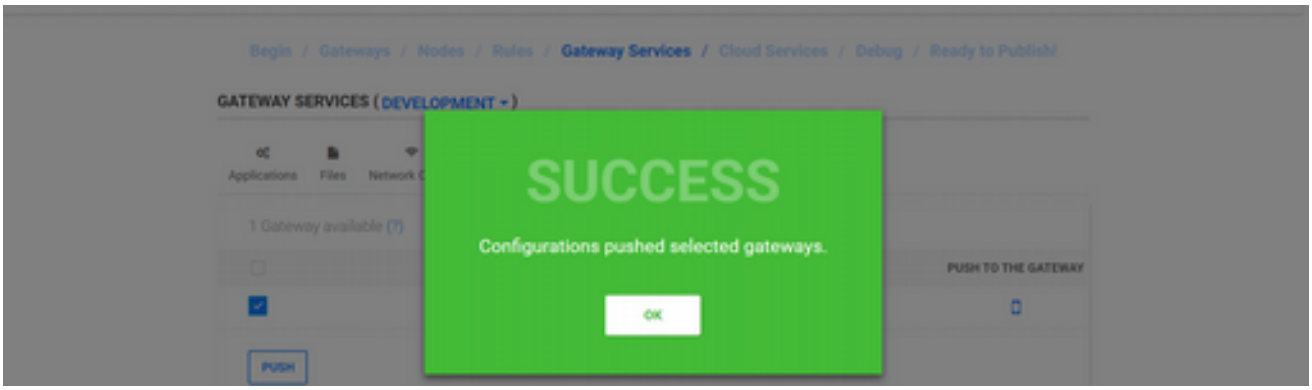
DEVELOPMENT modunda **Temperature** ve **Lamp** için tanımladığımız yapılandırmayı aktif ettik. Gateway'e yeni **DEVELOPMENT** modunu göndermek için yukarıda bulunan **Push** sekmesine tıklayınız.



Listeden Gateway'i seçip aktif edelim.



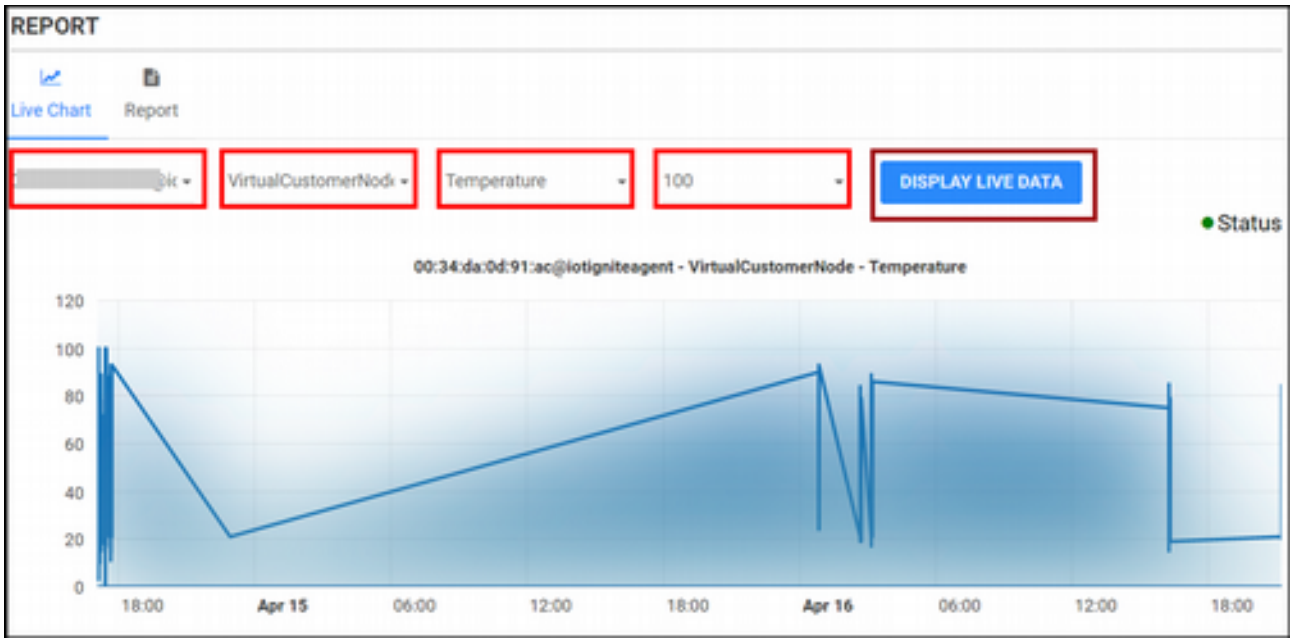
Ardından sağ tarafta yer alan Gateway ikonuna tıklayıp push edelim.



İşlem başarılı olduğunda yukarıdaki gibi bir bilgilendirme mesajı alacaksınız. Gateway, yeni modu aldığında da ekranında "*Updating device policies*" mesajı çıkacaktır. Bu mesaj çıktığında, artık mod güncellenmiş demektir.

Temperature sensör verilerinin yeni yapılandırma ile nasıl davranış gösterdiğini denemek için Devzone'da üst menüden **REPORT** (Rapor)'a tıklayın.

Live Chart sekmesindeyken Gateway'i, Node'u (VirtualCustomerNode), sensörü (Temperature) seçip, veri sayısını **100** yapın. **DISPLAY LIVE DATA** butonuna tıklayın.

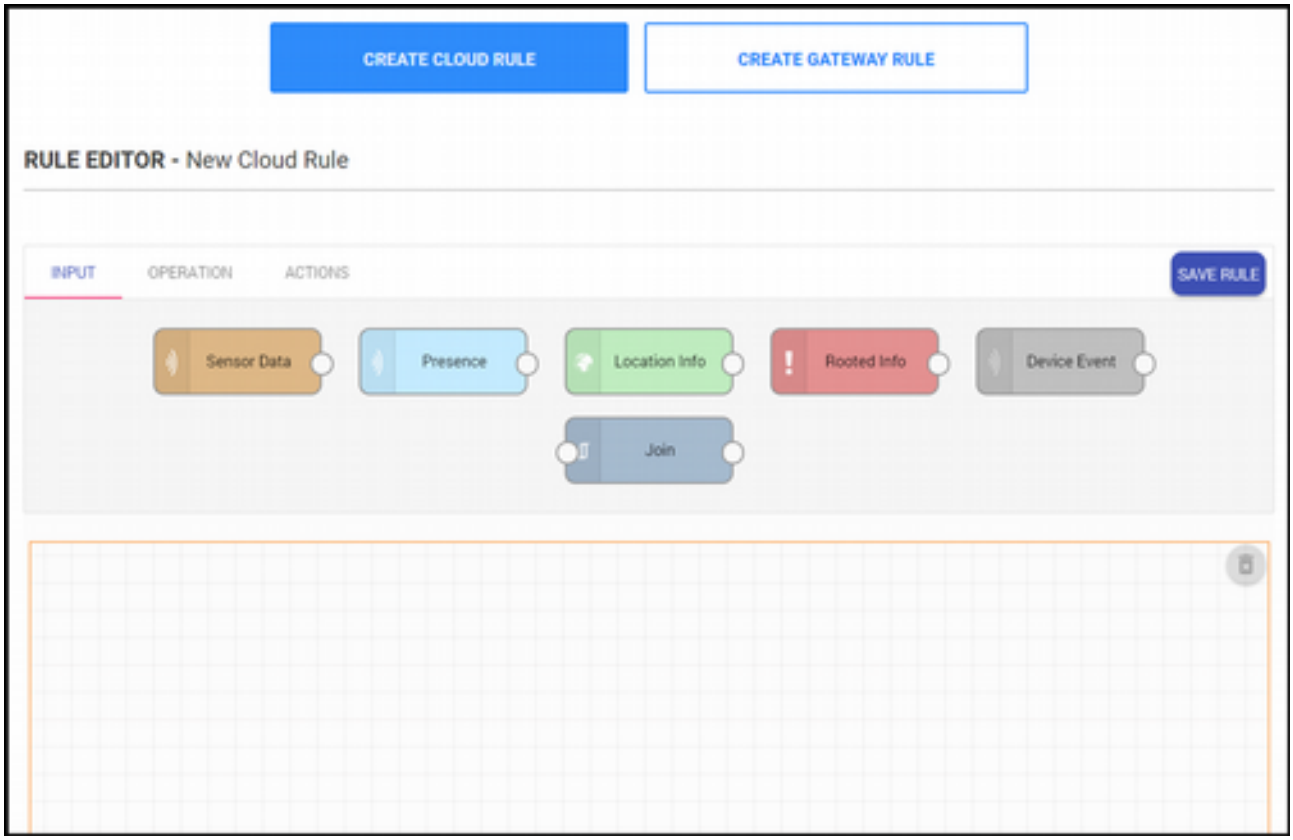


Açılan grafikte, son 100 veri gösterilmektedir. İsterseniz veri sayısını, listeden 10-1000 arası değiştirebilirsiniz.

Cloud Rule ile Kural Oluşturulması

Cloud Rule ile bulut tarafında IoT uygulamaları için kurallar oluşturabiliriz. Kurallar ile IoT ekosisteminde bulunan sensör ve actuator bileşenlerini rahatlıkla kontrol edebiliriz. Bunun için Devzone ortamındayken **Rules** sekmesini tıklayalım.

Açılan sayfada **Cloud Rule** veya **Gateway Rule** olmak üzere iki çeşit kural oluşturabiliriz. Amacımız **Cloud Rule** yani bulut tarafında bulunan kuralları oluşturmaktır. Bundan dolayı yukarıda bulunan **CREATE CLOUD RULE** seçeneğine tıklayalım.



Yukarıda açılan sayfayı kullanarak bulut tarafında iki adet kural oluşturacağız. Kurallarımız aşağıdaki gibidir:

- İlk kuralın amacı sıcaklık sensöründen gelen veri, 50 °C'nin üstüne çıktığında Müşteri uygulamasında bulunan lambanın yanmasını sağlamaktır.
- ikinci kuralın amacı ise sıcaklık sensöründen gelen veri, 50 °C'nin altına düştüğünde Müşteri uygulamasında bulunan lambanın sönmesini sağlamaktır.

Şimdi bu işlemleri yapacak kurallarımızı oluşturalım.

Sıcaklık 50 °C'nin Üstüne Çıktığında Lambanın Yanması

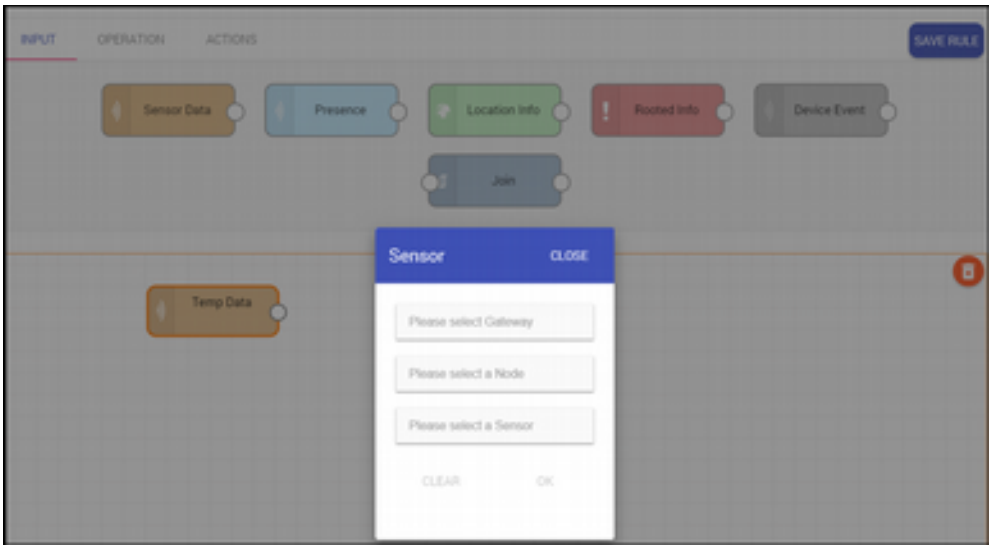
CREATE CLOUD RULE seçeneğine tıkladıktan sonra açılan pencereden kuralımızı oluşturmak için aşağıdaki adımları takip etmemiz gerekiyor.

1) **Rule Editor** çalışma alanında yer alan **INPUT** sekmesindeki **Sensor Data** diyagramını kural oluşturma alanına sürükleyiniz.

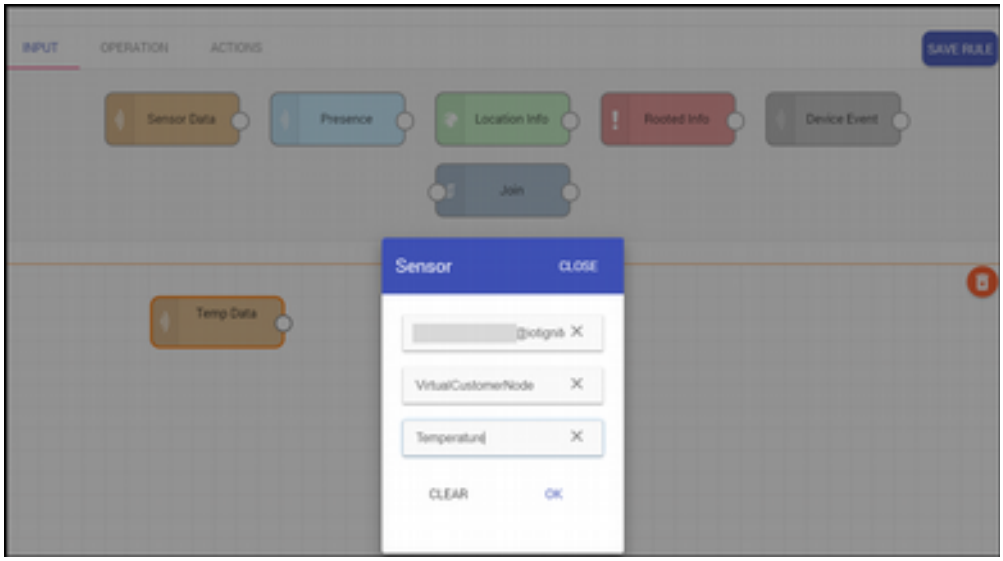
2) Sürükleme işleminden sonra aşağıdaki pencere açılır. Buraya diyagram ismi olarak **Temp Data** ismini girip **OK** seçeneğine tıklayınız.

OK seçeneğine tıkladıktan sonra **Sensor Data** diyagramı aşağıdaki gibi eklenir.

3) **Temp Data** diyagramına çift tıklayarak aşağıdaki pencereyi açınız.

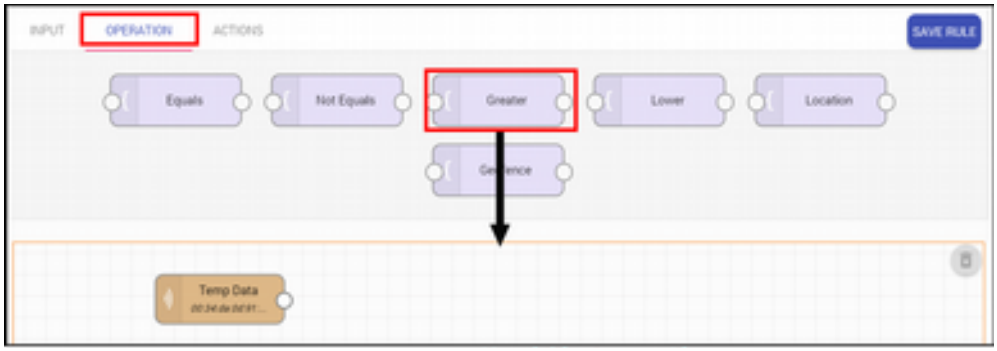


Bu pencerede verisi alınacak sensörün seçilmesi sağlanır. Tabii sensörü seçerken öncelikle sensörün bağlı olduğu Gateway ve Node bilgilerini doğru olarak set etmek gerekiyor. Amacımız akıllı telefon ya da tablette, **VirtualCustomerNode** node'na bağlı olan **Temperature** sensöründen gelen veriyi okumaktır. Yapılandırmaları aşağıdaki gibi yapalım.

