TÜBİTAK

Durum Farkında Servis Platform (CASP) Mimarisi, Tasarımı ve Geliştirilmesi

Program Kodu: 3001

Proje No: 114E938

Proje Yürütücüsü:
Yrd. Doç. Dr. Ufuk ÇELİKKAN

Araştırmacı:
Yrd. Doç. Dr. Kaan KURTEL

Bursiyerler:
Aykut GÜNER
Orkut KARAÇALIK

ARALIK 2017
İZMİR

ÖNSÖZ

Bu proje, İzmir Ekonomi Üniversitesi bünyesinde, Yrd. Doç. Dr. Ufuk ÇELİKKAN yürütücülüğünde, Yrd. Doç. Dr. Kaan KURTEL'in araştırmacı olarak, Aykut GÜNER ve Orkut KARAÇALIK'ın bursiyer olarak katkılarıyla gerçekleştirilmiştir. Proje 3001 Başlangıç AR-GE projesi kapsamında TÜBİTAK tarafından desteklenmiştir. Her iki bursiyerimiz de projenin 3. ayından itibaren 27 ay süreyle projeye dâhil olmuşlar, yüksek lisans tezlerini bu projede üzerlerine düşen konularda seçmişlerdir.

Projede akıllı nesneleri kullanan durum-farkında (*İngilizce:context-aware*) uygulamalar için genel bir yazılım altyapısı ve mimari çözüm üzerine geliştirme (PCAD) ve doğrulama çalışmaları yapılmıştır. Böylece, sistem yöneticilerinin yeni gereksinimlerden kaynaklanan yeni durum farkındalık bilgileri ile çeşitli servisler arasında, mevcut bilgi sistemine mimaride bir değişiklik yapmadan, sadece görsel arabirimi kullanarak, gelen verileri uygulamalar ile sorunsuz ve kolaylıkla entegre eden bir yapı geliştirilmiştir.

Proje kapsamında yapılan nitelikli yayınlar, toplantılarda sunulan bildiriler ve seminerler:

1. Celikkan, U., Kurtel K. 2017. “Application of Service-Oriented Context-Aware Architecture to Laundry Management System”, Studies in Informatics and Control, 26(2) 193-202.
2. Guner, A., Celikkan U., Kurtel K. 2017. “A message broker based architecture for context aware IoT application development”, International Conference On Computer Science And Engineering/IEEE Xplore, 233-238.
3. Celikkan U., Kurtel K. 2015. “A Platform for Context-Aware Application Development: PCAD”, FedCSIS, 5, 1481-1488.
4. Güner, A. 2016. “The Use of Relational and NoSQL databases and their performance comparison based on a Node.JS implementation of a Context Aware System”, İzmir Ekonomi Üniversitesi Bölüm Semineri.
5. Karaçalik, O. 2016. “Evaluation of Actor Model Formalism in the Implementation of a Context-Aware Platform”, İzmir Ekonomi Üniversitesi Bölüm Semineri.
6. Çelikkan U., Kurtel K., Yanığlı, B. “Nesnelerin İnterneti için Sensör Tabanlı Yazılım Uygulaması: Veteriner Tanı Destek Sistemi”, Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi. (Yayın başvurusu yapıldı)

Proje kapsamında yapılan web sitesi ve sektör paylaşımları:

7. Proje web sitesi: http://deviot.ieu.edu.tr/. Context aware solutions using IoT.

8. Teta Teknik Tarım Hayvancılık sektör paylaşımı. 2017. Nesnelerin interneti için sensör tabanlı yazılım uygulaması: veteriner tanı destek sistemi.
9. Probel Yazılım sektör paylaşımı. 2016. PCAD Durum Farkında Servis Platform (PCAD) Mimarisi ve Akıllı Çöp Sepeti Uygulaması ve Çöp Toplama Sistemi.

Proje sürecinde yapılan yayınlar, yukarıda listelenen çalışmalar ile akademiye ve endüstriyle bilgi paylaşımı gerçekleştirilmiştir. Elde edilen geri dönüşler projenin gelişimine ve sonuçlanmasına olumlu katkı sağlamıştır. Projede yapılan çalışmalar, proje yürütücüsünün ve araştırmacının akademik kariyerlerini olumlu etkilemiş, araştırma becerilerinin gelişmesine yardımcı olmuştur. Araştırmacı, güncel yazılım geliştirme yöntem ve araçları hakkında bilgisini genişletmiş ve vermekte olduğu derslerde belirgin bir olgunluk sağlamıştır. Proje aynı zamanda iki yüksek lisans öğrencisinin tez çalışması olmuştur. Bu tezler halen sürmekte olup ve her iki tezden de önümüzdeki zaman diliminde nitelikli yayın çıkma beklentisi vardır.

Projede özellikle durum farkında ve nesnelerin interneti konusundaki gelişmelere koşut olarak farklı alanlarda uygulanabilme özelliğine sahip bir uygulama geliştirilmiştir. Böylelikle, platforma ve ara yüzlere ilişkin yapılan çalışmaların, farklı alanlarda kullanılması mümkün olacaktır.

Bu proje TÜBİTAK tarafından desteklenmiştir. Bu desteği vererek çalışmaların yapılmasını mümkün kılan TÜBİTAK'a, proje ekibi olarak teşekkür ederiz.

TÜBİTAK

İÇİNDEKİLER

TABLO, ŞEKİL VE PROGRAM LİSTELERİ

TABLOLAR

ŞEKİLLER

PROGRAMLAR

ÖZET

Modern bilişim ve iletişim teknolojileri, çok geniş bir çevrede etkili olan bilgisayar ağları ve uygulamaları aracılığı ile çeşitli kaynaklardan verileri toplamakta ve işlemektedir. Nesnelerin interneti olarak da tanımlanan bu kavram, çeşitli haberleşme protokolleri sayesinde bilgi paylaşan ve haberleşen karmaşık bir ağ oluşturmuş cihazlar sistemidir. Bu sistem karmaşık bir yapı göstermekte, farklı yapısal özelliklerde ve değişmeye meyilli fazla sayıda yazılım ve donanım ürününü içinde barındırmaktadır. Nesnelerin interneti kavramını en fazla destekleyen olgu ise, akıllı nesneler olarak tanımlanan sensör, kamera, mikro yongalar ve RFID teknolojileridir.

Bu projede, akıllı nesneleri kullanan durum-farkında uygulamalar için genel bir altyapı ve mimari çözüm geliştirilmiştir. İşletim sistemlerinden esinlenen servis tabanlı ara yazılım katmanı içeren mimari sisteminin tasarımında kullanılmıştır. Sensörler, cihazlar, servisler ve uygulamalar birbirinden ayrılmıştır. Projenin temel amacı, sistem yöneticilerinin yeni gereksinimlerden kaynaklanan yeni durum farkındalık bilgileri ile çeşitli servisler arasında, mevcut bilgi sistemine mimaride bir değişiklik yapmadan, gelen verileri uygulamalar ile sorunsuz ve kolaylıkla entegre eden bir yapı geliştirilmesidir.

Projede Durum Farkında Uygulama Geliştirme Platformu-PCAD olarak adlandırılan bir platform ve kullanıcı geliştiricileri için bir arayüz kütüphanesi geliştirilmiştir. Projede iki farklı geliştirme iskeleti kullanarak iki alternatif geliştirme yapılmıştır. Bunlardan ilkinde aktör tabanlı bir formalizm olan AKKA iskeleti ile Scala dili ve MySQL ilişkisel veri tabanı kullanılmıştır. İkinci gerçekleştirmede ise Node.js iskeleti, MQTT simsar mimarisi ve NoSQL veri tabanı kullanılmıştır. Her iki gerçekleştirmede de, yazılım uygulamalarının kullanması için RESTful kullanan bir ara yüz sunmaktadır. Kullanıcılar ayrıca bir web arayüzü aracılığı ile veri alabilmekte, uygulamalar ve sensör yazılımlarını kayıt ettirebilmekte, kimlik doğrulayabilme ve erişim haklarını belirleyebilmektedir. Ayrıca AKKA gerçekleştirilmesinde PYTHON dilinde programlama arayüzü kütüphanesi de sunulmaktadır. Her iki geliştirme iskeleti de yazılım mühendisliği esaslarına göre test edilmiş, sistemlerin kurulma ve kullanım kılavuzları hazırlanmıştır.

Projede yapılan bu çalışmaların akıllı nesneleri kullanan durum-farkında uygulamalar konusunda yeni projeler yapılmasına önayak olması beklenmektedir. Proje, ekipte yer alan kişilerin akademik ve araştırma becerilerini geliştirmelerine de katkı sağlamıştır.

Anahtar Kelimeler: Durum Farkında Uygulama Geliştirme Platformu, Yazılım Mimarisi, Uygulama İskeleti, Algılayıcılar

ABSTRACT

Applications using Information and Communication Technologies are collecting and processing a diverse range of data using networks of machines connected to each other through networks. This phenomenon is known as term Internet of Things, which encompasses a complex interconnection of devices that share data using various communication protocols. This architecture is further complicated due to continuous changes in hardware and software components, with differing operations and designs. Devices such as sensors, cameras, microchips, and RFID based products play an important role in support of the IoT paradigm.

The project delivered an infrastructure architecture and a solution for a context-aware application development that uses smart objects. The platform architecture was inspired by an operating system and modeled after a service-based middleware design. The sensors, devices and the services are decoupled from the applications. One of the fundamental goals of the project was to develop a platform that can react to new requirements stemming from changes in the context information without making any fundamental or substantial changes to the architecture. The platform seamlessly integrates data providers and applications.

The project delivered both the platform, called PCAD-Platform for Context Aware Development, and a public programming interface to enable application development. There are two different implementations, each with a different framework. The first implementation is based on AKKA framework, and used Scala as the programming language, and MySQL as the relational database. The second implementation is based on Node.js framework and MQTT broker architecture and used document database NoSQL as the data storage. Both implementations provide RESTful services for applications to access data and PCAD services, and a web interface allowing users to interact with the platform. Users can use the web interface to retrieve sensor data, and register sensors and application software to the platform, and for authentication and authorization. Additionally, AKKA implementation provided a PYTHON Application Programming Interface library. Both implementations were tested according to the software testing guidelines, after which user manuals were prepared.

The project has contributed to the advancement of the knowledge, development experience and research abilities of the project team members. In the future, it is expected to lead to the development of new context aware applications involving smart objects.

Keywords: Platform for Context Aware Development, Software Architecture, Framework, Sensors

1.GİRİŞ

Bu dokümanda, “Durum Farkında Servis Platformunun Mimarisi, Tasarımı ve Geliştirilmesi” projesinin sonuç raporu sunulmaktadır.

Projede, geliştirilen yazılım altyapısı ve mimari çözümü, akıllı nesneleri kullanan durum farkında uygulamalar için genel bir yapı sunmaktadır. Bu yapı, günümüzün internet ve mobil kullanım yaygınlığı ve IoT uygulamaların gelişimi dikkate alındığında önemli bir endüstriyel uygulama alanına sahiptir. Geliştirilen mimari yapı, temelde işletim sistemlerinin geleneksel ve güvenilir yapısal özelliklerini esas almaktadır. Temel amaç, yazılım servislerini katmanlar halinde tasarlayarak, sistemdeki farklı servis ve uygulamaları birbirinden ayırmak olmuştur. Bunu yapılırken aynı zamanda, servislerin ve uygulamaların birbirleriyle veri alış verişini gevşek bağlayarak, sistem genişletilebilir olarak tasarlanmıştır. Bu özellikle, çok çeşitli kurumsal ve/veya kişisel uygulamalar ve de sensörler gibi veri kaynakları dikkate alındığında, PCAD platformunun, tak-çalıştır özelliğine sahip, olası genişleme isteklerini olumlu yanıtlayan ve kırılganlığı yenebilen bir sistem olmasını sağlamıştır. Dolayısı ile sistem yöneticilerine, çevresel durumda meydana gelen değişikliklerin farkına vararak, sistemin yapısal özelliklerini değiştirmeden olası yeni gelişmeleri hızla yanıtlayabilecekleri bir platform sunulmuştur.

Projenin başlangıç aşamalarında yapılan çalışmaların neticesinde ve de yukarıda sunulan amaca uygun olarak gerçekleştirme çalışmaları iki farklı uygulama iskeleti kullanılarak yapılmıştır. Bu iskeletler: Akka ve Node.js olarak belirlenmiştir. Böylelikle raporun ilerideki bölümlerinde de sunulduğu üzere çalışma alanı genişletilmiş, bu da çalışmanın etki alanının genişlemesine neden olmuştur. Ayrıca başlangıç safhasında platformun CASP olarak sunulan ismi, bu ismin yurtdışında başka araştırıcılar tarafından kullanılmış olmasından dolayı PCAD (Durum Farkında Uygulama Geliştirme Platformu, *Platform for Context Aware Application Development*) olarak değiştirilmiştir.

Projede ilk olarak kapsamlı bir literatür araştırmasını takiben, platformun gerçekleştirilmesine yönelik araçlar, uygulama iskeletleri, bu iskeletlerle uyumlu çalışan programlama dilleri ve veri tabanları araştırılmıştır. Gerçekleştirme metodolojileri konusunda yapılan araştırmada, özellikle hazır uygulama iskeleti yazılımlarının ve araç takımı geliştirme desteklerinin üzerinde durulmuştur. Hem hazır kütüphanelerin kullanımı ve hem de kütüphanelerin proje içinde geliştirilmesi konusuna çalışılmış, üstünlük ve zaafları tartışılmıştır. Bu araştırmanın

sonucunda, PCAD servislerini geliştirmede Akka ve Node.js uygulama iskeletlerinin uygun fonksiyonlara sahip olduğu anlaşılmıştır.

Projenin devam eden aşamalarında, çalışmalar yazılım mühendisliği ilkelerine göre sürdürülmüştür. Her iki uygulama iskeleti geliştirilmiş, test edilmiş ve dokümantasyonu tamamlanmıştır.

Proje sürecinde, elde edilen sonuçların akademi ve sektör ile paylaşımı yapılmıştır. Proje çalışanları yaptıkları teorik ve uygulamalı çalışmalar sonucunda mevcut akademik ve araştırma becerilerini geliştirmişlerdir. İki yüksek lisans bursiyeri, ders aşamalarını başarı ile tamamlamışlar ve yazımına devam etmekte olup, tez çalışmalarını sonuçlandırmalarını takiben başka yayınların da çıkması beklenmektedir.

Bu raporun geri kalan kısmın şu şekilde oluşmaktadır: Bölüm 2'de literatür özeti verilmiş. Bölüm 3'de amaç ve yöntemler belirtilmiştir. Bölüm 4’de bulgular ve tamamlanan yazılımın çıktıları verilmektedir. Bu bölümde mimariyi oluşturan servislerin tasarımı ve platformun AKKA ve Node.js iskeletleri kullanılarak yapılan gerçekleştirmesi detaylı anlatılmaktadır Bölüm 5 ve 6’da ise sırası sistemin sınanması ve kullanıcı kılavuzu yer almaktadır. Bölüm 7'de ise sonuç kısmı verilerek rapor sonlandırılmıştır.

2. LİTERATÜR ÖZETİ

2.1 Durum Farkında Sistemler ve Yazılım Mimarisi

Durum-farkında sistemler adaptif sistemler olup çevrelerinde oluşan değişikliklere, kullanıcının bir müdahalesi olmadan tepki verebilen sistemlerdir. Durum kavramının içine objelerin, nesnelerin ve uygulamaların içinde bulundukları durumu anlatmakta kullanılan, zamanla değişen her türlü veri, çevreleriyle olan ilişkilerinde gösterdikleri davranış biçimleri ve hareketleri dâhil edilmektedir (Dey, 2000, 2001).

Literatürde durum farkında sistemler gerçekleştirilmesinde (*İngilizce*: *implementation*) kullanılması önerilen çeşitli mimari yapılar mevcuttur. Bu mimari yapıların ortak özelliği bilgi kaynaklarından durum bilgisini toplayan fonksiyonlar ile bu bilgiyi kullanan fonksiyonların birbirinden ayrıştırmasıdır. Bu amaca ulaşmanın en kabul edilen yolu ise katmanlı bir mimari yapı kullanarak, alt katmanlardaki işlem detaylarının üst katmanlardan saklanmasıdır. Bu sayede, sensörlerden gelen bilginin toplanması, bilginin iş mantık süreçlerinde kullanılması ve en son olarak kullanıcıya sunulması işlevleri birbirinden ayrılır. Böyle bir yapı kavramsal olarak beş katman şeklinde Ailisto ve diğerleri tarafından önerilmiştir (Ailisto vd., 2002). Bu beş katman fiziki, veri toplama, çıkarım, veri depolama ve kullanıcı ara yüzü olarak önerilmiştir. Katmanlı yapı kullanımı sistemin genişlemesini ve modüllerin tekrar kullanılmasını daha kolay hale getirir. MVC paradigmasına aşina olanlar için bu beş katmanı MVC'deki katmanlarla eşleştirmek oldukça kolaydır. Rehman ve diğerleri (Rehman vd., 2007) MVC tabanlı bir durum-farkında mimari yapı önermişlerdir. Durum-farkında sistem mimarileri iki temel kıstas göz önüne alınarak sınıflandırılmışlardır (Baldauf vd., 2007; Hristova, 2009):

1. Durum bilgisi edinme biçimine göre
2. Durum yönetimi ve durum-farkında sistemin bileşenlerinin yönetimi biçimine göre

Birinci durum bilgisi edinme biçimi kategorisinde genel hatlarıyla üç tür mimari yer almaktadır:

1. Doğrudan sensör ulaşımı kullanan mimariler
2. Ara yazılım (*İngilizce: middleware*) kullanan mimariler
3. Durum sunucusu (*İngilizce: context server*) kullanan mimariler

Uygulamaların sensörlerle direk iletişimi üzerine kurulmuş olan mimari yapılarda uygulamalar ile sensör'leri birbirlerine sıkı bir şekilde bağlanırlar (*İngilizce: tightly coupled*). Bu tür yapılar gerçekleşme bakımında daha kısa süre alsalar da esnek olmamaları yüzünden eleştiri almaktadırlar, bu da uzun vadede geliştirme maliyetlerini artırmaktadırlar. Sensörlere ulaşım koordine edilemediğinden dağıtık sistemler için uygun değildirler. Buna karşın, bir durum suncusunun kullanıldığı mimari yapılarda sensörlere ulaşım bir durum sunucusu tarafında koordine edilerek, uygulamaların eş zamanlı olarak sensörlere ulaşımı sağlanmıştır. Bu tür bir mimari dağıtık bir yaklaşım için çok uygundur. Ara yazılım kullanan yapılar katmanlı mimariye uygun oldukları için yukarıda sayılan katmanlı mimarilerin avantajlarına sahiptirler (Baldauf vd., 2007).

İkinci durum yönetimi ve durum-farkında sistemi oluşturan bileşenlerin yönetilme biçimi kategorisinde üç ana kısımda incelenebilir:

1. Widget
2. Bağlantılı servisler
3. Karatahta (*İngilizce*:*Blackboard*)

Widget'ler birer yazılım bileşenleri olup bu bileşenler birbirleri ile mesaj yollamak suretiyle iletişimde bulunurlar. Widget'lar bileşenle ilgili detayları saklayarak kuşatma (*İngilizce: encapsulation*) kavramını yerine getiririler. Widget yöneticisi, widget'lerin yönetiminden sorumludur. Uygulamalar, detayları bilmeden widget'ların yolladığı veriyi kullanırlar. Bunun alternatifi bir mimari olan Bağlantılı Servislerde ise widget'lar bağımsızdır ama birbirleri ile bağlantı halindedir. Fakat bu bağlantılar bir önceki mimarinin aksine bir widget yöneticisi tarafından yönetilmezler. Karatahta mimarisi veriyi ön planda tutan genel katmanlı bir modelidir. Bu model abonelik (*İngilizce: publish-subscribe*) üzerine kurulmuş olup sistemi oluşturun bileşenler ilgilendikleri sensör verisine kayıt olduktan sonra bir filtreleme sistemi ile sensör verileri kayıtlı bileşenlere ulaştırmaktadır. Böyle bir mimari, sisteme yeni durum bilgi sağlayıcılarının eklenmesini çok kolaylaştırmaktadır. Buna karşılık abonelik ve filtreleme işlemleri ekstra bir adım eklediğinden dolayı, uygulamaların veriye ulaşım süreci artmaktadır (Baldauf vd., 2007).

Son yıllarda popüler bir araştırma alanı olan servis tabanlı mimariler de (Papazoglou, 2008) durum-farkında sistemlerin gerçekleştirilmesinde kullanılmıştır (Truong ve Dustdar, 2009; Miraoui vd., 2008). Bu tür bir mimaride her bileşen birer servis olarak tasarlanmış, servislerin birbirlerini bulabilmesi ya da servislerin durum bilgi sağlayıcılarını bulabilmesi için (*İngilizce*: *service locating service*) bir servis bulucu kullanılmıştır (Gu vd., 2004). Bu çalışmalara

ilaveten, bağlamsal verilerin elde edilmesini ve saklanmasını iş mantığından ayıran çalışmalar da yapılmıştır (Merezeanu vd., 2016). Aynı araştırıcılar, endüstriyel uygulamalara bağlam-duyarlı algılama, bilgi işlem ve iletişim yetenekleri sağlamak için katmanlı bir mimari olarak inşa edilmiş bir çerçevede Kablosuz Algılayıcı Ağları, IoT ve Bulut Bilişim teknolojilerini entegre etmişlerdir.

Durum-farkında sistemlerin sağlaması gereken bir diğer fonksiyon ise durum bilgi sağlayıcılarının gönderdikleri bilginin depolanması ve durum bilgisinin modellenmesidir (*İngilizce: context modeling*). Durum modellemesi özellikle bilginin gösterimi, otomatik olarak işlenmesi ve transferi açısından önemlidir. Modelleme yöntemleri arasında sağladığı özellikler açısından en avantajlı olanı ontoloji tabanlı modellerdir (Strang ve Linnhoff-Popien, 2004; Perttunen vd., 2009; Gruber 2004). Diğer yöntemler arasında anahtar-değer (*İngilizce: key-value*) (ör. ısı=25), nesne tabanlı, markup ve grafiksel olanlar sayılabilir (Strang and Linnhoff-Popien, 2004; Perttunen vd. 2009).

Bu proje durum farkında mimarilere bir işletim sistemi (Stallings 2012) yaklaşımı getirmektedir. Projede önerilen mimari, yukarıda özeti verilen mimari yapıların avantajlı özelliklerini kullanan, katmanlı bir yapıda olup karatahta (*İngilizce: blackboard*) ve ara yazılım (*İngilizce: middleware*) modelini kullanmaktadır. Bu iki modelin birleşimi bir işletim sisteminin fonksiyonlarına benzerlik göstermektedir. İşletim sisteminde aygıtların oluşturduğu olaylar, nasıl uygulamalara dağıtılıyorsa, aynı şekilde karatahta modelinde de sensörlerden gelen durum bilgileri kayıtlı uygulamalara dağıtılır. Nasıl bir işletim sisteminin çekirdeği bütün uygulamaların ihtiyacı olan ortak servisleri sağlıyorsa (ör. dosya depolama) projede önerilen mimaride, bu genel servisleri uygulamalara sağlanır, çekirdek burada ara yazılım vazifesi görmektedir. PCAD, katmanlı bir yapı olduğu için genişlemesi kolay olacaktır. Proje bir işletim sistemi çekirdeği gibi hafif ve çevik (*İngilizce: agile*) bir platform önermektedir. Uygulamalar sensör bilgilerini rahatça alarak, isterlerse bu bilgiyi kendi istedikleri format ve yöntem kullanarak da depolayabilirler.

2.2 Yazılım Geliştirme Araçları, Veri Tabanları ve Programlama Dilleri

Projenin gerçekleştirilmesinde pek çok yazılım geliştirme aracı, veri tabanı ve programlama dili birlikte kullanılmıştır. PCAD servislerinin ve bu servislerin sensörler ve kullanıcılar ile haberleşmesi oldukça karmaşık ve fazla sayıda yazılım aracının birlikte çalışmasını gerektirmektedir. Bunun yanı sıra gerçekleştirme çalışmalarının iki ayrı kanaldan, Akka ve Node.js uygulama iskeletleri üzerinde yapılması da kullanılan yazılım araçlarını çeşitlendirmiştir.

Akka

Akka (Akka) Aktör modelinin bir uygulamasıdır. Aktör modeli, koşutzamanlı (*İngilizce: concurrent*) ve dağıtık sistemlerin ihtiyaçlarını karşılamak için geliştirilmiş bir modeldir. Akka, Hewitt (Hewitt vd., 1973, Hewitt, 1972) tarafından tanımlanan Aktör modelinden küçük sapmalar gösterse de, bunlar Aktör modelinin pratiğe uyarlanması için yapılmıştır. Aktör, içinde durum (*İngilizce: state*) bilgisini barındıran ve bu bilgiyi işleyen bir iş parçacığı ünitesine sahip birim olarak tanımlanmaktadır. Aktör modeli ve dolayısı ile Akka uygulama iskeleti ölçeklenebilir, dağıtık çalışabilir, eşzamanlık gibi avantajlar içermektedir. Aktörler arasında haberleşme, aktörlerin birbirlerini mesaj alışverişi kullanarak etkilemesi ile gerçekleşmektedir.

Akka'da abonelik (*İngilizce: publish-subscribe*) mekanizmasını kullanan olaya dayalı veriyolu (*İngilizce: event-bus*) özelliği mevcuttur. Bu özellik bire çok iletişime olanak sağlamaktadır. Böylece bir aktör bir olay yayınladığında, bu olaya abone olan bütün aktörlerin olaydan haberi olmaktadır. Bu modelin aksine, aktörler birbirlerinin posta kutusuna direk mesaj yollamak suretiyle bire bir mesajlaşma da yapabilmektedir.

Akka düşük hafıza tüketimi sayesinde yüksek ölçeklenebilir yazılımlar geliştirmeye olanak sağlamaktadır. Bir aktör sadece 300 bayt bellek tüketimine sahip olup, 1 gigabayt bellekte 3 milyon aktör yer alabilmektedir (Akka). Geliştiricilere yönelik iyi tanımlanmış uygulama geliştirme arayüzü sayesinde, tek makine üzerinde çalışan büyük düzeyde eşzamanlı, ölçeklenebilir dağıtık ve az hafıza kullanan yazılımların gelişmesine olanak sağladığından dolayı Akka uygulama iskeleti seçilmiştir.

Node.js

Platformun paralel olarak gerçekleştirildiği diğer bir uygulama iskeleti de Node.js'dir. Node.js V8 javascript motoru üzerinde çalışan, tek iş parçacığını kullanan (*İngilizce: single threaded*), olay-güdümlü, bloke olmayan giriş/çıkış modeli kullanan, ölçeklenebilir uygulamalar geliştirmek için tasarlanmış bir geliştirme platformdur (Node.js). V8 javascript motoru ise Google tarafından geliştirilen Chrome tarayıcısının üzerinde çalışan, javascript kodlarını makine diline çeviren bir motordur.

Node.js'in giriş/çıkış işlemlerinde asenkron (*İngilizce: asynchronous*) çalışması, uygulamanın bu süreçler devam ederken bloke olmaması ve her yeni gelen isteğe cevap vermesini sağlayan en önemli özelliğidir. Bu özellik Node.js'de geri çağırma adı verilen yapı ile sağlanmaktadır. Bu sayede Node.js, bir fonksiyon çağrıldığında fonksiyonun göndereceği

sonuçları beklemeden ve daha fazla sayıda isteğe cevap verebilmektedir. Bu durum Node.js kullanarak yazılan uygulamaları oldukça ölçeklenebilir hale getirmektedir.

Node.js kullanarak yazılmış uygulamalarda eğer merkezi işlemciyi yoğun kullanan kodlar varsa bunların ana iş parçacığından alınıp oluşturulan bir alt işlem parçası tarafından yapılması, merkez, işlemci ünitesinin bloke olmasını önlemektedir. PCAD sisteminin gerçekleştirilmesinde, servislerin hayata geçirilmesi alt işlem parçası kullanılarak yapılmıştır. Aynı makinada yaratılan alt işlem parçaları, ilk defa UNIX sistemleri ile ortaya çıkmış olan IPC metodunu kullanarak mesaj alış verişi yapmaktadır.