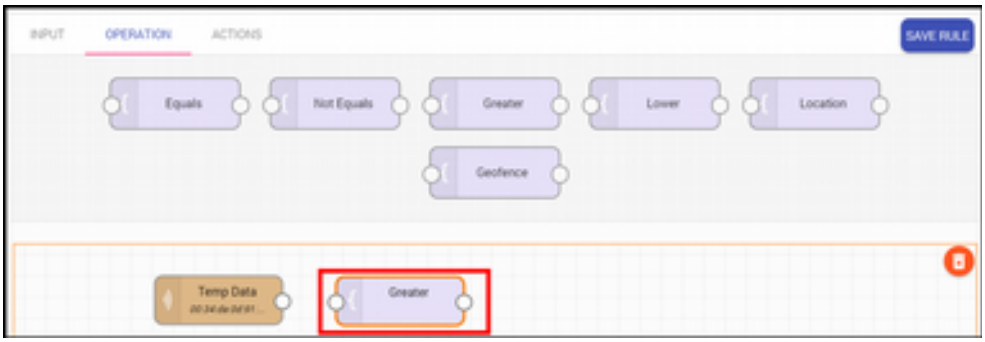


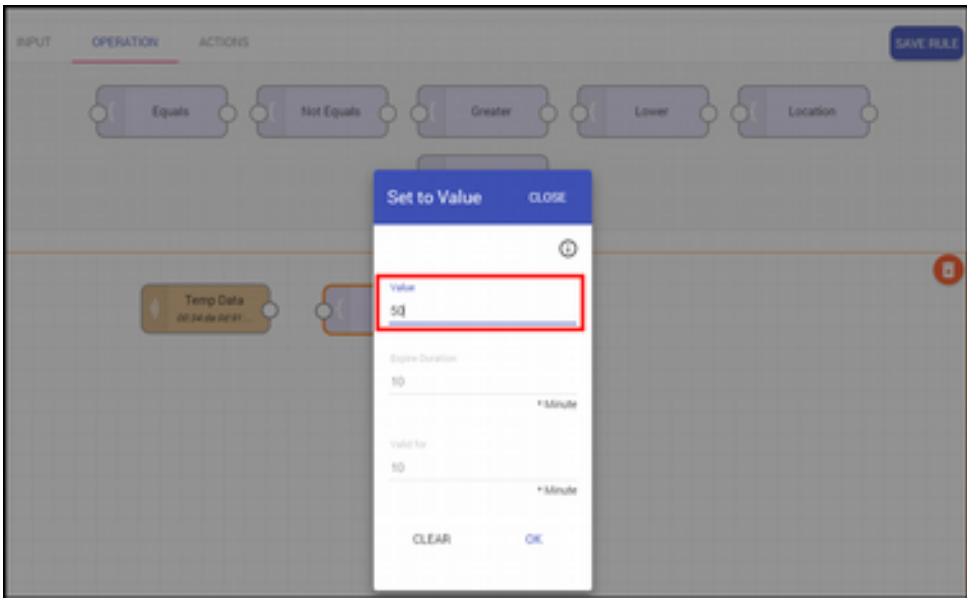
İşlemleri yukarıdaki gibi tamamladıktan sonra **OK** seçeneğine tıklayalım.

4) Yukarıdaki işlemi yaptıktan sonra şimdi **Rule Editor** çalışma alanında yer alan **OPERATION** sekmesindeki **Greater** diyagramını kural oluşturma alanına sürükleyiniz.



Sürükleme işleminden sonra diyagram aşağıdaki gibi çalışma alanına eklenir. Burada eklenmesi sırasında herhangi bir pencere açılmaz.





5) **Greater** ile sıcaklık sensöründen gelen verinin belirli bir değerden büyük olup olmama durumu kontrol edilir. Burada sensörden gelen verinin 50'den büyük olma durumu ele alınacaktır. Çalışma alanında bulunan **Greater** diyagramına çift tıklayalım.

   Açılan pencerede kırmızı alan içinde bulunan Value alanına 50 değerini girip **OK** butonuna tıklayalım.

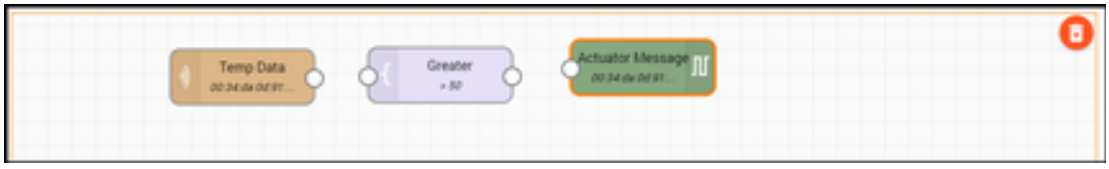
6) **Greater** diyagramı için yapılandırmayı sağladıktan sonra şimdi **Rule Editor** alanında bulunan **ACTIONS** sekmesine tıklayıp **Actuator Message** diyagramını çalışma alanına sürükleyelim.

Sürükleme işleminden sonra diyagram aşağıdaki gibi çalışma alanına eklenir. Burada eklenmesi sırasında herhangi bir pencere açılmaz.

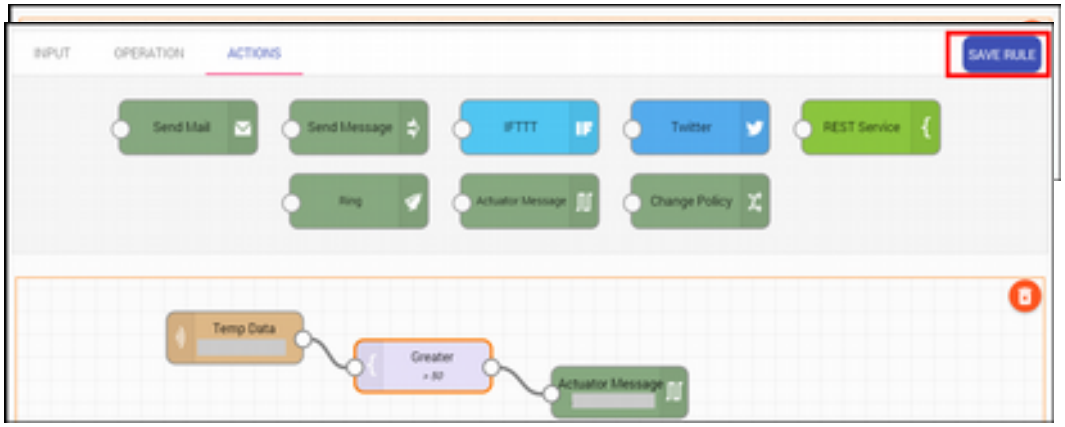
7) Burada sensörden gelen veri 50'den büyük olduğunda Müşteri uygulamasında bulunan lambanın yanması sağlanacaktır. Bu işlemi Actuator Message ile yapabiliriz. Bu yapabilmek için çalışma alanında bulunan **Actuator Message** diyagramına çift tıklayalım.

Açılan pencerede Sensor Data diyagramında olduğu gibi sırasıyla **Gateway**, **Node** ve **Sensör** seçimi yapılır. En altta kırmızı alanda bulunan **Message** alanına **1** değerini girdik. Bu bilgi Android Müşteri uygulamasında alınarak lambanın yanması sağlanır. **OK** butonuna tıklayarak işlemi uygulayabiliriz.

8) Yukarıdaki adımdan sonra çalışma alanı aşağıdaki gibi olur.



9) Bu haliyle bıraktığımızda yapılan işlemlerin hiçbir anlamı olmaz. Kuralı tam olarak oluşturmak için çalışma alanında bulunan diyagramların aşağıdaki gibi birbirine bağlanması gerekir.



10) Yukarıda bağlantı işlemi yapılan kuralın çalışma mantığında Temp Data ile sıcaklık sensöründen gelen veri alınır. Greater ile gelen verinin 50'den büyük olup olmadığı kontrol edilir. Eğer sıcaklık 50 dereceden yüksek ise Android uygulamasında bulunan lambanın yanması sağlanır.

Oluşturduğumuz bu kuralı kayıt etmek için **SAVE RULE** butonuna tıklayınız. Bu butona tıkladıktan sonra aşağıdaki pencere bizlere gösterilir.

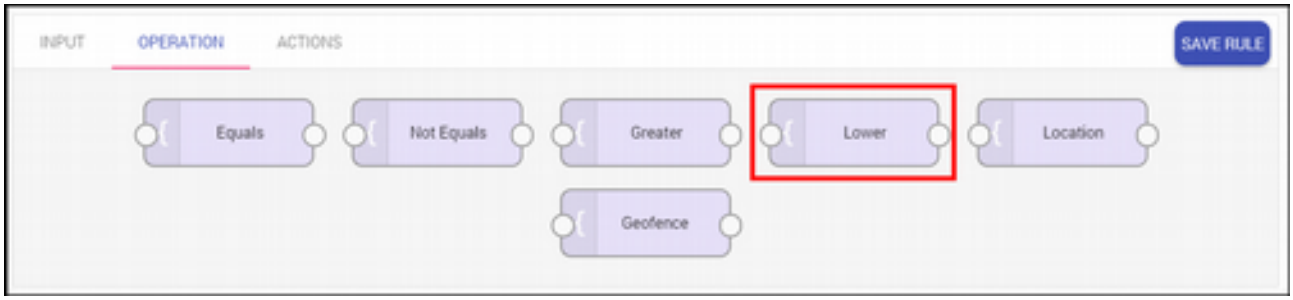
Açılan pencerede kural ismini ve açıklamasını girerek **OK** butonuna tıklayınız. Bu işlemden sonra kuralımız IoT-Ignite platformuna kayıt edilir.

Kuralı kayıt ettikten sonra Gateway cihaza herhangi bir şey gönderilmez. Çünkü yapılan işlem Ignite ortamında gerçekleşmektedir.

Sıcaklık 50 °C'nin Altına Düştüğünde Lamba'nın Sönmesi

Sıcaklık sensöründen gelen veri 50 derecenin üstünde ise Müşteri uygulamasında bulunan lambanın yanması sağlanır. Bir de 50 derecenin altında düştüğünde lambanın söndürülmesini sağlayan Cloud kuralını tanımlayalım. Bir önceki kuralımızı oluştururken yapılacak işlemleri ayrıntılı olarak sizlere sunduk. Burada sadece farklı olan yerleri sizlere aktaracağız. Öncelikle yukarıdaki kural için yaptığımız ilk üç adımı bu kural içinde birebir uygulayınız.

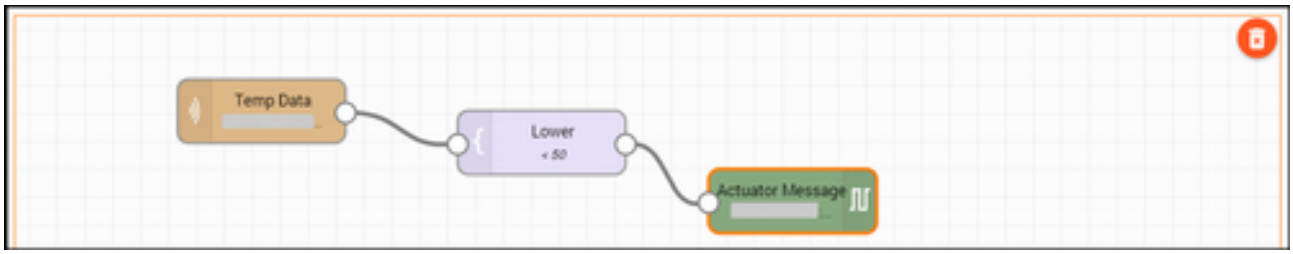
4. adımda **OPERATION** sekmesinde bulunan **Greater** yerine aşağıda verilen **Lower** diyagramını ekleyelim.



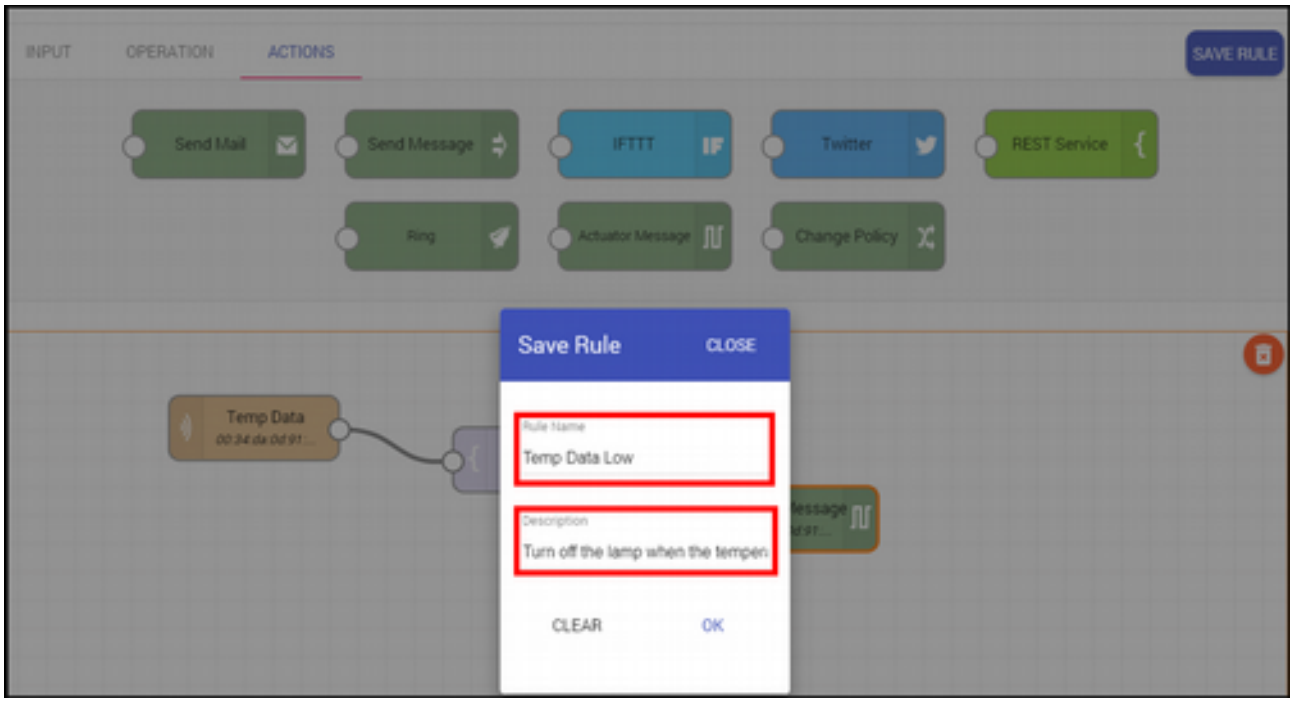
Greater diyagramı verilen parametreden büyük olan değerleri baz alırken, **Lower** ise belirtilen parametreden düşük olan değerleri baz alır. **Lower** diyagramına **Greater** için yaptığımız yapılandırmanın birebir aynısını yapalım. Burada amaç sıcaklık değeri 50 derecenin altında olduğu durumları tespit etmektir.

Son olarak **ACTIONS** sekmesinde bulunan **Actuator Message** diyagramını ekleyip Message alanına 0 değerini girelim. Burada **Temp Data** diyagramından gelen sensör verisi **Lower** ile 50 derecenin altında olup olmadığı kontrol edilir. Eğer sensör verisi 50 derecenin altında ise lambanın sönmesi sağlanır.

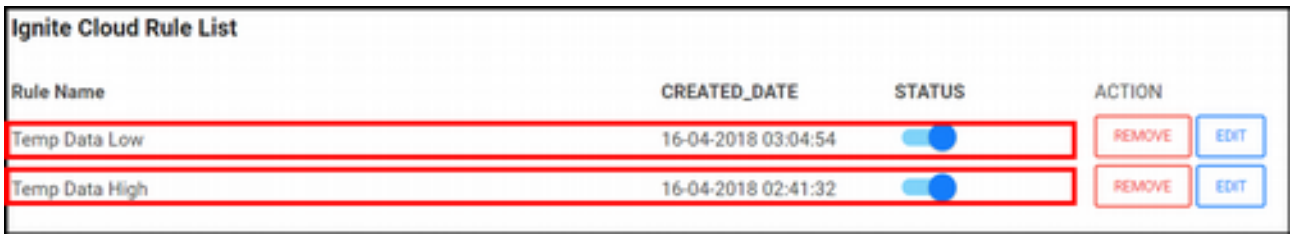
Diyagramları yukarıdaki gibi tanımladıktan sonra aşağıdaki gibi birbirine bağlayalım.



SAVE RULE seçeneğine tıklayarak kuralımızı aşağıdaki gibi kaydedelim.



Kayıt işleminden sonra tüm kurallarımız aşağıdaki gibi listelenir.



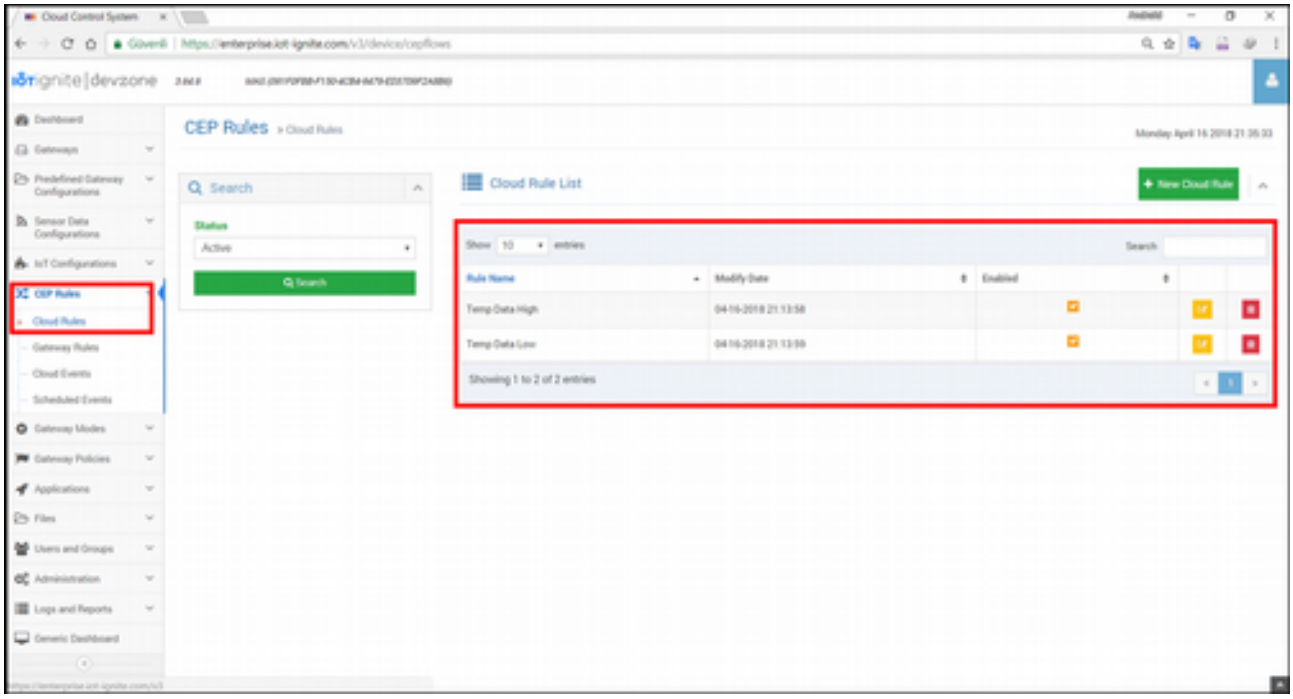
Şimdi bu kuralların nasıl çalıştığını test edebiliriz.