Veri Tabanı

PCAD platformunun Veri Yönetim Servisi, çeşitli kaynaklardan elde edilen verilerin saklanması ve gereksinim duyulduğunda servis edilmesi ile ilgilidir. Platformun geliştirilmesinde her iki uygulama iskeleti için ayrı ayrı veri tabanı sistemleri kullanılmıştır. Bunlar MySQL ve NoSQL veri tabanlarıdır. MySQL ilişkisel veri tabanlarında, veriler tablolar içinde satırlar ve sütunlar halinde tutulmaktadır. SQL standart dili yardımı ile veri tabanı yönetimi sağlanmaktadır. İlişkisel veri tabanında yeni bir alan eklenmek istendiğinde tabloyu baştan ele alıp yeni bir sütun eklenmesi gerekmektedir (MySQL). NoSQL veri tabanları, verileri ilişkisel tutmak yerine anahtar-değer çiftleri şeklinde tuttukları için ilişkisel veri tabanlarına nazaran daha hızlı sorgulama, güncelleme, ekleme ve silme işlemi sağlamaktadır. Tablo ve sütün kavramı olmadığı için yeni bir alana ihtiyaç duyulduğunda bunu doğrudan bir anahtar-değer ikilisi ekleyerek yapılması mümkün olmaktadır (NoSQL). Bu da hız ve performans açısından önemli kazanımlar sağlanması anlamına gelmektedir (Nayak vd. 2013; Liv vd., 2013). MongoDB ilişkisel veri tabanlarından farklı olup doküman temelli bir yapıya sahiptir. Bu yapı özellikle veri yükü fazla olan sistemlerde performans anlamında katkı sağlamaktadır (MongoDB).

Play Framework

Play Framework (Play) yüksek performanslı web uygulamaları yazmak için Scala dil desteğide bulunan bir Web uygulama iskeletidir. Akka akımları (*İngilizce:stream)* üzerinde inşa edilmiş oldukça ölçeklenebilir minimum kaynak kullanan, durum bilgisi tutmayan *(İngilizce: stateless),* web dostu ve bileşenlerin kolaylıkla entegre edilebildiği bir teknolojidir. Akka ile yapılan gerçekleştirmede Play iskeleti kullanılmıştır.

WebSoket

Websoket TCP (*İngilizce: Transmission Control Protocol*) protokolü üzerinden çift yönlü veri haberleşmesi sağlayan bir veri transfer kanalıdır (Pimentel vd., 2012). HTTP tabanlı veri alışverişinde tarayıcı yaptığı HTTP isteğine karşı cevap aldıktan sonra bağlantı kesilir. Takip eden her yeni istek için yeni bir bağlantı oluşturulur. Buna karşılık Websoket kullanıldığında sunucu ile web tarayıcı arasında kalıcı bir bağlantı açılır. Sunucuda herhangi bir veri güncellendiği zaman, web tarayıcının istek yapmasına gerek kalmadan, sunucu kendisi istemciye veri gönderebilir. PCAD gerçekleştirilmesinde uygulamalar PCAD'den sürekli veri almak istedikleri durumda bir websoket açarak bağlantı kurarlar ve bu bağlantı bitirilinceye kadar aktif kalır. Sensörler platforma veri yolladıkça bu veriler uygulamaya iletilir.

Slick

Slick, Scala için modern bir veri tabanı sorgu ve erişim kütüphanesidir. Depolanan veriye Scala komutlarını kullanıyormuş gibi erişilmesini sağlamaktadır. Diğer bir deyişle, sorguların SQL yerine Scala ile yazılmasını sağlamaktadır. Aynı zamanda veri tabanı erişiminin ne zaman gerçekleştiğine ve hangi verinin transfer edildiğine dair tam kontrol imkânı vermekte, böylece statik kontrol ve derleme güvenliği sağlanmaktadır. Slick, farklı arka veri tabanı tipleri için (MySQL, PostgreSQL) için kod üreten genişletilebilir bir sorgu derleyicisine sahiptir.

MQTT

Bir mesajlaşma protokolünün doğru seçimi, servisler ve paydaşlar arasındaki verimli etkileşimi, kullanımı kolaylığı, güvenilirliği ve uygulanabilirliği açısından önemlidir. Çeşitli araştırmacılar (Karagiannis vd., 2015; Yassein vd., 2016; Babovich vd., 2016) CoAP, MQTT, XMPP, RESTFul Services, AMQP, DSS ve Websockets gibi mesajlaşma protokollerini karşılaştırmış ve değerlendirmiştir. Özellikle, Toma ve Popa (2014) IoT tabanlı Atık Yönetim Çözümü uygulaması bağlamında CoAP, MQTT ve REST veri protokollerini araştırmıştır. Söz konusu araştırıcıların çalışmaları, ve bulguları projenin Node.js uygulama iskeletinin gerçekleştirilmesinde MQTT protokolünün değişim protokolü olarak seçilmesinde etkin olmuştur.

MQTT, 1999'da IBM ve Arlen Nipper'dan Andy Stanford-Clark tarafından geliştirilen, makineler arası iletişim (M2M) ve nesnelerin interneti (IoT) mesajlaşma protokolüdür. Basit ve kolay bir şekilde uygulanabilen bir yayıncı/abone mesaj taşıma protokolüdür ve özellikle küçük bellek ve güç kaynağı gerektiren sınırlı donanıma sahip yaygın cihazlar için

kullanılmaktadır (MQTT ve MQTT-IBM). Bir açık kaynak uygulaması olan MQTT protokolünün popüler ticari uygulamaları da başarı ile kullanılmaktadır. Bunların belli başlıları; Mosquitto, VerneMQ ve HiveMQ (Mosquitto; VerneMQ; HiveMQ) dir.

Nesnelerin interneti konusunda aracı (*İngilizce*:*broker*) haberleşme sistemlerini kullanan çeşitli çalışmalar ve öneriler araştırmacılar tarafında yapılmıştır. Collina vd. (2012), MQTT ve REST hizmetlerini kullanarak, akıllı cihazlar ve son kullanıcılar arasındaki boşluğu kapatmak için QEST adlı yeni bir etkileşim protokolü önermişlerdir. Fox vd. (2012), bir Java mesaj aracısı olan JMS ve diğer sistem bileşenlerini yönetmek ve sensörlerin kayıtlanması, bulunması, abonelik ve kontrol hizmetlerinin sağlanması için denetleyici içeren IoT bulut servisini sunmuşlardır.

RESTful Servisler

REST (*İngilizce:Representational State Transfer*) servis yönelimli mimari üzerine oluşturulan yazılımlarda kullanılan bir veri transfer yöntemidir. REST mimarisini kullanan servislere genel olarak RESTful servis denmektedir. HTTP protokolü üzerinde kurulu olup veri alışverişi en basit ve minimum içerikle yapılır. İstemci ve sunucu arasında kullanılan en popüler veri taşıma formatı JSON'dur. Fakat XML, CSV veya HTML formatında da veri taşınabilir. REST mimarisin en önemli kavramı kaynaklardır (*İngilizce: Resource)*. Kaynaklar istemciye sunduğunuz her şeydir. Kaynaklara URI üzerinden HTTP GET, PUT, DELETE ve POST metodları kullanılarak ulaşılır ve değiştirilir.

2.3 Uygulama Alanları

Yukarıda anlatılan mimari yapıları kullanan çeşitli uygulamalar geliştirilmiştir. Uygulama alanları arasında ev, akıllı sınıf (Sung vd., 2005), hastane (Munõz 2003), turizm (Abowd vd., 1997) karar destek sistemleri, iletişim sistemleri (Hong vd., 2009), çamaşır ve çamaşırhane (Celikkan vd., 2017; Celikkan vd., 2013; Lu ve Yu, 2010; Lu vd., 2010; Tajima vd., 2011, Van vd., 2012; Noor vd., 2012) sayılabilir. Bu konuda yapılmış uygulamalar ve mimariler hakkında kapsamlı bir araştırma Hong'da yer almaktadır (Hong vd., 2009).

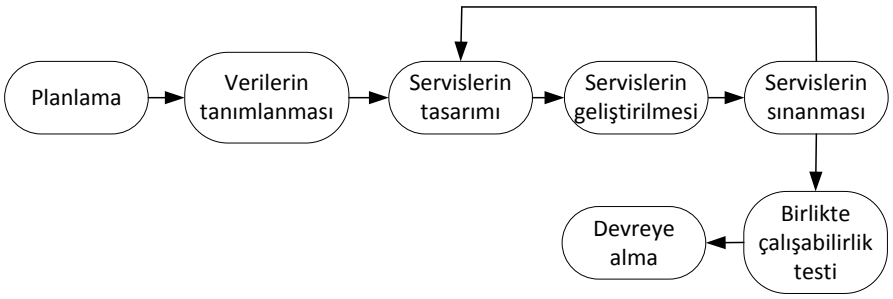
MQTT haberleşme protokolünün endüstriyel alanda pek çok kullanımı mevcuttur. Akıllı park sistemleri ve akıllı şehir hizmetleri (Kayal vd., 2017; Dhar vd., 2016; Antonic vd. 2015), akıllı pisuar yıkayıcı (Osathanunkul vd., 2017), akıllı ev IoT tasarımı (Kang vd. 2017) gibi M2M ve IoT tabanlı endüstriyel uygulamalarda geniş bir yelpazede kullanım için idealdir. Mobil uygulamalar (Doblander vd., 2016) ve engelli insanlar için hizmetler (Domingo, 2012) bulunmaktadır.

3. AMAÇ VE YÖNTEM

Bu bölümde projenin amacı ve bu amacı gerçekleştirmek üzere kullanılan yöntem ve süreçler sunulmaktadır.

Bu projede üç temel amaç saptanmıştır. Bunlar; 1) Durum Farkında Servis Platformunu ortaya koymak, 2) Elde edilen sonuçların araştırmacılar ve sektör ile paylaşmak ve 3) Proje çalışanlarının akademik ve araştırma becerilerini geliştirmektir.

Bu bölümde yukarıda tanımlanan amaçlara ulaşmak için kullanılan yöntem, genel yazılım geliştirme proje adımlarına uygun olarak düzenlenmiştir. Gerçekleştirme işlemleri, PCAD servislerinin, programlar, API'ler, ara yüzler ile bütünleştirilmesinin sistematik yapısal yaklaşımını sunmaktadır. Gerçekleştirme faaliyetleri ve birbirleriyle olan ilişkisi Şekil 1'de sunulmaktadır.



Şekil 1. PCAD projesi gerçekleştirme süreci

Bu adımlar:

- **Planlama:** Bu adımda gerçekleştirme işlemlerinin tanımlanması ve planlanması yapılmıştır. Ortak çalışma düzeni kurulmuş, haberleşme ve haftalık toplantı düzeni belirlenmiştir.
- **Verilerin tanımlanması:** Bu adımda sensörlerin verilerinin sanal sensörler aracılığıyla elde edilmesi tanımlamıştır. Bu veriler projenin ilerleyen geliştirme safhalarında ve özellikle de test aşamasında kullanılmıştır.
- **Servislerin tasarımı:** Bu adımda PCAD servislerinin detaylı tasarımı yapılmıştır. Bu tasarım çalışması sırasında işletim sistemi, yazılım mimarisi, durum farkında uygulama teknolojileri ve akıllı sistemlerini içeren ve literatür bölümünde de sunulan geniş bir okuma çalışması yapılmıştır.
- **Servislerin geliştirilmesi**: Bu aşamada geliştirme araçları seçilen ve tasarımı yapılan servislerin geliştirilmesi işlemine devam edilmiştir. Servislerin geliştirilmesinin hangi sırada yapılacağı için planlama adımında sunulan hususlar dikkate alınmıştır.

- **Servislerin sınanması:** Servislerin, servislerin birlikte çalışabilirliğinin, grafik arayüzlerinin ve performansları test edilmiştir. Bu faaliyette gerçekleştirimi tamamlanan PCAD servislerinin bir bütün olarak her iki geliştirme iskeleti için de testi yapılmıştır.
- **Devreye alma ve dokümantasyon:** Geliştirimi tamamlanan PCAD platformunun gerekli testleri tamamlandıktan sonra, donanım ve yazılım araçlarının ayarlarının yapılarak paket haline getirilmesi ve ilk çalıştırımının sağlanması, işlevini düzgün bir şekilde yerine getirdiğinin görülmesi işlemlerini kapsamaktadır. Bu adımda her iki geliştirme iskeleti içinde gerekli kullanım kılavuzları, sistem kurulumu, sıkça sorulan sorular gibi dokümantasyonlar yapılmıştır.

Çalışmaya rehberlik etmesi açısından projedeki kavramlara ilişkin bir sözlük geliştirilmiş, böylece projenin yürütücüsü, araştırıcısı ve bursiyerleri tarafından ortak kavramların tanımlanması ve isimlendirilmesi gerçekleştirilmiştir. Bu çalışma proje boyunca sürdürülmüş olup Ek-A'da sunulmaktadır.

Gerçekleştirme çalışmalarının planlanması aşamasında, bir gerçekleştirme stratejisi üzerinde çalışılmıştır. Projede önerilen sistemin gerçekleştirilmesinde ara yazılım (*İngilizce: middleware*) ve karatahta (*İngilizce: blackboard*) özelliklerini barındıran bir model seçilmiştir. Karatahta abonelik (*İngilizce: publish-subscribe*) üzerine kurulmuş bir model olup uygulamalar ve sensörler platform aracılığı ile birbirlerine gevşek bir bağla bağlanmıştır. Bunun sunucunda sistemin her hangi bir katmanında yapılan değişiklik diğer bileşenleri en az şekilde etkilemektedir. Platforma bilgi sağlayıcılar ve sensörlerin eklenip çıkarılması kolaylıkla gerçekleştirilebilmektedir. Prototipin geliştirilmesinde kullanılacak uygulama iskeleti, araç takımı ve programlama dilleri gibi seçeneklerden bir veya birkaçının seçimi için detaylı bir çalışma gerçekleştirilmiştir. Bu çalışmada öne çıkarılan sonuç, seçilecek yöntemin ara yazılım ve abonelik modellerinin gerçekleşmesine yardım etmesi olmuş ve bu doğrultuda çalışmalar yapılmıştır. Yapılan çalışmalar sonucunda iki temel seçeneğin olduğu anlaşılmıştır. Bunlar:

1) Her hangi bir uygulama iskeleti ya da araç takımı kullanmadan, sadece bir programlama dilinin temel yapılarını ve kütüphanelerini kullanarak bir gerçekleştirme yapmak.

   Bu seçenekte Java programlama dili üzerinde durulmuş, bütün sistemin tabandan tavana inşasının Java kullanılarak gerçekleştirilmesi durumu araştırılmıştır. Böyle bir çözümde özellikle küçük iş parçacıklarının yazılması (*İngilizce: threading*) başta olmak üzere sistemin diğer tüm alt yapı bileşenlerinin temelden inşa edilmesi

gerektiği ortaya çıkmıştır. Dolayısı ile sistemin doğru çalışması, performans ve ölçeklenebilme kriterlerini sağlaması bakımından oldukça büyük bir çabaya ihtiyaç duyulacağı anlaşılmıştır. Bunun yerine bu özellikleri sağlayan uygulama iskeleti ve araç takımı alternatifleri üzerine odaklanılmıştır.

2) Uygulama iskeleti ve araç takımı geliştirme desteklerini kullanarak gerçekleştirmeyi yapmak.

   Bu seçeneğin araştırılması esnasında iki alternatif üzerinde durulmuştur. Bunlardan birincisi Scala tabanlı Akka, diğeri ise JavaScript tabanlı Node.js uygulama iskeletleridir. Akka, aktör modeli kullanan koşutzamanlı ve dağıtık sistemlerin gerçekleştirilmesinde kullanılan bir uygulama iskeletidir. Araştırılan diğer uygulama iskeleti Node.js'de ise benzer şekilde ölçeklenebilir, asenkron çalışan sunucular üzerinde çalışması için tasarlanmış bir iskelettir.

Bir programlama dili kullanarak sistemin sıfırdan inşa edilmesinin sağlayacağı avantajların en önemlisi esnekliğidir. Sistem tasarımında uygulama iskeleti ve araç takımlarının kullanımında karşılaşılması muhtemel kısıtlayıcı faktörler minimize edilmektedir. Buna karşılık hem uygulama iskeletlerinin hazır bir şekilde sunduğu fonksiyonların yeniden yazılması hem de bunların istenilen bir şekilde gerçekleştirilmesi için hatırı sayılır bir çaba gerekmektedir. Bu noktada PayPal şirketinin yaptığı deney bunu destekler niteliktedir (PAYPAL). PayPal şirketi web sitelerinin en çok trafik çeken sayfasını ve bunun arkasındaki uygulamayı yeniden gerçekleştirmek ve iyileştirmek istemektedir. Bunu yaparken olası riskleri azaltmak için paralel takımlardan oluşan bir geliştirme ekibi kurmuştur. Bu ekiplerden birini yeni bir teknoloji olan Node.js'i kullanmakla görevlendirmiş, diğerini ise tecrübeli programcılardan oluşturmuş ve bilinen bir teknoloji olan Java'yı kullandırmıştır. Uygulamalar bittiğinde her ikisi de aynı fonksiyonel testleri başarıyla geçmiştir. Fakat Node.js daha az programcı sayısıyla iki kat daha hızlı yapılmış, %33 daha az kod yazılmış ve dosya sayısı Java'ya göre %40 daha azalmıştır. Tablo 1'de bu üç seçeneğin avantajları ve dezavantajları karşılaştırılmıştır.

Bu sonuçların ışığında, gerçekleştirmenin Akka ve Node.js uygulama iskeletleri üzerinde yapılmasına karar verilmiştir. Öncelikle bu gerçekleştirme çalışmalarının, Akka uygulama iskeleti için yapılması, sonrasında da Akka üzerinde elde edilecek uygulama deneyiminin, Node.js uygulamalarının geliştirilmesinde kullanılmasının uygun olacağı düşünülmüştür. Ancak projeye zaman planı açısından bakıldığında, ikinci uygulamanın, birincinin tamamlanmasını beklemeden başlamasının uygun olacağına karar verilmiştir. Böylelikle, uygulama iskeletlerinin geliştirme süreçlerinin birlikte ve belirli bir faz farkı ile yapılmasına

karar verilmiştir. Bu ikili geliştirme stratejisinin bir yararı da, Node.js iskeletinin, projenin planlanan şekilde yürütülmesini önemli ölçüde aksatan öngörülmemiş gelişmelerle karşılaşılması durumunda bir "B Planı" veya "Risk Planı" olarak işlev görecek olmasıdır.

Tablo 1. Java, Node.js ve Akka karşılaştırılması

	Avantajlar	**Dezavantajlar**	**Uygulamalar**
Java	Geliştirme esnekliği, platform bağımsızlığı, kaynak zenginliği, olgun bir dil olması	Platformun yapılması için daha çok zaman ve efor gerektirmesi	Oldukça fazla
Akka	Ölçeklenebilir, dağıtık çalışabilir, eşzamanlık, az hafıza kullanması	Programlamayı öğrenme zamanı, aktör modelini etkin kullanabilme, kodların kompleks hale gelmesi	CSC, Trafik Yönetim Sistemi (Hollanda), SVT (İsveç Televiz.) Alt yazı sistemi
Node.js	Asenkron, Ölçeklenebilir, bloke olmayan giriş/çıkış işlemleri	Asenkron programlama ilkelerini öğrenme zaman alabiliyor, her bir alt iş parçacığı 10MB ram tüketiyor.	Paypal, Linkedin

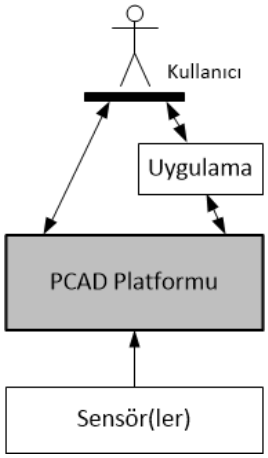
Planlama aşamasında, her iki uygulamada PCAD servislerinin gerçekleştirilme sürecindeki ortak ve farklılıkları açısından incelenmiştir. Akka ve Node.js uygulama iskeletlerinin gerçekleştirilmesinde tespit edilen benzerlikler ve farklılıklar aşağıda sunulmaktadır.

- Benzerlikler: 1) Veri Yönetim, Güvenlik ve Gizlilik, Birlikte Çalışabilirlik ve Raporlama servisleri her iki uygulama iskeletinde de büyük ölçüde benzerlikler göstermektedir, 2) Geliştirme işlemlerinin doğrulanması her iki uygulama için benzerlikler göstermektedir.
- Farklılıklar: 1) Node.js uygulamasında Kural Servisinde yorumlayıcı (İngilizce:*interpreter*) yer almamaktadır. Buna bağlı olarak Uyarı ve Bildirim Servisinde farklılıklar bulunmaktadır.

Uygulama iskeletleri arasındaki benzerlikler ve farklıların tespit edilmesini takip eden çalışmalarda, servisler arasındaki iletişim özellikle veri akışı açısından incelenmiştir. Böylece hangi servislerin öncelikli olarak geliştirileceğine karar verilmiştir. Bu çalışmanın sonucuna göre Kural, Veri Yönetim ve Uyarı-Bildirim Servislerinin öncelikle geliştirilmesinin proje

açısından yararlı olacağı sonucuna varılmıştır. Ayrıca Raporlama Servisinin projede kavramsal olarak yer almasının ve ancak geliştirilmesinin gerekli olmadığı sonucuna varılmıştır. Bunun nedeni Raporlama Servisinin PCAD platformunun fonksiyonel bir parçası olmasına rağmen, platformun proje bahsedilen genel amacına sağladığı katkının görece az olması ve benzer çalışmalarda genellikle üçüncü parti yazılım araçları ile çözülüyor olmasıdır. Ayrıca bu adımda söz konusu geliştirme işlemlerinin hangi yazılım araçları, programlama dilleri ve veri tabanı sistemleri kullanılarak yapılacağı çalışmaları sonuçlandırılmıştır.

Planlama aşamasında çalışma ortamının düzenlenmesi işlemleri de gerçekleşmiştir. Bu adımda, İEÜ tarafından projeye tahsis edilen çalışma ortamı düzenlenmiş, gerekli donanım ve yazılım ürünlerinin kuruluşları yapılmıştır. Her iki geliştirme ortamına ait gerekli yazılım araçları, sürücüleri kurulmuş ve gerekli diğer düzenlemeler yapılmıştır. PCAD istemine ait bilgi giriş ve çıkışları ve bunları gerçekleştirenler kavramsal olarak analiz edilmiş,sonuç Şekil 2'de sunulmuştıur.



Şekil 2. PCAD platformu girdi ve çıktıları

Araştırma sonucunda, PCAD platformundan bilgi isteyen kullanıcıların bu işlemi iki farklı biçimde gerçekleştireceği anlaşılmıştır. Bunlar: 1) Kullanıcı geçerli URL adresine isteklerini tanımlayarak PCAD platformuna doğrudan istekte bulunabilir veya 2) Kullanıcılar, çeşitli uygulama geliştiricilerinin geliştirmiş oldukları uygulamaları kullanarak PCAD platformundan istekte bulunabilmektedir. Bu tip kullanıcılar, “Uygulama kullanıcısı” olarak adlandırılmıştır.

Projenin ikinci amacı olarak tespit edilen, elde edilen sonuçların araştırmacılar ve sektör ile paylaşılması konusunda yapılan çalışmalar detaylı olarak sonuç bölümünde sunulmaktadır. Bu amacın gerçekleştirilmesi için projenin gelişimine koşut olarak yayınların da gerçekleşmesi hedeflenmiştir. Böylelikle, hem akademiden hem de endüstriden geri

bildirimler elde edilmiş ve bu bilgilenme sayesinde proje süreçleri gözden geçirilerek tekrardan düzenlenmiştir. Proje kapsamında SCI indeksli bir adet makale, C-SCI indeksli bir adet konferans bildirisi ve bir adet de ulusal konferans makalesi yayınlanmıştır. Bir makalede ulusal bir dergiye yollanmıştır. Projede yer alan iki yüksek lisans öğrencisi çalıştıkları proje konuları hakkında seminer düzenlemişlerdir. Bunlara ilaveten, proje kapsamında web sitesi geliştirilmiş ve iki firma ile de görüşmeler yapılmış, proje çıktıları paylaşılmıştır.

Projenin üçüncü amacı olarak tespit edilen proje çalışanlarının akademik ve araştırma becerilerini geliştirmesi konusunda ise, projenin gerçekleştirme aşamalarında elde edilen teorik ve pratik bilgiler olumlu katkı sağlamıştır. Proje yöneticisi ve diğer ekip üyeleri cesurca yeni geliştirme araçlarının ve uygulamaların üzerine gitmişlerdir, bu konudaki öğrenme, uygulama ve yenilik fırsatlarını değerlendirmişlerdir. Bu da proje ekibinin güncel geliştirme teknolojilerini takip etmesi sağlamıştır. Böylelikle tüm proje ekibinin akademik kariyeri olumlu etkilenmiş, araştırma beceri ve pratiklerinin gelişmesi sağlanmış, vermekte oldukları derslerde belirli bir olgunluk sağlamıştır. İki yüksek lisans öğrencisi, projede kapsamındaki konularda tezlerini hale sürdürmekte olup ve yazım aşamasına gelmiş bulunmaktadırlar. Bu tezlerin sonuçlanmasını takiben başka nitelikli yayınların daha yapılması beklenmektedir.

Yukarıda sunulan çalışmalar, proje kapsamında satın alınan sunucu ve dizüstü bilgisayarlar üzerinde, TÜBİTAK ve İzmir Ekonomi Üniversitesi tarafından sağlanan insan kaynağı kullanılarak, İzmir Ekonomi Üniversitesi'nin projeye özel olarak tahsis ettiği araştırma laboratuvarında ve üniversitenin diğer olanakları kullanılarak yapılmıştır.

4. BULGULAR ve YAZILIM ÇIKTILARI

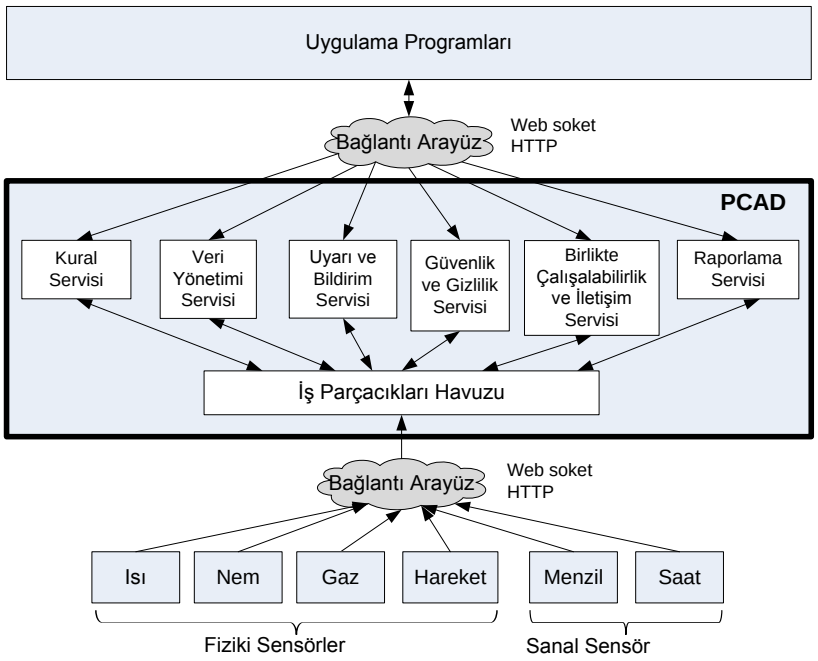
Bu bölümde proje süresince geliştirilen platform yazılımının mimarisi, tasarımı ve gerçekleştirme detayları verilmektedir. Ortak bir tasarım ve mimariye sadık kalınarak platformun iki farklı sürümü değişik yazılım iskeletleri ve programlama dilleri kullanılarak gerçekleştirilmiştir. Bölüm 4.1'de mimariyi oluşturan servislerin tasarımı verilmiştir. Bölüm 4.2 ve 4.3'de platformun AKKA ve Node.js iskeletleri kullanılarak yapılan gerçekleştirmesi anlatılmıştır. Yazılım süreçlerinin önemli bir parçası olan sınama ile ilgili sonuçlar takip eden Bölüm 5'de verilmiştir. Yazılımın nasıl kurulacağını ve kullanılacağını anlatan kullanım kılavuzu kendine ait bir bölümde sunulmuştur. Kullanıcı kılavuzunu kapsayan Bölüm 6'nın platformun ileriki sürümlerinde ayrı bir doküman haline getirilmesi planlanmaktadır. Bundan dolayı kılavuz 6. Bölümde sunulmaktadır.

4.1 PCAD Servislerinin Tasarımı

PCAD platformu işletim sistemlerinden esinlenerek tasarlanmış, bunun ışığında bir işletim sisteminde bulunan yönetim modüllerine benzer bir yapı, servis kavramı altında platformun içerisinde yer almıştır. PCAD platformunun genel mimari yapısı Şekil 3'de gösterilmiştir. Platforma veri sağlayan sensörler ile platformun sunduğu verileri kullanan uygulamalar platform ile bağlantılarını bir ara katman aracılığı ile yaparlar. Bu sayede platformda oluşacak değişiklikler ve eklentilerden en az şekilde etkilenirler. Platformun uygulamalar ve sensörler arasındaki bağlantısı websoket (*İngilizce*: *web socket*) ve HTTP protokolleri aracılığı ile gerçekleştirilmiştir. PCAD platformunda aşağıda verilen gerçekleştirilmesi yapılmış beş servis yer almaktadır.

1. Kural Servisi (KS)
2. Veri Yönetim Sistemi (VYS)
3. Uyarı ve Bildirim Servisi (UBS)
4. Güvenlik ve Gizlilik Servisi (GGS)
5. Birlikte Çalışabilirlik ve İletişim Servisi (BÇS)

Raporlama Servisinin, PCAD platformunun fonksiyonel bir parçası olmasına rağmen, platformun proje bahsedilen genel amacına sağladığı katkının görece az olması ve benzer çalışmalarda genellikle üçüncü parti yazılım araçları ile çözülüyor olması gerekçesiyle kavramsal olarak yer almış ancak geliştirilmemiştir. Takip eden bölümde gerçekleştirilen beş servisin fonksiyonları hakkında detaylı bilgi verilmektedir. Gerçekleştirme ile ilgili detaylar Bölüm 4.2 ve 4.3 'de verilmiştir.



Şekil 3. PCAD platformu genel mimari yapısı

4.1.1 Kural Servisi

Verilerin uygulamalar tarafından belirtilen kurallar doğrultusunda işlenerek uygulamaya sunulmasına ve örneğin farklı sensörlerden gelen verileri birleştirmek gibi, uygulama isteklerine cevap vermek kural servisinin görevidir. Kural servisi bu görevi uygulama tarafından gönderilen kural dosyaları, ya da kullanıcının platforma yolladığı JSON formatında yazılmış isteğin içinde bulunan filtreleme kuralları aracılığı ile yerine getirir.

4.1.2 Veri Yönetim Servisi

Veri Yönetim Servisi (VYS) sisteme gelen verilerin saklanması ve bu verilerin sistem içerisindeki servislere dağıtılmasını sağlamakla görevlidir. Veri tabanı için iki alternatif üzerinde durulmuştur. Bunlar ilişkisel (SQL) ve doküman (noSQL) veri tabanlarıdır.

a) İlişkisel veri tabanlarında veriler tablolar içinde satırlar ve sütunlar halinde tutulmaktadır. SQL standart dili yardımı ile veri tabanı yönetimi sağlanmaktadır. İlişkisel veri tabanında yeni bir alan eklenmek istendiğinde tabloyu baştan ele alıp yeni bir sütun eklenmesi gerekmektedir.

b) NoSQL veri tabanları verileri ilişkisel tutmak yerine anahtar-değer çiftleri şeklinde tuttukları için ilişkisel veri tabanlarına nazaran daha hızlı sorgulama, güncelleme,

ekleme ve silme işlemi sağlamaktadır. Tablo ve sütün kavramı olmadığı için yeni bir alana ihtiyaç duyulduğunda bunu doğrudan bir anahtar-değer ikilisi ekleyerek yapılması mümkün olmaktadır. Bu da hız ve performans açısından önemli kazanımlar sağlanması anlamına gelmektedir. PCAD platformunda sensörlerden gelen veriler çok hızlı ve çok sayıda geleceği için yazma performansı üst seviyelerde olan NoSQL veri tabanı kullanımı iyi bir alternatif olarak gözükmektedir. Tutulan veri yüksek seviyede ilişkisel olmadığı ve bir işlem hareketine (*İngilizce*: *transaction*) ihtiyaç olmadığı için performans ve hız bakımından ilişkisel veri tabanlarına göre sağladığı avantajlar göz önünde bulundurulduğunda NoSQL veri tabanı SQL'e göre tercih edilebilir (Nayak vd., 2013, Li vd., 2013). PCAD, uygulamalar ile olan iletişiminde yap ve unut (*İngilizce*: *fire-and-forget*) prensibine göre çalışmaktadır. Uygulama bir sensör için istekte bulunmakta, veriyi aldığında işlem tamamlanmaktadır.

Platformun AKKA ile yapılan gerçekleştirilmesinde ilişkisel (SQL) veri tabanı tipi kullanılmıştır. İlişkisel veri tabanı için ise MySQL'den faydalanılmıştır. Node.js ve Broker tabanlı gerçekleştirmede ise NoSQL kullanılmıştır.

4.1.3 Uyarı ve Bildirim Servisi

Uyarı ve Bildirim Servisinin görevi, kullanıcılar ve uygulamalar tarafından belirtilen kıstaslara göre yapılan isteklere yanıt vermektir. Buna örnek olarak kullanıcıya sensör verilerinin sadece belli bir eşik aralığında olduğunda gönderilmesi verilebilir. Uyarı ve Bildirim Servisi, kuralların çalıştırılmasından sorumlu olan Kural Servisi ile işbirliği içinde çalışır. Uygulamalar herhangi bir kural belirtmeyerek, verilerin herhangi bir filtrelemeye tabi tutulmadan gönderilmesini de isteyebilir. Uygulamalar PCAD platformu ile bağlantılı ya da bağlantısız iletişim metotlarından birini kullanarak etkileşime geçmektedirler. Bağlantılı iletişimi kullanmadaki temel amaç verilerin uygulamaya gönderilmesinde bir süreklilik sağlamaktır. Bu bağlantı uygulama tarafından yıkılana kadar aktif şekilde kalmaktadır. AKKA ile yapılan gerçekleştirmede platform ile uygulamalar bağlantılı iletişimi Web soket bağlantısı açarak sağlanmaktadır. Node.js gerçekleştirmesinde bu bağlantıyı sağlamakta simsar (*İngilizce*: *broker*) modeli kullanılmıştır. Bağlantısız iletişim metodu basit bir veri alış veriş olanağı sunmaktadır. Hem Node.js hem de AKKA gerçekleştirilmesinde http istek/yanıt mekanizmasını kullanılmış, uygulama http protokolünü kullanarak bir kereye mahsus bir istek göndermekte ve yanıtını beklemektedir.

Bağlantılı İletişim Metodu

Bağlantılı iletişim metodu iki farklı yol kullanarak çalıştırılmaktadır. Birinci yolda sensör verileri platform tarafından alınır alınmaz hemen istemciye iletilirken aynı zamanda da paralel

olarak veriler platform içinde gerekli servisler tarafından işlenmeye başlanır ve en son olarak veri tabanına yazılır. Sensörden veri gelmediği zaman istekte bulunan istemci/kullanıcı bloke olmakta ve yeni verinin gelmesini beklemektedir. Bu sayede servis uygulamalara esnek gerçek zamanlı veri sağlama gereksinimini yerine getirir.

İkinci yolda veri direkt olarak veri tabanından okunmakta ve sabit aralıklara istemciye yollanmaktadır. Kullanılan bu yolda istemcinin belirlenen zaman aralığının sonunda veri alması garanti edilmektedir. İstemciye her zaman veri tabanına yazılmış en son veri yollanmaktadır. Eğer sorgulanan sensör verilen aralıkta platforma herhangi yeni bir veri sağlamamış ise, istemciye bir önceki, yani güncel olmayan veri yollanmaktadır. Verinin güncel olup olmadığını verideki zaman mührüne bakarak kontrol etmek istemcinin görevidir.

Bağlantsız İletişim Metodu

Bağlantısız iletişim metodu basit bir veri alış veriş olanağı sunmaktadır. Uygulama http istek/yanıt mekanizmasını kullanarak bir kereye mahsus bir istek göndermekte ve yanıtını beklemektedir. İstenilen veri, veri tabanından sağlandıktan sonra veri alış verişi bitmekte ve http oturumu sonlandırılmaktadır. Veri tabanı yerine, doğrudan sensörden bir kereye mahsus veri almak istediğinde bağlantılı iletişim metodu kullanılmalıdır. Bu bağlantı kurulup ilk veri geldikten sonra bağlantının hemen kapatılması suretiyle bu gerçekleştirilir.

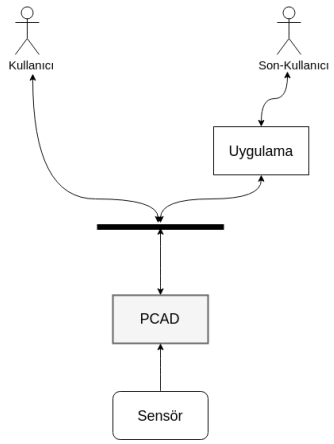
Uygulama, PCAD ile altı değişik etkileşim mekanizmasından birini kullanarak veri alış verişinde bulunur:

A. Uygulama ile PCAD arasında bağlantı tabanlı (*İngilizce: connection-oriented*) sürekli veri alış verişi. Bu veri alışverişi dört farklı şekilde gerçekleştirilmektedir.
    1. **(A1)** filtre belirtmeden sensörden doğrudan veri istemi (gerçek zamanlı)
    2. **(A2)** filtre belirterek sensörden doğrudan veri isteminde bulunur (gerçek zamanlı)
    3. **(A3)** filtre belirtmeden veri tabanından veri istemi (periyodik)
    4. **(A4)** filtre belirterek veri tabanından veri istemi (periyodik)
B. Uygulama ile PCAD arasında tek seferlik veri alış verişi
    1. **(B1)** herhangi bir filtre belirtmeden bir kereye mahsus veri tabanından veri istemi
    2. **(B2)** bir filtre belirterek bir kereye mahsus veri tabanından veri istemi

4.1.4 Güvenlik ve Gizlilik Servisi

Sensörlerden elde edilen ortam hakkındaki verilerin çeşitli uygulamalar ile paylaşılacak olması, bu verilere bir yetkilendirme sonucunda ulaşılmasını zorunlu kılmaktadır. Güvenlik ve Gizlilik Servisinin amacı kullanıcıların ve uygulamaların yetkilerini belirlemek, sistem üzerindeki verilere erişimi sınırlamak ve kontrol etmektir. PCAD platformunun

gerçekleştirilmesinde, Uygulama, Sensör ve Kullanıcı olarak üç kategoride sınıflandırılan istemciler (Şekil 4), hangi işlemleri yapıp yapamayacakları konusunda yetkilendirilmiştir. Kullanıcılar sistem üzerinde uygulama ve sensör tanımlarken, sensörler sadece sistem için veri sağlarlar. Uygulamalar ise sensörlerin ürettiği verilere ulaşıp, kendi amaçları doğrultusunda kullanırlar.



Şekil 4. PCAD faydalanıcıları

Kullanıcıların platforma erişimi iki yolla olmaktadır:

- Platforma doğrudan bağlanmak (Kullanıcı).
- Platforma bir uygulama aracılığı ile bağlanmak (Uygulama Kullanıcısı).

Uygulamaları kullanan kullanıcıların yetkileri, uygulamanın kullanıcılar için belirlediği yetkilerle sınırlıdır. Uygulama kullanıcılarının sisteme ulaşımı bir uygulama üzerinden olduğu için ilk aşamada uygulamaya verilen haklar kullanılacak, bu haklar gerekirse uygulama geliştiricisi tarafından sınırlanacaktır. Uygulamalar kendi veri tabanlarını yaratarak kullanıcı profilleri oluşturmak suretiyle erişimi bu şekilde kısıtlama olanağına sahiptirler, böylece uygulamadan faydalananların istenmeyen haklara sahip olması engellenmiş olmaktadır.

Denetleme işlemi token'lar kullanılarak yürütülür. Geçerli bir erişim token'a sahip olan istemciler PCAD servislerini kullanırlar. Erişim token'larına sahip olacak istemciler; kullanıcılar, uygulamalar ve sensörlerdir. Kullanıcılar kendilerini sisteme kullanıcı ismi ve şifreyle tanıttıktan sonra erişim token'i elde ederler. Uygulamalar erişim tokenlarını uygulama geliştiricisi aracılığı ile platforma kayıt esnasında alırlar (uygulama geliştiricisi de bir kullanıcı olmaktadır). Benzer şekilde Sensörler ve sensör yazılımları da platforma kayıt esnasında bir erişim token'i alırlar. Uygulamalar ve Sensörler daha sonra yapacakları platform iletişimlerinde ilk kayıt esnasında kendilerine verilen erişim token'ını kullanarak kimliklerini doğrulatırlar. Bunun sonucunda süreli yeni bir *oturum* token'ı elde ederler. Oturum token'ı

platforma yollanan her isteğin içine konulmak zorundadır. Anlatılan bu işlemler Şekil 5'de sunulmaktadır.

Şekil 5. Kimlik doğrulama ve yetkilendirme

Erişim tokenları JSON Web Token (JWT) formatında gerçekleştirilmiştir. Uygulamaların ve kullanıcıların sensörlere erişimi çok temel ve basitleştirilmiş bir Rol Tabanlı Erişim Denetimi - RTED (*Role Based Access Control - RBAC*) (Ferraiolo vd., 1995) modeli üzerinden tasarlanmıştır. Bu model sensör verilerinin hangi uygulama için erişilebilir olduğuna rehberlik edecek, roller aracılığı ile kaynağa erişim yetkileri ve çeşitleri ile ilgili çeşitli kısıtlamalar tanımlanabilecektir. Bu model Zorunlu Erişim Denetimi (*İngilizce: Mandatory Access Control)* mekanizması ile yakınlık göstermektedir. Bu noktada sensör verileri için bir gizlilik düzeyi belirlenirken, uygulamalarda hangi düzeydeki bilgilere ulaşabileceğini belirten bir erişim düzeyi atanır. Roller de platforma dayalı politikalar ile tanımlanacaktır. Örneğin hava durumu uygulamasının, ulaşım verisine erişmesine gerek yok iken, bunun tam tersi olarak, ulaşım uygulamasının hava durumu bilgisine ulaşmasına izin verilecektir.

Her bir kullanıcı veya uygulamanın sensör erişiminde kullandığı bir rolü vardır ve bu rol *durum*, *bilgi* ve *veri* adı verilen alt yetkilerden oluşmaktadır. Bu üç yetkinin birleşimi rolün tam yetkisini belirlemektedir. Bu yetkilerin tanımı aşağıda verilmiştir.

- **Bilgi**: Sensör meta bilgisi sorgulama hakkı (yeri, tipi, üretici ve diğer özellikler)
- **Veri:** Sensörden veri alma hakkı (sensör verileri)
- **Durum:** Sensör durum bilgisini sorgulama (aktif-pasif olup olmadığı).

Güvenlik ve gizlilik servisin gerçekleştirilmesi ile ilgili detaylar Kısım 4.2.4 ve 4.3.4'de sunulmuştur.

4.1.5 Birlikte Çalışabilirlik ve İletişim Servisi

Bu servis PCAD platformunun, PCAD benzeri platformlarla iletişime geçip veri alışverişinde bulunmasını sağlamak için tasarlanmıştır. Kullanıcılar PCAD platformundan HTTP İstek/Yanıt mekanizması kullanarak sensör verisi alabilmektedirler. Kullanıcıların platforma yapılan istekler ve gönderilen cevaplar JSON formatındadır. JSON veri formatları Kısım 4.2.6 ve 4.3.6'da detaylı şekilde anlatılmıştır. HTTP İstek/Yanıt çok yaygın bir protokol olduğu için diğer platformların da PCAD platformundan veri isteğinde bulunması benzer şekilde mümkün olmaktadır. PCAD'dan veri isteyen diğer platformlar kendilerini sisteme kayıt ettirdikten sonra bir kullanıcı gibi HTTP İstek/Yanıt metodunu ya da kütüphane API'lerini kullanarak veri almaktadır.

PCAD'in diğer platformlardan veri alabilmesi için ilgili platforma ait arayüzün bilinmesi gerekmektedir. Bunun için tasarım şablonlarında bulunan adaptör yapısına (Gamma, vd. 1994) benzer eklenebilecek modül sistemi düşünülmüştür. Fakat bu fonksiyon platformun bu sürümünde yer almamaktadır.

4.2 PCAD Platformunun Akka Uygulama İskeleti Kullanılarak Gerçekleştirilmesi

Bu bölümde PCAD platformuna ait servislerin Akka araçları ve Scala programlama dili kullanılarak gerçekleştirilmesi sürecindeki işlemler sunulmaktadır. Akka uygulama iskeleti, Aktör modelinin bir uygulamasıdır. Aktör modeli, koşutzamanlı (*İngilizce: concurrent*) ve dağıtık sistemlerin ihtiyaçlarını karşılamak için geliştirilmiş bir modeldirAkka uygulama iskeletinin gerçekleştirilmesinde kullanılan yazılım araçları ve bu araçların PCAD servisleri ile eşleşmesi Tablo 2'de sunulmaktadır.

Tablo 2. PCAD platformunun geliştirilmesinde kullanılan yazılım araçları

Geliştirme aracı	İşlevi	PCAD Servisi
Scala	Programlama dili	Tüm servisler
Play Framework	Web Framework	Platformun kendisi
MySQL	Veri tabanı	Veri Yönetim Servisi
Akka	Servis Modelleme	Tüm servisler
StandardTokenParser Kütüphanesi	Ayrıştırıcı ve yorumlayıcı	Kural Servisi
Websocket	İletişim protokolü	Uyarı ve Bildirim Servisi
Slick	VT giriş, sorgu kütüphanesi	Veri Yönetim Servisi

4.2.1 Kural Servisi

Verilerin uygulamalar tarafından belirtilen kurallar doğrultusunda işlenerek uygulamaya sunulmasına ve örneğin farklı sensörlerden gelen verileri birleştirmek gibi, uygulama isteklerine cevap vermek kural servisinin görevidir. Kural servisi bu görevi uygulama tarafından gönderilen kural dosyaları, ya da kullanıcının platforma yolladığı JSON formatında yazılmış isteğin içinde bulunan filtreleme kuralları aracılığı ile yerine getirir. Kuralların tanımlanması amacı doğrultusunda projede küçük bir kural dili geliştirilmiştir. Bu dili kullanarak oluşturulan kurallar kural dosyası içinde yer alır. PCAD kütüphanesi tarafından önceden tanımlanmış olan fonksiyonlar kuralları desteklemektedir. Kullanıcıdan gelen kural dosyasının çalıştırılması için PCAD içerisinde çalışan bir Kural Motoru tasarlanmıştır. İlk önce kullanıcı isteği bir dönüştürücü aracılığı ile kural motoruna gönderilmekte, sonrasında bu motorun içerdiği küçük bir dil ile gelen kural dosyası ayrıştırılmakta (*İngilizce: parse*), son olarak da bu dosya yorumlanmaktadır (*İngilizce*: *interpretation*).

Kural Dosyası

Kural dosyaları uygulamalar tarafından gönderilen ve belirli şartlar sağlandığında, platformun şartlara uygun eylemlerde bulunmasını tanımlayan özel dosyalardır. Dosya içinde yer alan kuralların nasıl yazılması gerektiğini belirten BNF gösterimi aşağıda sunulmaktadır.

```
<Rule>          → <RuleConf> <RuleAttr> <RuleBody>
<RuleConf>      → name = <ident>
<RuleAttr>      → <VariableDef>
                  | <VariableDef> <whitespace> <RuleAttr>
<RuleBody>      → when { <Condition> } then { <Action> }
<CompositePredicate> →<Condition> | <Predicates>
<Predicates>    → <Predicate>
                  | <Predicate> <Logical Operator> <Predicate>
<Condition>     → <Expression>
                  | <Expression> <Comparator> <Expression>
<Action>        → <Function>
                  | <Function> <whitespace> <Action>
<Expression>    → <Variable> | number
<Comparator>    → == | != | < | > | <= | >=
<Function>      → <ident> ( <Parameters> )
<Parameters>    → <Parameter>
                  | <Parameter> <whitespace> <Parameters>
<Parameter>     → <VariableDef> | <Variable>
<Variable>      → <ident>
<VariableDef>   → <ident> = <Definition>
<Definition>    → <Function> | <ident> | number
<Logical Operator>→ and | or
<ident>         → string
<whitespace>    → whitespace
```

Bu dilde yazılmış örnek bir kural dosyası aşağıdaki verilmiştir.

```
name = rule

threshold = 15
sensorValue = getSensorValue(id)

when {
   sensorValue > threshold
}
then {
   notify()
}
```

Kural dosyalarının ayrıştırılması ve yorumlanması Kural Servisinin görevleri arasındadır. Ayrıştırma işlemi, tanımlanan BNF temellerine göre yapılır. Ayrıştırma işleminin sonucunda dosya içinde belirtilen kuralların doğru yazılıp yazılmadığı belirlenir. Yorumlama işlemi ise kuralların çalıştırılmasına verilen addır. Kurallar, kural servisi tarafından çalıştırılıp sonuçlandırılmaktadır.

Kural dosyası iki ana parçadan oluşmaktadır: başlangıç (*İngilizce*: *preamble*) ve gövde (*İngilizce*: *body*). Başlangıç kısmında kural dosyasının adı belirtilmekte ve değişken tanımlamaları yapılmaktadır. Bu kısımda değişkenlere eşik değerleri, veri tabanı değerleri veya gerçek zamanlı gelen değerlerin ataması yapılmaktadır. Değişkenlere kütüphane fonksiyonları tarafından hesaplanan değer atamasıda yapılabilmektedir. Bu değişkenlerin değerleri hazır oldukları anda kural çalıştırılır.

Takip eden gövde kısmı **'when'** ve **'then'** bloklarını kapsayan işlenecek kuralı tarif eden mantıksal ifade ve aksiyon bölümlerinden oluşur. **'when'** bloğu içerisinde yer alan mantıksal ifadelerin ve/veya gibi mantıksal operatörlerle birleştirilmesi yapılır. Bu şekilde oluşturulan ifade **'true'** veya **'false'** sonucunu verir ve takip eden **'then'** bloğunun çalışması bu sonuca göre belirlenir. Bildirim işlemi, fonksiyonların çağrılması gibi işler bu blok kullanılarak gerçekleştirilir. Şekil 6'da bir uygulama tarafından gönderilen kural dosyasının içeriği sunulmaktadır. Bu örnekte 2 ve 7 numaralı sensörlerden gelen değer 14.4'ün üzerinde ise istemciye *Notify()* fonksiyonu kullanarak bildirim yollanmaktadır.

```
threshold = 14.4
primary_value = get_value(sensor=2)
secondary_value = get_rt_value(sensor=7)
when {
  (primary_value > threshold) and (secondary_value >
threshold)
} then {
  Notify()
}
```

Şekil 6. Kural örneği

Kurallar içerisinde kullanılmak üzere PCAD Kütüphanesi içerisinde fonksiyonlar tanımlanmıştır. Bu fonksiyonlar hakkında bilgi aşağıdaki Tablo 3'de verilmiştir.

Kural dosyasının söz dizin analizi için Scala için yazılmış olan *StandardTokenParser* kütüphanesi kullanılmaktadır. Bu kütüphane ayrıştırma işlemini kolaylaştırmak için Scala dilinin örüntü eşleme özelliğinden faydalanmaktadır. Kural dosyası öncelikle ayrıştırma işlemine tabi tutulmakta, geçerli bir formda oluşturulduysa, bir kural objesine dönüştürülmekte ve çalıştırma işlemi için hazır hale gelmektedir. Kural objesi, kural dosyası ve kurala ait meta veriden oluşmaktadır (örneğin kuralı yollayan uygulama ile ilgili veriler). Kuralların çalıştırılması için, yorumlama işlemi koşmaktadır. Kural dosyası yorumlayıcı tarafından adım adım çalıştırılmaktadır. Bunun için kural doyası içindeki komutlar kural ismi, değişken atamaları, mantıksal ifadeler ve aksiyonlar olmak üzere dört farklı parçaya ayrılmaktadır. Her bir parça ardı sıra çalıştırılmaktadır. İlk adımda, kural ismi alınmakta, tanınma işlemi gerçekleştirilmektedir. İkinci adımda, değişkenlerin atamaları gerçekleştirilmektedir. Üçüncü adımda ise mantıksal ifadenin sonucu elde edilmekte ve sonuca göre de, son kısımdaki işlemler gerçekleştirilmektedir. Üçüncü ve dördüncü adımların çalıştırılması kural işleyici, tarafından yapılmaktadır.

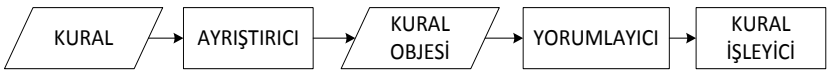
Tablo 3. Kurallar içerisindeki fonksiyonlar

İsim	**Parametre**	**İşlem**	**Örnek**
get_value	Sensör numarası	Varsayılan sensörden* verinin ölçülen değerini getirir.	get_value(sensor=1)
get_time	-	Varsayılan sensörden* verinin zaman olarak saatini getirir.	get_time()
get_rt_value	Sensör numarası	Tanımlı sensör numarasını kullanarak gerçek zamanlı	get_rt_value(sensor=1)

		verinin ölçülen değerini getirir.	
get_rt_time	Sensör numarası	Tanımlı sensör numarasını kullanarak gerçek zamanlı verinin zaman olarak saatini getirir.	get_rt_time(sensor=1)
get_db_value	Sensör numarası	Tanımlı sensör numarasını kullanarak veri tabanındaki en son verinin ölçülen değerini getirir.	get_db_value(sensor=1)
get_db_time	Sensör numarası	Tanımlı sensör numarasını kullanarak veri tabanındaki en son verinin zaman olarak saatini getirir.	get_db_time(sensor=1)

* Varsayılan sensör: İstek içerisinde tanımlı olan başlıca sensördür.

Şekil 7 uygulamadan alınan kural dosyasının çalıştırılmasında kullanılan safhalar gösterilmektedir. PCAD kural servisinin kullanımına alternatif olarak, uygulamalar kendi durum süreçlerini yürütmeyi seçip, PCAD'ın kural servisini devre dışı bırakabilirler.

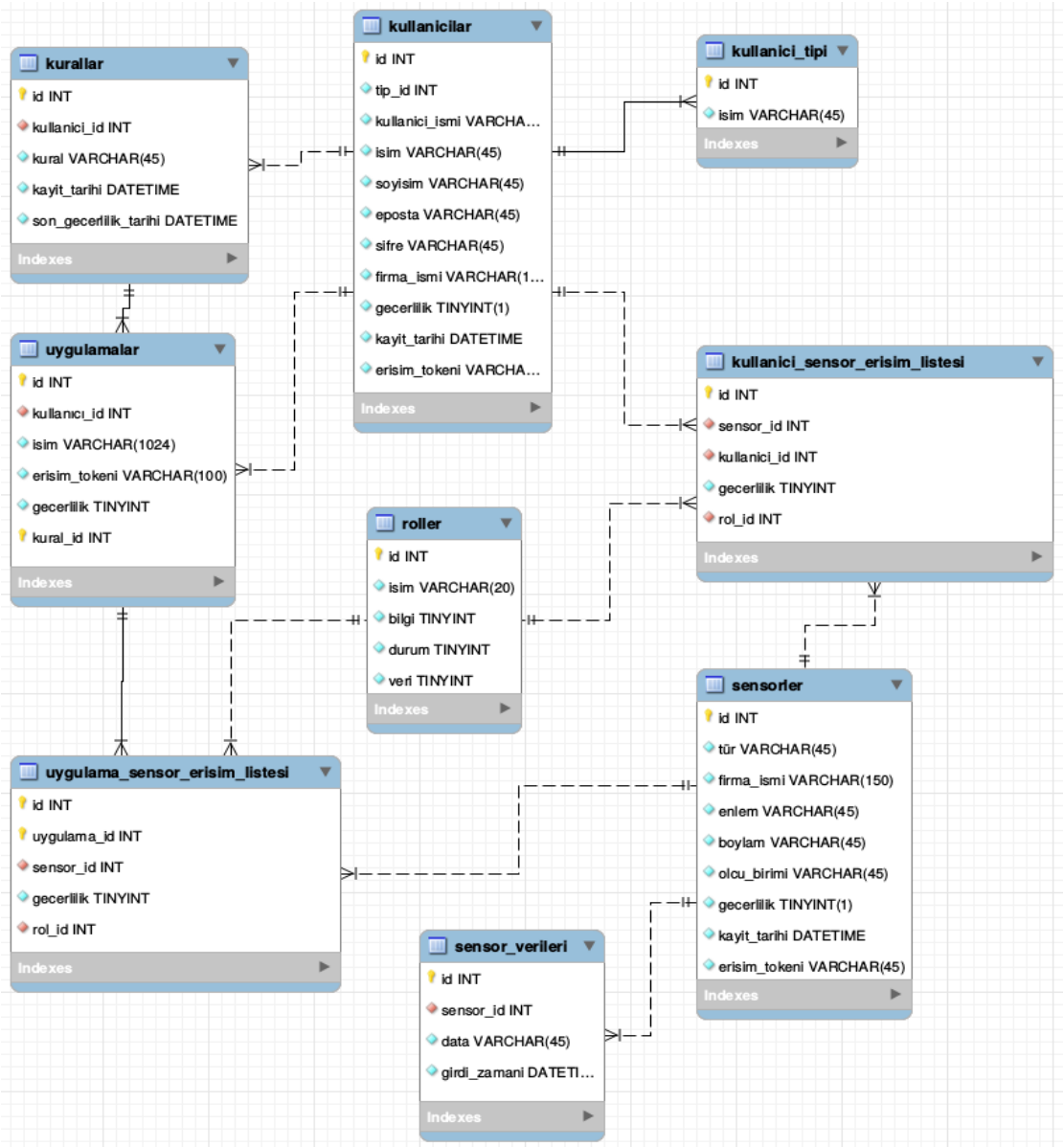


Şekil 7. Kural işleme akış şeması

4.2.2 Veri Yönetim Servisi

Platformun AKKA ile yapılan gerçekleştirilmesinde ilişkisel (SQL) veri tabanı tipi kullanılmıştır. İlişkisel veri tabanı için ise MySQL'den faydalanılmıştır. Veri tabanlarına yapılan isteklerin bloke etmeyecek şekilde çalışması için Slick kütüphanesi kullanılmıştır. Slick, Scala için modern bir veri tabanı sorgu ve erişim kütüphanesidir. Depolanan veriye Scala komutlarını kullanıyormuş gibi erişilmesini sağlamaktadır. Diğer bir deyişle, sorguların SQL yerine Scala ile yazılmasını sağlamaktadır. Aynı zamanda veri tabanı erişiminin ne zaman gerçekleştiğine ve hangi verinin transfer edildiğine dair tam kontrol imkânı vermekte, böylece statik kontrol ve derleme güvenliği sağlanmaktadır. Slick, farklı arka veri tabanı tipleri için (MySQL, PostgreSQL) için kod üreten genişletilebilir bir sorgu derleyicisine sahiptir.