Yapılandırmaların Test Edilmesi

Bu başlık yukarıda yapılan yapılandırma işlemlerinin test edilmesini sağlayacağız. Eğer yukarıda yapılan işlemler doğru ise Müşteri uygulamamız aşağıdaki gibi çalışacaktır.

EHUB Ortamında Tweet Atma Kuralı Tanımlamak

Devzone ortamında kural tanımlayabildiğimiz gibi aynı işlemi EHUB ortamında da yapabiliriz. EHUB ortamında kural oluşturmak için **CEP Rules** > **Cloud Rules** yolunu takip ederek aşağıdaki sayfayı açalım.



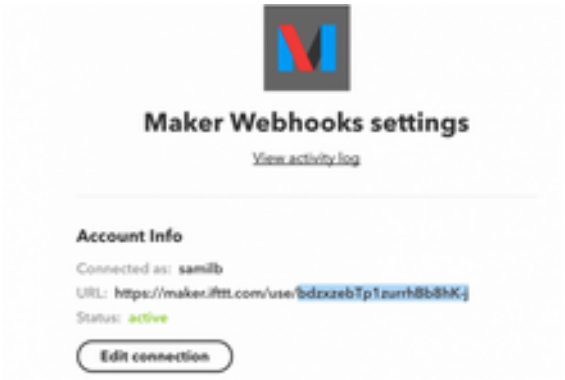
Yukarıda Devzone ortamında oluşturduğumuz iki kuralın EHUB ortamında listelendiği görülmektedir. Aynı şekilde EHUB ortamında oluşturulan bir kurala Devzone ortamında da erişim sağlayabiliriz.

Bu başlık altında sıcaklık sensöründen gelen veri 60 derecenin üstüne çıktığında tweet atmayı sağlayacağız. Tweet atma işlemi harici bir işlem olduğundan dolayı öncelikle aşağıda verilen işlemlerin yapılması gerekiyor. Burada verilen başlıklar hakkında ayrıntılı bilgileri kitabın 3. bölümünde ele aldığımız için burada kısaca değineceğiz.

IFTTT Bağlantısı için API Key Almak

IoT-Ignite platformunda bu işlemi yapabilmek için **IoT Configurations > External Services** yolunu takip ediniz ve aşağıdaki adımları gerçekleştiriniz:

- IFTTT hesabınızı giriniz, **Maker Channel API**'sı oluşturup anahtarı kopyalayın.
- **API** anahtarı bilgilerini yapıştırın.



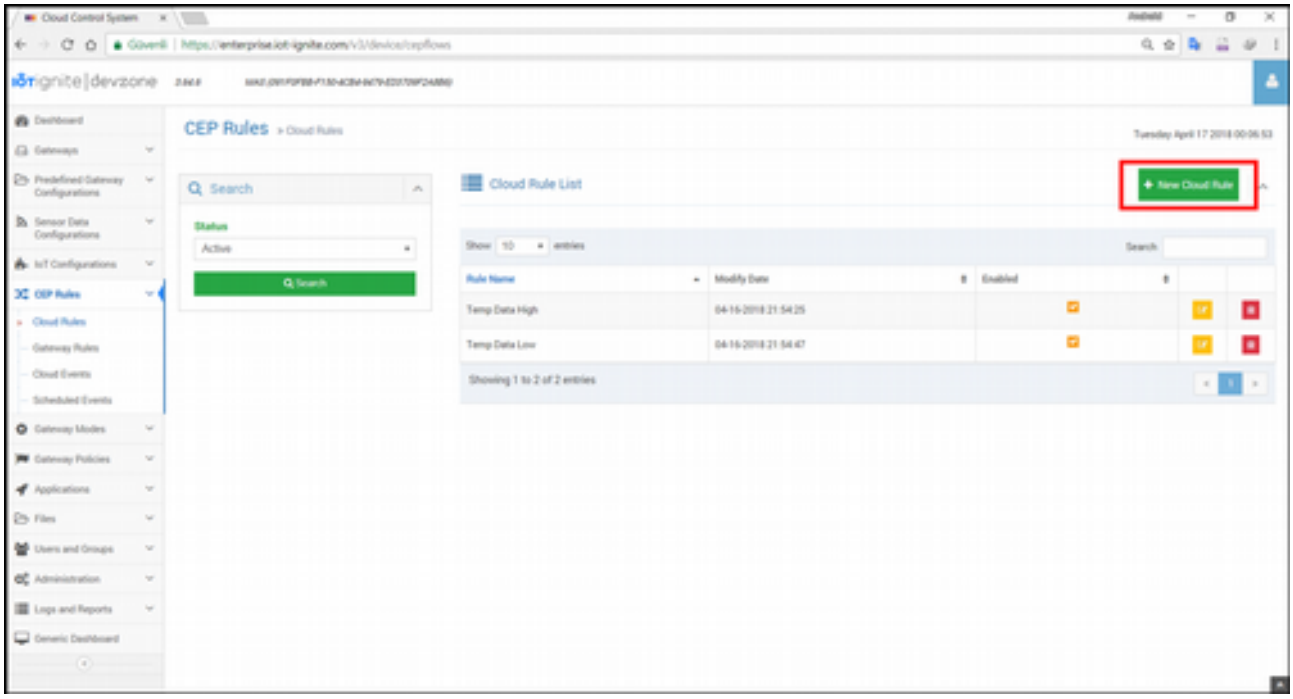
Twitter Bağlantısı Sağlamak

Twitter bağlantısını IoT-Ignite'den sağlamak için **IoT Configurations > External Services** yolunu takip ediniz ve **Twitter** tabına tıklayınız. Twitter sekmesi altında Twitter hesabınıza giriş yapmak için **connect** butonuna tıklayın ve IoT-Ignite uygulamasına izin verin.

Sensör Verilerini Tweetlemek

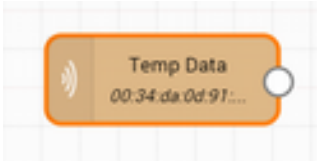
Sensör verilerini Tweetlemek için aşağıda verilen adımları takip ediniz.

- **CEP Rules > Cloud Rules** menüsüne gidiniz.

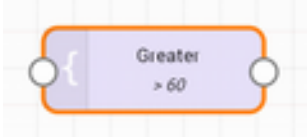


- **New Cloud Rule** butonuna tıklayınız.

- **INPUT** sekmesinde bulunan Sensor Data diyagramını çalışma alanına sürükleyip daha önce yaptığımız gibi sıcaklık sensörü için yapılandırmayı sağlayınız.

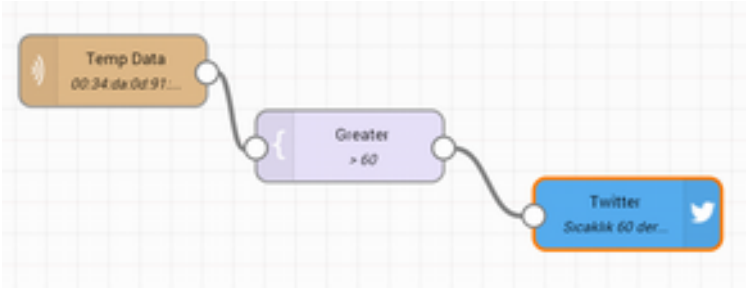


- **OPERATION** sekmesinde bulunan **Greater** diyagramını ekleyip 60 değerini veriniz.



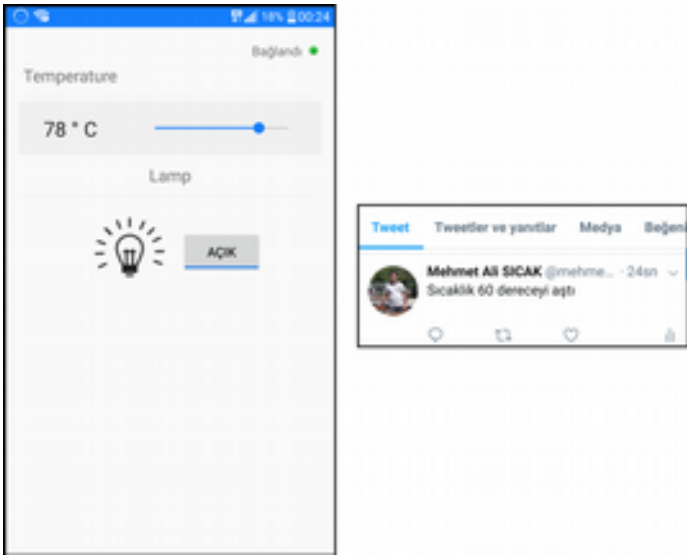
- **ACTIONS** sekmesi altında bulunan Twitter diyagramını çalışma alanına sürükleyip aşağıdaki yapılandırmayı sağlayınız.

- Son olarak eklediğimiz diyagramları aşağıdaki gibi bağlayıp kuralı kayıt ediniz.



Kuralı Test Etmek

Kuralı kayıt ettikten sonra uygulamamızı test ettiğimiz zaman aşağıdaki gibi mesaj atıldığını görebiliriz.



BÖLÜM 11

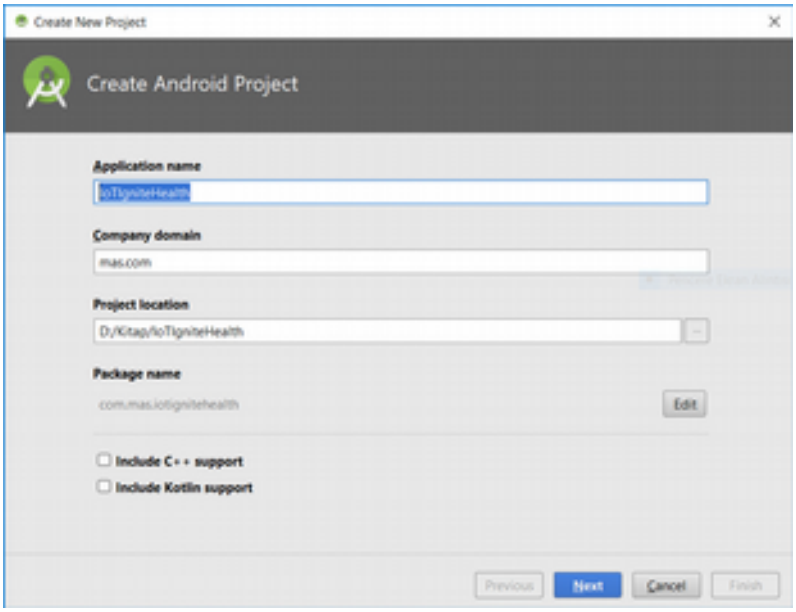
Uçtan Uca Uygulama

Hasta Takip Sistemi

Bu uygulamada daha önce yapılmayan bir işlemi gerçekleştireceğiz. Amacımız; Smart Watch donanımında bulunan nabız sensörünün verilerini IoT-Ignite ortamına aktarmaktır. Bu veriyi aktardıktan sonra tanımlayacağımız bir Cloud Rule ile hastadan sorumlu olan doktorun akıllı telefonuna bir uyarı mesajı göndermeyi sağlayacağız.

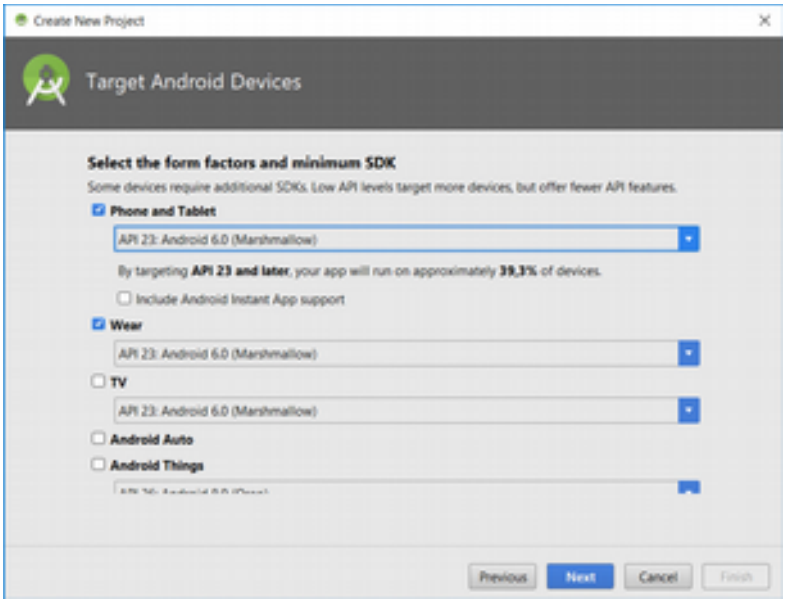
Yeni Proje Oluşturmak

Android Studio geliştirme ortamını açınız ve yeni bir proje oluşturunuz.

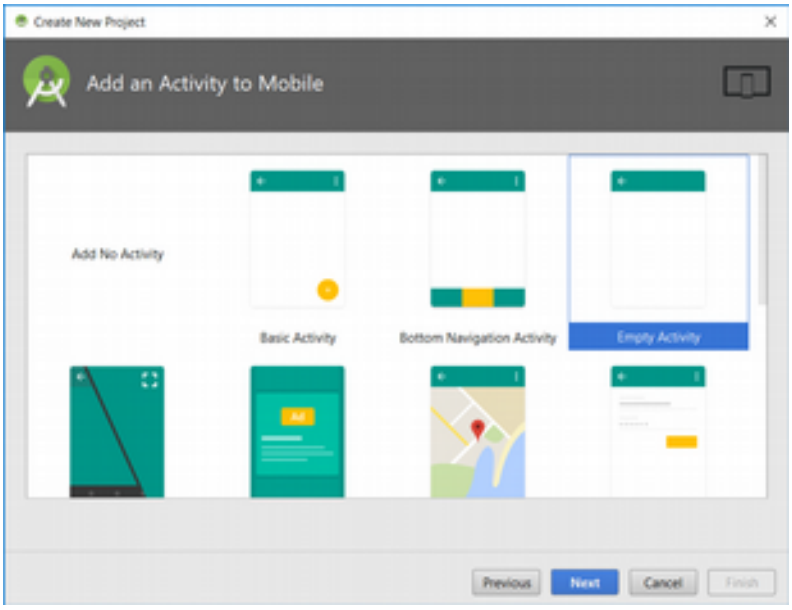


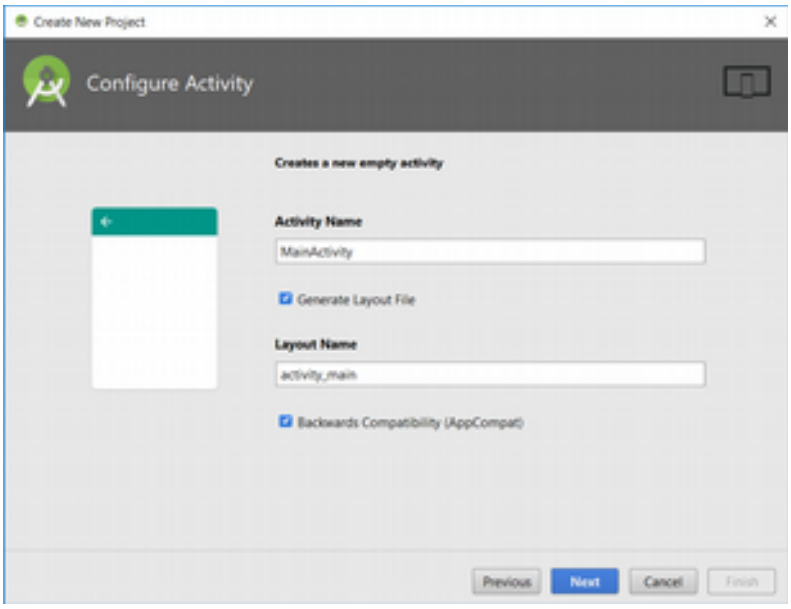
Projeye **IoTIgniteHealth** ismini verdikten sonra **Next** butonuna tıklayalım.