Platformun servislerin kullanacağı MySQL ilişkisel veri tabanı tabloları Şekil 8'de sunulmaktadır.



Şekil 8. SQL ilişkisel veri tabanı tabloları

TABLO kullanicilar: Temelde iki tip PCAD kullanıcısı olacaktır. Bunlar uygulama geliştiricisi ve kullanıcı tipleridir. İlki PCAD altyapısını kullanan uygulamalar geliştirecektir. İkincisi platformdan direk ulaşım yaparak sensörler ile ilgili bilgi alabileceklerdir. İkinci grupta yer alanların derinlemesine programlama bilgisine sahip olmaları gerekmemekte, sadece platforma yollanacak istek URL'nin yükünde yer alacak formatı bilmeleri yeterli olmaktadır. Kullanıcı bilgileri bu tabloda yer alacaktır. Tablodaki **gecerlilik** sahası kullanıcının sistem üstünde yönetici tarafından aktif/pasif olduğu bilgisinin tutulması için kullanılacaktır. Ayrıca **erisim_tokeni** kolonu da güvenlik anlamında yetkilendirme için kullanılacaktır.

TABLO kullanici_tipi: Kullanıcı tipleri bu sahada yer alacaktır. Örneğin, geliştirici, yönetici vb.

TABLO uygulamalar: Kullanıcılar kendi uygulamaları için PCAD'den veri alabilmek adına uygulamalarını PCAD'e kaydettirmek zorundadırlar. Her kullanıcının her uygulaması için benzersiz bir **erisim_tokeni** üretilerek kullanıcıya verilecektir. Uygulamalar aynı zamanda kurallara sahip olabilmektedir. Bu **kural_id** kolonunda saklanmaktadır.

TABLO kurallar: Kullanıcılar sisteme kural kaydetme imkânına sahiptirler. Kurallar son geçerlilik ve kayıt tarihi ile birlikte bu tabloda saklanmaktadır. Kurallar tablosunda dosyasının içeriği ASCII formatında tutulacaktır. Bu dosyanın büyüklüğü 1024 byte ile sınırlı olacaktır.

TABLO kullanici_sensor_erisim_listesi: Kullanıcıların hangi sensörün hangi özelliğine (konum, durum vb.) ve verisine erişim izninin belirlenmesi için tasarlanan bu tabloda sensör ve kullanıcılar **sensor_id** ve **kullanici_id** sahaları ile eşleşeceklerdir. Erişim haklarını belirleyen **rol_id** roller tablosuna referans vererek, bu tablodan erişim haklarına bakılabilmektedir.

TABLO uygulama_sensor_erisim_listesi: Kullanıcı uygulamalarının hangi sensörün hangi özelliğine (konum, durum vb.) ve verisine erişim izninin belirlenmesi için tasarlanan bu tabloda sensör ve uygulamalar **sensor_id** ve **uygulama_id** sahaları ile eşleşeceklerdir. Erişim haklarını belirleyen **rol_id** roller tablosuna referans vererek, bu tablodan erişim haklarına bakılabilmektedir. Bu ve **kullanici_sensor_erisim_listesi** tablolarının kullanımı hakkındaki geniş bilgi Güvenlik ve Gizlilik Servisleri kısmında verilmiştir.

TABLO roller: Uygulamaların veya kullanıcıların sensörler üzerinde hangi izinlere sahip olduğu bu tabloda saklanır. Bu izinler sensör meta ***Bilgi***si, ***Durum*** ve ***Veri*** olarak sınıflandırılmaktadır. Güvenlik ve Gizlilik Servisleri kısmında söz konusu sınıflandırma detaylandırılmıştır.

TABLO sensorler: Bu tabloda sensörlerin bilgileri (konum, ilgili şirket bilgisi vb.) tutulacaktır. Herhangi bir firma yeni bir sensör eklemek istediğinde bu tabloda yeni bir satır yer alacaktır.

TABLO sensor_verileri: Sensörlerin verileri bu tabloda saklanacaktır. Sensörlerin yazıldığı ham verinin yorumlanması uygulamalara ait olacaktır. Verinin yazıldığı zaman ait zaman mührü de bu tabloda tutulmaktadır. Bu tablo sistemin ölçeklenebilmesi açısından en kritik tablodur.

Veri tabloların tasarımında Boyce-Code-Normal-Form (BCNF) kuralları göz önünde bulundurulduğu için **uygulama_sensor_erisim_listesi** ve **kullanici_sensor_erisim_listesi** tabloları birleştirilmemiştir.

4.2.3 Uyarı ve Bildirim Servisi

Kısım 4.1.3 de anlatıldığı üzere Uyarı ve Bildirim Servisinin görevi, kullanıcılar ve uygulamalar tarafından belirtilen kıstaslara göre yapılan isteklere yanıt vermektir. Filtreleme kıstasları kullanıcıdan alındıktan sonra uygulama tarafından bir kural dosyasına dönüştürülür ve platforma yollanır. Uyarı ve Bildirim Servisi kural dosyasını, kuralların çalıştırılmasından sorumlu olan Kural Servisi ile işbirliği içinde çalıştırır. Ayrıca uygulamalar herhangi bir kural belirtmeyerek, verilerin herhangi bir filtrelemeye tabi tutulmadan gönderilmesini de isteyebilir. Uygulamalar PCAD platformu ile bağlantılı ya da bağlantısız iletişim metotlarından birini kullanarak etkileşime geçerler. Bağlantılı iletişim, platform ile bir Web soket bağlantısı açılarak sağlanır. Bu bağlantı uygulama tarafından yıkılana kadar aktif şekilde kalmaktadır. Bağlantılı iletişimi kullanmadaki temel amaç verilerin uygulamaya gönderilmesinde bir süreklilik sağlamaktır. Bağlantısız iletişim metodu basit bir veri alış veriş olanağı sunmaktadır. Uygulama http istek/yanıt mekanizmasını kullanarak bir kereye mahsus bir istek gönderir ve yanıtını bekler. Uygulama, PCAD ile altı değişik etkileşim mekanizmasından birini kullanarak veri alış verişinde bulunur. Tablo 4'te bu etkileşim mekanizmalarının özellikleri sunulmaktadır.

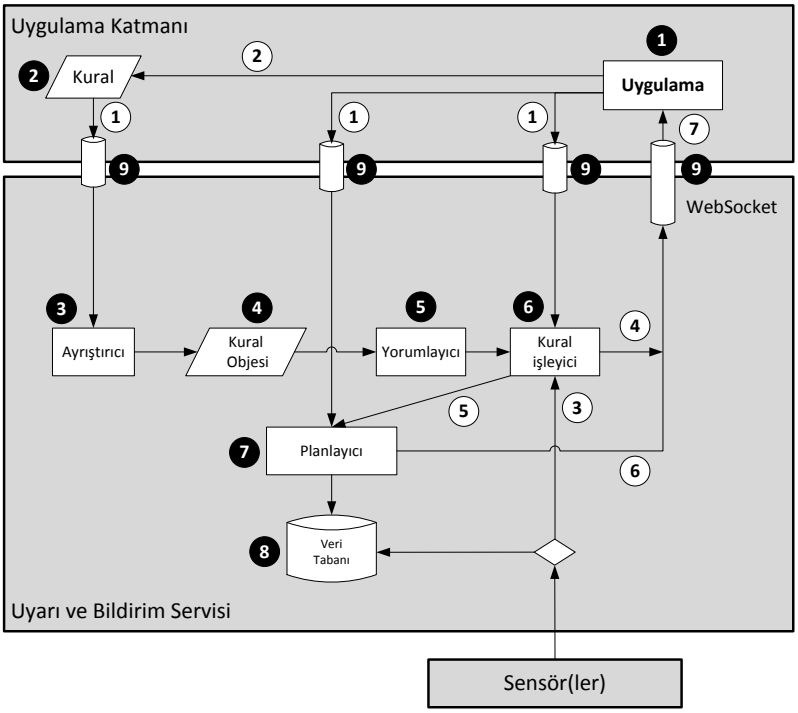
Tablo 4. Etkileşim mekanizmaları özellikleri

	Bağlantılı	**Bağlantısız**	**Filtre**	**Gerçek zamanlı**	**Periyodik**
A1	✓			✓	
A2	✓		✓	✓	
A3	✓				✓
A4	✓		✓		✓
B1		✓			
B2		✓	✓		

Şekil 9'da servisin ilk dört mekanizmasının işleyişi, Tablo 5'de söz konusu mekanizmaların sorumlulukları sunulmaktadır. Tabloda siyah daire ile gösterilen numaralar işlemleri ve dosyaları, beyaz daire ile gösterilen numaralar ise iş akışını göstermektedir.

Tablo 5. Uyarı ve Bildirim Servisi'nin işleyiş adımları

No	Açıklama
❶	**Uygulama:** Sensör verilerine ihtiyaç duyan uygulamalar.
❷	**Kural dosyası:** Kural içeren kullanıcı isteklerinin, uygulama tarafından bir kural dosyası haline getirilmesi sonucunda ortaya çıkan dosyadır.
❸	**Ayrıştırıcı:** Kural dosyası, ayrıştırıcı (*parser*) tarafından ele alınmakta ve sözdizimi ve anlam bilgisine göre işlevsel öğelerine ayrıştırmaktadır. Detayı 4.2.1 Kural Servisi bölümünde sunulmaktadır.
❹	**Kural objesi:** Ayrıştırılan dosya kural objesi haline dönüştürülmektedir.
❺	**Yorumlayıcı:** Yorumlayıcı (*interpreter*) kendisine gelen kural dosyasını çalıştırmakta ve kural işleyicisine göndermektedir.
❻	**Kural işleyicisi:** Kural işleyicisi (*rule handler*), kendisine gelen kural dosyasını ve sensör verisini alarak kuralı çalıştırmaktadır.
❼	**Planlayıcı:** Planlayıcı *(scheduler),* periyodik veri istekleri olduğunda planlayıcı zaman planına uygun şekilde iş parçasını oluşturmaktadır.
❽	**Veri tabanı:** Sensörlerden gelen verileri tutmaktadır.
❾	**Web soket:** Uygulamaya tarafından isteklerin PCAD platformuna gönderilmesini ve PCAD tarafından Uyarı ve Bildirim Servisi aracılığı ile istenilen verilerin uygulamaya geri gönderilmesini sağlamaktadır. PCAD'a yapılan istek ile açılmakta ve kurallar sağlandığı sürece veri yollamayı sürdürmektedir. Daha sonra istek karşılandığında isteği yapan çağrı tarafından kapatılmaktadır.
①	Uygulama, uygulama programındaki ara yüz fonksiyonlarını kullanarak PCAD'e web soket üzerinden istekte bulunmaktadır.
②	Uygulama, kural dosyası oluşturmaktadır.
③	PCAD platformu, sensörlerden gelen verileri Kural İşleyicisine göndermektedir.
④	PCAD platformu, sensörlerden gelen verileri kural işleyici üzerinden uygulamaya göndermektedir.
⑤	Planlayıcı, kural işleyicisinin gönderdiği kuralı almaktadır.
⑥	Planlayıcı, veri tabanından aldığı verileri kuralı dikkate alarak veya almayarak websocket aracılığı ile uygulamaya göndermektedir.
⑦	Uyarı ve Bildirim Servisi verileri uygulamaya göndermektedir.



Şekil 9. Uyarı ve bildirim servisinin bağlantı tabanlı işleyişi

Etkileşim Mekanizması A1

Etkileşim mekanizması A1'de uygulama herhangi bir filtre tanımlamadan sensörden doğrudan veri isteminde bulunmaktadır. Bu mekanizma filtrelenmemiş veri göndermek için en basit aktarma yöntemdir. Bu yöntem esnek gerçek zamanlı veri aktarımını temsil etmektedir. Veri kaynağı doğrudan sensörlerdir ve sensör verisi web soket aracılığıyla doğrudan uygulamalara gönderilmektedir (Şekil 9). Kural İşleyici bu mekanizmada boş (*null*) olarak çalışmaktadır. Etkileşim Mekanizması A1'in işleyişi sırasında {❶, ①, ❾,❻, ③, ④, ❾, ⑦, ❶} nolu adımlar takip edilmektedir.

Etkileşim Mekanizması A2

Etkileşim mekanizması A2'de uygulama, bir kural dosyası belirterek sensörden doğrudan veri isteminde bulunmaktadır. Böylelikle sensör verisi kural kullanılarak filtrelenmektedir. Bunu gerçekleştirmek için uygulama öncelikle istekle birlikte bir kural dosyası göndermektedir. Bu kural dosyası, ilk olarak ayrıştırıcıda doğrulama işlemine tabi tutulmaktadır. Bunun sonucunda kural dosyası kural objesine dönüştürülmektedir. Obje içerisinde bulunan kural dosyası kural işleyicisi üzerinde, sensör verisi ile birlikte çalıştırılmakta ve sonrasında da uygulamalara gönderilmektedir (Şekil 9). Etkileşim Mekanizması A2'in işleyişi sırasında {❶, ②, ❷, ①, ❾, ❸, ❹, ❺, ③, ❻, ④, ❾, ⑦, ❶} nolu adımlar takip edilmektedir.

Etkileşim Mekanizması A3

Etkileşim Mekanizması A3'de uygulamadan veri tabanındaki verilerin belirli bir aralıkla sorgulanması istenmektedir. Söz konusu aralık uygulama tarafından belirlenmekte ve Planlayıcı'ya gönderilmektedir. Bu durumda kural dosyasına ihtiyaç bulunmamakta ve böylece de sistemde performans ve verimlilik artışı sağlanmaktadır. Periyodik gönderim işlemi zaman planı uygulayıcı sayesinde yapılmaktadır. Örneğin, uygulama her 5 dakikada veri gönderimi yapılması için istekte bulunmaktadır ve zaman planı uygulayıcısı içerisinde yeni bir iş tanımlanarak Veri Yönetim Servisi aracığıyla veri tabanı sorgusu yapılması istenmektedir. Sonrasında da bu veriler uygulamaya gönderilmektedir. Etkileşim Mekanizması A3'ün işleyişi sırasında {❶, ①, ❾, ❼, ❽, ⑥, ❾, ⑦, ❶} nolu adımlar takip edilmektedir.

Etkileşim Mekanizması A4

Etkileşim Mekanizması A4'te, mekanizma A3'e benzer şekilde veri tabanındaki verilerin belirli aralıklarla elde edilmesi istenmektedir. Ancak Mekanizma A4'te bu verilerin bir kural ile elde edilmesi söz konusudur. Bu kural çalıştırdıktan sonra, planlayıcı sonuçları uygulamaya bildirilmektedir. Etkileşim Mekanizması A4'ün işleyişi sırasında {❶, ②, ①, ❾, ❸, ❹, ❺, ❻, ⑤, ❼, ❽, ⑥, ❾, ⑦, ❶} nolu adımlar takip edilmektedir.

Etkileşim Mekanizması B1 ve B2

Kalan durumlar sürekli bağlantı zorunluluğu getirmemektedir. Kesintisiz iletişim gerektirmediklerinden dolayı basit bir HTTP istek/yanıt mekanizması yeterlidir. Etkileşim Mekanizması B1 ve B2'de Web soket kullanmaya ihtiyaç bulunmamaktadır. B1 için uygulama herhangi bir kısıt tanımlaması yapmadan istekte bulunmaktadır. B2 durumda ise bir kısıt tanımlaması sonucu sınırlandırmak için yapılmaktadır. Platforma yapılan istekler JSON formatında olacaktır. Aşağıdaki platforma yollanan HTTP istek/cevap mekanizmasında kullanılan JSON veri örneği gösterilmektedir.

```
query = {
  value: {
    min: Float,
    max: Float
  }
  time: {
    min: Timestamp,
    max: Timestamp
  }
}
```

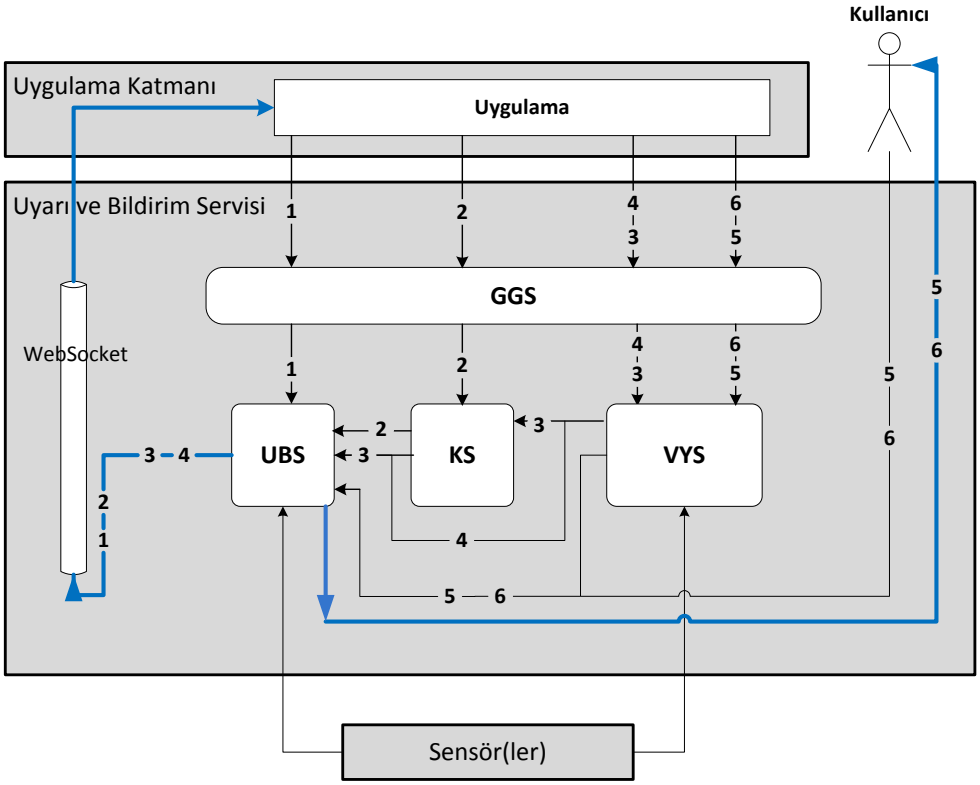
"*value*" istekte bulunulan sensörün değer aralıklarını "*time*" ise hangi zaman aralığındaki değerlerin istendiğini belirtmektedir.

Etkileşim Mekanizmaları Servis kullanım Eşleşmesi

Bu bölümde hangi isteğin hangi servis ya da servisler tarafından yerine getirildiği anlatılmaktadır. Platforma gelen bütün istekler Gizlilik ve Güvenlik ile Alarm ve Bildirim servislerini kullanmaktadır. Gelen istekler ancak güvenlik kontrollerini geçtikten sonra işlenmeye başlanır. Kullanıcılar PCAD platformu ile etkileşime girdiklerinde yollanan istekler platform içindeki servisler tarafından işlenmekte ve elde edilen sonuçlar geri gönderilmektedir. A1 ve A2 mekanizmaları bilgiyi direk sensörden almaktadırlar. Bunlardan ilkinde herhangi bir kural dosyası ile filtreleme yapılmazken, ikincisinde bir filtreleme olmaktadır. Bundan dolayı A2 mekanizması Kural Servisinin sağladığı işlevleri kullanmaktadır. A3 ve A4 mekanizmaları veri kaynağı olarak veri tabanını kullanmakta ve bir zaman ayarlayıcısı yardımı ile verileri sabit aralıklarla kullanıcıya yollanmaktadır. Veri tabanından verinin çekilmesi Veri Yönetimi Servisi tarafından yapılmaktadır. A4 mekanizmasında, A2 mekanizmasına benzer olarak verinin filtrelenmesi istendiğinde kural servisi kullanılmaktadır. B1 ve B2 mekanizmaları veriyi veri tabanından http istek/yanıt mekanizması kullanarak aldıkları için ayrıca bir bağlantı kurma safhasına ihtiyaç olmamaktadır. Bu son iki mekanizmada veri filtrelemesi çok basit ve kısıtlı olarak sunulmakta, bunun içinde bir kural dosyasına ihtiyaç duyulmamaktadır. Bu nedenden ötürü Kural Servisi kullanılmamaktadır. Filtre parametreleri platforma http isteğinin yük kısmında yollanmaktadır. Tablo 6'da servis mekanizma eşleşmesi, Şekil 10'da ise etkileşim mekanizmaları için servis akışı sunulmaktadır. Şekildeki sayılar, ilgili mekanizmaları göstermektedir (5 ve 6, B1 ve B2'ye karşılık gelmektedir).

Tablo 6. Servis mekanizma eşleşmesi

Etkileşim Mekanizması	Kural Servisi	Veri Yönetim Servisi	Uyarı Bildirim Servisi	Gizlilik Güvenlilik Servisi	Birlikte Çalışabilirlik Servisi
A1			X	X	
A2	X		X	X	
A3		X	X	X	
A4	X	X	X	X	
B1		X	X	X	
B2		X	X	X	



Şekil 10. Etkileşim mekanizmaları için servis akışı

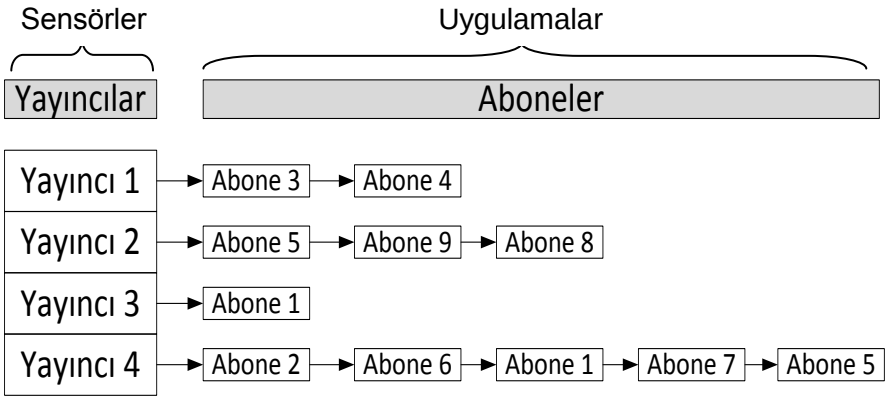
Filtreleme işlemleri

Gelen verinin filtrelenmesi iki şekilde olmaktadır: gelişmiş ve basit filtreleme. Gelişmiş filtreleme, Kural Servisi bölümünde belirtilen kural dosyası aracılığı ile yapılır. Kural dosyası sadece Python kütüphanesinde bulunan programlama API'leri kullanıldığında belirtilebilmektedir. Basit filtreleme ise REST-API kullanıldığında JSON yükünün içinde verilmektedir. Gerçek zamanlı durumda veriler platforma sensörler tarafından yollandıkları anında işleme alınırken, periyodik durumda veriler, uygulamaların belirttikleri aralık doğrultusunda işleme alınmakta ve veri tabanı, veri kaynağı olarak kullanılmaktadır.

Uyarı ve Bildirim Servisi'nde veri akışını doğru şekilde organize etmek amacıyla, Abone-Yayıncı (*İngilizce: publisher-subscriber*) modeline benzeyen bir yapı kullanılmıştır. Uygulamalar *Subscriber*, veri sağlayan sensörler ise *Publisher* olarak isimlendirilmiştir. *Publisher*, fiziksel sensörlerden veya veri tabanından gelen verileri doğrudan ya da belirli işlemlerden geçirerek *Subscriber* öğelerine göndermekle sorumludurlar. *Publisher* öğelerine veriler sadece bir kaynaktan gelmek zorunda değildir. Gerektiğinde birden fazla kaynaktan gelen verileri işleyerek ilgili öğelere gönderebilirler. Servis, en basit şekliyle bu modele göre çalışmaktadır. Ancak, bu akış içerisine kural çalıştırmayla ilgili işlemler girdiğinde, Kural

servisi de bu sürece dâhil olmaktadır. Ayrıca duruma göre Güvenlik ve Gizlilik Servisi ve Birlikte Çalışabilirlik ve İletişim Servisi de süreçte yer almaktadır.

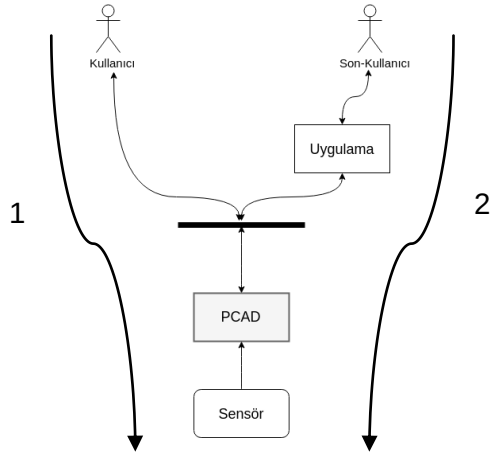
Uyarı ve Bildirim Servisi, veri akışını düzenlemek için hangi uygulamalara (abone), hangi veri kaynaklarından veri göndereceğine dair bilgiyi bir tabloda tutmaktadır. Bu tablo sayesinde veriler güvenli şekilde transfer edilirler. Bu tablo Uyarı ve Bildirim Servisi tarafından dinamik şekilde güncellendiğinden dolayı Uyarı ve Bildirim Servisine ait aktör tekil olmak durumundadır. Bu ilişkiler tablosu, Şekil 11'de sunulmaktadır. Birden fazla UBS oluşumunun tabloyu güncellemesine izin vermek, onarılmaz hatalara sebep olabilir. Sonuç olarak bu servisin tasarımı, çalışma mekanizmalarını gözeterek hataya yer vermeyecek şekilde yapılmıştır.



Şekil 11. Uyarı ve Bildirim Servisi sensör ve uygulamalar abone ilişkileri

4.2.4 Güvenlik ve Gizlilik Servisi

PCAD platformunun gerçekleştirilmesinde, Uygulama, Sensör ve Kullanıcı olarak üç kategoride sınıflandırılan istemciler (Şekil 12) hangi işlemleri yapıp yapamayacakları konusunda yetkilendirilmiştir. Kullanıcılar sistem üzerinde uygulama ve sensör tanımlarken, sensörler sadece sistem için veri sağlarlar. Uygulamalar sensörlerin ürettiği verilere ulaşıp, kendi amaçları doğrultusunda kullanırlar.



Şekil 12. PCAD faydalanıcıları erişim yolları

Kullanıcıların platforma erişimi iki yolla olmaktadır:

1. Kullanıcı bir web tarayıcısı aracılığı ile platforma doğrudan bağlanarak geçerli URL adresine isteklerini REST-API kullanarak yollamakta ve sensör verilerine ulaşmaktadır (Kullanıcı).
2. Kullanıcılar bir uygulama aracılığı ile platformun sağladığı verilere ulaşmaktadırlar (Uygulama Kullanıcısı).

Uygulamaları kullanan kullanıcıların yetkileri, uygulamanın kullanıcılar için belirlediği yetkilerle sınırlıdır. Uygulama kullanıcılarının sisteme ulaşımı bir uygulama üzerinden olduğu için ilk aşamada uygulamaya verilen haklar kullanılacak, bu haklar gerekirse uygulama geliştiricisi tarafından sınırlanacaktır. Uygulamalar kendi veri tabanlarını yaratarak kullanıcı profilleri oluşturmak suretiyle erişimi bu şekilde kısıtlama olanağına sahiptirler, böylece uygulamadan faydalananlarının istenmeyen haklara sahip olması engellenmiş olmaktadır. Kullanıcı kılavuzunda yukarıdaki adımların web arayüzü kullanılarak nasıl yapıldığı ekran çıktılarıyla verilmiştir.