Uygulamanın çalışacağı hedef aygıtlar aşağıda açılan yeni pencerede belirlenir. Amacımız; uygulamayı akıllı telefonlarda ve akıllı saatlerde çalışacak şekilde kullanmaktır. Akıllı telefon ve akıllı saat için minimum SDK veya API değerini **API 23** olarak belirledik.

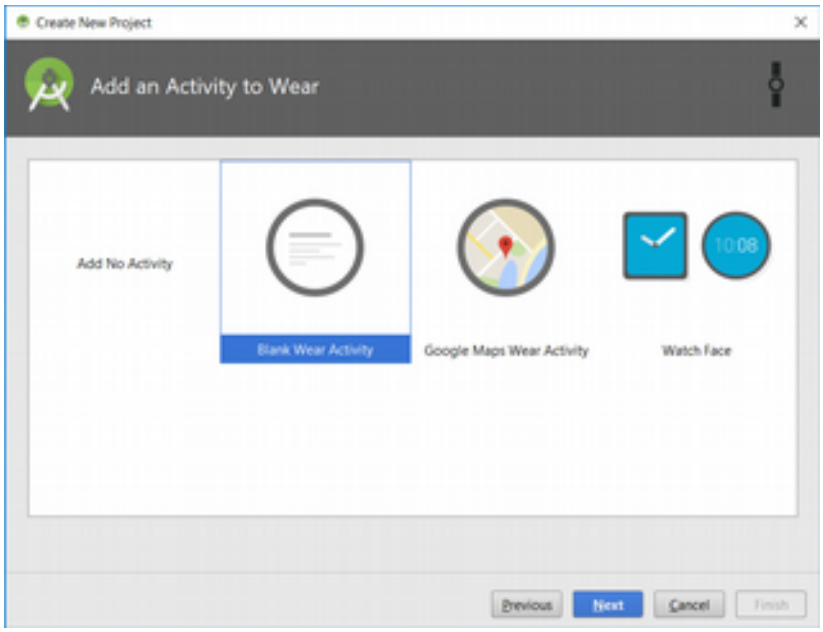


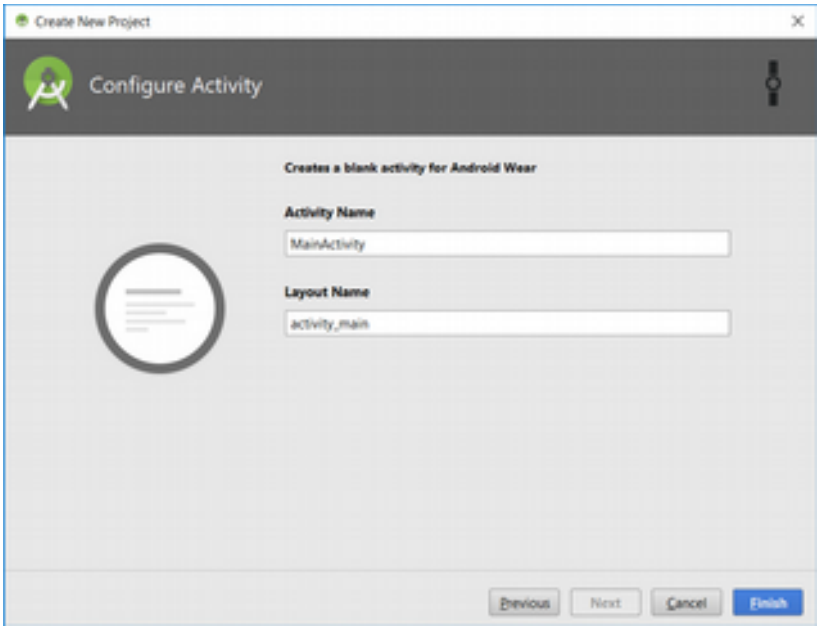
Hedef aygıtları belirledikten sonra **Next** butonuna tıklayıp devam edelim. Açılan yeni pencerede akıllı telefon ve akıllı saat için sırasıyla yeni bir **Activity** eklememiz gerekiyor.





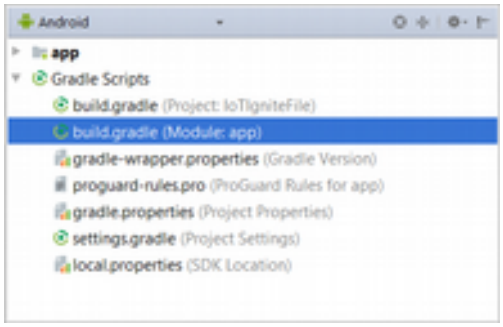
Akıllı telefon için yukarıdaki adımları geçtikten sonra akıllı saat için aşağıdakileri yapmamız gerekiyor.





IoT-Ignite için oluşturduğumuz bu yeni projede IoT-Ignite ile uygulama geliştirmek için bazı ayarlamalar yapmamız gerekiyor. Bunları yapmadan IoT-Ignite için geliştirilen kütüphaneleri kullanamayız.

Öncelikle aşağıda verilen dosyayı açalım. Dosyayı kolay bir şekilde bulabilmek için proje görünümünü Android olarak değiştirmenizi tavsiye ediyoruz.



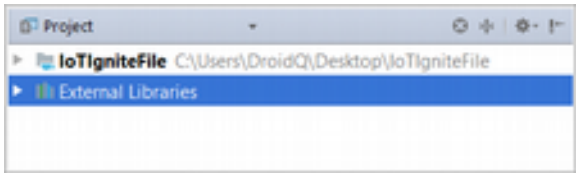
Dosya açıldıktan sonra aşağıda koyu renkli gösterilen satırları dosyanıza ekleyiniz.

```
20    repositories {
21        mavenCentral()
22        maven {
23            url "http://repo.maven.apache.org/maven2"
24        }
25        maven {
26            url "https://repo.iot-ignite.com/content/repositories/releases"
27        }
28    }
29    dependencies {
          ...
36        compile 'com.ardic.android:IoTIgnite:0.8.2'
37        compile 'com.google.code.gson:gson:2.7'
38    }
```

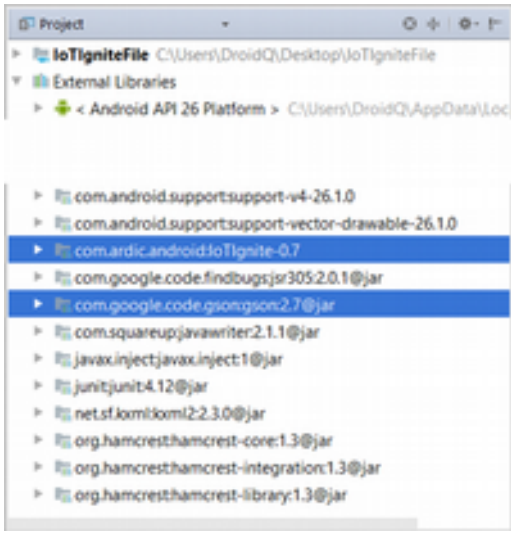
Yukarıda verilen **repositories** blogu olduğu gibi eklenirken, **dependencies** içerisine ise sadece koyu renkli iki satır eklenmelidir. Dependencies içine eklenen ilk satır, Google tarafından geliştirilen **Gson** kütüphanesini eklemek için kullanılır. Bunu ekleme sebebimiz IoT-Ignite platformunda veri transferi için **JSON** yapısının kullanılmasıdır. Gson kütüphanesi, JSON verilerini ayrıştırmak için kullanılır. Bundan dolayı bunu mutlaka eklemeniz gerekiyor. Diğer satır ise **ARDIC** tarafından geliştirilen IoT-Ignite kütüphanesini projeye eklemek için kullanılmaktadır.

Bunları ekledikten hemen sonra **Build > Rebuild** yolunu takip ederek ilgili kütüphaneleri projeye ekleyebiliriz.

Kütüphanelerin projeye eklenip eklenmediğini kontrol etmek adına öncelikle Android görünümünden **Project** görünümüne geçiş yapalım.

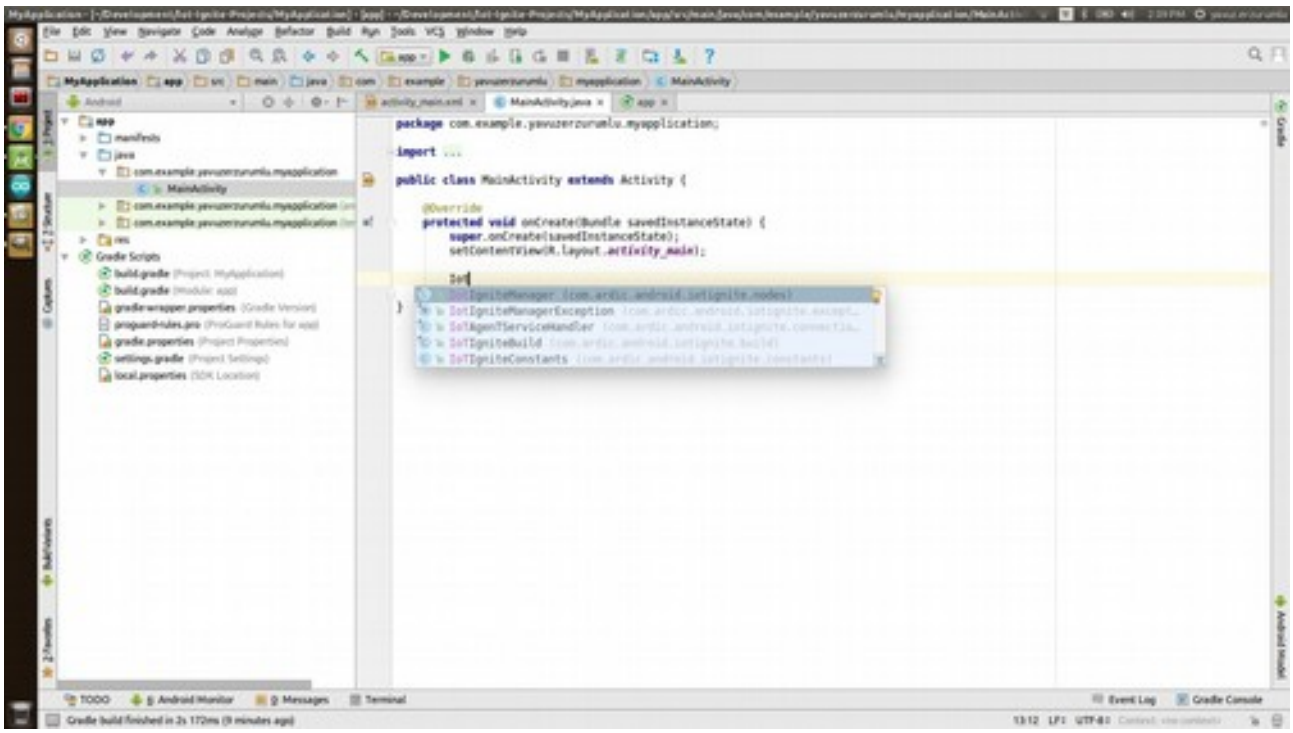


Project görünümü altında yer alan **External Libraries** açılır listesini genişlettiğimiz zaman, harici olarak eklenen kütüphaneleri görebiliriz. Aşağıda verilen listede eğer işaretli kısımları görüyorsanız, kütüphanelerin projeye eklendiğinden emin olabilirsiniz.



Kütüphaneleri ekledikten sonra ilk test aşamasını gerçekleştirebiliriz. Buradaki amaç editör alanında IoTIgnite ile eklenen kütüphanelere erişim sağlayıp sağlamadığımızı kontrol edebilmektir.

MainActivity içinde bulunan **onCreate()** metodunda **IoTIgnite** yazdığınızda aşağıda verilen sonuç ile karşılaşırsanız, bu durumda işlemlerin başarıyla gerçekleştiğini söyleyebiliriz.



Akıllı Saat Uygulaması

Akıllı saat için uygulama kodlarımız aşağıdaki gibidir.

Arayüz Kodu

Uygulama arayüzü aşağıdaki gibi olacaktır.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.wearable.view.BoxInsetLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/dark_grey"
    android:padding="0dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_gravity="center_vertical"
        app:layout_box="all">

        <!--Nabız verisinin yazılacağı kontrol -->
        <TextView
            android:id="@+id/heart"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Kalp Atış Hızı:" />


        <!--Hastanın kalp krizi geçirdğini bildiren uyarıcı -->
        <Button
            android:id="@+id/btnSend"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:onClick="sendRate"
            android:text="Sanal Sensör"
            android:textAllCaps="false" />

    </LinearLayout>
</android.support.wearable.view.BoxInsetLayout>
```

Uygulamanın arayüzü aşağıdaki gibi görünecektir.

Java Kodu

Amacımız akıllı saat ile nabız ölçümü yapmak ve bunu akıllı telefona iletmektir. Bunun için ihtiyacımız olan java kodları da aşağıdaki gibidir.

DataSendActivity.java

```
1    package com.mas.iotignitehealth;
2
3    import android.hardware.Sensor;
4    import android.hardware.SensorEvent;
5    import android.hardware.SensorEventListener;
6    import android.hardware.SensorManager;
7    import android.os.Bundle;
8    import android.support.annotation.NonNull;
9    import android.support.annotation.Nullable;
10   import android.support.wearable.activity.WearableActivity;
11   import android.view.View;
12   import android.view.WindowManager;
13   import android.widget.TextView;
14
15   import com.google.android.gms.common.ConnectionResult;
16   import com.google.android.gms.common.api.GoogleApiClient;
17   import com.google.android.gms.wearable.DataMap;
18   import com.google.android.gms.wearable.PutDataMapRequest;
19   import com.google.android.gms.wearable.PutDataRequest;
20   import com.google.android.gms.wearable.Wearable;
21
22
23   /*GoogleApiClient.ConnectionCallbacks arayüzünü activity'ye uyguladıktan sonra,
24   onConnected() ve onConnectionSuspended() metotlarını eklemeliyiz.
25   GoogleApiClient.OnConnectionFailedListener arayüzünü activity'ye uyguladıktan
26   sonra, onConnectionFailed() metodunu eklemeliyiz.*/
27   public class DataSendActivity extends WearableActivity implements
SensorEventListener, GoogleApiClient.ConnectionCallbacks,
28           GoogleApiClient.OnConnectionFailedListener {
29
30       /*Değişkenlerimiz*/
31       String rate = "250";
32       private TextView textViewHeart;
33       String wearable_data_path;
34       GoogleApiClient googleClient;
35       SensorManager sensorManager;
36       Sensor heartRateSensor;
37
38       @Override
39       protected void onCreate(Bundle savedInstanceState) {
40           super.onCreate(savedInstanceState);
41           setContentView(R.layout.activity_main);
42           getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
43           textViewHeart = findViewById(R.id.heart);
44           getHeartRate();
45
46       }
47
48       public void sendRate(View v) {
49           createGoogleApiClient();
50       }
51
52       private void createGoogleApiClient() {
53           /*Data Layer API'sine ulaşmak için GoogleApiClient sınıfını kullanırız.
54           Bu sınıf ile Google Play servislerine ulaşabiliriz. Diğer tüm hizmetlerin
ana giriş noktası bu sınıftır.
55
56           Bu sınıf içerisinde çeşitli statik metotlar bulunmaktadır.
57           Amacımız, sadece Wearable.API servisine erişmektir. Bu servisi kullanarak
58           iki cihaz arasında veri senkronizasyonu işlemini yapacağız.
59           Activityye eklediğimiz, onConnected(), onConnectionSuspended() ve
60           onConnectionFailed() metotlarını bu istemci ile ilişkilendirmek için
61           .addConnectionCallbacks(this)
62           .addOnConnectionFailedListener(this)
63           verilen metotları kesinlikle istemci ile ilişkilendirmeliyiz.
64           bunları yazmadığımızda, eklediğimiz metotlar çalışmayacaktır.
65           dolayısıyla herhangi bir veri senkronizasyonu da olmayacaktır.
66           build(): Bu metot ile istemci oluşturulur. Tabi bu işlemler ile sadece
istemciyi oluşturduk. istemciyle çalışabilmek için connect() metodunu kullanmalıyız.*/
```