Denetleme işlemi *token*'lar kullanılarak yürütülür. Tokenlar sisteme kayıtlanma suretiyle elde edilirler. Geçerli bir erişim token'a sahip olan istemciler PCAD servislerini kullanırlar. Erişim token'larına sahip olacak istemciler; kullanıcılar, uygulamalar ve sensörlerdir. Kullanıcının sisteme kaydı ve sistemden çıkış dışındaki bütün işlemler erişim token'leri sağlanarak yapılır. Kullanıcılar web arayüzünü kullanarak sisteme kayıt olduktan sonra bir erişim token'i oluşturulur. Her kayıt olan uygulama ve sensör öğesi için bir erişim tokeni oluşturulur. Kullanıcılar bu erişim token'ını kullanarak sistem üzerinde uygulama ve sensör kaydı yaparlar.

Uygulamalar ve sensörlerin platformla iletişimi, kullanıcı grubundan farklı gerçekleşmektedir. Kullanıcılar kendilerini sisteme kullanıcı ismi ve şifreyle tanıtabilirler. Ancak bu tanıtım işlemi sensörler ve uygulamalar için bazı güçlükleri içermektedir. Uygulama ve sensör iletişimini başlatmak için birincil olarak erişim tokenlarıyla kendilerini doğrulamak zorundadırlar. Bu işlemin olumlu geçmesi durumunda, yeni bir oturum token'i elde edeceklerdir ve bu belli bir süre sonunda kullanımı sona erecek olan bir token olacaktır. Veri alışverişi sırasında bunu kullanmak zorundadırlar. Böylelikle kimlik doğrulama işlemleri tamamlanmış olacaktır.

Tokenlar JSON Web Token (JWT) formatında gerçekleştirilmiştir. Örnek bir JSON Web Token Şekil 13'de yer almaktadır. Web token *header, claim* ve *signature* olmak üzere üç bölüme ayrılmıştır. Şekil 13'de token'nın *header* ve *claim* içeriği sunulmaktadır. Kullanıcılar yarattıkları öğeler için erişim token'larını görüntüleyebilirler. Bu token HS-256 (HMAC-SHA256) kullanılarak imzalanmıştır.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJQQ0FEIiwidWlkIj
oxNSwiZ3JvdXAiOiJzZW5zb3IifQ.8ifEzEbLGwMWvs__IvGXKX9Eh056HWZYXH
krcytcoPY
```

```
header = '{"alg":"HS256","typ":"JWT"}'
claim = '{"iss":"PCAD","group":"sensor","uid":"1"}'
```

Şekil 13. JSON Web Token örneği

Uygulamalar oturum *token*'larını her REST-API çağrısında PCAD'e iletirler. Güvenlik ve Gizlilik Servisi istekte bulunulan verilere erişim iznini veri tabanından sorgulayarak verir. Eğer kullanıcı ve uygulama bazında gerekli yetkilendirmeler var ise, bu istek diğer PCAD servislerine iletilir ve hazırlanan cevap istek yapana döndürülür.

Erişim ve oturum token'ları sensörlerin platforma bağlanıp veri yollanması da aynı şekilde gerçekleşecektir. Sensörler ve sensör yazılımları platforma kayıt esnasında bir erişim *token*'i alacaklardır ve onu takiben bir oturum token'i alacaklardır. Sensör yazılımı platforma her veri yolladığında bu oturum *token*'i da beraberinde yollayacaktır. Sensörün kimlik doğrulaması tamamlandıktan sonra sensöre veri yollaması için izin verilir. Kullanıcının platforma erişimi ile sensörlerin platforma erişimi arasındaki en önemli fark, sensörler için erişim hakkının seviyeli olmamasıdır. Diğer bir deyişle sensörün platforma erişim hakkı ya vardır ya da yoktur.

Özetle, ilk olarak istemciler kendilerini sisteme doğrulatmak zorundadırlar. Bunun için isteklerle birlikte gönderilen tokenler geçerli olmalıdır. Yapılan isteklerin hiyerarşisi birbiriyle çelişmemeli ve uyumlu olmalıdır. Örneğin uygulamalar platforma sensör verisi yollayamazlar. Son olarak da yapılan istek için gerekli izinleri sağlanmış olması gerekmektedir. Bu adımlardan herhangi biri için aksama, isteğin reddedilmesine neden olacaktır.

Uygulamaların ve kullanıcıların sensörlere erişimi çok temel ve basitleştirilmiş bir Rol Tabanlı Erişim Denetimi modeli kullanılarak gerçekleştirilmiştir. Bu model sensör verilerinin hangi uygulama için erişilebilir olduğuna rehberlik edecek, roller aracılığı ile kaynağa erişim yetkileri ve çeşitleri veya çeşitli kısıtlamalar tanımlanmıştır. Sensör verileri için bir gizlilik düzeyi belirlenirken, uygulamalarda hangi düzeydeki bilgilere ulaşabileceğini belirten bir erişim düzeyi atanır. Roller de platforma dayalı politikalar ile tanımlanır.

Her bir kullanıcı veya uygulamanın sensör erişiminde kullandığı bir rolü vardır ve bu rol *bilgi,veri* ve *durum* adı verilen alt yetkilerden oluşmaktadır. Bu üç yetkinin birleşimi rolün tam yetkisini belirlemektedir. Bu yetkilerin tanımı aşağıda verilmiştir.

- **Bilgi**: Sensör meta bilgisi sorgulama hakkı (yeri, tipi, üretici ve diğer özellikler)
- **Veri:** Sensörden veri alma hakkı (sensör verileri)
- **Durum:** Sensör durum bilgisini sorgulama (aktif-pasif olup olmadığı).

Örneğin bir uygulamanın sensörün bulunduğu yeri öğrenmesi, durumunu sorgulaması ve veri alabilmesi için **Bilgi**, **Veri**, ve **Durum** haklarını barındıran bir rolü olmalıdır. Böyle bir rol aşağıda tanımlanmıştır.

RolA = (B, V, D)

B,V ve D; **Bilgi**, **Veri** ve **Durum** haklarının kodlanmış şeklidir. Aynı metodu kullanarak diğer rollerde benzer şekilde tanımlanmıştır.

RolB = (-,V,-)

RolC = (-,-,D)

Bu yetkiler kullanıcılara aşağıda sunulan adımlar sonucunda verilir:

1. Kullanıcı sensöre erişim isteğinde bulunur.
2. Bu istek sistem yöneticisi tarafından onaylanır.
3. Kullanıcı kendisine verilen yetkiyi aşmayan ikinci bir yetkiyi sahip olduğu uygulamalar için ister.
4. Sistem yöneticisi uygulama yetkilerini onaylar.
5. Uygulama verilen izinler doğrultusunda bir role sahip olur ve sensöre erişim hakkı kazanır.
6. Uygulama isterse uygulama kullanıcıları için bu yetkileri daha da kısıtlayabilir.

Örnek Senaryo: Uygulama geliştiricisi uygulamasını kaydetmek için PCAD portal sayfasına giriş yapar ve uygulamasını kayıt eder. Bu işlem sistem yöneticisi tarafından onayladığında kendisine uygulamanın kullanması için erişim token'ı verilir. Kullanıcı verileri *kullanicilar* tablosuna uygulama verileri *uygulamalar* tablosuna yazılır. Sistem yöneticisi onayı ile her iki

tabloda bulunan *gecerlilik* sahası aktif hale getirilir. Kayıt esnasında uygulama geliştirici bu uygulamanın sensörler için nasıl bir erişim hakkı istediğine dair bir istekte bulunur. Bu istek sistem yöneticisi tarafından onayladığında kendisine erişim *token*'ı verilir. Uygulamaya *roller* tablosunda bulunan rollerden biri atanır. Eğer *roller* tablosundaki mevcut roller yeterli bir yetkilendirme vermiyorsa yeni bir rol oluşturulur ve tabloya yazılır. Uygulama, yetkileri atanan rol ile birlikte *uygulama_sensor_erisim_listesi* tablosunda tutulur. *uygulama_sensor_erisim_listesi* tablosuna eklenen her bir satır atanan rol ile uygulamanın her bir sensöre nasıl bir yetki ile ulaşacağını belirler. Sistem yöneticisi onayı ile birlikte ilgili tablodaki geçerlilik sahası aktif hale gelir.

Platforma direk ulaşmak isteyen kullanıcılar ile ilgili işlemler benzer bir şekilde yerine getirilir ve kullanıcıya rolü atanır. *kullanicilar* ve *kullanici_sensor_erisim_listesi* tablolarında gerekli satırlar eklenir. *kullanici_sensor_erisim_listesi* rol-kullanıcı-sensor eşleşme bilgisini tutar. Erişim tablolarında bulunan geçerlilik alanı erişim haklarının yürürlükte olup olmadığını gösterir. Bu nokata da uygulamaları kullanan kullanıcıların erişim hakları, uygulamalar tarafından kıstılanabilir ya da azaltılabilir. Böylece uygulamadan faydalananların istenmeyen haklara sahip olması engellenmiş olunmaktadır.

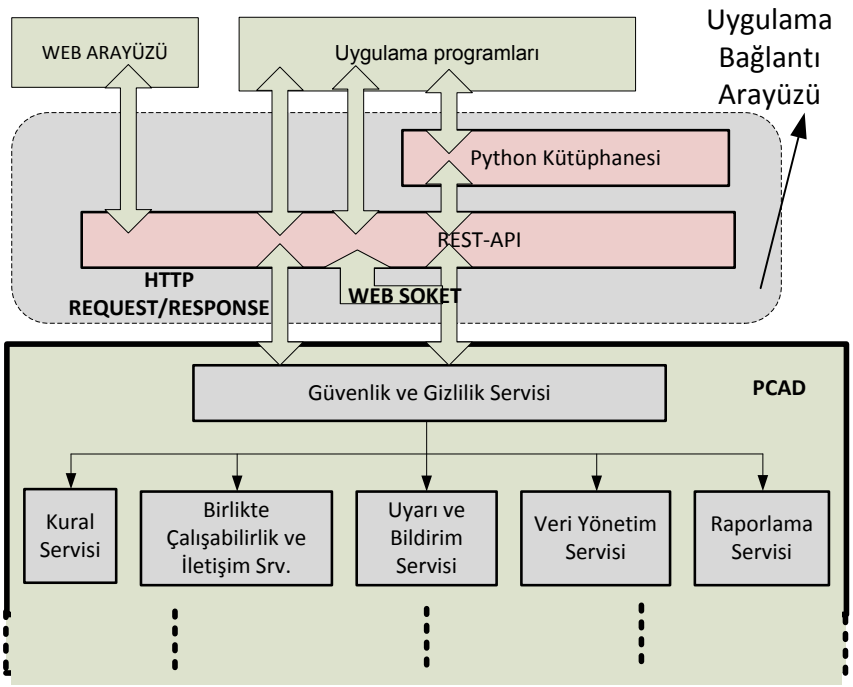
4.2.5 Birlikte Çalışabilirlik ve İletişim Servisi

Bu servis PCAD platformunun, PCAD benzeri platformlarla iletişime geçip veri alışverişinde bulunmasını sağlamak için kullanılır. Bu servisin gerçekleştirilmesi için ek bir çabaya gerek yoktur. Çok yaygın bir protokol olan HTTP Request/Response kullanarak diğer platformların da PCAD platformundan veri isteğinde bulunması bu şekilde mümkün olmaktadır. PCAD'dan veri isteyen diğer platformlar kendilerini sisteme kayıt ettirdikten sonra bir kullanıcı gibi HTTP Request/Response metodunu ya da kütüphane API'lerini kullanarak veri alabileceklerdir. Platforma yollanan isteklerin nasıl yapılacağı ve isteğin formatı takip eden Kısım 4.2.6 ve 4.3.6 da detaylı anlatılmıştır.

4.2.6 PCAD platformuna bağlanma arayüzleri

Platformla iletişim temel olarak, REST arayüzleri kullanılarak HTTP protokolü üzerinden yürütülmektedir. Arayüz tanımlanmasının amacı yazılım geliştirmeyi kolaylaştırmaktır. Böylelikle, günümüzde benzerleri çokça kullanılan arayüzler sayesinde, hem uygulama hem de sensör geliştiricilerinin platforma uyum sağlamaları kolaylaşacaktır. Buna ek olarak, kullanıma sunulan Python kütüphanesi sayesinde alt katmanda çalışan REST API kullanımını saklanarak, geliştiriciye daha basit bir arayüz kullanımı sunulmaktadır. Ancak isteyen kullanıcılar, REST API'ler aracılığı ile platform bağlantısı sağlayabilirler. Uygulamalar ve platforma veri sağlayan sensörler platform ile iki şekilde iletişime geçmektedir (Şekil 14):

- HTTP istek/yanıt aracılığı ile REST API kullanarak ve
- Programlama arayüzü (API) barındıran Python kütüphanesi kullanarak.



Şekil 14. Uygulamaların platform ile bağlantıları

REST API arayüzü

REST API arayüzü ile ilgili detaylar Tablo 7'de sunulmaktadır. Bu süreçte, verilen URL üzerinden gerekli HTTP metotuyla istek yapılmakta, gerektiği durumlarda ise JSON formatında yük gönderilmektedir. Tablo 8'de JSON yüküyle ilişkili format gösterilmiştir. Tablo 7'de görülen iki istek dışında (*signup, signout*) istekler erişim *token*'ının kullanmaları gerekmekte ve bunu URL üzerinden sorgu parametresi olarak göndermek zorundadır. Kullanıcıların listelenmesi için örnek URL şu şekilde olmalıdır:

```
http://pcad.com/users?access_token=eyklş12930KIM.9023JKDFM3.FDK390KKAGN
```

Uyarı ve Bildirim Servisinde bahsi geçen altı etkileşme mekanizmasından bağlantı tabanlı (*connection oriented*) olan dört tanesi Websoket protokolü kullanılarak gerçekleştirilmiştir.

Websoket protokolü üzerinden çalışan sadece bir adet URL mevcuttur. Tablo 7'te HTTP protokolünü kullanan URL'ler sunulmaktadır. Websoket protokolü üzerinden çalışan bağlantı kapatılana kadar sürekli açık kalır ve HTTP protokolünde olduğu gibi cevap alındığında

sonlanmaz. UBS bölümünde anlatılan ilk dört senaryo Websoket protokolü kullanılarak aşağıda örneği verilen URL aracılığı ile gerçekleştirilmektedir. Kısım 6.1.2'da bu protokolün kullanımı ile ilgili detaylı örnekler verilmiştir. Bağlantı kurarken oturum token'ı URL'in içinde gönderilmelidir.

Websocket URL : `ws://pcad.com/sensors/stream?access_token=OTURUM_TOKENI`

Tablo 7'de verilen *signup* ve *signout* haricindeki diğer end-point'ler için access_token URL'in içerisinde olmalıdır. Tablo 7'de altı çizili end-point'ler kütüphane tarafından kullanılmakta, geri kalan end-point'ler ve /auth/signin ise Web arayüzü tarafından kullanılmaktadır.

Tablo 7. REST API

end-point	Metot	Yük	İşlev
/auth/signup	POST	newUser	Kullanıcı kaydı
/auth/signin	POST	-	Kullanıcı girişi
/auth/signout	GET	-	Kullanıcı çıkışı
/users	GET	-	Kullanıcıların listelenmesi
/users/:id	GET	-	Kullanıcı görüntüleme
/users/:id	PUT	User	Kullanıcı güncelleme
/sensors	GET	-	Sensörlerin listelenmesi
/sensors	POST	newSensor	Sensör kaydı
/sensors/:id	GET	-	Sensör görüntüleme
/sensors/:id	PUT	Sensor	Sensör güncelleme
/sensors/:id	DELETE	-	Sensör silme
/applications	GET	-	Uygulamaların listelenmesi
/applications	POST	newApplication	Uygulama kaydı
/applications/:id	GET	-	Uygulama görüntüleme
/applications/:id	PUT	application	Uygulama güncelleme
/applications/:id	DELETE	-	Uygulama silme
/permissions/users	GET	-	Kullanıcı izinlerinin listelenmesi
/permissions/users	POST	newUserPermission	Kullanıcı izni kaydı
/permissions/users/:id	GET	-	Kullanıcı izni görüntüleme
/permissions/users/:id	PUT	userPermission	Kullanıcı izni güncelleme

/permissions/users/:id	DELETE	-	Kullanıcı izni silme
/permissions/applications	GET	-	Uygulama izinlerinin listelenmesi
/permissions/applications	POST	newAppPermission	Uygulama izni kaydı
/permissions/applications/:id	GET	-	Uygulama izni görüntüleme
/permissions/applications/:id	PUT	appPermission	Uygulama izni güncelleme
/permissions/applications/:id	DELETE	-	Uygulama izni silme
/sensors/:id/data	GET	-	Son sensör verisi görüntüleme
/sensors/:id/data	POST	sensorData	Sensör verisi kaydı
/sensors/:id/data/filter	POST	filter	Sensör verilerinin filtrelenmesi
/sensors/:id/info	GET	-	Sensör güncelleme
/sensors/:id/status	GET	-	Sensör silme
/roles	GET	-	Rollerin listelenmesi
/roles	POST	newRole	Rol kaydı
/roles/:id	GET	-	Rol görüntüleme
/roles/:id	PUT	role	Rol güncelleme
/roles/:id	DELETE	-	Rol silme

Tablo 8. JSON yük formatı

newUser (/auth/signup)	user (/users/:id)
{ firstName: String, lastName: String, email: String, company: String, username: String, password: String }	{ id: Long, firstName: String, lastName: String, email: String, company: String, username: String, password: String access_token: String registration_time: Timestamp, valid: Boolean }
newSensor (newSensor)	**sensor (/sensors/:id)**

{
type: String,
company: String,
unit: String,
latitude: String,
longitude: String
}
	{ id: Long,
	type: String,
	company: String,
	unit: String,
	latitude: String,
	longitude: String,
	access_token: String,
	valid: Boolean
	}

newApplication (/applications)	application (/applications/:id)

{
user: Long,
name: String
}
	{ id: Long,
	user: Long,
	name: String,
	access_token: String,
	registration_time: Timestamp
	}

newUserPermission (/permissions/users)	userPermission (/permissions/users/:id)

{
user: Long,
sensor: Long,
role: Long
}
	{ id: Long,
	user: Long,
	sensor: Long,
	role: Long,
	valid: Boolean
	}

newAppPermission /permissions/applications	appPermission (/permissions/applications/:id)

{
application: Long,
sensor: Long,
role: Long
}
	{ id: Long,
	application: Long,
	sensor: Long,
	role: Long,
	valid: Boolean
	}

sensorData (/sensors/:id/data)	filter (/sensors/:id/data/filter)

{
sensor: Long,
time: Timestamp,
value: Float
}
	{
	value: {
	min: Float,
	max: Float
	}
	time: {
	min: Timestamp,
	max: Timestamp
	}
	}

newRole /roles	role (/roles/:id)

{
name: String,
status: Boolean,
info: Boolean,
data: Boolean
}
	{
	id: Long
	name: String,
	status: Boolean,
	info: Boolean,
	data: Boolean
	}

Python kütüphanesi

Python kütüphanesi ile ilgili uygulamalar ve Sensörler tarafından ortak kullanılan ara yüzler Tablo 9'da sunulmaktadır.

Tablo 9. Uygulamalar ve Sensörler tarafından ortak kullanılan ara yüzler

İsim	Açıklama
pcad_init_context	Platform arasındaki iletişime başlayabilmek için kullanılacak ilk ara yüzdür. Bu metot düzenleme dosyası dizinini alır ve dosya içerisinde yer alan erişim tokeninin atamasını yapar.
pcad_get_credentials	Kimlik doğrulama yapılıp bağlantı kurulduktan sonra yapılacak iletişimde kullanılmak üzere oturum token'i almak için kullanılır. Oturum token'i zaman sınırlıdır. Geçerlilik süresi bittiğinde yenisini almak için pcad_init_context ara yüzünü kullanarak tekrardan kimlik doğrulaması yapılmalıdır.
pcad_free_context	Bağlantı bitirilmek istendiğinde çağrılır. Disk'de ve hafızada bulunun bilgileri temizler.

Platform ile sensörler arasında kullanılan API'ler Tablo 10'da, platform ile uygulamalar arasında kullanılan API'ler ise Tablo 11'de verilmiştir.

Tablo 10. Sensör platform bağlantı API'si

İsim	Açıklama
pcad_post_data	Sensörün platforma veri yollarken kullanacağı ara yüzdür. Veri JSON formatında yollanmalıdır.

Tablo 11. Uygulama platform bağlantı API'leri

İsim	Açıklama
pcad_init_connection	Uygulama ile platform arasındaki bağlantıyı oluşturur. Bu bağlantı web soket tabanlı bir bağlantıdır. Bağlantı kesilene kadar aktif durumda bulunur. Bağlantı parametrelerini kullanarak bağlantının nasıl olması gerektiği ve hangi sensörden veri alınacağı belirtilir. Bağlantının esnek gerçek zamanlı olup olmayacağı bağlantı kurulurken sabitlenir. Bunun alternatifi olarak uygulamaların veri tabanından gerçek zamanlı olmayan ama sürekli veri alma istekleri de bağlantı kurulacağı zaman platforma gönderilir.
pcad_attach_sensor	Bildirim alınmak istenen sensörle ilgili istek gönderimini

	parametreler almak yardımıyla yapar.
pcad_on_message	Platformdan mesaj almak için kullanılır. Uygulamalar bu API ile platformu dinleme moduna geçerler. Platform sensörden veri geldikçe uygulamanın API çağrılırken belirlediği fonksiyonu geri çağırır.
pcad_terminate_connection	Bağlantıyı sonlandırmakta kullanılır.
pcad_send_query	Bu API platformla devamlı aktif bir bağlantı yerine bir kerelik veri isteğinde bulunulduğunda kullanılır. Uygulama *pcad_send_query* kullanarak JSON formatında veri isteğinde bulunur. Platform uygulamaya istenen veriyi yollar. Bu tür iletişimde başlatılması ya da sonlandırılması gereken bir bağlantı yoktur. Platform ile iletişim HTTP istek/yanıt mekanizması kullanılarak yapılır.

Sensörlerin PCAD platformu ile bağlanma adımları

Sensörlerin platform ile bağlantısı için bir sensör yazılımı gerekmektedir. Bu genellikle sensörün bağlı olduğu bir mikro denetleyici aracılığı ile olmaktadır. Aşağıdaki bölümde sensörlerin ve uygulamaların platform ile iletişimi için yapılması gereken adımlar sunulmaktadır. Programlama ara yüz kütüphanesi için PYTHON programlama dili seçilmiştir. Sensör yazılımları sensörün bağlandığı mikro denetleyiciye göre değişkenlik göstermektedir. Arduino kullanıldığında C programlama dili yazılmış olan program sensörden gelen bilgiyi okurken Arduinoya bağlı ikincil bir cihaz Python ile yazılmış programı kullanarak Arduino'dan gelen bilgiyi alarak PCAD'a iletmektedir. Rasperry PI gibi bir mini bilgisayar kullanıldığında bu cihaz hem sensörden veriyi doğrudan alabilecek hem de PCAD'a yollayabilecektir.

Sensörlerin PYTHON programlama arayüzünü (API) kullanarak platforma bağlantısı

Sensörler platforma veri yollamaya başlamadan önce bir doğrulama adımı gerçekleştirilmedir. Bu sayede platforma sadece veri göndermeye yetkisi olan sensörler veri yollayabileceklerdir. Doğrulama adımı tamamlandıktan sonra veri yollama işlemi başlayacaktır.

1. Öncelikle bir JSON dosyası içerisine sensöre ait olan erişim tokeni yazılır. Anahtar olarak "access_token", değer olarak da erişim tokeni yazılır. *access_token*, sensör yazılımcısı tarafından sensör sisteme kayıt edilirken alınmalıdır. *access_token*

alınırken https kullanarak platforma bağlantı yapılır. *sensor_config.json* isimli örnek bir dosya aşağıda verilmiştir.

```
{
  "access_token":
  "oxLCJncm91cCI6InVzZXIifQ.jQpT5bGz2Mck76faNdewtQX2"
}
```

2. Sensör, platform ile bağlantıyı sağlamak için `pcad_init_context` fonksiyonu kullanılır. Fonksiyona parametre olarak verilen dosyanın içerisinde, erişim token'ı bulunur. Bu erişim token'ı sensör kaydı yapılırken alınan erişim token'ı ile aynıdır.

```
Context context;
context = pcad_init_context("sensor_config.json")
```

3. Platformla oturum açılabilmesi `pcad_get_credentials` fonksiyonu kullanılır. Bu doğrulama işleminin tamamlar ve sensör ile platform arasında veri yollamada kullanılmak üzere bir oturum açılır.

```
int result = pcad_get_credentials(context)
```

4. Veri göndermek için ***pcad_post_data*** fonksiyonuna, gönderilecek olan veri parametre olarak verilir. İstenildiği kadar veri bu metot kullanılarak gönderilir. Gönderilecek veri JSON formatında olmalıdır.

```
String payload = {
  'sensor': 1,
  'time': '2016-12-02T23:39:30.0+03:00',
  'value': 14.6
}
int response = pcad_post_data(context, payload)
```

5. İş bitince kimlik doğrulama bilgilerini temizlemek için ***pcad_free_context*** metotu çağrılabilir.

```
pcad_free_context(context)
```

Program 1'de Python kütüphanesini kullanan örnek bir sensör programı bulunmaktadır.

```python
# -*- coding: utf-8 -*-
import datetime
import random
from connection import Connection
from dateutil.tz import tzlocal

if __name__ == "__main__":
    INSTANCE = Connection()
    CONTEXT = INSTANCE.pcad_init_context("sensor_config.json")
    SECURE_CONTEXT = INSTANCE.pcad_get_credentials(CONTEXT)
    NOW = datetime.datetime.now(tzlocal())
    CURRENT_TIME = NOW.strftime('%Y-%m-%dT%H:%M:%S.%f%z')
    RANDOM_VALUE = round(random.uniform(-10.0, 20.0), 2)
    PAYLOAD = {'sensor': 15, 'time': CURRENT_TIME, 'value': RANDOM_VALUE}
    RES = INSTANCE.pcad_post_data(SECURE_CONTEXT, PAYLOAD)
```

Program 1. Python API kullanan sensör programı (*api_sensor.py*)

Sensörlerin REST-API kullanarak platforma bağlantısı

Sensörler platforma veri yollamak için arayüz kütüphanesi yerine REST API metotunu kullanarak platforma doğrudan istek yollayabilirler. Buradaki işlemlerde kullanılacak REST arayüzü 4.2.6 bölümündeki Tablo 7'de listelenmiştir. Bunun için sensör yazılımlarının bu isteklerini yollayabilecekleri bir kütüphaneye ihtiyaç vardır. Python dilinde yazılmış *requests* kütüphanesi bu işlemi yapmaktadır. Bu kütüphane, HTTP istekleri yollama işleminde kullanılmaktadır. Bu kütüphane kullanılarak yapılan işlemler aşağıda sunulmaktadır.

1. Kimlik doğrulama için daha önceden alınmış `ACCESS_TOKEN` ve `/auth/signin` uzantılı URL kullanılır. Bu erişim token'ı kullanılarak oluşturulan URL, *requests* kütüphanesi kullanılarak platforma yollanır.

```
URL = http://localhost:9000/api/v1/auth/signin?
                  access_token = ACCESS_TOKEN
response = request.request(
   "POST", URL, data="{}",
   headers = {'content-type': 'application/json'}
)
```

   Cevap olarak platform yeni bir *oturum token'*ı yollar. Bu token daha sonraki yapılacak istekler için kullanılır. Gelen cevabın içinden oturum tokenı çekilmeli ve bundan sonraki isteklerde bu token kullanılmalıdır.

```
SESSION_TOKEN  =  response.text
```

2. Veri gönderme için `/sensor/:id/` uzantılı URL kullanılarak gönderilecek olan veri isteğe eklenir. Gönderilecek olan yükün formu 4.2.6 numaralı bölüm, Tablo 8'de **sensorData** kısmında belirtilmiştir ve "id" sensör numarasına karşılık gelmektedir. Alttaki örnekte bu 1 olarak belirtilmiştir.

```
URL = http://localhost:9000/api/v1/sensors/1/data?
                                access_token = SESSION_TOKEN
payload = "{\"sensor\": 1,
           \"time\": \"2019-02-10T10:54:57.0+03:00\",
           \"value\": 1.11}"

response = request.request(
   "POST", URL, data=payload,
   headers={'content-type': 'application/json'}
)
```

Program 2'de REST-API kullanan örnek bir sensör programı bulunmaktadır.