```
67
68
69              googleClient = new GoogleApiClient.Builder(this)
70                      .addApi(Wearable.API)
71                      .addConnectionCallbacks(this)
72                      .addOnConnectionFailedListener(this)
73                      .build();
74
75              /*path bilgisi tanımlanır.*/
76              wearable_data_path = "/wearable_data";
77
78              /*connect() bu metot ile onConnected() metodu çağrılır.*/
79              googleClient.connect();
80          }
81
82          private void getHeartRate() {
83              /*Nabız ölçümü yapan sensör çalışmaya başlar.*/
84              sensorManager = ((SensorManager) getSystemService(SENSOR_SERVICE));
85              heartRateSensor = sensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE);
86              sensorManager.registerListener(this, heartRateSensor,
SensorManager.SENSOR_DELAY_NORMAL);
87          }
88
89
90          public void onAccuracyChanged(Sensor sensor, int accuracy) {
91
92          }
93
94          public void onSensorChanged(SensorEvent event) {
95              /*Nabız sensöründen gelen veriler saat ekranında gösterilir.*/
96              if (event.sensor.getType() == Sensor.TYPE_HEART_RATE) {
97                  rate = "" + (int) event.values[0];
98                  textViewHeart.setText("Kalp Atış Hızı:" + rate);
99              }
100         }
101
102         /*onConnected(): connect() metodundan sonra, bağlanma isteği
103         başarıyla tamamlandığında bu yöntem asenkron olarak çağrılır.*/
104         @Override
105         public void onConnected(@Nullable Bundle bundle) {
106
107             /*DataMap: Öncelikle DataMap yapısını oluşturmalıyız. Bu yapı
108             içerisine, google clienta/akıllı telefona göndermek istediğimiz değerleri
eklemeliyiz.*/
109             DataMap dataMap = new DataMap();
110
111             /*DataMap sınıfı yapı olarak Bundle sınıfında benzer. Veriler
112             key-value olarak tutulur. key ifadesinin bu veriye erişmek için
113             kullanacağız.*/
114             dataMap.putString("rate", rate);
115
116             /*DataMap oluşturduktan sonra PutDataMapRequest nesnesi
117             oluşturmalıyız. Bu nesnenin temel amacı, iletilecek verinin path
118             yani yol bilgisini belirtmektir. Bunun için create() metodu
119             kullanılır. Path bilgisini akıllı saatte bu veriye erişmek için
120             kullanacağız.
121             setUrgent(): Eğer yapılacak işlem çok önemli ise setUrgent()
122             metodu ile işlemin acil olduğunu belirtmelisiniz. Örneğin, akıllı
123             saatten, akıllı telefondaki müzik uygulaması kontrol edildiğinde
124             işlemin gecikmeden yapılması beklenir. Bu durumda yukarıdaki
125             metodun kullanımı çok önemlidir.
126             setUrgent() metodu çağrılmadığı zaman, sistem yapılan
127             istekleri 30 dakika kadar geciktirebilir. Bu maksimum değerdir.
128             Genellikle birkaç dakika içinde işlem yapılır. */
129             PutDataMapRequest putDataMapRequest =
PutDataMapRequest.create(wearable_data_path).setUrgent();
130
131             /*DataMap ile oluşturulan verilerin, PutDataMapRequest nesnesine
132             eklenmesi için getDataMap() ve putAll() metotlarını kullanırız.
133             İletilecek veriler aşağıdaki gibi nesneye eklenir.*/
134             putDataMapRequest.getDataMap().putAll(dataMap);
135
136             /*PutDataMapRequest içinde bulunan verilere sahip olan
137             PutDataRequest nesnesi oluşturulur. */
138             PutDataRequest request = putDataMapRequest.asPutDataRequest();
139
140             /*request yani istek ağa gönderilir. Eğer akıllı saat ve telefon bağlı
141             değilse, bağlantı yeniden kurulduğunda veriler belleğe alınır ve
```

```
142             senkronize edilir. Yani cihazın o an bağlı olma şartı yoktur.
143             Bağlantı kurulduğunda veri otomatik olarak akıllı saate iletilir.*/
144             Wearable.DataApi.putDataItem(googleClient, request);
145         }
146 
147         /*onConnectionSuspended(): İstemci bağlantısı geçici olarak kesildiği
148         durumlarda bu metot çağrılır*/
149         @Override
150         public void onConnectionSuspended(int i) {
151 
152         }
153 
154         /*onConnectionFailed(): Bağlantının başarısız olduğu olayları
155         dinlemek için bir dinleyici kayıt eder. */
156         @Override
157         public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {
158 
159         }
160 
161         @Override
162         protected void onPause() {
163             super.onPause();
164             /*Kullanıcı activityden ayrıldığında, eğer istemci bulunuyorsa ve bu
istemci aktif ise istemci sonlandırılır.*/
165 
166             if (googleClient != null && googleClient.isConnected()) {
167                 googleClient.disconnect();
168 
169             }
170             sensorManager.unregisterListener(this, heartRateSensor);
171         }
172 
173     }
```

Manifest Dosyası

Akıllı saat için manifest dosyamız aşağıdaki gibi olacaktır.

```
1     <?xml version="1.0" encoding="utf-8"?>
2     <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3         package="com.mas.iotignitehealth">
4 
5         <uses-feature android:name="android.hardware.type.watch" />
6         <!--Nabız ölçümü için izin eklendi. -->
7         <uses-permission android:name="android.permission.BODY_SENSORS" />
8         <uses-permission android:name="android.permission.WAKE_LOCK" />
9 
10        <application
11            android:allowBackup="true"
12            android:icon="@mipmap/ic_launcher"
13            android:label="@string/app_name"
14            android:supportsRtl="true"
15            android:theme="@android:style/Theme.DeviceDefault">
16 
17            <activity
18                android:name=".DataSendActivity"
19                android:label="@string/app_name">
20                <intent-filter>
21                    <action android:name="android.intent.action.MAIN" />
22 
23                    <category android:name="android.intent.category.LAUNCHER" />
24                </intent-filter>
25            </activity>
26        </application>
27 
28    </manifest>
```

Uygulamayı Akıllı Saate Yüklemek

Geliştirilen uygulamayı akıllı saate yüklemek için şu adımları takip etmeniz gerekiyor:

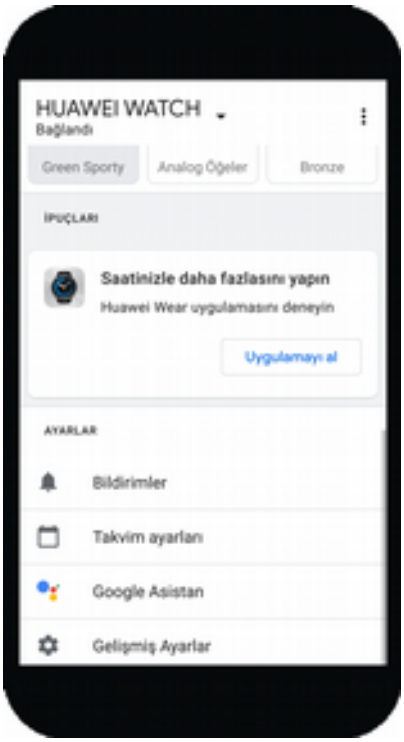
Öncelikle akıllı saatinizin geliştirici ayarlarını aktifleştiriniz. Bunun için akıllı saatinizde **Ayarlar > Sistem > Hakkında** yolunu takip ederek aşağıdaki arayüze erişim sağlayınız.

Arayüzde bulunan **Derleme Numarasına art arda 7 kez** tıklayınız. Bunu yaptıktan sonra geliştirici seçenekleri aktif olacaktır. Bir **Ayarlar** menüsüne döndüğünüzde aşağıdaki gibi **Geliştirici Seçenekleri'**nin aktif olduğunu görebilirsiniz.

Burada bulunan **Geliştirici Seçenekleri'**ne tıklayarak aşağıdaki iki seçeneği kontrol ediniz.

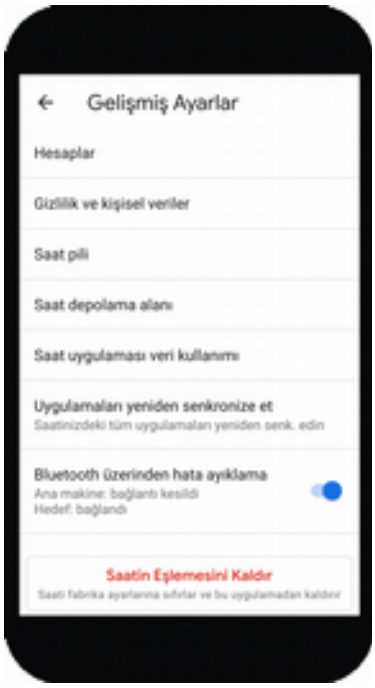
Bunları yaptıktan sonra akıllı telefonda bulunan Wear OS uygulamasını aşağıdaki gibi açınız. Uygulamaya aşağıdaki link'ten ulaşabilirsiniz:

https://play.google.com/store/apps/details?id=com.google.android.wearable.app&hl=it



Uygulamanın en altında yer alan **Gelişmiş Ayarlar** seçeneğine tıklayınız.

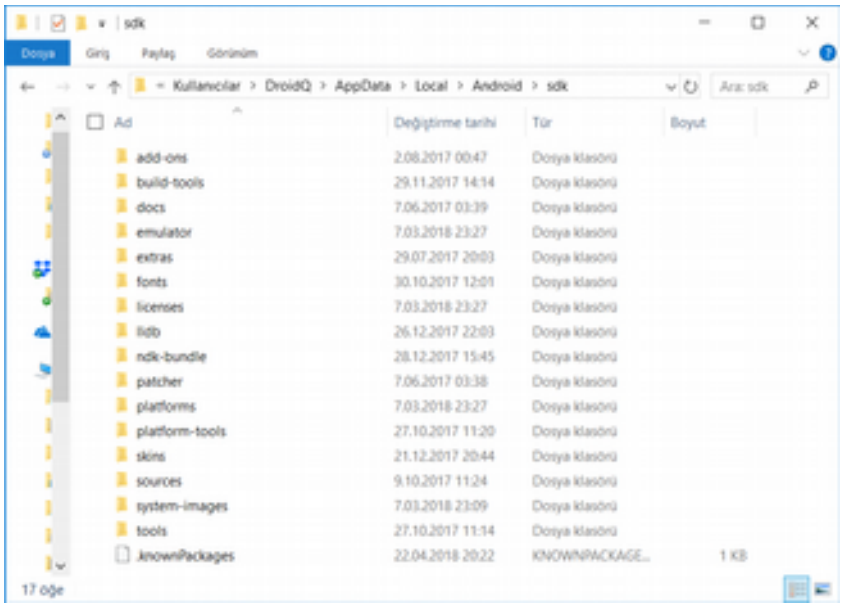
Açılan sayfada aşağıda görüleceği üzere Bluetooth üzerinden hata ayıklama seçeneğini kontrol ediniz. Burada Hedef: Bağlandı mesajı görülmektedir. Buradan akıllı saat tarafında yapılan ayarların doğru olduğu ve akıllı saat için geliştirici seçeneklerinin aktif olduğu anlaşılır. Ancak akıllı saate uygulama yükleyebilmek için Ana Makine içinde bağlandı mesajını almamız gerekiyor.



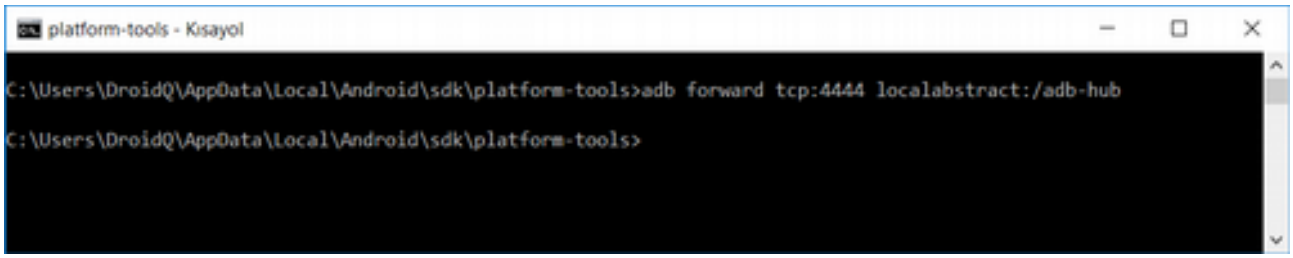
Akıllı saate uygulama yüklemek için öncelikle akıllı telefonunuzu Android Studio açıkken bilgisayarınıza bağlayınız. Daha sonra aşağıda verilen yolu takip ediniz.

C:\Users\[KULLANICI ADI]\AppData\Local\Android\sdk

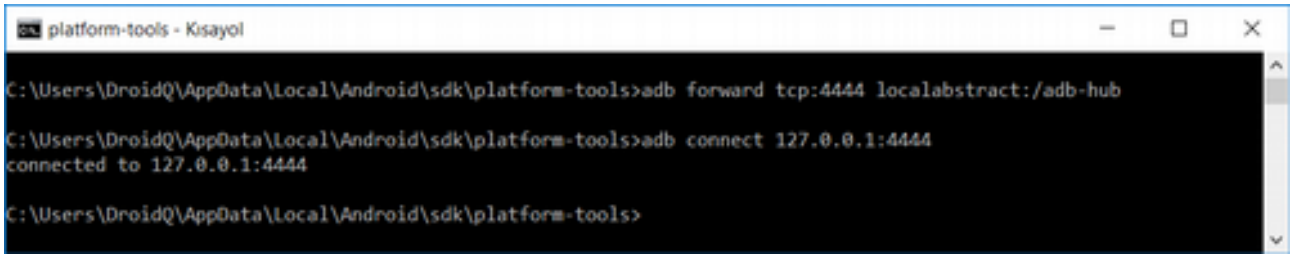
Kullanıcı Adı yazılan yere sizin kendi bilgisayarınızın kullanıcı adını girmeniz gerekiyor. Bu yolu takip ettiğinizde aşağıdaki pencere açılır.

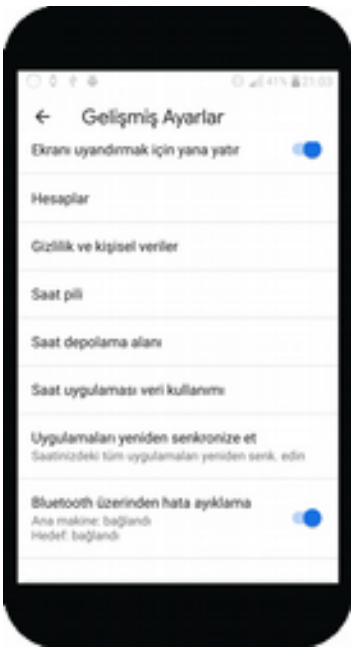


Burada bulunan **platform-tools** klasörüne sağ tıklayıp komut satırını açınız. Açılan komut satırına öncelikle aşağıdaki komutu giriniz ve Enter tuşuna basınız.



Daha sonra aşağıdaki komutu giriniz.





connected to mesajını gördükten sonra akıllı telefonunuzda bulunan Wear OS uygulamasında **Ana Makine: Bağlandı** mesajını görmeniz gerekiyor.

Şimdi uygulamayı akıllı saatinize yükleyebilirsiniz.

Akıllı Telefon Uygulaması

Akıllı saatten alınan nabız verisini IoT-Ignite ortamına iletmek için akıllı telefon tarafında aşağıdaki uygulamayı geliştirdik. Bu uygulamanın amacı; gelen nabız verisini IoT-Ignite platformuna iletmektir.

Arayüz Kodu

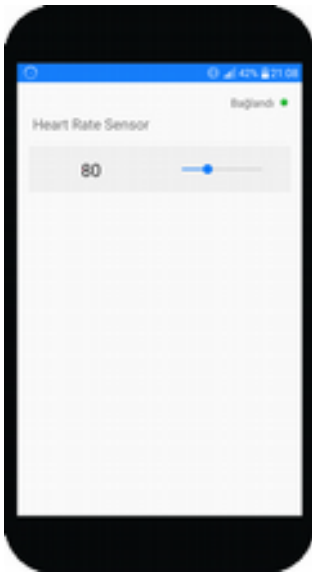
Uygulama arayüzü için kodlarımız aşağıdaki gibidir.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="@dimen/activity_vertical_margin">

    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <!--IoTIgnite bağlantı durumu -->
        <TableRow
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="right">

            <TextView
                android:id="@+id/textConnection"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="right|center"
                android:layout_marginRight="5dp"
                android:layout_span="2"
                android:text="Disconnected"
                android:textAppearance="?android:attr/textAppearanceSmall" />

            <ImageView
                android:id="@+id/imageViewConnection"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="right|center"
                android:src="@drawable/disconnected" />

        </TableRow>

        <!--Heart Rate sensörü -->
        <TableRow
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <TextView
                android:id="@+id/textRate"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_margin="5dp"
                android:text="Heart Rate Sensor"
                android:textAlignment="center"
                android:textAppearance="?android:attr/textAppearanceMedium" />
        </TableRow>

        <View
            android:layout_height="1dp"
            android:layout_gravity="center_vertical"
            android:layout_margin="5dp"
            android:background="@color/colorLightGrey" />

        <TableRow
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="@color/colorLightGrey">

            <TextView
                android:id="@+id/heartRateSensorValue"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="15dp"
                android:text="80"
                android:textAlignment="center"
                android:textAppearance="?android:attr/textAppearanceLarge" />

            <SeekBar
                android:id="@+id/heartRateSensor"
                android:layout_height="wrap_content"
```

```
                    android:layout_margin="20dp"
                    android:layout_weight="1" />
            </TableRow>

        </TableLayout>

    </LinearLayout>
```

Arayüz aşağıdaki gibi olacaktır.



Java Kodu

Uygulamanın Java kısmında çeşitli sınıflar geliştirdik. Bu sınıflarımız ve kodlarımız sırasıyla aşağıdaki gibi olacaktır.