```
1  # -*- coding: utf-8 -*-
2  import requests
3
4  AUTH_URL = "http://localhost:9000/api/v1/auth/signin"
5  QUERY_STRING =
   {"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJQQ0FEIiwidWl
   kIjoxNSwiZ3JvdXAiOiJzZW5zb3IifQ.8ifEzEbLGwMWvs__IvGXKX9Eh056HWZYXHkrcytcoPY"}
6  HEADERS = {'content-type': 'application/json'}
7  RESPONSE = requests.request("POST", AUTH_URL, data="{}", headers=HEADERS,
   params=QUERY_STRING)
8
9  DATA_URL = "http://localhost:9000/api/v1/sensors/19/data"
10 QUERY_STRING = {"access_token":RESPONSE.content.replace("\"", "")}
11 PAYLOAD = "{\n  \"sensor\": 19,\n  \"time\": \"2019-02-10T10:54:57.0+03:00\",
   \n  \"value\": 1.11\n}"
12 RESPONSE = requests.request("POST", DATA_URL, data=PAYLOAD, headers=HEADERS,
   params=QUERY_STRING)
13 print(RESPONSE.text)
14
```

Program 2. REST-API kullanan sensör programı (*rest_sensor.py*)

Uygulama ile PCAD platformu bağlantısı

Bu Bölümde uygulamalar ile PCAD platformunun nasıl bağlanacağına ilişkin yapılan çalışmalar örnekler ile sunulmaktadır. Söz konusu bağlantılarda, bir üsteki bölümde de sunulanlara benzer şekilde iki farklı yöntem kullanılmıştır. Bunlar, programlama arayüzünü (API) ve REST-API'dir.

Uygulamaların PYTHON programlama arayüzünü (API) kullanarak platforma bağlantısı

Uygulama bağlantısı sensörlerin aksine daha karmaşık bir yapıya sahiptir. Bunun iki temel sebebi bulunmaktadır. Bunlar, uygulamaların sensörlerden periyodik zamanlı veri istemesi ve kural belirtme seçeneklerinden kaynaklanmaktadır. Ancak veri isteme şekli ne olursa olsun, Uygulamaların platforma bağlanması ve kimlik doğrulaması sensörlerin bağlanırken kullandıkları yöntemle aynıdır. `pcad_init_context`, `pcad_get_credentials`, `pcad_free_context` fonksiyonları sensörler için belirtilen kimlik doğrulama işlemlerinde

kullanıldığı şekliyle aynen uygulanır. Kullanılan dosyası sensörler için belirtilen JSON dosyası da aynı formattadır. Bölüm 6.1.2'de PCAD platformu ile uygulama arasında kullanılacak API'leri listesi verilmiştir.

Aşağıdaki örnekte, Bölüm 4.2.3'de sunulan **bağlantı tabanlı** etkileşim mekanizmalarından **A1** anlatılmaktadır.

Uygulamalar PCAD platformu ile bağlantılı iletişimi Web soket bağlantısı açarak sağlarlar. Bu bağlantı uygulama tarafından bitirilene kadar aktif şekilde kalmaktadır. Bağlantılı iletişimi kullanmadaki temel amaç, verilerin uygulamaya gönderilmesinde bir süreklilik sağlamaktır. Aşağıda yapılması gereken adımlar sıralanmıştır.

1. `pcad_init_context`, `pcad_get_credentials` foksiyonları kullanılarak kimlik doğrulaması yapılır.

```
context = pcad_init_context("sensor_config.json")
secure_context = pcad_get_credentials(context)
```

2. `pcad_init_connection` metotuyla aktif bağlantı açılması sağlanır. Bu kullanılarak platformdan uygulamaya devamlılığı olan veri akışı sağlanır.

```
connection = pcad_init_connection(secure_context)
```

3. `pcad_attach_sensor` metotuyla aktif olan bağlantı yoluyla veri isteği yapılır. Veri istenen sensör ve ilgili ayarlamalar yapılır. İlk parametre olarak *sensör*, *mode* ("*real*" ya da "*periodic*") ve aralık JSON formatında verilecektir, burada aralık sadece mod "*periodic*" olarak yazıldığında verilmelidir ve ikinci parametre olarak opsiyonel olarak kural dosyası dizini verilebilir. Kurallar bir dosya içerisinde tanımlanır ve `.rl` uzantısına sahiptir.

```
pcad_attach_sensor({"mode":"real","sensor":15})
```

4. `pcad_on_message` metotu aktif bağlantı yolu ile gelen mesajları kullanılmasını sağlar. Örneğin gelen mesajların ekrana yazılmasını sağlayan fonksiyon bu metota parametre olarak verilir ve ekrana yazdırma işlemi gerçekleşir.

```
def print_message(message):
      print message
pcad_on_message(connection, print_message)
```

5. `pcad_terminate_connection` aktif bağlantıyı sonlandırmak için için kullanılır.

```
status = pcad_terminate_connection(connection)
```

Bağlantı tabanlı etkileşim mekanizmalarından **A1**'e ilişkin program örneği Program 3'te sunulmaktadır.

```python
# -*- coding: utf-8 -*-
from connection import Connection

def print_message(message):
    print message

if __name__ == "__main__":
    try:
        INSTANCE = Connection()
        CONTEXT = INSTANCE.pcad_init_context("application_config.json")
        SECURE_CONTEXT = INSTANCE.pcad_get_credentials(CONTEXT)
        CONNECTION = INSTANCE.pcad_init_connection(SECURE_CONTEXT)
        INSTANCE.pcad_attach_sensor({"mode":"real", "sensor": 15})
        INSTANCE.pcad_on_message(CONNECTION, print_message)
    except (Exception, KeyboardInterrupt, SystemExit) as e:
        INSTANCE.pcad_terminate_connection(CONNECTION)

```

Program 3. Etkileşim Mekanizması A1 (*api_01_app_real_time.py*)

Aşağıdaki etkileşim mekanizmalarından **B2**'yi anlatmaktadır. Program 4'de kaynak kodu verilmiştir. Uygulama, platformun verileri depoladığı veri tabanından belirli bir zaman aralığında, gerçek zamanlı olmayan ve bir kereye mahsus filtreli bir veri sorgulaması yapmaktadır. Bu etkileşim mekanizmasında devam eden bir bağlantı yoktur.

1. `pcad_init_context`, `pcad_get_credentials` fonksiyonları kullanılarak kimlik doğrulaması yapılır.
2. `pcad_send_query` fonksiyonu aracılığı ile veri sorgulaması yapılır. Bu fonksiyona JSON formatında bir filtre verilir. Bu filtre opsiyoneldir. Filtrenin yapısı Tablo 8'de **"filter"** altında görülebilir. Bu yapı Python dilinde *dictionary* veri yapısı oluşturularak sağlanabilir.

```
filter = {
   "value": null,
   "time":       {
     "min": "2016-12-02T23:39:30.0+03:00",
     "max": "2016-12-07T23:39:30.0+03:00"
   }
 }

String response = pcad_send_query(filter);
```

Gelen veri aşağıdaki formda olacaktır:

```
[
{ "id": 51, "sensor": 1, "time": "2016-12-
03T18:20:14.000+03:00","value": 5.84},
{"id": 52,"sensor": 1,"time": "2016-12-
03T16:00:36.000+03:00","value": 12.25}
…
]
```

```python
# -*- coding: utf-8 -*-
from connection import Connection

if __name__ == "__main__":
    try:
        INSTANCE = Connection()
        CONTEXT = INSTANCE.pcad_init_context("application_config.json")
        SECURE_CONTEXT = INSTANCE.pcad_get_credentials(CONTEXT)
        FILTER = {"value": None, "time": {"min":
"2016-12-02T23:39:30.0+03:00", "max": "2016-12-07T23:39:30.0+03:00"}}
        RESULT = INSTANCE.pcad_send_query(SECURE_CONTEXT, 1, FILTER)
        print RESULT
    except (Exception, KeyboardInterrupt, SystemExit) as e:
        print "connection killed " + str(e)
```

Program 4. Etkileşim Mekanizması B2 (*api_06_app_one_time_filter.py*)

Diğer etkileşim mekanizmalarına ait örnekler sistemim dokümantasyonunun sunulduğu 6. Bölümde bulunabilir.

Uygulamaların REST-API kullanarak platforma bağlantısı

Uygulamalar için REST API üzerinden kimlik doğrulama işlemleri sensörler için kullanılan metotlarla aynıdır. Aktif bağlantı kurmak için temelde websoket protokolü kullanılır. Bunun içinde websoket kütüphanesi kullanılmalıdır. Aktif bağlantı açma ve veri isteği, /sensor/stream uzantılı URL üzerinden kullanılarak bağlantı sağlanır. Sonra, bu bağlantı üzerinden JSON yapısında mesaj yollanır. Bu mesajda sensör, mode (opsiyonel), interval (opsiyonel), kural (opsiyonel) bulunmalıdır. Böylelikle bağlantı istenen ayarlarda sağlanmış olur. Örnekte *websocket-client* kütüphanesi kullanılarak etkileşim mekanizması **A2** gerçekleştirilmiştir.

```
url = "http://localhost:9000/api/v1/sensors/stream?access_token=
SESSION_TOKEN"
websocket.connect(url)
rule = "..."
payload = "{\"mode\":\"real\",\"sensor\":1,\"rule\":rule}"
websocket.send(payload)
```

Mesajları almak için gelen bağlantıyı dinlemek gerekir ve aşağıdaki gibi yapılır.

```
response = websocket.recv()
```

Aktif bağlantı sonlandırmak aşağıdaki gibidir.

```
websocket.close()
```

REST-API kullanan diğer etkileşim mekanizmalarına ait örnekler sistemin dokümantasyonunun sunulduğu 6. Bölümde bulunabilir.

4.3 PCAD Platformunun Node.js Uygulama İskeleti Kullanılarak Gerçekleştirilmesi

Bu bölümde PCAD platformuna ait servislerin Node.js araçları ve JAVA programlama dili kullanılarak gerçekleştirilmesi sürecindeki işlemler sunulmaktadır. Node.js uygulama iskeletinin gerçekleştirilmesinde kullanılan yazılım araçları ve bu araçların PCAD servisleri ile eşleşmesi Tablo 12'de sunulmaktadır.

Tablo 12. Node.js uygulama iskeletinin gerçekleştirilmesinde kullanılan yazılım araçları ve bu araçların PCAD servisleri ile eşleşmesi.

	Kural Servisi	Veri Yönetim Servisi	Uyarı ve Bildirim Servisi	Güvenlik ve Gizlilik Servisi	Birlikte Çalışabilirlik ve İletişim Servisi
Mosquitto (MQTT)			✓		
LoopBack		✓	✓	✓	

Söz konusu gerçekleştirme sürecinde Yayıncı-Abone mimarisinin "*loose-coupling*"ve "*scalability*" gibi avantajlarından dolayı MQTT (*İngilizce*: *Message Queuing Telemetry Transport*) protokolü seçilmiş ve bu protokolü gerçekleştiren Mosquitto yazılımı kullanılmıştır. MQTT protokolü birçok yazılım geliştirme dili tarafından desteklenmekte ve kullanılmaktadır. Mosquitto yazılımı MQTT protokolünü kullanan bir simsar veya aracı (*broker*) yazılımı olup mesajı yayan birimlerle (*publisher*), bu mesajı kullanmak isteyen aboneler (*subscriber*) arasında verinin iletilmesini sağlamaktadır. MQTT protokolü bir mesaj kuyruklama ve iletimi protokolü olup, en önemli özelliklerinden biri, farklı platformlarda kolay kullanımıdır. Uyarı ve Bildirim Servisi'nin aşağıdaki bölümde sunulan gerçekleştiriminde, ilgili yazılıma ait detaylar sunulmaktadır.

Veri Yönetim Servisi, Güvenlik ve Gizlilik Servisinin kimlik doğrulama kısmı, LoopBack uygulama iskeletinin içerdiği araçlar kullanarak geliştirilmiştir. Araçların bulunmadığı ya da yetersiz kaldığı durumlarda LoopBack uygulama iskeleti içine gerekli fonksiyonlarda saf Node.js yazılarak işlemler tamamlanmıştır (Örneğin Güvenlik ve Gizlilik Servisinin yetkilendirme kısmı gibi). Kural Servisi ile Uyarı ve Bildirim Servisi, Mosquitto yazılımının kullanımı ile birlikte LoopBack uygulama iskeletinin içinde, gerekli modellerde gerçekleştirilmiştir.

4.3.1 Kural Servisi

Kural Servisi, JSON yükünün içinde gelen filtre objesinin ayrıştırılması ve yorumlanması görevlerinden sorumludur. Filtreler uygulamalar tarafından gönderilen ve belirli şartlar sağlandığında, PCAD'in şartlara uygun verileri istek yapan kullanıcıya bildirmesini sağlayan yapılardır. Aşağıda bulunan ve filtre objesi içeren örnek bir JSON yükü gösterilmektedir.

```
{
    "sensors": ["id1"],
    "filter": {
          "id1": {
                "lowerThan": "12",
                "operator": "and",
                "greaterThan": "6"
          }
    },
       "mode":"real"

}
```

Buradaki "filter" objesi PCAD tarafından şu şekilde işleme alınmaktadır. "sensors" anahtarı ile ifade edilen array içindeki "id1" ID'li sensör bu ID ile "filter" objesinin içinde bulunan bu sensörün verileri üzerinde istenen filtre tanımlanır ve veri tabanına kaydedilir. "id1" ID'li sensörden veri geldiğinde veri tabanındaki filtreler kontrol edilir ve filtrede istenen veri ile sensörden gelen veri uyuyorsa ise kullanıcıya veri gönderilir.

Kural servisinin gerçekleştirilmesi için PCAD içerisinde özel bir ara yüz (*Ingilizce*: *interface*) yazılmıştır. Bu ara yüz kullanıcıdan gelen JSON verisindeki kural parametrelerini, programlama dilinde dinamik olarak kullanılabilecek bir yapıya çevirmektedir.

4.3.2 Veri Yönetim Servisi

Veri tabanı olarak NoSQL tipinde olan MongoDB kullanılmıştır. MongoDB ilişkisel bir veri tabanı değildir. MongoDB'de veriler, ilişkisel veri tabanlarındaki tablo yapısı yerine doküman yapısında saklanırlar. Doküman temelli veri tabanları, her anahtara karşılık gelen bir veri yapısı saklar. Bu veri yapısı, **array**, **string, sayı** veya iç içe geçmiş birden fazla dokümanlar da içerebilir. Aynı zamanda doküman temelli veri tabanları (MongoDB) dinamik olarak (*runtime* esnasında) var olan dokümana gelecek olan veride bulunan yeni bir sahanın eklenmesine, yeni bir doküman yapısı eklenmesine olanak sağlamaktadır. Bu özellikleri ile PCAD'in ilerde duyabileceği yeni gereksinimleri geliştirme de minimum çaba gerektirecek şekilde karşılayacaktır. PCAD'in veri tabanına ait, MongoDB tarafından saklanan doküman yapısı Tablo 13'de sunulmaktadır.

MongoDB Şemaları

MongoDB ilişkisel veri tabanlarından farklı olup doküman temelli bir yapıya sahiptir. Bu yapı özellikle veri yükü fazla olan sistemlerde performans anlamında katkı sağlamaktadır. MongoDB veri tabanı doküman şemaları aşağıda sunulmaktadır.

Tablo 13. MongoDB doküman yapısı

KULLANICILAR:
{
"userType" : "int",
"name" : "string",
"surname" : "string",
"companyName" : "string",
"valid" : "boolean",
"registerDate" : ISODate(""),
"userName" : "string",
"password" : "string",
"email" : "string",
"emailVerified" : "boolean"
}
	ROLLER:
	{
	"name" : "string",
	"information" : "boolean",
	"status" : "boolean",
	"data" : "boolean"
	"id": "string"
	}
	SENSÖR VERİLERİ
	{
	"sensorID" : "string",
	"data" : "",
	"timestamp" : ISODate("")
	}

KULLANICI SENSÖR ERİŞİM YETKİLERİ
{
"sensorID" : "string",
"userName" : "string",
"roleID" : "",
"valid" : "boolean"
}
	SENSÖRLER
	{
	"type": "string",
	"coordinate": "geopoint",
	"metric": "string",
	"valid": "boolean",
	"registerDate": "date"
	}

ERİŞİM_YETKİSİ
{
"ttl" : "number",
"created" : ISODate(""),
"userId" : ObjectId("")
}
	UYGULAMALAR:
	{
	"id": "string",
	"owner": "string",
	"name": "string",
	"registration_time": "date"
	}

UYGULAMA SENSÖR ERİŞİM YETKİLERİ
{
"sensorID": "string",
"appID": "string",
"roleID": "string",
"valid": "boolean"
}	

MongoDB klasik SQL sorgularına benzeyen sorgulama yöntemini desteklemektedir. Ancak birden fazla veri tabanı sistemi ile uyumluluk olması açısından veri yönetim servisinin geliştirilmesin de Nesne-İlişkisel Eşleme yazılım geliştirme tekniği kullanılmıştır. Bu teknik sayesinde veri tabanı tipinden bağımsız olarak veriler modeller ile eşlenmiş, uygulama seviyesinde bu modeller üzerinde işlem yapılmıştır. PCAD kullanıcıları “pcad_user” adında

bir modele eşlenmiş ve bu modelden ID'si "userId" değişkeninin değeri olan kullanıcı veri tabanından çağrılmıştır. Bu sayede veri tabanı tipi değişse dahi veriler modele eşitlendiğinden dolayı sorgu tipi hep aynı kalacak ve uygulama çalışmaya devam edecektir. Bu yapı ilerde yeni veri tabanı yapılarının sisteme eklenmesinde/değiştirilmesinde minimum çaba ve kod değişikliği ile sistemi ayağa kaldırmaya ve devreye sokmaya yardımcı olur. Örneğin bağlantı tipi "mysql" olarak değiştirildiğinde başka her hangi bir kaynak kod değişikliğine gerek kalmadan ilişkisel tabanlı bir veri tabanı yapısı (MySQL) kullanılmaktadır. Aşağıda kullanıcılar tablosunda ID'si userID olan kullanıcı sorgulaması sunulmaktadır.

```
pcad_user.find( {where : {_id: userId  }},
    function(err, userInstance) {
       if (err) {
          console.log(err);
       }
       console.log(userInstance);
    });
```

4.3.3 Uyarı ve Bildirim Servisi

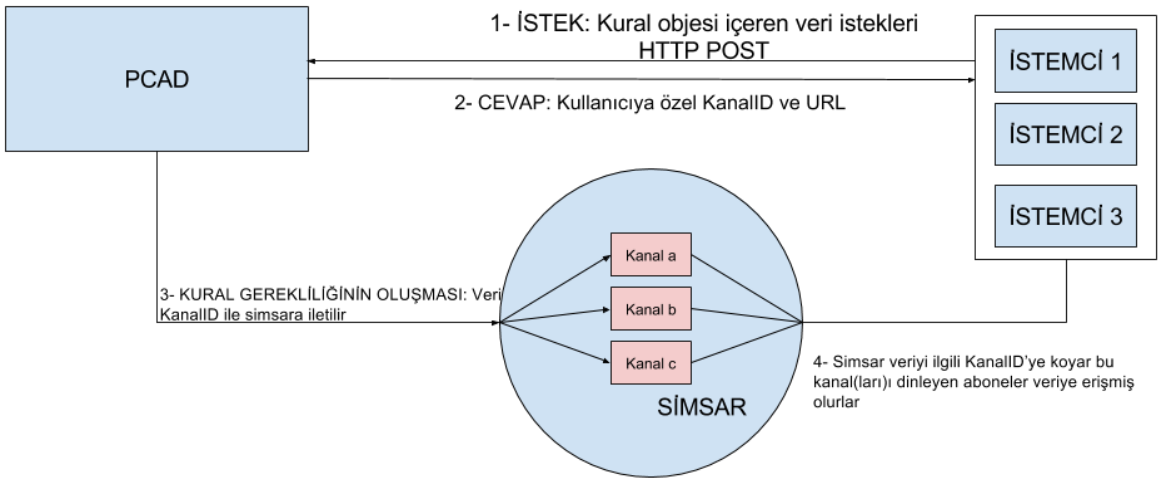
UBS kullanıcılarından kural içeren veri isteği geldiğinde bunu sisteme kaydetme ve kural gereklilikleri oluştuğunda ilgili veriyi istek yapan kullanıcılara iletmekle yükümlüdür. Kuralda yer alan gereklilikler belirsiz bir zaman aralığında gerçekleşebileceğinden dolayı, bu kullanıcı PCAD ile bağlı kalmak durumundadır. HTTP "*stateless protocol*" olduğundan ötürü kullanıcılar sisteme bağlantının aktif kaldığı başka bağlantı türü ile bağlanmalıdır. Örneğin websoket, yayıncı/abone, HTTP *long pulling* gibi. Her protokol kendi içinde avantaj dezavantaj içerse de UBS için en uygun protokol yayıncı/abone mimarisidir. Yayıncı/abone mimarisinin "*loose-coupling*" ve "*scalability*" gibi avantajları bulunmaktadır. PCAD "*loose coupling*" avantajı sayesinde herhangi bir abonenin veri alışı sırasında yaşadığı sorundan etkilenmeden aynı ya da farklı kanallara veri yollayabilir, diğer işlemleri gerçekleştirebilir konumda kalacaktır. Veri gönderim ihtiyacının artması durumunda ise "*scalability*" avantajı sayesinde PCAD çalışmaya devam ederken yayıncı/abone yapısı kaynak olarak genişleyebilir durumda olacaktır.

Yayıncı/abone mimarisinde mesajlar simsar olarak adlandırılan uygulama vasıtasıyla iletilir ve alınır. Hangi mesajın nereye yollanacağı ve kimin tarafından alınacağı KanalID ile belirlenir. Yayıncı mesaj iletmek istediği KanalID'yi ve veriyi belirlenen protokol ve port üzerinden simsara ulaştırır. Simsar bu KanalID'ye veriyi iletir ve bu sayede ilgili KanalID'ye abone olan tüm kullanıcılar mesajı almış olurlar.

Uyarı ve Bildirim Servisinin geliştirilmesinde Mosquitto simsar yazılımının kullanıldığı bu mimaride yayıncı görevini PCAD, abone(ler) rolünü istemci(ler) üstlenmektedir. Bir istemci

(kullanıcı), kural objesi içeren bir veri isteği yaptığında, PCAD'de Şekil 15'de sunulan işlemler gerçekleşmektedir.

1. İstek yapılan sensöre istek yapan kullanıcının erişim yetkisi kontrol edilmektedir.
2. Erişim yetkisi var ise kendisine kanalID ve URL içeren cevap döndürülmektedir (Şekilde 15'teki 2. durum), eğer yok ise kendisine "Erişim yetkiniz bulunmamaktadır" mesajı döndürülmektedir.
3. Kural gereklilikleri oluştuğunda istemci için oluşturulmuş KanalID ile ve veri ile simsara mesaj iletilmektedir.
4. İstemciler veriyi bu kanalID'den okumaktadır.



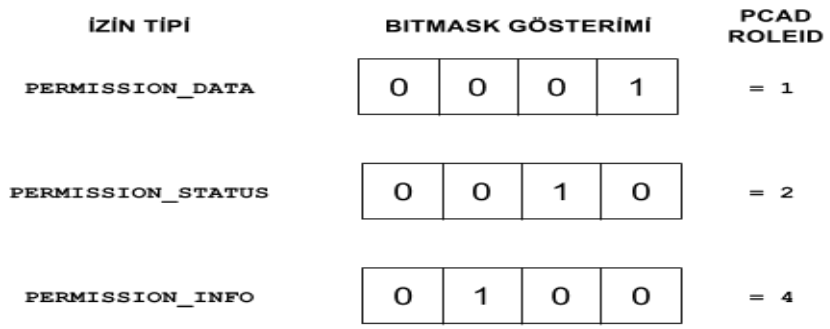
Şekil 15. Devamlı veri isteği için akış diagramı

4.3.4 Güvenlik ve Gizlilik Servisi

Güvenlik ve Gizlilik servisinin çalışması PCAD'e gelen erişim *token*'i ve kullanıcıların sensör bilgilerine erişim yetkilerinin kontrolü ile başlar. PCAD'e gelen her istek "*access_token*" değişkenini içermelidir (ilk kayıt hariç). Bu değişken kullanıcıya özel olarak üretilen yapıdadır ve kullanıcı hesabını oluşturduktan sonra giriş yaptığında kendisine verilir. Bir istek PCAD'e geldiğinde bu değişkene göre kullanıcı yetkilendirilir ya da erişimi engellenir. Kullanıcı yetkilendirmesi olduktan sonra kullanıcının talep ettiği sensör verilerine ve bilgilerine erişim izinleri kontrol edilir. Yetkisi bulunup bulunmamasına göre ilgili bilgiler döndürülür. Sistem üzerinde, bu kontrol mekanizmasında *bitmask* mekanizması kullanılmıştır. Bu mekanizma da izinler ikinin üstleri olarak tanımlanmış ve kullanıcıların sensörlerin veri, durum gibi bilgilerine erişimi kontrolü bu şekilde sağlanmıştır (Şekil 16). Bu mekanizma ile yeni yetkiler tanımlamanın kolaylaştırılması amaçlanmıştır. Örneğin, "test_user" kullanıcısı 123 nolu sensörünün hem verisini hem de bilgilerini okuma yetkisine sahip olması durumunda, bu

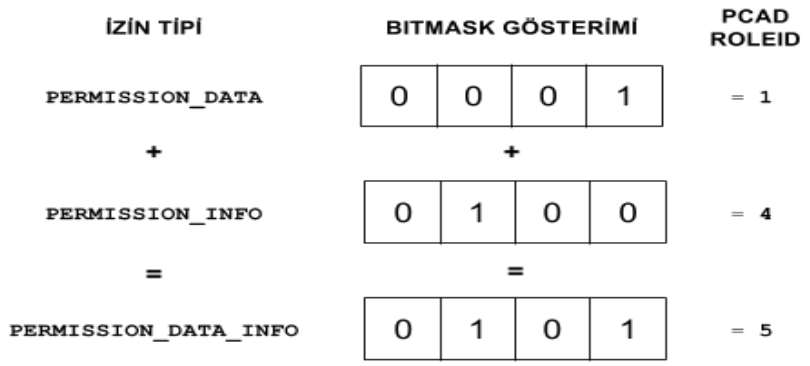
kullanıcının izni, veri tabanı roller tablosunda (Tablo 13) “id” ‘si 5 olan role karşı gelecektir. Veri tabanında kayıtlı bir rol örneği aşağıda verilmiştir.

```
{
  "name": "Info and Data Access",
  "information": true,
  "status": false,
  "data": true,
  "id": “5”
}
```



Şekil 16. PCAD izin tanımlarını gösteren bitmask’lar

Bu iznin gösterimi Şekil 17’de belirtilmiştir.



Şekil 17. Bilgi ve veri yetkilerine sahip test_user kullanıcı örneği

Bu kullanıcının bu sensör’e erişimi hakkında bilgi yetki tablosunda “roleID = 5” (Tablo 13) olarak belirtilecektir. Veri tabanında kayıtlı bir kullanıcı erişim yetkisi örneği aşağıda verilmiştir.

```
{
  "sensorID" : "123",
  "userName" : "test_user",
  "roleID" : "5",
  "valid" : false
}
```

4.3.5 Birlikte Çalışabilirlik ve İletişim Servisi

Bu servis Bölüm 4.1.5 de de anlatıldığı gibi HTTP istek/yanıt kullanarak gerçekleştirilmiştir. Kullanıcılar PCAD platformundan HTTP istek/yanıt kullanarak sensör verisi alabilmektedirler. Kullanıcıların platforma yapılan istekler ve gönderilen cevaplar JSON formatındadır. Bölüm 4.2.6'da REST-API kullanılması detaylı şekilde anlatılmıştır. HTTP istek/yanıt çok yaygın bir protokol olduğu için diğer platformların da PCAD platformundan veri isteğinde bulunması benzer şekilde mümkün olmaktadır. PCAD'dan veri isteyen diğer platformlar kendilerini sisteme kayıt ettirdikten sonra bir kullanıcı gibi HTTP istek/yanıt metodunu ya da kütüphane API'lerini kullanarak veri alabileceklerdir.

4.3.6 PCAD platformuna bağlanma arayüzleri

Sensörlerin PCAD platformu ile bağlantısı

PCAD ile sensörler HTTP REST API kullanarak iletişime geçerler. POST ve PATCH içeren metotlar JSON yükü alırken GET metotları herhangi bir yük almazlar. PCAD-Sensör bağlantı API'leri Tablo 14'de sunulmaktadır.

Tablo 14. Sensör Platform bağlantısı REST API'leri

URL	Metot	Yük	İşlev
/api/sensors?access_token=	POST	{ "type": "string", "coordinate": { "lat": 0, "lng": 0 }, "metric": "string" }	Sensör Kaydı
/api/sensors?access_token=	GET	-	Sensör Listeleme
/api/sensors/{id}?access_token=	GET	-	ID'si verilen sensör bilgilerini listeleme
/api/sensors/{id}?access_token=	PATCH	{ "type": "string", "coordinate": { "lat": 0, "lng": 0 }, "metric": "string" }	ID'si verilen sensör bilgilerini güncelleme
/api/sensorData?access_token=	GET	-	İzni bulunan sensör verilerini listeleme
/api/sensorData?access_token=	POST	{ "sensorID":"string",	Sisteme yeni bir sensör verisi kaydetme

		"data": "string", "timestamp": "DATETIME" }	

Uygulama ile PCAD platformu bağlantısı

PCAD ile uygulamalar HTTP REST API kullanarak iletişime geçerler. POST ve PATCH içeren metotlar JSON yükü alırken GET metotları herhangi bir yük almazlar. PCAD Uygulama bağlantı REST-API'leri Tablo 15'de sunulmaktadır.

Platform ile uygulama arasındaki bağlantı, Bölüm 4.1.3 belirtilen altı etkileşme mekanizması kullanılarak gerçekleştirilecektir. Bu etkileşme mekanizmalarının ortak tarafı hepsinde de istekle ilgili bilgilerin JSON formatında yollanmasıdır.

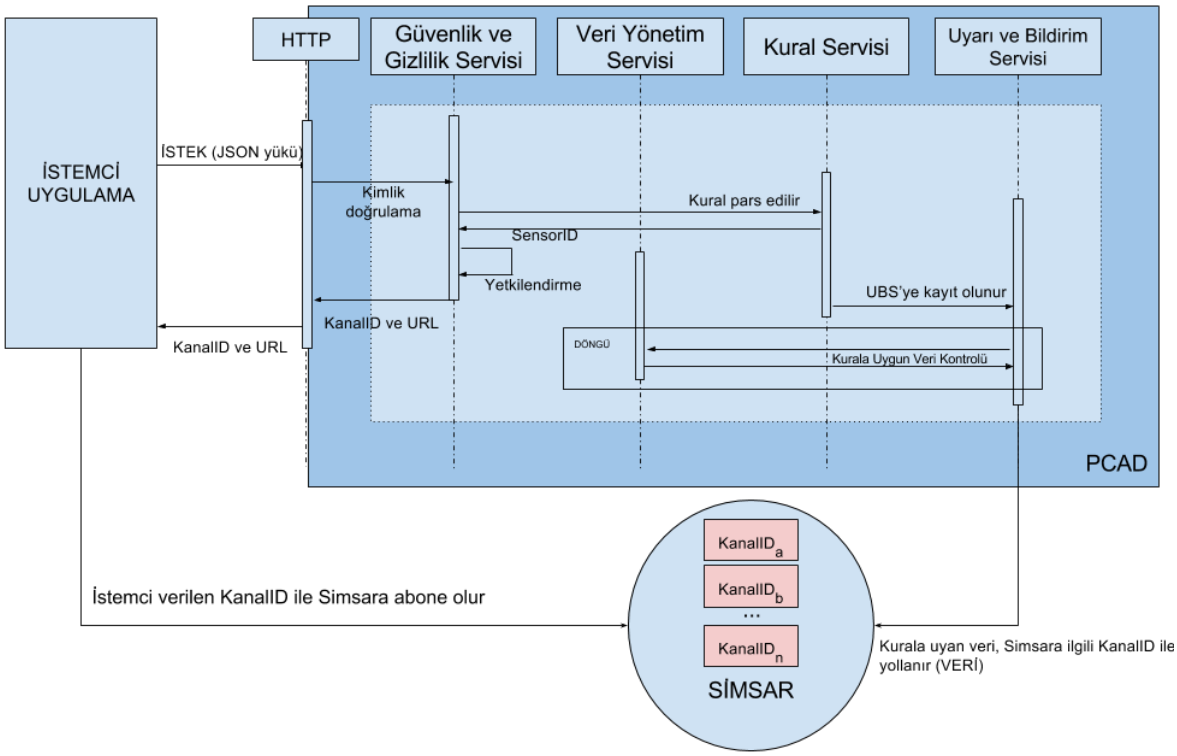
Sürekli veri alış verişi yapılan durumunda (A kodlu etkileşme mekanizmaları) uygulama ile PCAD arasında bir bağlantı kurulmalı ve uygulama ile PCAD istemci/sunucu modelinde çalışmalıdır. Bu modeli gerçekleştirmekte kullanılacak programla dilleri ve kütüphaneleri mevcuttur. Java'da bulunan "`org.eclipse.paho.client.mqttv3`", Node.JS'de "`mqtt`", C#'da "`uPLibrary.Networking.M2Mqtt`" kütüphanesi MQTT protokolü için kullanılan Mosquitto simsarı için bağlantı arayüzleri sağlamaktadır.

Sürekli veri alışverişi gerçekleştirilmesinde Java ve MQTT kütüphanesi kullanan bir örnek Bölüm 6.2.2'de verilmiştir. Tek seferlik veri alış verişi yapılan durumlarda (B kodlu etkileşme mekanizmaları) ise REST API, HTTP istek/yanıt ve JSON kullanılarak gerçekleştirilmiştir. Uygulamayla PCAD arasında sürekli veri alışverişini gösteren ardışık işlemler Şekil 18'de sunulmaktadır. Bu mekanizmada, filtreleme içeren bir isteğin PCAD'e ulaşması ve sonrasındaki işlemler göstermektedir. Filtre, HTTP üzerinden gelen istekte JSON yükünün içinde bulunur. Bu istek PCAD'e ulaştığında "access_token" değişkeni ile Güvenlik ve Gizlilik Servisinde kullanıcının kimlik doğrulaması yapılır ve kişinin sisteme erişim yetkisi kontrol edilir. Bu istek onaylandıktan sonra Kural Servisi filtreyi ayrıştırarak, verisi istenen sensör numarasının bulur. Takip eden adımda, ayrıştırılan sensör numarası Güvenlik ve Gizlilik Servisine gönderilir ve kullanıcının sensör bilgilerine erişim yetkisi kontrol edilir. Başarılı yetkilendirmenin ardından filtre veri tabanına kaydedilir ve kullanıcıya kanalID ve URL içeren bir cevap gönderilir. MQTT protokolü üzerinde Mosquitto simsarının çalıştığı bu URL'ye istemci kendini abone eder. Bu esnada PCAD Mosquitto üzerinde kullanıcıya iletilen kanalID ile bir kanal açar. Bu sayede kullanıcı ile PCAD arasında eş zamanlı tek yönlü çalışan bir oturum başlatılmış olur. UBS LoopBack içinde bulunan *operation hook*'lar sayesinde veri

tabanından okuma ya da sensör üzerinden gelen verinin direk kullanıcıya iletilme durumunu sağlamaktadır.

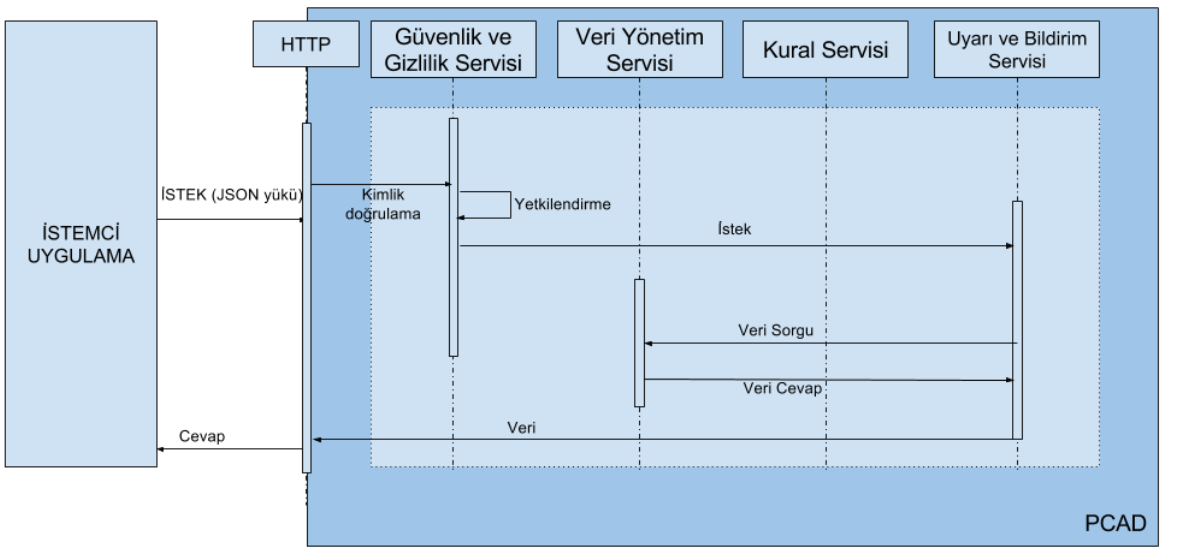
Tablo 15. Uygulama-Platform bağlantısı REST API'leri

URL	Metot	Yük	İşlev
api/pcadUsers?access_token=	GET	-	Kullanıcı bilgilerini listeleme
api/pcadUsers?access_token=	POST	{ "userType": "string", "name": "string", "surname": "string", "companyName": "string", "userName": "string", "email": "string", "password":"string" }	Yeni kullanıcı kaydı
api/pcadUsers/{id}?access_token=	PATCH	{ "userType": "string", "name": "string", "surname": "string", "companyName": "string", "userName": "string", "email": "string", "password":"string" }	{id} sahasında belirtilen ID'ye sahip kullanıcı bilgileirini günceller
/api/userSensorDataAccesses?access_token=	POST	{ "sensorID": "string", "userName": "string", "roleID": "string" }	Rol tanımlama
api/sensorData/get_data?access_token=	POST	{ "sensors": ["id"], "rules": { "id": { "lowerThan": "", "operator": "", "greaterThan": "" } }, "mode":"real/periodic" }	Sensör verisi isteme
/api/sensors?access_token=	GET	-	Sistemdeki sensörleri listeler



Şekil 18. Sürekli veri alış verişi sürecindeki ardışık işlemler

Uygulama ile PCAD arasında tek seferlik veri alış verişini gösteren ardışık işlemler Şekil 19'da sunulmaktadır. Bu mekanizmada, üstte sunulan A kodlu etkileşme mekanizmalarından farklı olarak MQTT protokolü ve Mosquitto simsar yazılımı yer almamaktadır. Tek seferlik veri alışverişinde, istek önce GGS tarafından değerlendirildikten sonra sırasıyla UBS ve VYS'yi takip ederek istek yapan kullanıcıya UBS tarafından cevap döndürülür.



Şekil 19. Uygulama ile PCAD arasında tek seferlik veri alış verişi sürecindeki ardışık işlemler

Uygulama ile PCAD arasında sürekli veri alışverişi durumu

Sürekli veri alış verişi yapılan durumda (A kodlu etkileşme mekanizmaları) uygulama, PCAD ile yetkilendirme safhasını bitirdikten sonra Mosquitto sunucusu ile istemci/sunucu modelinde çalışmaktadır. Arka planda PCAD, Mosquitto sunucusuna Abone-Yayıncı modelini kullanarak veri yollarken, Mosquitto sunucusu bu veriyi dinleme modunda olan uygulamaya yollamaktadır.

Uygulama tarafından tarafından yapılan isteğin içinde bulunan ve PCAD ile sürekli alış verişi sağlayacak JSON yüküne ait örnekler aşağıda sunulmaktadır.

1. Uygulama filtre belirtmeden sensörden doğrudan veri isteminde bulunur (**A1**).

```
{
     "sensors": [""],
     "mode":"real"
}
```

2. Uygulama filtre belirterek sensörden doğrudan veri isteminde bulunur (**A2**).

```
{
     "sensors": [""],
     "filter": {
           "": {
                 "lowerThan": "",
                 "operator": "",
                 "greaterThan": ""
           }
     },
     "mode":"real"
}
```

3. Uygulama filtre tanımlamadan veri tabanından veri isteminde bulunur (**A3**).

```
{
     "sensors": [""],
     "mode":"periodic"
}
```

4. Uygulama filtre belirterek veri tabanından veri isteminde bulunur (**A4**).

```
{
     "sensors": [""],
     "filter": {
           "": {
                 "lowerThan": "",
                 "operator": "",
                 "greaterThan": ""
           }
     },
     "mode":" periodic"
}
```

Uygulamaların simsar ile iletişimde bulunabilmek için MQTT protokolü kullanmaların olanak verecek bir kütüphaneye ihtiyaçları vardır. Java dilinde yazılmış böyle bir kütüphane mevcuttur. Bölüm 6.2.2'de yer alan dokümantasyonda bir uygulamanın simsar aracılığı ile

PCAD platformundan veri alması için kullanabileceği Java dilinde yazılmış örnek bir program verilmiştir.

Uygulama ile PCAD arasında tek seferlik veri alışverişi durumları

Aşağıda, Uygulama ile PCAD arasındaki veri alışverişinin tek seferlik olması durumuna ilişkin istekte bulunacak JSON yüküne ait örnekler sunulmaktadır.

1) Uygulama filtre belirtmeden bir kereye mahsus veri tabanından veri isteminde bulunur **(B1).**

```
{
     "sensors": [""],
}
```

Not: Bu istek herhangi bir JSON yükü kullanılmadan GET metotu ile de aşağıdaki şekilde yapılabilmektedir. Burada bulunan parametreler URL formatında kodlanmış olmalıdır.

```
/api/sensorData?filter=%7B%22where%22%20%3A%20%7B%22sensorID%22%3A%2
258f7b4a1927d01aafd365614%22%7D%7D&access_token=ofg8RPCxT3HJCpqY27Kr
2PIXV1B5q0WPgEY55EH6FDWTzCCOHbzGoUm2cLFJTBrg
```

URL formatında kodlanmış parametrelerin açılımı:

```
{"where" : {"sensorID":"58f7b4a1927d01aafd365614"}}&
access_token=ofg8RPCxT3HJCpqY27Kr2PIXV1B5q0WPgEY55EH6F
DWTzCCOHbzGoUm2cLFJTBrg
```

2) Uygulama filtre belirterek bir kereye mahsus veri tabanından veri isteminde bulunur **(B2)**.

```
{
     "sensors": [""],
     "filter": {
           "": {
                 "lowerThan": "",
                 "operator": "",
                 "greaterThan": ""
           }
     }
}
```

5. SİSTEMİN SINAMASI

5.1 Temel Test Süreci ve Uygulanan Yöntem

Bu bölümde PCAD platformunun test sürecinde uygulanılan yöntem açıklanmaktadır. Bu yöntem, IEEE Std 829-2008 standardında belirtilen temel test süreçleri esas alınarak oluşturulmuştur (IEEE Std. 829, 2008). Standardın yönlendirmesine göre, platformun test süreci; Planlama, Testlerin Tasarımı, Testlerin Çalıştırılması ve Sonuçların Analizi adımlarını içermektedir.

PCAD platformunun test edilmesindeki temel amaç, platformun yayınlanmadan önce, kendisinden beklenen fonksiyonları yerine getirip getirmediğinin anlaşılması ve platformdaki hataların tespit edilmesidir. Bu amaca uygun şekilde, dört temel testin yapılması kararlaştırılmıştır. Bunlar:

1. API testleri
2. GUI testleri
3. Entegrasyon testleri
4. Performans testleri

API testleri içerisinde platform için bağlantı ara yüzlerini sağlayan RESTful ara yüzlerinin test edilmesi amaçlanmıştır. GUI testlerinde, platformun web ara yüzlerinin testlerinin yapılması hedeflenmektedir. Entegrasyon testleri ile platform içerisinde yer alan servislerin birbirleriyle çalışmaları test edilmiştir. Performans testleri ile de geliştirilen platformun genel performansının test edilmesi planlanmıştır. Yukarıda sunulan testlerin yapılabilmesi için tasarım aşamasında, her bir test grubu için test senaryoları (*İngilizce: test case*) geliştirilmiş, bu senaryolara ilişkin girdiler, operasyonlar ve çıktılar tanımlanmıştır.

Tasarım sürecini takip eden adımda ise takip eden bölümde sunulduğu şekilde testler çalıştırılmış, elde edilen sonuçlara göre gerekli düzeltmeler ve iyileştirmeler yapılmış ve belgelendirilmiştir. Böylece, test süreci tamamlanmıştır.

5.2 Testlerin Tasarımı

Bu bölümde yapılmış olan dört temel testin tasarımı anlatılmaktadır.

5.2.1 API Testleri

Bu bölümde, PCAD platformunun RESTful ara yüzleri içerisinde sunulan uygulama programlama ara yüzlerinin genel test amaçlarına uygun şekilde somut test koşullarına ve test senaryolarına dönüştürülmesi anlatılmaktadır. Bunun için öncelikle test tasarım spesifikasyonları yazılmış ve bunlar ile ilgili amaç, ayarlar ve girdi formatları belirlenmiştir. Her bir spesifikasyon için birden fazla test senaryosu üretilmiştir. Her bir senaryo için örnek girdiler belirlenmiş ve beklenen çıktılar belgelenmiştir.

Sağlanan programlama ara yüzü RESTful olduğundan dolayı, HTTP protokolü istek yapmak için kullanılmak zorundadır. Bir sonraki kısımda, test tasarım spesifikasyonları ilgili tablolar "Test - XX" olarak adlandırılmış olup, HTTP isteklerinin nasıl yapılacağına ilişkin metot, URL, sorgu parametreleri ile yük değerlerin formatlarını içermektedir. Testlerde amaç, metot ve yük parametreleri sabittir. Gereksiz tekrarlardan kaçınmak için, bu bilgi sadece testlerde sunulmakta, dolayısıyla test senaryolarında gösterilmemektedir. Sabit olmayan parametreler ise, her bir test senaryosu için örneklendirilmiştir. Sorgu parametreleri, URL içerisinde verilecek girdileri ifade etmektedir ve buna ihtiyaç duyulmaması halinde “-” ile gösterilmektedir. Ayrıca, yük kısmında gönderilecek yükün JSON formatını tanımlamaktadır. Test senaryolarında verilen girdiler örneklendirilerek sunulmuş ve beklenen çıktılar da aynı formatta belirtilmiştir. Böylelikle, verilen test senaryoları standardize edilerek tamamlanmıştır.

Test senaryolarının sağlıklı olarak tamamlanması için öncesinde yapılması gerekenler tespit edilmiştir. Testler öncesinde veri tabanında gerekli veriler yaratılmıştır. Bu veriler, sağlanmış olan `init_test_db.sh` isimli betiğin çalıştırılmasıyla oluşturulmuş. Tablolarda yaratılan verilerin sayıları verilmiştir. `users`, `user_permissions`, `application_permissions` tablolarında birer, `sensors`, `applications`, `roles` tablolarında ikişer adet örnek veri bulunmaktadır. Bu sayılar, testler senaryolarında kullanılmıştır. Ayrıca, test senaryolarında verilmesi gereken `TRUE_SESSION_TOKEN` iki numaralı testte sağlanan metotla elde edilmektedir. Cevap olarak gelen `SESSION_TOKEN` `TRUE_SESSION_TOKEN`’e karşılık gelmektedir. Böyle yazılmasının nedeni, bu değerin dinamik olarak şifrelendiği için sabit bir değerinin olmamasıdır. `WRONG_SESSION_TOKEN` ise herhangi bir dizgi (*string*) değeri olabilmektedir.

API testlerine ait test senaryoları Ek-C’de sunulmaktadır.

5.2.2 GUI Testleri

Bu bölümde, platform için hazırlanan web ara yüzlerinin testi için oluşturulan test senaryoları sunulmaktadır. Test senaryolarının geliştirilmesinde, temel olarak ara yüzlerin doğru çalışıp çalışmadığı ve dolayısıyla kendilerinden beklenen fonksiyonları yerine getirip getirmediklerinin testi amaçlanmıştır. Bunun için öncelikle platformun web ara yüzlerinin ekran görüntüleri alınmıştır ve hemen takibinde fonksiyonelliği sınayan test senaryoları yazılmıştır.

Ara yüzlerdeki başta kozmetik özellikler olmak üzere çeşitli özelliklerin test edilmesi için başka test senaryolarının da yazılması mümkündür. Ancak bu noktada kozmetik hataların testi için test senaryoları üretilmemiştir. Bunun yerine geliştiriciler, hızlı bir şekilde ekran görüntülerini test senaryosu üretmeksizin görsel olarak test etmişlerdir. Bu testler sırasında özellikle yazı karakterleri, fontları ve renkleri, butonların rengi ile ekran görüntüsünün genel durumu ve düzeni esas alınmıştır.

GUI testlerine ait test senaryoları Ek-D'de sunulmaktadır.

5.2.3 Entegrasyon Testleri

Entegrasyon testleri ile platform içerisinde çalışan servisleri birbirleriyle uyumlu çalışması test edilmesi planlanmıştır. Daha önceki raporlarda belirtildiği üzere platform için beş servis tanımlıdır: VYS, UBS, GGS, KS ve BÇS. Burada amaçlanan, servislerin birbirleriyle çalışmalarını test etmektir. Ancak platform içerisindeki her servis her servisle iletişime geçmemektedir. Tablo 16'de servislerin kullandığı REST-API *endpoint*'leri sunulmaktadır. Bu tabloda, bir istek geldiğinde, yapılan isteğe bağlı olarak çalışan servisler listelenmektedir. Örneğin, ilk satırda yer ala URL, kullanıcı kaydı işlemini yapar ve tabloda görüleceği üzere, sorgu parametresi almaz, yük ile birlikte POST isteği yapılmaktadır. Bu istek için GGS ve VYS servislerinin işaretli olduğunu ve sonuç olarak, bunların birlikte çalıştığını anlayabiliriz. Yazılan her bir test senaryosu, ancak yapılan her işlem doğru sonuçlanırsa başarılı olmaktadır. Bu açıdan değerlendirildiğinde, önceki bölümde tasarlanmış test senaryoları entegrasyon testlerine karşılık gelir. Bu servislerin en basit yöntemiyle test edilmesi, yani entegrasyon testleri, API testleri sayesinde sağlanmaktadır ve yapılan her bir HTTP isteği bir veya birden fazla servisin çalışmasıyla cevaplanmaktadır. Bu yöntemle entegrasyon testleri tamamlanmaktadır.

Tablo 16. API testleri ve servislerin birbirleriyle etkileşimi

URL	Metot	Sorgu	Yük	GGS	VYS	KS	UBS
/auth/signup	POST	-	+	+	+	-	-
/auth/signin	POST	-	-	+	+	-	-
/auth/signin	POST	+	-	+	+	-	-
/auth/signout	GET	+	-	+	-	-	-
/users	GET	+	-	+	+	-	-
/users	POST	+	+	+	+	-	-
/users/:id	GET	+	-	+	+	-	-
/users/:id	PUT	+	+	+	+	-	-
/users/:id	DELETE	+	-	+	+	-	-
/sensors	GET	+	-	+	+	-	-
/sensors	POST	+	+	+	+	-	-
/sensors/:id	GET	+	-	+	+	-	-
/sensors/:id	PUT	+	+	+	+	-	-
/sensors/:id	DELETE	+	-	+	+	-	-
/sensors/stream	GET	+	-	+	+	+	+
/applications	GET	+	-	+	+	-	-
/applications	POST	+	+	+	+	-	-
/applications/:id	GET	+	-	+	+	-	-
/applications/:id	PUT	+	+	+	+	-	-
/applications/:id	DELETE	+	-	+	+	-	-
/permissions/users	GET	+	-	+	+	-	-
/permissions/users	POST	+	+	+	+	-	-
/permissions/users/:id	GET	+	-	+	+	-	-
/permissions/users/:id	PUT	+	+	+	+	-	-
/permissions/users/:id	DELETE	+	-	+	+	-	-
/permissions/applications	GET	+	-	+	+	-	-
/permissions/applications	POST	+	+	+	+	-	-
/permissions/applications/:id	GET	+	-	+	+	-	-
/permissions/applications/:id	PUT	+	+	+	+	-	-
/permissions/applications/:id	DELETE	+	-	+	+	-	-
/sensors/:id/status	GET	+	-	+	+	-	-
/sensors/:id/info	GET	+	-	+	+	-	-
/sensors/:id/data	GET	+	-	+	+	-	-
/sensors/:id/data	POST	+	+	+	+	-	-
/sensors/:id/data/filter	POST	+	+	+	+	-	-

5.2.4 Performans Testleri

Bu bölümde performans ile ilgili yapılan testlerin sonuçları sunulmaktadır. İkinci proje raporunda, ilk olarak platformun henüz geliştirilme aşamasında yapılan test sonuçları sunulmuştur. Bu raporda ise platformun son haliyle yapılan testlerin sonuçları sunulmaktadır. Böylelikle projenin iki farklı aşaması hakkında bilgi edinmek mümkün olmaktadır. Tablo 17'de platformun başlangıç ve geliştirme sonrasındaki performansı gönderilen verinin gönderilme ile veri tabanına yazılma arasındaki farkı olarak sunulmaktadır.

Tablo 17. Performans testi sonuçları

# Sensör	Toplam Veri	Süre(*sn*)	Başlangıç - fark (*ms*)	Geliştirme sonrası – fark(*ms*)
1	30	60	2	3
10	300	60	49	56
50	1.500	60	51	52
100	3.000	60	52	51
250	7.500	60	52	60
375	11.250	60	50	66
500	15.000	60	85	87

5.3 Testlerin Çalıştırılması

API testleri için toplamda 87 test senaryosu, GUI testleri için toplam 69 test senaryosu tasarlanmıştır. Testleri çalıştırmak için Java ve Scala için web uygulama iskeleti Play Framework (Play Framework) içerisinde sağlanan test kütüphanelerinden faydalanılmıştır. Kütüphaneler sayesinde, programsal olarak taklit HTTP istekleri oluşturulup, API testleri yapılmıştır. GUI testleri de benzer şekilde tamamlanmıştır, test kütüphanesi sayesinde, program yordamıyla bir tarayıcı açılarak, bunun üzerinde ara yüz olayları (*İngilizce:event*) taklit edilebilmektedir. Örneğin, butona tıklama işlemi bunlardan bir tanesidir. Testler yukarıda sunulduğu gibi yazılmış ve çalıştırılmıştır.

5.4 Testlerin Analizi ve Test Sonuçları

Bu bölümde testlerin çalıştırılması sonucunda bulunan hataların nedenleri ve kök sebepleri araştırılmıştır. Yazılımın geliştirilmesini takiben yapılan fonksiyonel testlerin sonuçları Tablo 18'de sunulmaktadır. Test senaryolarının ilk çalıştırılması sonucunda, API testlerinden 14 test senaryosu, GUI testlerinde de 10 test senaryosu başarısızlıkla sonuçlanmıştır. API testlerindeki bu hataların tamamı JSON yükü gerektiren HTTP isteklerinden kaynaklanmıştır.

Bu istekler içerisinde gelen yükler platformda ayrıştırılırken, yükte en ufak bir hata olduğunda, sistem hata vererek HTTP 500 kodlu hata cevabı vermiştir. Asıl verilmesi gereken 400 kodlu HTTP cevabıdır. Bunun çözümü için ayrıştırma yapılan kısımlar tekrar gözden geçirilerek gerekli iyileştirmeler yapılmıştır. Hataların düzeltilmesini takiben, tester tekrardan çalıştırılmış ve elde edilen sonuçlar Tablo 19'de sunulmuştur. PCAD platformu kabul testlerinden başarılı bir şekilde geçmiştir.

Tablo 18. Testlerin ilk çalıştırma sonuçları

Testler	Planlanan	Çalıştırılan	Başarılı	Başarısız
API	87	87	73	14
GUI	69	69	59	10

Tablo 19. Kabul testi sonuçları

Testler	Planlanan	Çalıştırılan	Başarılı	Başarısız
API	87	87	87	0
GUI	69	69	69	0

GUI testlerindeki hatalar ise form girdilerinden kaynaklanmaktadır. Ayrıca, yapılan eklemeler, bildirim mesajlarıyla desteklenerek kullanıcı deneyimini arttırmak amaçlanmaktadır. Bunun için ilgili sayfalardaki girdi doğrulama mekanizmalarına eklemeler yapılarak iyileştirmeler yapılmıştır.

Performans test sonuçlarından görüldüğü üzere (Tablo 17), çok küçük miktarlarda yazma ve gönderme değerlerinde milisaniye cinsinden yavaşlamalar görülmektedir. İlk testin yapıldığı zamandaki platformun hali ile son testin yapıldığı haliyle, çok fazla değişiklik ve eklemeler vardır ve bu da yavaşlamanın bir sebebi olarak yorumlanmaktadır. Bu farklar platformun çalışmasında büyük farklara sebep olmayacağı düşünülmektedir.

Sonuç olarak, yapılan testlerle sistemdeki hatalar bulunarak, gerekli düzeltmeler yapılmıştır. Düzeltmeler sonrasında test senaryoları tekrar çalıştırılarak sistemin, tanımlanan test senaryoları açısından hatasız olarak çalıştığı saptanmıştır. Performans testlerinin, sistemin tasarımı sırasındaki durumu ile kıyaslandığında farklılıklar olduğu ölçülmüştür. Ancak bu farklar sistemin genel işleyişi açısından tehdit oluşturmamaktadır.

6. PCAD GERÇEKLEŞTİRMESİNİ DEVREYE ALMA VE DOKÜMANTASYON

Bu kısımda AKKA ve Node.JS ile yapılan gerçekleştirmeyle ilgili kullanıcı dokümantasyonu yer almaktadır.

6.1 AKKA Gerçekleştirmesini Devreye Alma ve Dokümantasyon

Bu bülümde AKKA kullanılarak yapılan gerçekleştime ile ilgili devreye alma ve dokümantasyon bilgisi verilmektedir.

6.1.1 AKKA ile gerçekleştirilen sistemin kurulumu

Bu bölümde, İP5'e uygun olarak, sistemin nasıl kurulacağına yönelik yapılan çalışmalar sunulmaktadır.

Sistemin kurulumu için iki seçenek vardır. Birincisi sistemin tam sürümünün kurulumu, ikincisi geliştirme yapmak için yapılan kurulum. Geliştirme yapmak için yapılacak kurulum EK-B'de verilmiştir. Kurulum dosyasına ufuk.celikkan@ieu.edu.tr adresine elektronik posta gönderilerek ulaşılabilir.

Tam Sürüm Kurulumu:

Tam sürüm kurulumu, sadece sistemin son halinin alınıp çalıştırılmasını içerir ve bu sistem üzerine yeni eklemeler yapılmasına imkân sağlamaz. Bu kurulum, en az kurulum çabasını gerektirmektedir. Bunun yanında isteğe bağlı olarak `.conf` dosyası içerisinden değişiklikler yapılabilir. Örneğin, PCAD'in çalışacağı port numarasını değiştirmek.

Gereksinimler:

- Linux Ubuntu 14.04
- Java 8 - http://www.oracle.com/technetwork/java/javase/downloads/index.html
- mysql 5.7 - https://dev.mysql.com/downloads/

Gerekli yazılımlar kurulduktan sonra aşağıdaki komutlar ile kontrol edilmelidir. 'x' minör sürümü belirtir, değişkenlik gösterebilir, çalışmada etki yapmaz.