Constants

Uygulamada kullanacağımız sabitleri tanımladığımız sınıfımız.

```
package com.mas.iotignitehealth;

/*Uygulamada kullanacağımız sabitleri tanımladığımı sınıfımız*/
public class Constants {

    /*Kurucu metodumuz*/
    private Constants() {
    }

    /*Sanal sensörler için VENDOR_INFO bilgisi*/
    public static final String VENDOR_INFO = "Health App";

    /*Sanal node ismi*/
    public static final String NODE_ID = "HumanNode";

    /*Sanal sensör ismi*/
    public static final String HEART_RATE_SENSOR = "HeartRateSensor";


    /*Heart Rate sensörü için ilk değer*/
    public static final int FIRST_VALUE_FOR_HEART_RATE = 80;


}
```

ListenerService

Akıllı saatten **Data Layer** ile gelen nabız verisini alacak servisimiz için kodlar aşağıdaki gibidir. Bu servis akıllı saatten nabız verilerini aldığı anda **MainActivity** isimli activity otomatik olarak başlatılır ve veri bu activity'de gösterilir.

```
1    package com.mas.iotignitehealth;
2
3    import android.content.Intent;
4    import com.google.android.gms.wearable.DataEvent;
5    import com.google.android.gms.wearable.DataEventBuffer;
6    import com.google.android.gms.wearable.DataMap;
7    import com.google.android.gms.wearable.DataMapItem;
8    import com.google.android.gms.wearable.WearableListenerService;
9
10   /*WearableListenerService: Bu sınıf, veri katmanı olaylarını (data layer event) bir
11   servis içinde dinlemeyi sağlar. Servisin yaşam döngüsü sistem tarafından yönetilir.
12   Veri öğeleri ve mesajları göndermek istediğinizde servise bağlanılır, aksi durumda
13   yani ihtiyaç olmayan durumlarda bağlantı kesilir. Servislerle çalışmak için bu
14   sınıf kullanılır.*/
15   public class ListenerService extends WearableListenerService {
16
17       /* onDataChanged(): Data Layer da meydana gelen veri değişiklikleri algılayan
18       metodumuz */
19       @Override
20       public void onDataChanged(DataEventBuffer dataEvents) {
21
22           /*Akıllı saatte veri olaylarını kontrol eden döngümüz*/
23           for (DataEvent event : dataEvents) {
24
25               /*DataEvent.TYPE_CHANGED: Data Layer üzerinde veri değişikliği
26               olduğunda (örneğin data layer'a yeni bir veri eklendiğinde) yapılacak
27               işlemleri belirlemede kullanılır.*/
28               if (event.getType() == DataEvent.TYPE_CHANGED) {
29
30                   /*Verinin path bilgisi alınır ve değişkene atanır.*/
31                   String path = event.getDataItem().getUri().getPath();
32
33                   /*Path bilgisi /wearable_data ise, akıllı saatten metin
34                   gönderildiği anlaşılır.*/
35                   if (path.equalsIgnoreCase("/wearable_data")) {
36
37                       /*DataItem yapısında gönderilen metnimizi DataMap
38                       nesnesine atarız*/
39                       DataMap dataMap =
DataMapItem.fromDataItem(event.getDataItem()).getDataMap();
40
41                       /*Yeni bir intent oluşturduk. Bu intent ile MainActivity
42                       isimli activity başlatılacaktır.*/
43                       Intent i = new Intent(this, MainActivity.class);
44
45                       /*Aldığımız DataMap verisini intent içerisine ekleriz.
46                       DataMap doğrudan eklenemediğinden toBundle() metodu ile veri
47                       Bundle nesnesine dönüştürülür. MainActivity isimli activity'de
48                       veriyi almak için "datamap" key değerini kullanırız.*/
49                       i.putExtra("data", dataMap.toBundle());
50
51                       /*Activity için bayrak tanımladık. Her veri değişiminde
52                       activity yeniden başlatılır ve gelen yeni veriyi kullanıcıya
53                       gösterir FLAG_ACTIVITY_CLEAR_TASK: Çalışan bir
54                       görev varsa temizlemeyi sağlar. FLAG_ACTIVITY_NEW_
55                       TASK: Activity'yi yeni bir görev içinde başlatır.*/
56                       i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK |
Intent.FLAG_ACTIVITY_NEW_TASK);
57
58                       /*Activity başlatılır.*/
59                       startActivity(i);
60                   }
61
62               }
63           }
64       }
65   }
```

VirtualHumanNodeHandler

IoT-Ignite ortamında sanal sensörler oluşturabiliriz. Burada sensörleri EHUB'ta değil, Android kodları ile oluşturacağız. Bunun için **VirtualHumanNodeHandler** sınıfını geliştirdik. Bu sınıf özetle aşağıda verilen işlemleri yapmaktadır.

- IoT-Ignite platformuna bağlantı sağlamak ve bağlantı durumunu kontrol etmek.
- Thing oluşturup bunları IoT-Ignite ortamına kayıt etmeyi sağlamak.
- Arayüzde bulunan sanal sensörlerden gelen verileri okumak ve bu verileri IoT-Ignite platformuna göndermek.

Bunlar genel olarak yapılan işlemler. Ancak kodumuzu incelerseniz, daha birçok işlemin gerçekleştirdiğini görebilirsiniz.

Bu sınıfın kodları aşağıdaki gibi olup, her kodun açıklamasını beraberinde verdik. Bu sayede uygulamanın çalışma mantığını daha iyi kavrayabilirsiniz.

VirtualHumanNodeHandler

```
1    package com.mas.iotignitehealth;
2
3    import android.app.Activity;
4    import android.content.Context;
5    import android.util.Log;
6    import android.widget.ImageView;
7    import android.widget.SeekBar;
8    import android.widget.TextView;
9
10   import com.ardic.android.iotignite.callbacks.ConnectionCallback;
11   import com.ardic.android.iotignite.enumerations.NodeType;
12   import com.ardic.android.iotignite.enumerations.ThingCategory;
13   import com.ardic.android.iotignite.enumerations.ThingDataType;
14   import com.ardic.android.iotignite.exceptions.AuthenticationException;
15   import com.ardic.android.iotignite.exceptions.UnsupportedVersionException;
16   import com.ardic.android.iotignite.listeners.NodeListener;
17   import com.ardic.android.iotignite.listeners.ThingListener;
18   import com.ardic.android.iotignite.nodes.IotIgniteManager;
19   import com.ardic.android.iotignite.nodes.Node;
20   import com.ardic.android.iotignite.things.Thing;
21   import com.ardic.android.iotignite.things.ThingActionData;
22   import com.ardic.android.iotignite.things.ThingConfiguration;
23   import com.ardic.android.iotignite.things.ThingData;
24   import com.ardic.android.iotignite.things.ThingType;
25
26   import java.util.Hashtable;
27   import java.util.Timer;
28   import java.util.TimerTask;
29   import java.util.concurrent.Executors;
30   import java.util.concurrent.ScheduledExecutorService;
31   import java.util.concurrent.ScheduledFuture;
32   import java.util.concurrent.TimeUnit;
33
34   /*VirtualHumanNodeHandler bu sınıf özetle
35   IOTIGNITE PLATFORMUNA BAĞLANTI,
36   THING OLUŞTURMA VE IOT-IGNITE PLATFORMUNA KAYIT ETMEK,
37   MAINACTIVITY ARAYÜZÜNDE BULUNAN SENSÖR VERİLERİNİ OKUMAK VEYA OKUMAYI İPTAL ETMEK,
38   SENSÖR VERİLERİNİ IOT-IGNITE ORTAMINA GÖNDERMEK ve daha birçok
39   işlemi yapmak için kullanılmaktadır.*/
40
41   /*onConnected() ve onDisconnected() metotlarını eklemek için
42   sınıfa ConnectionCallback arayüzü uygulanmalıdır.*/
43   public class VirtualHumanNodeHandler implements ConnectionCallback {
44
45       /********DEĞİŞKENLER********/
46
47       /*Uygulama içinde kullanacağımız değişkenlerimiz*/
48       private static final String TAG = "Health App";
49
50       /*ScheduledExecutorService: Komutları belirli bir gecikmeden sonra çalıştırmayı
veya periyodik olarak yürütmeyi sağlar.*/
```

```
51        private static volatile ScheduledExecutorService mExecutor;
52
53        /*İş parçacığı havuzunda tutulacak iş parçacığı sayısı 2 tane olacaktır.*/
54        private static final int NUMBER_OF_THREADS_IN_EXECUTOR = 2;
55
56        /*EXECUTOR_START_DELAY: iş parçası için gecikme süresi tanımladık*/
57        private static final long EXECUTOR_START_DELAY = 100L;
58
59        /*Thing bileşenlerden veri okumayı sağlayan bir görev tanımladık.
60        Hashtable: Bu sınıf, key-value çiftlerinden oluşan bir karma tablo oluşturmayı
sağlar.*/
61        private Hashtable<String, ScheduledFuture<?>> tasks = new Hashtable<>();
62
63        /*IotIgniteManager: IoT-Ignite pltaformuna bağlanmayı sağlayan temel
sınıftır.*/
64        private static IotIgniteManager mIotIgniteManager;
65
66        /*Sanal node oluşturmak için Node sınıfı kullanılır.*/
67        private Node mNode;
68
69        /*Sanal Thing oluşturmak için Thing sınıfı kullanılır.*/
70        private Thing mRateThing;
71
72        /*ThingType: Sensörün türünü belirtir. Sensör tipi ve üretici hakkında bilgi
depolamayı sağlar.*/
73        private ThingType mRateThingType;
74
75        /*ThingDataHandler: Thing verilerini Ignite ortamına göndermeyi sağlayan iş
parçacığı.*/
76        private ThingDataHandler mThingDataHandler;
77
78        /*IoT-Ignite ortamına bağlantı durumunu tutan bir değişken tanımladık*/
79        private boolean igniteConnected = false;
80
81        /*Bağlantı işlemini kontrol etmek için oluşturulan görevin bekleme süresi*/
82        private static final long IGNITE_TIMER_PERIOD = 5000L;
83
84        /*Timer: Bir iş parçacığını gelecekte yürütmek için kullanılır.*/
85        private Timer igniteTimer = new Timer();
86
87        /*IgniteWatchDog İş parçacığı ile Ignite bağlantısı kontrol edilir.*/
88        private IgniteWatchDog igniteWatchDog = new IgniteWatchDog();
89
90        private Context ctx;
91        private Activity activity;
92
93        /********KURUCU METODUMUZ********/
94        public VirtualHumanNodeHandler(Activity activity) {
95            this.ctx = activity.getApplicationContext();
96            this.activity = activity;
97        }
98
99
100       /********IOT-IGNITE PLATFORMUNA BAĞLANTI ********/
101       /*IgniteWatchDog İş parçacığı ile Ignite bağlantısı kontrol edilir.
102       Bunun için sınıfın TimerTask sınıfından türetilmesi gerekiyor.
103       TimerTask sınıfı bir defalık veya tekrarlanan görevlerin yürütülmesi
sağlanır.*/
104       private class IgniteWatchDog extends TimerTask {
105           @Override
106           public void run() {
107               /*IoT-Ignite platformuna bağlantı yoksa bağlantı işlemi yapılmaya
çalışılır.*/
108               if (!igniteConnected) {
109                   Log.i(TAG, "Ignite bağlantısı kuruluyor...");
110                   /*start metodu ile bağlantı sağlanır.*/
111                   start();
112               }
113           }
114       }
115
116       /*Ignite platformuna bağlanmayı sağlayan metodumuz.*/
117       public void start() {
118
119           try {
120               /*IotIgniteManager.Builder: IgniteManger oluşturulur.
121               Bu sınıf IoT-Ignite platformuna bağlanmayı sağlar.
122               setContext(): Bağlantı işleminin hangi uygulama için yapılacağı
```

```
123                 belirtilir. Buraya parametre olarak Context verilir.
124                 setConnectionListener: Parametre olarak ConnectionCallback arayüzü
125                 alır. Bağlantı sağlandığı zaman bu arayüz ile gelen onConnected()
126                 metodu çağrılır. Bu arayüzü VirtualHumanNodeHandler sınıfına yukarıda
127                 uygulamıştık. build(): bu metot ile bağlantı işlemi sağlanır.*/
128                 mIotIgniteManager = new IotIgniteManager.Builder()
129                         .setContext(ctx)
130                         .setConnectionListener(this)
131                         .build();
132
133             } catch (UnsupportedVersionException e) {
134                 Log.e(TAG, e.toString());
135
136             }
137             /*Bağlantı sağlandıktan sonra Timer durdurulur.*/
138             cancelAndScheduleIgniteTimer();
139         }
140
141         /*cancelAndScheduleIgniteTimer: Ignite platformuna bağlantı sağlandıktan sonra
Timer sonlandırılır.*/
142         private void cancelAndScheduleIgniteTimer() {
143             /*cancel: Planlanmış görevleri iptal ederek Timer nesnesini sonlandırır.*/
144             igniteTimer.cancel();
145             igniteWatchDog.cancel();
146
147             /*Timer nesneleri yeniden oluşturulur.*/
148             igniteWatchDog = new IgniteWatchDog();
149             igniteTimer = new Timer();
150
151             /*schedule(): igniteWatchDog iş parçası 500 milisaniye sonra tekrar
başlatılır.*/
152             igniteTimer.schedule(igniteWatchDog, IGNITE_TIMER_PERIOD);
153         }
154
155         public void stop() {
156             if (igniteConnected) {
157                 /*IoT-Ignite ile bağlantı kesildiği zaman Node ve Thing bileşenleri
158                 çevrimdışı yapmak için setConnected() metodunu kullanırız.
159                 Bu metot bağlantı durumunu ayarlamayı sağlar. Metodun ikinci
160                 parametresine istediğiniz değeri verebilirsiniz.*/
161                 mRateThing.setConnected(false, "Application Destroyed");
162                 mNode.setConnected(false, "ApplicationDestroyed");
163             }
164             /*mExecutor null ise sonlandırılır.*/
165             if (mExecutor != null) {
166                 mExecutor.shutdown();
167             }
168         }
169
170         /*onConnected() ve onDisconnected() metotları ConnectionCallback arayüzü ile
171         gelen metotlardır.
172         IoT-Ignite platformuna bağlantı sağlanırsa onConnected() metodu çağrılır.*/
173         @Override
174         public void onConnected() {
175
176             Log.i(TAG, "Ignite Bağlantısı Kuruldu!");
177             igniteConnected = true;
178
179             /*Bağlantı durumu aşağıdaki metot ile kullanıcı arayüzüne uygulanır.*/
180             updateConnectionStatus(true);
181
182             /*Ignite platformuna bağlantı sağlanırsa aşağıdaki metotlar icra edilir.*/
183             /*Node ve Thing oluşturulup ignite platformuna kayıt edilir. Ayrıca Thing
184             bileşenlere ilk değerleri atanır.*/
185             initIgniteVariables();
186             sendInitialData();
187             cancelAndScheduleIgniteTimer();
188         }
189
190         /*IoT-Ignite platformuna bağlantı kurulamaz ise onDisconnected() metodu
çağrılır.*/
191         @Override
192         public void onDisconnected() {
193             Log.i(TAG, "Ignite Bağlantısı Kesildi!");
194             igniteConnected = false;
195             /*Bağlantı durumu aşağıdaki metot ile kullanıcı arayüzüne uygulanır.*/
196             updateConnectionStatus(false);
197             cancelAndScheduleIgniteTimer();
```

```
198       }
199
200       /*IoT-Ignite platformu ile bağlantının olup olmadığını
201       MainActivity'de göstermek için bu metot kullanılır.
202       Amaç kullanıcıya bağlantı durumu hakkında bilgi vermektir.*/
203       private void updateConnectionStatus(final boolean connected) {
204           if (activity != null) {
205               /*runOnUiThread(): Main Thread üzerinde değişiklik yapmak için bir
206               Runnable tanımlar. Runnable ile arayüz güncellenir.*/
207               activity.runOnUiThread(new Runnable() {
208                   @Override
209                   public void run() {
210                       /*MainActivity arayüzünde bulunan iki kontrole erişim
sağlanır.*/
211                       ImageView imageViewConnection =
activity.findViewById(R.id.imageViewConnection);
212                       TextView textViewConnection =
activity.findViewById(R.id.textConnection);
213
214                       /*connected true ise kullanıcı arayüzünde bağlantının kurulduğu
215                       false ise bağlantının kurulamadığı bilgisi verilir.*/
216                       if (connected) {
217
imageViewConnection.setImageDrawable(activity.getResources().getDrawable(R.drawable.conn
ected));
218                           textViewConnection.setText("Bağlandı");
219                       } else {
220
imageViewConnection.setImageDrawable(activity.getResources().getDrawable(R.drawable.disc
onnected));
221                           textViewConnection.setText("Bağlanamadı");
222                       }
223                   }
224               });
225           }
226       }
227
228       /********THING OLUŞTURMA VE IOT-IGNITE PLATFORMUNA KAYIT ETMEK********/
229       /*initIgniteVariables(): Bu metot ile Node ve Thing oluşturulup IoT-Ignite
platformuna kayıt edilir.*/
230       private void initIgniteVariables() {
231           /*ThingType: Sensörün türünü belirtir. Sensör tipi ve üretici hakkında
bilgi depolamayı sağlar.
232           Bu sınıfın kurucu metodu için üç parametre alır:
233           THING_TYPE: Thing tipi bilgisi tanımlar.
234           VENDOR: Thing hakkında üretici bilgisi tanımlar.
235           THING_DATA_TYPE: Thing için veri tipi tanımlar.*/
236           mRateThingType = new ThingType(Constants.HEART_RATE_SENSOR,
Constants.VENDOR_INFO, ThingDataType.FLOAT);
237
238           /*NodeFactory(): Node nesnesi oluşturulur.
239           createNode() Bu metot oluşturulacak Node'un bilgilerini tanımlamayı sağlar.
240           Parametre olarak şu bilgileri kullanır.
241           NODE_ID: Node için benzersiz bir ID bilgisi.
242           NODE_ID: Node için label etiket tanımlar. Benzersiz olmak zorunda değil.
243           NODE_TYPE: Desteklenen Node türlerini tanımlar. GENERIC, RASPBERRY_PI ve
ARDUINO_YUN parametrelerini alabilir.
244           Burada genel amaçlı sanal bir sensör tanımladığımız için NodeType.GENERIC
parametresini kullandık.
245           4. parametreye bu gibi durumlarda null verilir.
246           NodeListener: Kayıt işlemini dinlemek için tanımlanır.*/
247           mNode = IotIgniteManager.NodeFactory.createNode(Constants.NODE_ID,
248                   Constants.NODE_ID,
249                   NodeType.GENERIC,
250                   null,
251                   new NodeListener() {
252                       @Override
253                       public void onNodeUnregistered(String s) {
254                           Log.i(TAG, Constants.NODE_ID + " kayıt edilmedi!!");
255                       }
256                   });
257
258           /*Node kayıtlı değilse kayıt edilir ve bağlantı sağlanır.*/
259           if (!mNode.isRegistered() && mNode.register()) {
260               mNode.setConnected(true, Constants.NODE_ID + " online");
261               Log.i(TAG, mNode.getNodeID() + " kayıt edildi!");
262           } else {
263               mNode.setConnected(true, Constants.NODE_ID + " online");
```

```
264                 Log.i(TAG, mNode.getNodeID() + " zaten kayıtlı!");
265             }
266
267             /*Node daha önce kayıtlı ise bu durumda bu Node'a bağlı olan Thing yani
sensör ve aktüatör oluşturabiliriz.*/
268             if (mNode.isRegistered()) {
269
270                 /*createThing(): Thing oluşturmak için kullanılır. Şu parametreleri
alır.
271                 THING_ID: Thing için ID bilgisi. Bunun eşsiz olması zorunludur.
272                 Thing Type: Thing tipini tanımlar. EXTERNAL, BUILTIN veya UNDEFINED
değerlerini alır.
273                 Burada EXTERNAL kullandık. Çünkü sensörlerimiz Gateway dışında
tanımlıdır.
274                 FLAG_STATE: Thing'in bir actuator gibi hareket edip etmeyeceğini
tanımlar.
275                 mRateThing için bu değer false. Çünkü bunu aktüatör değil ritim sensörü
olarak kullanacağız.
276                 Thing Listener: Olay dinleyici tanımlamayı sağlar. Bunun için
rateThingListener olay dinleyicisini inceleyiniz.
277
278                 Son parametre null olacaktır.*/
279                 mRateThing = mNode.createThing(Constants.HEART_RATE_SENSOR,
280                         mRateThingType,
281                         ThingCategory.EXTERNAL,
282                         false,
283                         rateThingListener,
284                         null);
285
286                 /*registerThingIfNotRegistered(): Bu metot ile sensörlerin Ignite
287                 platformuna kayıt edilmesi sağlanır.*/
288                 registerThingIfNotRegistered(mRateThing);
289             }
290         }
291
292         /*Tanımladığımız ritim sensörünü dinlemek için ThingListener tanımladık.
293         Bu olay dinleyicide 3 metot bulunmaktadır. Konfigürasyon ve action message
verileri buradan alınır.*/
294
295         private ThingListener rateThingListener = new ThingListener() {
296
297             /*onConfigurationReceived(): Yapılandırma IoT-Ignite tarafından
ayarlandığında bu metot icra edilir.*/
298             @Override
299             public void onConfigurationReceived(Thing thing) {
300                 Log.i(TAG, "Konfigürasyon alındı " + thing.getThingID());
301                 /*applyConfiguration: Thing bileşenlerden veri okumayı sağlayan metot
*/
302                 applyConfiguration(thing);
303             }
304
305             /*onActionReceived(): Thing bir aktüatör olarak ayarlanmışsa, gönderilen
eylem mesajları burada ele alınır.
306             Ritim sensörü bir aktüatör olmadığından burada bir işlem yapılmayacak*/
307             @Override
308             public void onActionReceived(String nodeId, String sensorId,
ThingActionData thingActionData) {
309
310             }
311
312             /*onThingUnregistered(): Thing, Ignite ortamına kayıt edilmediğinde bu
metot çağrılır.*/
313             @Override
314             public void onThingUnregistered(String nodeId, String sensorId) {
315                 Log.i(TAG, "Ritim sensörü kayıt edilmedi!");
316                 /*Thing kayıt edilemediği zaman veri okumayı sağlayan taskın
durdurulması gerekir.*/
317                 stopReadDataTask(nodeId, sensorId);
318             }
319         };
320
321         /*registerThingIfNotRegistered(): Kod içinde tanımlanan Thing nesnelerini
322         ignite ortamına kayıt etmeyi sağlar.*/
323         private void registerThingIfNotRegistered(Thing t) {
324             /*Thing bileşenlerinin kayıtlı olup olmadığı kontrol edilip,
325             kayıtlı olmayanların kayıt edilmesi sağlanır.*/
326             if (!t.isRegistered() && t.register()) {
327                 /*Ignite ile bağlantı kesildiği zaman Node ve Thing bileşenleri
```

```
328                 çevrimdışı yapmak için setConnected() metodunu kullanırız.
329                 Bu metot bağlantı durumunu ayarlamayı sağlar. Metodunu ikinci
330                 parametresine istediğiniz değeri verebilirsiniz.*/
331                 t.setConnected(true, t.getThingID() + " bağlandı");
332                 Log.i(TAG, t.getThingID() + " kayıt edildi!");
333             } else {
334                 t.setConnected(true, t.getThingID() + " bağlandı");
335                 Log.i(TAG, t.getThingID() + " zaten kayıtlı!");
336             }
337             /*applyConfiguration: Thing bileşenlerden veri okumayı sağlayan metot */
338             applyConfiguration(t);
339         }
340 
341         /********MAINACTIVITY ARAYÜZÜNDE BULUNAN SENSÖR VERİLERİNİ OKUMAK VEYA OKUMAYI
İPTAL ETMEK********/
342         /*applyConfiguration: Thing bileşenlerden veri okumayı sağlayan metot*/
343         private void applyConfiguration(Thing thing) {
344 
345             if (thing != null) {
346 
347                 /*stopReadDataTask(): Konfigürasyon alındığı zaman öncelikle veri
okumayı sağlayan görevin durdurulması gerekir.*/
348 
349                 stopReadDataTask(thing.getNodeID(), thing.getThingID());
350 
351                 /*getThingConfiguration(): Thing için gelen konfigürasyonu almayı
sağlar.
352                 getDataReadingFrequency(): Veri okuma sıklığını milisaniye cinsinden
alır. Varsayılan olarak READING_DO_NOT_READ değerine sahiptir. Hiçbir yapılandırma yoksa
bulut için veri gönderilmez.
353                 Eğer bu değer 0 ise, verinin geldiği anlaşılır.*/
354 
355                 if (thing.getThingConfiguration().getDataReadingFrequency() > 0) {
356 
357                     /*ThingDataHandler: Thing verilerini Ignite ortamına göndermeyi
sağlayan iş parçacığı.*/
358                     mThingDataHandler = new ThingDataHandler(thing);
359 
360                     /*Executors: ScheduledExecutorService nesnesi oluşturmayı sağlar.
361                     newScheduledThreadPool: Belirli bir gecikme sonrasında komut
çalıştırmak veya düzenli olarak yürütmek için komutları zamanlayabilen bir iş parçacığı
havuzu oluşturur.*/
362 
363                     mExecutor =
Executors.newScheduledThreadPool(NUMBER_OF_THREADS_IN_EXECUTOR);
364 
365                     /*ScheduledFuture: ScheduledExecutorService için planlı bir eylem
tanımlar.
366                     scheduleAtFixedRate(Runnable command, long initialDelay, long
period, TimeUnit unit):
367                     Verilen ilk gecikmeden sonra (EXECUTOR_START_DELAY), verilen
periyotta (thing.getThingConfiguration().getDataReadingFrequency())
368                     bir eylem oluşturmayı sağlar. Aynı zamanda bu eylemi yürütmeyi de
sağlar. Sonuç olarak Thing verisi Ignite ortamına gönderilir.*/
369 
370                     ScheduledFuture<?> sf =
mExecutor.scheduleAtFixedRate(mThingDataHandler, EXECUTOR_START_DELAY,
thing.getThingConfiguration().getDataReadingFrequency(), TimeUnit.MILLISECONDS);
371 
372                     /*thing ve node için ID bilgileri alınıp tek bir key elde edilir.*/
373                     String key = thing.getNodeID() + "|" + thing.getThingID();
374 
375                     /*put: Hashtable'da yani task içine key-value eşleştirmesi ekler.*/
376                     tasks.put(key, sf);
377                 }
378             }
379         }
380 
381         /*MainActivity'den gelen verilerin alınması sağlanır.
382         Daha sonra bu verilerin IoT-Ignite ortamına aktarılması sağlanır.
383         Bunu Runnable sınıfından türettik.*/
384         private class ThingDataHandler implements Runnable {
385 
386             /*Gelen Thing bileşeni tutacak nesnemiz*/
387             Thing mThing;
388 
389             /*Kurucu metodumuz Thign bileşeni alır.*/
390             ThingDataHandler(Thing thing) {
```

```
391                 mThing = thing;
392             }
393
394             @Override
395             public void run() {
396
397                 /*ThingData: IoTIgnite ortamına Thing verilerini göndermek için
kullanılır.
398                 Her Thing nesnesi veri göndermek veya aktüatörler gibi eylem mesajı
almak için bir ThingData nesnesine ihtiyaç duyar.*/
399
400                 ThingData mThingData = new ThingData();
401
402                 /*Gelen Thing, ritim sensörü ise*/
403                 if (mThing.equals(mRateThing)) {
404                     /*MainActivity arayüzünde bulunan kalp ritim değerini üreten
SeekBar kontrolüne erişim sağlanır.*/
405                     SeekBar seekBarRate = activity.findViewById(R.id.heartRateSensor);
406
407                     /*SeekBar'dan gelen veri ThingData nesnesine eklenir.*/
408                     mThingData.addData(seekBarRate.getProgress());
409                 }
410
411                 /*ThingData nesnesinde bulunan veri IoT-Ignite ortamına gönderilir.*/
412                 if (mThing.sendData(mThingData)) {
413                     Log.i(TAG, "VERİ GÖNDERİMİ BAŞARILI : " + mThingData);
414                 } else {
415                     Log.i(TAG, "VERİ GÖNDERİMİ BAŞARISIZ");
416                 }
417             }
418         }
419
420         /*Thing bileşenlerden veri okumayı sağlayan görevi durdurmayı sağlar.
421         Özellikle Thing bileşen ignite ortama kayıtlı olmadığında veya yeni gelen
422         bir konfigürasyon bilgisini alırken veri okumanın bu metot ile durdurulması
423         gerekiyor. Parametre olarak nodeId ve thingId bilgilerini vermemiz gerekiyor.*/
424         public void stopReadDataTask(String nodeId, String sensorId) {
425             /*nodeId ve thingId bilgileri birleştirilerek key bilgi elde edilir.*/
426             String key = nodeId + "|" + sensorId;
427
428             /*Task nesnesi gelen key bilgisini içeriyorsa if bloğu çalışır.*/
429             if (tasks.containsKey(key)) {
430                 try {
431                     /*Task içinde belirtilen key bilgisine sahip olan görev iptal
edilir.*/
432                     tasks.get(key).cancel(true);
433
434                     /*Task içinde belirtilen key bilgisine sahip olan görev silinir.*/
435                     tasks.remove(key);
436                 } catch (Exception e) {
437                     Log.d(TAG, "Veri alma durdurulamadı" + e);
438                 }
439             }
440         }
441
442         /********SENSÖR VERİLERİNİ IGNITE ORTAMINA GÖNDERMEK********/
443         /*IoT-Ignite ortamına bağlantı sağlandığı zaman aşağıdaki metot çağrılır.
444         Burada sensör için ilk değer ataması gerçekleşir. */
445         private void sendInitialData() {
446             /*sendData(): Bu metot ID bilgisi verilen Thing bileşenlere değer atamak
için kullanılır.*/
447             sendData(Constants.HEART_RATE_SENSOR,
Constants.FIRST_VALUE_FOR_HEART_RATE);
448
449         }
450
451         /*Ignite ortamına veri göndermeyi sağlar. Bu işlemin yapılabilmesi için
getDataReadingFrequency() metodu READING_WHEN_ARRIVE değerine sahip olmalıdır.*/
452
453         public void sendData(String thingId, int value) {
454
455             /*Ignite bağlantısı varsa*/
456             if (igniteConnected) {
457                 try {
458
459                     /*Node ID bilgisi Ignite ortamından alınır.*/
460                     Node mNode = mIotIgniteManager.getNodeByID(Constants.NODE_ID);
461
```

```
462                     /*Node null değilse işlem yapılır.*/
463                     if (mNode != null) {
464
465                         /*Node'a bağlı olan ve thingId ID bilgisine sahip olan Thing
nesnesi alınır.*/
466                         Thing mThing = mNode.getThingByID(thingId);
467
468                         /*Thing null değilse işlem yapılır.*/
469                         if (mThing != null) {
470
471                             /*ThingData: IoT-Ignite ortamına Thing verilerini göndermek
için kullanılır.
472                             Her Thing nesnesi veri göndermek veya aktüatörler gibi
eylem mesajı almak için bir ThingData nesnesine ihtiyaç duyar.*/
473
474                             ThingData mthingData = new ThingData();
475
476                             /*Gelen veri ThingData nesnesine eklenir.*/
477                             mthingData.addData(value);
478
479                             /*getDataReadingFrequency() metodu READING_WHEN_ARRIVE
değerini üretiyorsa Ignite platformuna veri gönderilebilir.
480                             Aksi durumda gönderilemez*/
481                             if (isConfigReadWhenArrive(mThing) &&
mThing.sendData(mthingData)) {
482                                 Log.i(TAG, "VERİ GÖNDERİMİ BAŞARILI : " + mthingData);
483                             } else {
484                                 Log.i(TAG, "VERİ GÖNDERİLEMEDİ!");
485                             }
486                         } else {
487                             Log.i(TAG, thingId + " kayıtlı değil!");
488                         }
489                     } else {
490                         Log.i(TAG, Constants.NODE_ID + " kayıtlı değil!");
491                     }
492                 } catch (AuthenticationException e) {
493                     Log.i(TAG, "AuthenticationException!");
494                 }
495             } else {
496                 Log.i(TAG, "Ignite Bağlantısı Kesildi!");
497             }
498         }
499
500         /*getDataReadingFrequency() metodu READING_WHEN_ARRIVE değerini üretiyorsa
Ignite platformuna veri gönderilebilir. Aksi durumda gönderilemez*/
501
502         private boolean isConfigReadWhenArrive(Thing mThing) {
503             if (mThing.getThingConfiguration().getDataReadingFrequency() ==
ThingConfiguration.READING_WHEN_ARRIVE) {
504                 return true;
505             }
506             return false;
507         }
508
509     }
```

MainActivity

Mainctivity sınıfı, uygulamanın ana sayfasıdır. Bu sınıfın amacı; **VirtualHumanNodeHandler** sınıfını kullanarak IoT-Ignite platformuna bağlanmayı sağlamak ve sanal sensörlerle arayüzde bulunan kontroller arasındaki iletişimi sağlamaktır. Sınıfın kodları aşağıdaki gibidir.

MainActivity

```
1    package com.mas.iotignitehealth;
2
3    import android.os.Bundle;
4    import android.support.v7.app.AppCompatActivity;
5    import android.widget.SeekBar;
6    import android.widget.TextView;
7    import java.util.Locale;
8
9
10   public class MainActivity extends AppCompatActivity {
11
12       /*Arayüzde bulunan kontroller için değişken tanımlama.*/
13       private TextView heartRateSensorValue;
14       private SeekBar heartRateSensor;
15
16       /*VirtualHumanNodeHandler bu sınıf özetle
17       IOTIGNITE PLATFORMUNA BAĞLANTI,
18       THING OLUŞTURMA VE IGNITE PLATFORMUNA KAYIT ETMEK,
19       MAINACTIVITY ARAYÜZÜNDE BULUNAN SENSÖR VERİLERİNİ OKUMAK VEYA OKUMAYI İPTAL
20       ETMEK, SENSÖR VERİLERİNİ IGNITE ORTAMINA GÖNDERMEK ve daha birçok
21       işlemi yapmak için kullanılmaktadır.*/
22       private VirtualHumanNodeHandler humanNodeHandler;
23
24
25       @Override
26       protected void onCreate(Bundle savedInstanceState) {
27           super.onCreate(savedInstanceState);
28           setContentView(R.layout.activity_main);
29
30           /*Parametre olarak MainActivity activity'sini gönderdik.*/
31           humanNodeHandler = new VirtualHumanNodeHandler(this);
32
33           /*start metodu ile IoT-Ignite platformuna bağlantı sağlanır.*/
34           humanNodeHandler.start();
35           getControlView();
36
37           /*getBundleExtra(): Bundle olarak gönderilen veri alınır. Bunun için
serviste tanımladığımız key bilgisini kullanırız.*/
38
39           Bundle data = getIntent().getBundleExtra("data");
40           if (data != null) {
41               /*Gelen metin değişkene atanır. Metne erişmek için Akıllı saat için
42               yazdığımız DataSendActivity isimli activityde tanımladığımız key
bilgisini kullanırız.*/
43
44               heartRateSensor.setProgress(Integer.parseInt(data.getString("rate")));
45               heartRateSensorValue.setText(data.getString("rate"));
46
47           }
48           /*Arayüz için bu metot çağrılır.*/
49           initUIComponents();
50       }
51
52       public void getControlView() {
53           /*Heart Rate sensörü için kontrol erişimi*/
54           heartRateSensor = findViewById(R.id.heartRateSensor);
55           heartRateSensorValue = findViewById(R.id.heartRateSensorValue);
56
57           /*Kontrollere ilk değerler atanır. Heart Rate 80 olur*/
58           heartRateSensor.setProgress(Constants.FIRST_VALUE_FOR_HEART_RATE);
59           heartRateSensorValue.setText(String.format(Locale.getDefault(), "%d",
Constants.FIRST_VALUE_FOR_HEART_RATE));
60
61           /*rate 0 ile 250 arasında üretilir.*/
62           heartRateSensor.setMax(250);
63       }
64
65       /*MainActivity arayüzünde bulunan kontrollere erişim sağlanır.*/
66       private void initUIComponents() {
67
68           /*Heart Rate değeri değiştiğinde algılamayı sağlayan bir listener
eklenir.*/
69           heartRateSensor.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
70
71               @Override
72               public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
```

```
73                      /*Yeni değer, kullanıcının okuyabilmesi için textview kontrolüne
yazılır.*/
74                      heartRateSensorValue.setText(String.format(Locale.getDefault(),
"%d", i));
75                  }
76
77                  @Override
78                  public void onStartTrackingTouch(SeekBar seekBar) {
79
80                  }
81
82                  @Override
83                  public void onStopTrackingTouch(SeekBar seekBar) {
84                      /*SeekBar üzerinde yapılan hareket durduğunda üretilen değer
85                      sendData metodu ile IoT-Ignite platformuna gönderilir.*/
86                      humanNodeHandler.sendData(Constants.HEART_RATE_SENSOR,
seekBar.getProgress());
87                  }
88              });
89          }
90
91          @Override
92          protected void onDestroy() {
93              super.onDestroy();
94              /*Uygulama sonlandığı zaman Ignite bağlantısı kesilir.*/
95              humanNodeHandler.stop();
96          }
97
98      }
```

Manifest Dosyası

Son olarak Akıllı telefon tarafında kullanacağımız uygulamanın manifest dosyası aşağıdaki gibi olmalıdır.

```
1       <?xml version="1.0" encoding="utf-8"?>
2       <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3           package="com.mas.iotignitehealth">
4
5           <uses-permission android:name="android.permission.WAKE_LOCK" />
6
7           <application
8               android:allowBackup="true"
9               android:icon="@mipmap/ic_launcher"
10              android:label="@string/app_name"
11              android:roundIcon="@mipmap/ic_launcher_round"
12              android:supportsRtl="true"
13              android:theme="@style/AppTheme">
14              <activity android:name=".MainActivity">
15                  <intent-filter>
16                      <action android:name="android.intent.action.MAIN" />
17
18                      <category android:name="android.intent.category.LAUNCHER" />
19                  </intent-filter>
20              </activity>
21
22             <!-- ListenerService servisini manifeste aşağıdaki gibi ekleyiniz. -->
23              <service android:name=".ListenerService">
24                  <intent-filter>
25
26                      <!-- Bu servis Data Layer da meydana gelen DATA_CHANGED olaylarına
duyarlı olacaktır. -->
27                      <action android:name="com.google.android.gms.wearable.DATA_CHANGED"
/>
28
29                      <!-- Saat ve telefon arasındaki veri transferinde wearable_data
path bilgisini kullanacağız. -->
30                      <data
31                          android:host="*"
32                          android:path="/wearable_data"
33                          android:scheme="wear" />
```

```
34                </intent-filter>
35            </service>
36
37        </application>
38
39    </manifest>
```

Uygulamayı AppStore'a Yüklemek

Hasta takip uygulamasını oluşturduk. Şimdi bu uygulamayı EHUB ortamında bulunan AppStore'a nasıl yükleyebiliriz, bundan bahsedelim. Bir uygulamanın IoT-Ignite platformu ile iletişime geçebilmesi için uygulamanın **Trusted**, yani güvenilir hale getirilmesi gerekir. Bu yapılmadığı zaman uygulamanız hiçbir şekilde IoT-Ignite platformu ile iletişime geçemez.

Uygulamayı AppStore'a yükleyebilmek için öncelikle Devzone ortamına üye olmanız ve bir Gateway cihazını sisteme kayıt etmeniz gerekiyor. Burada Gateway olarak Android bir telefonu kullandık.

Uygulamayı AppStore yükleme işlemini daha önceki bölümlerde ayrıntılı olarak ele aldığımızdan dolayı,burada tekrar aynı işlemleri göstermeyeceğiz.

Devzone'da Yapılması Gereken İşlemler

Amacımız; hasta takip uygulaması ile sanal sensörlerin oluşturulmasını sağlamakla birlikte ek olarak bu bölüme kadar anlatılanların daha iyi anlaşılması için Devzone ortamında bazı işlemler yapacağız. Bu başlık altında sırasıyla şu işlemleri yapacağız:

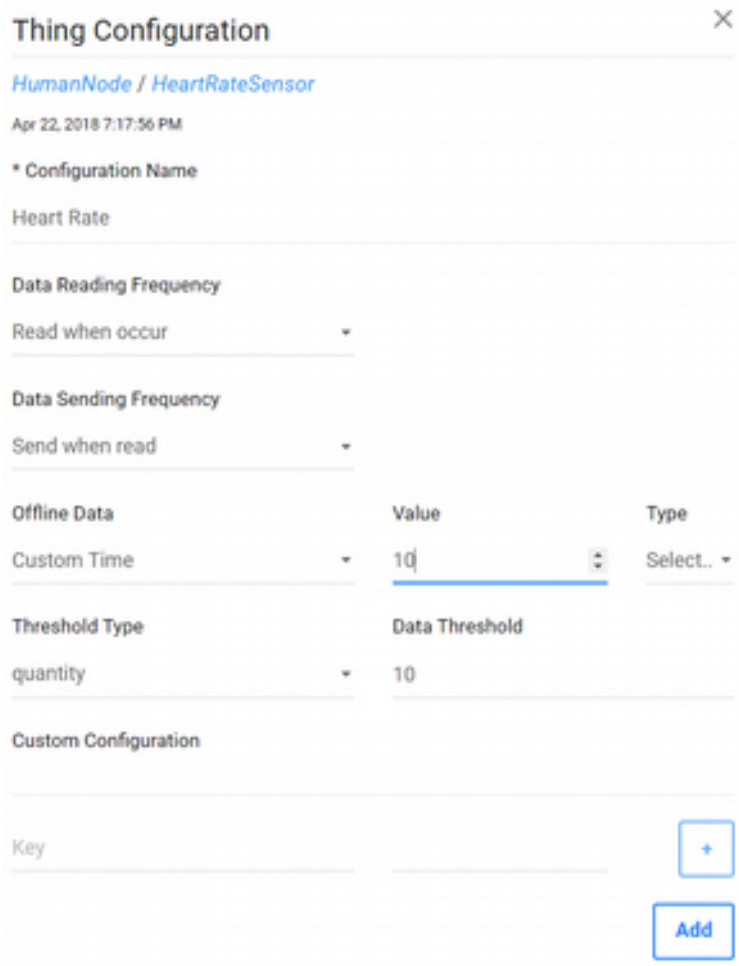
- Yeni bir Gateway eklemek. Bu Gateway, Android işletim sistemli telefon olmak zorundadır. Amacımız; hastadan gelen nabız verisi belirli bir değerin üstüne çıktığında ilgili kişiyi sesli ve mesaj olarak uyarmaktır.
- Sanal sensörlerin yapılandırılması,
- Cloud Rule ile bulut taraflı kural tanımlaması,
- Son olarak yukarıda yapılan işlemlerin Gateway'e gönderilmesini sağlamaktır.

Şimdi bunları nasıl yapacağımızı tek tek gösterelim.

Nabız Sensörünün Yapılandırılması

Sanal sensörleri Android kodları ile oluşturabiliriz. Ancak amacınız sensör verilerini IoT-Ignite platformuna aktarmak olduğunda sensörleri yapılandırmamız gerekiyor. Bunu yapmadığımız zaman sensörlerden gelen verileri okuyamayız. Burada kod ile oluşturduğumuz sıcaklık sensörü ve lamba actuatoru için yapılandırma ayarlarını yapmayı göstereceğiz. Daha önce Devzone ortamında **NODES** sekmesi altında oluşturduğumuz sensör ve node'ları görmüştük. Bu sayfada bizi ilgilendiren alan aşağıdaki gibidir.

Bu sensör için yapılandırmayı aşağıdaki gibi yapınız.

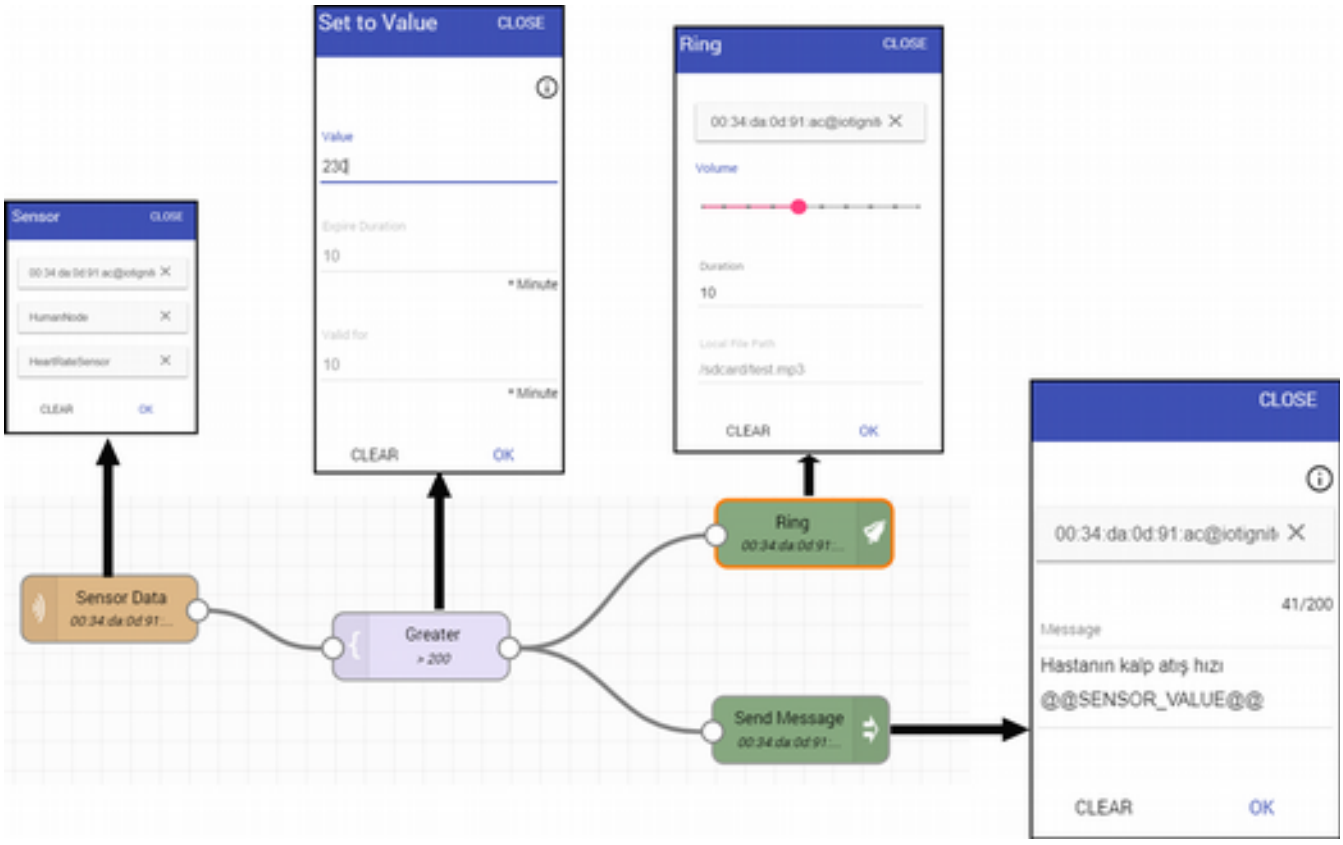


Cloud Rule ile Kural Oluşturulması

Cloud Rule ile bulut tarafında IoT uygulamaları için kurallar oluşturabiliriz. Kurallar ile IoT ekosisteminde bulunan sensör ve özellikle actuator bileşenlerini rahatlıkla kontrol edebiliriz. Bunun için Devzone ortamındayken **Rules** sayfasına girelim.

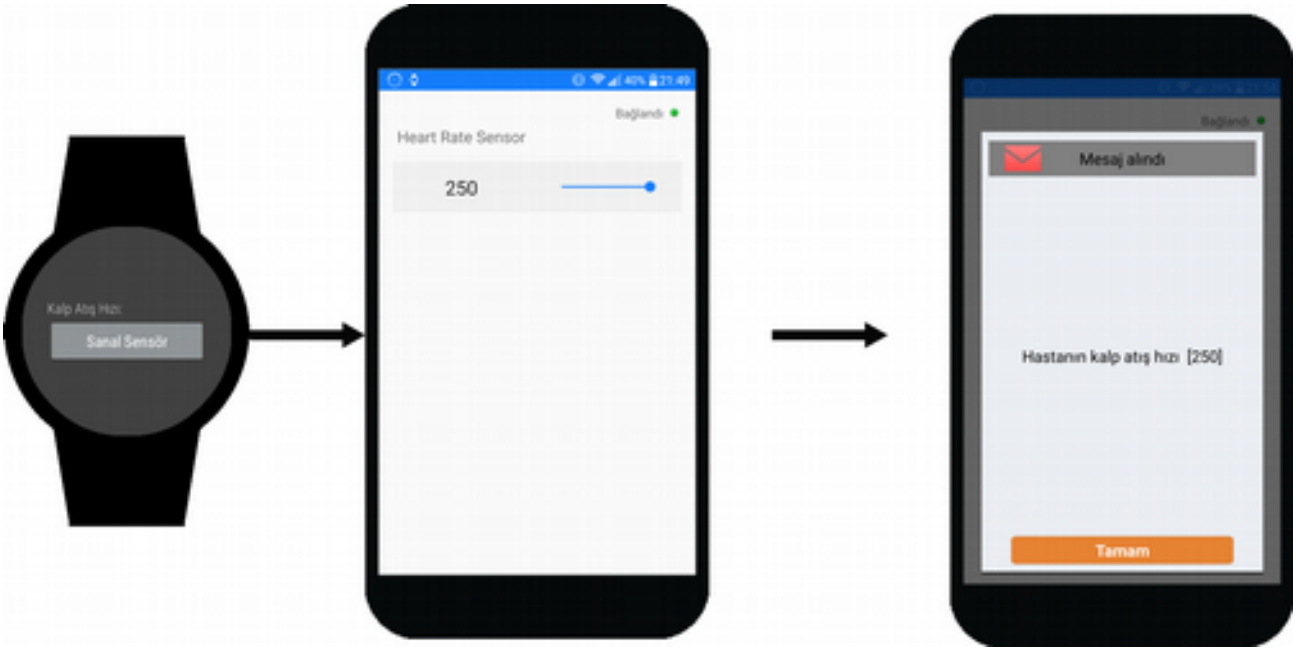
Açılan sayfada **Cloud Rule** veya **Gateway Rule** olmak üzere iki çeşit kural oluşturabiliriz. Amacımız **Cloud Rule** yani bulut tarafında bulunan kuralları oluşturmaktır. Bundan dolayı yukarıda bulunan **CREATE CLOUD RULE** seçeneğine tıklayalım.

Açılan kural editörünü kullanarak bulut taraflı bir adet kural oluşturacağız. Kuralın amacı; nabız sensöründen gelen veri, 230'un üstüne çıktığında ilgili kişide bulunan Gateway, yani akıllı telefona mesaj göndermeyi ve sesli uyarıda bulunmayı sağlamaktır.



Yapılandırmaların Test Edilmesi

Bu başlık yukarıda yapılan yapılandırma işlemlerinin test edilmesini sağlayacağız. Eğer yapılan işlemler doğru ise, ilgili kişinin telefonunda aşağıdaki sonucu görmemiz gerekiyor.



Referanslar

Referans	Link
IoT Raporu	https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf
IoT Sunumu	https://www.ituaj.jp/00_sg/20151012_TW15/slide/3_NTT.pdf
Raspberry Pi Resmi Sitesi	https://www.raspberrypi.org/
Devzone	https://devzone.iot-ignite.com
E-Hub Enterprise	https://enterprise.iot-ignite.com/v3/

Her türlü yardım ve önerileriniz için;
https://www.iot-ignite.com/contact/ sayfasından bize yazabilirsiniz.