```
$ java -version
```

Çıktı: `java version "1.8.x"`

```
$ mysql -version
```

Çıktı: `mysql 5.7.x`

Sunulan *pcad-play-1.0.*zip dosyasının içerisinde Linux işletim sisteminde çalışan *bin/pcad-play* isimli betik bulunmaktadır (Tablo 20). Ayrıca `conf` dizini içerisinden `application.conf` dosyasından veri tabanı için kullanıcı adı, şifre eklenmelidir.

Tablo 20. Kurulum dosyası pcad-play-1.0.zip içeriği

./	./samples:
|-- bin |-- conf |-- lib |-- logs |-- out |-- README.md |-- samples |-- init.sh	|-- api_01_app_real_time.py |-- api_02_app_real_time_rule.py |-- api_03_app_non_real_time.py |-- api_04_app_non_real_time_rule.py |-- api_05_app_one_time.py |-- api_06_app_one_time_filter.py |-- api_sensor.py |-- application_config.json |-- application_rest.py |-- connection.py |-- example.rl |-- rest_01_app_real_time.py |-- rest_02_app_real_time_rule.py |-- rest_03_app_non_real_time.py |-- rest_04_app_non_real_time_rule.py |-- rest_05_app_one_time.py |-- rest_06_app_one_time_filter.py |-- rest_sensor.py |-- sensor_config.json
./conf:	
|-- application.conf |-- logback.xml |-- routes	
./bin/logs	
|-- application.log	
./bin	
|-- logs |-- pcad-play	
./logs	
|-- application.log	
./lib	
Uzun bir liste. Play, database, angular gibi yazılımları barındırır.	

Zip dosyası içerisindeki init.sh ilk olarak çalıştırılmalıdır. Bu bir adet veritabanı şeması yaratmaktadır.

```
$ chmod +x init.sh
$ ./init.sh
```

Bunu takiben `bin` dizini içerisinde bulunan `pcad-play isimli` betik komut satırı üzerinden şekilde gösterildiği gibi çalıştırılır. Verilen argüman, uygulama içerisinde şifrelemeler için kullanılacak olan değişkene karşılık gelir.

```
$ pcad-1.0/bin/pcad-play -Dplay.crypto.secret=ABCD
```

PCAD böylelikle tam sürüm halinde çalışmaya başlayacaktır. Sistemin bir alan adı ile birlikte dünya ile iletişimi halinde olabilmesi için `nginx` yüklenmesi gerekir. `nginx` PCAD'in ihtiyacı olan web sunucusu işlevini sağlamaktadır. Aynı zamanda standard 80 portuna gelen http isteklerini PCAD'in çalıştığı porta yönlendirmektedir. Yükleme takip eden linkten yapılabilir: *http://nginx.org/en/download.html*

Linux için yapılması gerekenler, `/etc/nginx/sites-available/default` dizinli dosyayı açıp aşağıdaki gibi değiştirilmelidir.“`domain.com`” yerine sahip olunan adı yazılır. Diğer “`localhost:9000`” ise PCAD sisteminin çalışmaya başladığı makine ve port numarasını belirtir. Eğer varsayılan değerler değiştirilirse bunlar da değiştirilmelidir.

```
server {
    listen 80;
    server_name domain.com;
    location / {
        proxy_pass http://localhost:9000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

6.1.2 Programlama Arayüzü ve Örnekler

Platform bağlantı API’leri

Bu bölümde sensör yazılımlarının platforma veri yollarken ve uygulamaların platformdan veri alışlarında kullancakları API tanımları verilmiştir. Bu API kullanıcı kılavuzu İngilizce olarak hazırlanmıştır.

Uygulamalar ile Sensörler tarafından ortak kullanılan API’ler

Name

`pcad_init_context` - Creates the context to which will be used when calling library APIs.

Synopsis

```
Context context = pcad_init_context(String context_path)
```

Description

It accepts a configuration file as a parameter and initializes the sensor connection interface. The configuration file in JSON format contains information about sensor which includes access token for authentication.

Parameters

`context_path`: The location of the configuration file.

Return Value

It returns a context for to be used subsequent API calls.

Usage

```
1   String context_path = "/opt/pcad/config/app_config.json"
2   Context context = pcad_init_context(context_path)
```

Name

`pcad_free_context` - It frees context.

Synopsis

```
void pcad_free context(Context context)
```

Description

It deletes context object.

Parameters

`context`: The context acquired as a result of `pcad_init_context` call.

Return Value

It returns void, it only logs.

Usage

```
1   String context_path = "/opt/pcad/config/app_config.json "
2   Context context = pcad_init_context(context_path)
3   pcad_free_context(context)
```

Name

`pcad_get_credentials` - Retrieves session credentials.

Synopsis

```
Context pcad_get_credentials (Context context)
```

Description

It takes context object as an argument to retrieve session credentials from PCAD. It returns the modified context object augmented with the session credentials. Unlike access_token, session credential has duration. When it expires new credentials must be acquired.

Parameters

`context`: The context acquired as a result of `pcad_init_context`_call. At the end of the call context will be modified with the session credentials.

Return Value

It returns a secure context that contains session credentials.

Usage

```
1   String context_path = "/opt/pcad/config/app_config.json"
2   Context context = pcad_init_context(context_path)
3   Context secure_context = pcad_get_credentials(context)
```

PCAD Platformu ile Uygulama Arasındaki Bağlantı API'leri

Bu API kütüphanesi aracılığı ile uygulamalar platform ile etkileşimde bulunurlar. Etkileşim Mekanizmaları bu API'ler kullanılarak yapılmaktadır.

Name
`pcad_init_connection` - It opens a connection between PCAD and application which will be used for continuous data transfer.
Synopsis
`Connection connection = pcad_init_connection(Context secure_context)`
Description
`pcad_init_connection()` opens a connection with PCAD.
Parameters
The secure context acquired as a result of `pcad_get_credentials` call
Return Value
It returns connection instance which data will be received via this.
Usage

```
1   Context secure_context = pcad_get_credentials(context)
2   Connection connection = pcad_init_connection
(secure_context)
```

Name
`pcad_on_message` - It receives incoming messages from PCAD.
Synopsis
`int pcad_on_message(Connection connection, Function callback)`
Description
It listens incoming messages coming through connection instance. It cannot be used without a connection instance. Therefore, firstly, a connection must be established.
Parameters
`connection`: It is the connection handle that the applications receive messages and data. The connection handle is initialized by using `pcad_init_connection` method.
`callback(connection, message)`: callback function is taken as parameter. This function is written by the application developer. It will be registered to connection instance for listening messages. Method will take two parameters and the first one is connection handle, and other one is message received. The function uses message parameter to access to the data.
Return Value
-1: for error
non-negative: Success
Usage

```
1   String source = {mode: 'periodic', sensor: 51 ,interval:
5}
2   Connection connection = pcad_init_connection(source, null)
3   pcad_attach_sensor(connection,source,null)
4   pcad_on_message(connection, foo)
5
6   void foo(conn, message){
7      printf ('%s', message)
8   }
```

Name

`pcad_attach_sensor` – Binds the application to particular sensor for data retrieval.

Synopsis

```
void pcad_attach_sensor(Connection connection, JSON source,
                        String rule_file)
```

Description

`pcad_attach_sensor()`- Binds the application to a particular sensor for data retrieval. The applications use this call to get data and notifications for a particular sensor.

Parameters

`connection`: It is the connection handle that the applications receive messages and data. The connection handle is initialized by using `pcad_init_connection` method.

`source`: The source of data. It is a JSON formatted object which specifies the data source. It has three fields: mode of transfer, sensor id and interval. A valid JSON source object is given in the following code snippet.

```
1   source = {
2       mode : 'real' | 'periodic',
3       sensor_id : long,
4       interval : int // minute value and only required for
periodic
5   }
```

The mode field indicates if the data is to be sent in soft real time or not. If this field is set to '`real`', platform sends data as soon as it becomes available (i.e. delivered by the sensor) . When this field is set to '`periodic`' data will be delivered at predefined intervals from the database.

`rule_file`: It is a rule file to be interpreted by PCAD. If no rule file is given then this field must be specified as null.

Return Value

void

Usage

```
1   String source = {mode: 'real' , sensor_id: 23}
2   Connection connection = pcad_init_connection
(secure_context)
3   pcad_attach_sensor(connection,source,null)
```

Name

`pcad_terminate_connection` - Terminate connection.

Synopsis

```
int pcad_terminate_connection(Connection connection)
```

Description

Connection instance will be destroyed and communication will be ended.

Parameters

`connection`: It is handle of the connection to be destroyed.

Return Value

-1: for error

non-negative: Success

Usage

```
1   Context secure_context = pcad_get_credentials(context)
2   Connection connection =
pcad_init_connection(secure_context)
3   int result = pcad_terminate_connection(connection)
```

Name

`pcad send query` - It runs query on PCAD's sensor data.

Synopsis

```
String pcad_send_query(Context secure_context, long sensor_id,
JSON query)
```

Description

It is used for one time data requests. Querying pastime data is one of the use case for this function.

Parameters

`secure_context`: The secure context acquired as a result of `pcad_get_credentials` call

`sensor_id`: Sensor idç

`query`: It is a JSON object that filter data in database. It must be composed as following.

```
query = {
    value: {
      min: Float,
      max: Float
    }
    time: {
      min: Timestamp,
      max: Timestamp
     }
}
```

Return Value

It returns JSON response which consists of related sensor data. It may return empty JSON object, if no data to be found.

Usage

```
1  String query = "{
2     \"value\": null,
3     \"time\":     {
4        \"min\": "2016-12-02T23:39:30.0+03:00",
5        \"max\": "2016-12-07T23:39:30.0+03:00"
6     }
7  }"
8  String response = pcad_send_query(context, 1, filter);
```

PCAD Platformu ile Sensör Bağlantı API'leri

Name

`pcad_post_data` – Posts data to the platform.

Synopsis

```
int pcad_post_data(Context context, Data data)
```

Description

Data sources use this API to post data to the platform. It accepts a context and data in JSON format and sends the data to the platform.

Parameters

`context`: Configuration parameters which will be utilized for authentication.

`data`: It consists of sensor measurement and complementary information. It is a JSON formatted object as shown below.

```
data = {
  sensor_id : long,
  time : string,
  value: float
}
```

Return Value

-1: for error

non-negative: Success

Usage

```
String context_path =
"/opt/pcad/config/sensor_config.json"
Context context = pcad_init_context(context_path)
Context secure_context = pcad_get_credentials(context)
String data = {sensor_id: 1 , time:"2016-09-20", value:
23.2}
int response = pcad_post_data(secure_context, data)
pcad_free_context(context)
```

PCAD-Sensör İletişimi PYTHON Örnek Programları

Sensörler platforma veri yollamaya başlamadan önce bir erişim token'i almalıdırlar. Erişim token'ı bir JSON dosyası içerisinde platforma yollanır. Program 5 sensörün PCAD platformuna veri yollamısnı göstermektedir.

```
# -*- coding: utf-8 -*-
import datetime
import random
from connection import Connection
from dateutil.tz import tzlocal

if __name__ == "__main__":
    INSTANCE = Connection()
    CONTEXT = INSTANCE.pcad_init_context("sensor_config.json")
    SECURE_CONTEXT = INSTANCE.pcad_get_credentials(CONTEXT)
    NOW = datetime.datetime.now(tzlocal())
    CURRENT_TIME = NOW.strftime('%Y-%m-%dT%H:%M:%S.%f%z')
    RANDOM_VALUE = round(random.uniform(-10.0, 20.0), 2)
    PAYLOAD = {'sensor': 15, 'time': CURRENT_TIME, 'value': RANDOM_VALUE}
    RES = INSTANCE.pcad_post_data(SECURE_CONTEXT, PAYLOAD)
```

Program 5. Python API kullanan sensör programı *(api_sensor.py)*

PCAD-Sensör İletişimi REST-API Örnek Programları

Sensörler platforma veri yollamak için arayüz kütüphanesi yerine REST API metotunu kullanarak platforma doğrudan istek yollayabilirler. Program 6 böyle bir iletişimi göstermektedir.

```
# -*- coding: utf-8 -*-
import requests

AUTH_URL = "http://localhost:9000/api/v1/auth/signin"
QUERY_STRING =
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJQQ0FEIiwidWl
kIjoxNSwiZ3JvdXAiOiJzZW5zb3IifQ.8ifEzEbLGwMWvs__IvGXKX9Eh056HWZYXHkrcytcoPY"}
HEADERS = {'content-type': 'application/json'}
RESPONSE = requests.request("POST", AUTH_URL, data="{}", headers=HEADERS,
params=QUERY_STRING)

DATA_URL = "http://localhost:9000/api/v1/sensors/19/data"
QUERY_STRING = {"access_token":RESPONSE.content.replace("\"", "")}
PAYLOAD = "{\n  \"sensor\": 19,\n  \"time\": \"2019-02-10T10:54:57.0+03:00\",
\n  \"value\": 1.11\n}"
RESPONSE = requests.request("POST", DATA_URL, data=PAYLOAD, headers=HEADERS,
params=QUERY_STRING)
print(RESPONSE.text)
```

Program 6. REST-API kullanan sensör programı *(rest_sensor.py)*

PCAD- Uygulama İletişimi PYTHON Örnek Programları

A1, A1, A3, A4, B1 ve **B2** Etkileşim mekanizmalarının nasıl yapıldığına dair örnek programlar aşağıda verilmiştir. Bu programlar platformun sağladığı PYTHON kütüphanesi aracılığı ile yapılmaktadır.

PCAD-Uygulama Etkileşim Mekanizması B1

Program 7'de herhangi bir filtre belirlemeden veri tabanından bir kerelik sorgu yapan bir program sunulmaktadır. Bu etkileşim mekanizmasında devam eden bir bağlantı yoktur.

```
# -*- coding: utf-8 -*-
from connection import Connection

if __name__ == "__main__":
    try:
        INSTANCE = Connection()
        CONTEXT = INSTANCE.pcad_init_context("application_config.json")
        SECURE_CONTEXT = INSTANCE.pcad_get_credentials(CONTEXT)
        result = INSTANCE.pcad_send_query(SECURE_CONTEXT, 1)
        print result
    except (Exception, KeyboardInterrupt, SystemExit) as e:
        print "connection killed " + str(e)
```

Program 7. Etkileşim Mekanizması B1 *(api_05_app_one_time.py)*

PCAD-Uygulama Etkileşim Mekanizması B2

Program 8, uygulamanın veri tabanından belirli bir zaman aralığında, gerçek zamanlı olmayan ve bir kereye mahsus filtreli bir veri sorgulaması yapmasını göstermektedir. Bu etkileşim mekanizmasında devam eden bir bağlantı yoktur.

```
1  # -*- coding: utf-8 -*-
2  from connection import Connection
3
4  if __name__ == "__main__":
5      try:
6          INSTANCE = Connection()
7          CONTEXT = INSTANCE.pcad_init_context("application_config.json")
8          SECURE_CONTEXT = INSTANCE.pcad_get_credentials(CONTEXT)
9          FILTER = {"value": None, "time": {"min":
   "2016-12-02T23:39:30.0+03:00", "max": "2016-12-07T23:39:30.0+03:00"}}
10         RESULT = INSTANCE.pcad_send_query(SECURE_CONTEXT, 1, FILTER)
11         print RESULT
12     except (Exception, KeyboardInterrupt, SystemExit) as e:
13         print "connection killed " + str(e)
14
```

Program 8. Etkileşim Mekanizması B2 *(api_06_app_one_time_filter.py)*

PCAD-Uygulama Etkileşim Mekanizması A1

Web soket açarak herhang bir filtre belirtmeden sensörden doğrudan, sürekli veri okunması Program 9'da gösterilmiştir.

```
1  # -*- coding: utf-8 -*-
2  from connection import Connection
3
4  def print_message(message):
5      print message
6
7  if __name__ == "__main__":
8      try:
9          INSTANCE = Connection()
10         CONTEXT = INSTANCE.pcad_init_context("application_config.json")
11         SECURE_CONTEXT = INSTANCE.pcad_get_credentials(CONTEXT)
12         CONNECTION = INSTANCE.pcad_init_connection(SECURE_CONTEXT)
13         INSTANCE.pcad_attach_sensor({"mode":"real", "sensor": 15})
14         INSTANCE.pcad_on_message(CONNECTION, print_message)
15     except (Exception, KeyboardInterrupt, SystemExit) as e:
16         INSTANCE.pcad_terminate_connection(CONNECTION)
17
```

Program 9. Etkileşim Mekanizması A1 (*api_01_app_real_time.py)*

PCAD-Uygulama Etkileşim Mekanizması A2

Web soket açıldıktan bir filtre kullanarak sensörden doğrıdan ve sürekli veri okunması Program 10'da gösterilmektedir. **example.rl** dosyası filteryi belirtmekte kullanılmaktadır.

```python
# -*- coding: utf-8 -*-
from connection import Connection

def print_message(message):
    print message

if __name__ == "__main__":
    try:
        INSTANCE = Connection()
        CONTEXT = INSTANCE.pcad_init_context("application_config.json")
        SECURE_CONTEXT = INSTANCE.pcad_get_credentials(CONTEXT)
        CONNECTION = INSTANCE.pcad_init_connection(SECURE_CONTEXT)
        INSTANCE.pcad_attach_sensor({"mode":"real", "sensor": 19},
"example.rl")
        INSTANCE.pcad_on_message(CONNECTION, print_message)
    except (Exception, KeyboardInterrupt, SystemExit) as e:
        INSTANCE.pcad_terminate_connection(CONNECTION)
```

Program 10. Etkileşim Mekanizması A2 (*api_02_app_real_time_rule.py)*

PCAD-Uygulama Etkileşim Mekanizması A3

Program 11'de bir filtre belirtmeden veri tabanından periyodik veri okunması gösterilmektedir.

```python
# -*- coding: utf-8 -*-
from connection import Connection

def print_message(message):
    print message

if __name__ == "__main__":
    try:
        INSTANCE = Connection()
        CONTEXT = INSTANCE.pcad_init_context("application_config.json")
        SECURE_CONTEXT = INSTANCE.pcad_get_credentials(CONTEXT)
        CONNECTION = INSTANCE.pcad_init_connection(SECURE_CONTEXT)
        INSTANCE.pcad_attach_sensor({"mode":"periodic", "sensor": 12,
"interval": 10})
        INSTANCE.pcad_on_message(CONNECTION, print_message)
    except (Exception, KeyboardInterrupt, SystemExit) as e:
        INSTANCE.pcad_terminate_connection(CONNECTION)
```

Program 11. Etkileşim Mekanizması A3 *(api_03_app_non_real_time.py)*

PCAD-Uygulama Etkileşim Mekanizması A4

Program 12'de bir filtre kullanarak veri tabanından periyodik veri okunması gösterilmektedir.

```python
# -*- coding: utf-8 -*-
from connection import Connection

def print_message(message):
    print message

if __name__ == "__main__":
    try:
        INSTANCE = Connection()
        CONTEXT = INSTANCE.pcad_init_context("application_config.json")
        SECURE_CONTEXT = INSTANCE.pcad_get_credentials(CONTEXT)
        CONNECTION = INSTANCE.pcad_init_connection(SECURE_CONTEXT)
        INSTANCE.pcad_attach_sensor({"mode":"periodic", "sensor": 12,
"interval": 10}, "example.rl")
        INSTANCE.pcad_on_message(CONNECTION, print_message)
    except (Exception, KeyboardInterrupt, SystemExit) as e:
        INSTANCE.pcad_terminate_connection(CONNECTION)
```

Program 12. Etkileşim Mekanizması A4 (*api_04_app_non_real_time_rule.py)*

PCAD- Uygulama İletişimi REST-API Örnek Programları

A1, A1, A3, A4, B1 ve **B2** Etkileşim mekanizmalarının nasıl yapıldığına dair örnek programlar aşağıda verilmiştir. Bu programlar PYTHON dilinde yazılmış olup REST-API metodunu kullanmaktadır.

PCAD-Uygulama Etkileşim Mekanizması A1 (REST-API)

Program 13 REST-API metotunu kullanarak, bir filtre belirtmeden sensörden doğrudan ve sürekli veri okumaktadır.

```python
# -*- coding: utf-8 -*-
import requests
import websocket

AUTH_URL = "http://localhost:9000/api/v1/auth/signin"
QUERY_STRING =
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJQQ0FEIiwidWl
kIjoxLCJncm91cCI6ImFwcGxpY2F0aW9uIn0._DjfVW2emwWuXeZvCSipZKReWb06b9zq3_JM5CRk
bEE"}
HEADERS = {'content-type': 'application/json'}
RESPONSE = requests.request("POST", AUTH_URL, data="{}", headers=HEADERS,
params=QUERY_STRING)

DATA_URL = "ws://localhost:9000/api/v1/sensors/stream"
WEBSOCKET = websocket.WebSocket()
PAYLOAD = "{\"mode\":\"real\",\"sensor\":15}"
WEBSOCKET.connect(DATA_URL+"?access_token="+RESPONSE.content.replace("\"",
""))
WEBSOCKET.send(PAYLOAD)
RESPONSE = WEBSOCKET.recv()
print RESPONSE
WEBSOCKET.close()
```

Program 13. REST-API kullanan Etkileşim Mekanizması A1 (*rest_01_app_real_time.py)*

PCAD-Uygulama Etkileşim Mekanizması A2 (REST-API)

Program 14 REST-API metotunu kullanarak, bir filtre yardımıyla sensörden doğrudan ve sürekli veri okumaktadır.

```
# -*- coding: utf-8 -*-
import requests
import websocket

AUTH_URL = "http://localhost:9000/api/v1/auth/signin"
QUERY_STRING =
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJQQ0FEIiwidWl
kIjoxLCJncm91cCI6ImFwcGxpY2F0aW9uIn0._DjfVW2emwWuXeZvCSipZKReWb06b9zq3_JM5CRk
bEE"}
HEADERS = {'content-type': 'application/json'}
RESPONSE = requests.request("POST", AUTH_URL, data="{}", headers=HEADERS,
params=QUERY_STRING)

DATA_URL = "ws://localhost:9000/api/v1/sensors/stream"
RULE = "name = notification\nmin_value = 10.5\ncurrent_value =
pcad_get_time()\ndb_value = pcad_get_db_time(sensor=17) \nwhen
{\n(current_value < min_value) and (db_value < min_value)}\nthen\n{
Notify()}"
WEBSOCKET = websocket.WebSocket()
PAYLOAD = "{\"mode\":\"real\",\"sensor\":19, \"rule\":"+RULE+"}"
WEBSOCKET.connect(DATA_URL+"?access_token="+RESPONSE.content.replace("\"",
""))
WEBSOCKET.send(PAYLOAD)
RESPONSE = WEBSOCKET.recv()
print RESPONSE
WEBSOCKET.close()
```

Program 14. REST-API kullanan Etkileşim Mekanizması A2 (*rest_02_app_real_time_rule.py)*

PCAD-Uygulama Etkileşim Mekanizması A3 (REST-API)

Program 15 REST-API metotunu kullanarak, bir filtre belirtmeden veri tabanında periyodik veri okumaktadır.

```
# -*- coding: utf-8 -*-
import requests
import websocket

AUTH_URL = "http://localhost:9000/api/v1/auth/signin"
QUERY_STRING =
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJQQ0FEIiwidWl
kIjoxLCJncm91cCI6ImFwcGxpY2F0aW9uIn0._DjfVW2emwWuXeZvCSipZKReWb06b9zq3_JM5CRk
bEE"}
HEADERS = {'content-type': 'application/json'}
RESPONSE = requests.request("POST", AUTH_URL, data="{}", headers=HEADERS,
params=QUERY_STRING)

DATA_URL = "ws://localhost:9000/api/v1/sensors/stream"
WEBSOCKET = websocket.WebSocket()
PAYLOAD = "{\"mode\":\"periodic\",\"sensor\":1, \"interval\":5}"
WEBSOCKET.connect(DATA_URL+"?access_token="+RESPONSE.content.replace("\"",
""))
WEBSOCKET.send(PAYLOAD)
RESPONSE = WEBSOCKET.recv()
print RESPONSE
WEBSOCKET.close()
```

Program 15. REST-API kullanan Etkileşim Mekanizması A3 *(rest_03_app_non_real_time.py)*

PCAD-Uygulama Etkileşim Mekanizması A4 (REST-API)

Program 16'da REST-API metotunu kullanarak, bir filtre yardımıyla veri tabanında periyodik veri okumaktadır.

```
# -*- coding: utf-8 -*-
import requests
import websocket

AUTH_URL = "http://localhost:9000/api/v1/auth/signin"
QUERY_STRING =
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJQQ0FEIiwidWl
kIjoxLCJncm91cCI6ImFwcGxpY2F0aW9uIn0._DjfVW2emwWuXeZvCSipZKReWb06b9zq3_JM5CRk
bEE"}
HEADERS = {'content-type': 'application/json'}
RESPONSE = requests.request("POST", AUTH_URL, data="{}", headers=HEADERS,
params=QUERY_STRING)

DATA_URL = "ws://localhost:9000/api/v1/sensors/stream"
RULE = "name = notification\nmin_value = 10.5\ncurrent_value =
get_value()\ndb_value = get_db_value(sensor=17) \nwhen {\n(current_value <
min_value) or (db_value < min_value)}\nthen\n{ Notify()}"
WEBSOCKET = websocket.WebSocket()
PAYLOAD = "{\"mode\":\"periodic\",\"sensor\":19, \"interval\":5,
\"rule\":"+RULE+"}"
WEBSOCKET.connect(DATA_URL+"?access_token="+RESPONSE.content.replace("\"",
""))
WEBSOCKET.send(PAYLOAD)
RESPONSE = WEBSOCKET.recv()
print RESPONSE
WEBSOCKET.close()
```

Program 16. REST-API kullanan Etkileşim Mekanizması A4
(*rest_04_app_non_real_time_rule.py)*

PCAD-Uygulama Etkileşim Mekanizması B1 (REST-API)

Program 17'de REST-API kullanarak, herhangi bir filtre belirlemeden veri tabanından bir kerelik sorgu yapan bir program sunulmaktadır.

```
# -*- coding: utf-8 -*-
import requests

AUTH_URL = "http://localhost:9000/api/v1/auth/signin"
QUERY_STRING =
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJQQ0FEIiwidWl
kIjoxLCJncm91cCI6ImFwcGxpY2F0aW9uIn0._DjfVW2emwWuXeZvCSipZKReWb06b9zq3_JM5CRk
bEE"}
HEADERS = {'content-type': 'application/json'}
RESPONSE = requests.request("POST", AUTH_URL, data="{}", headers=HEADERS,
params=QUERY_STRING)

DATA_URL = "http://localhost:9000/api/v1/sensors/1/data"
QUERY_STRING = {"access_token":RESPONSE.content.replace("\"", "")}
RESPONSE = requests.request("GET", DATA_URL, headers=HEADERS,
params=QUERY_STRING)
print(RESPONSE.text)
```

Program 17. REST-API kullanan Etkileşim Mekanizması B1 (*rest_05_app_one_time.py)*

PCAD-Uygulama Etkileşim Mekanizması B2 (REST-API)

Program 18'de REST-API kullanarak, filtre kullanarak veri tabanından bir kerelik sorgu yapan bir program sunulmaktadır.

```
# -*- coding: utf-8 -*-
import requests

AUTH_URL = "http://localhost:9000/api/v1/auth/signin"
QUERY_STRING =
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJQQ0FEIiwidWl
kIjoxLCJncm91cCI6ImFwcGxpY2F0aW9uIn0._DjfVW2emwWuXeZvCSipZKReWb06b9zq3_JM5CRk
bEE"}
HEADERS = {'content-type': 'application/json'}
RESPONSE = requests.request("POST", AUTH_URL, data="{}", headers=HEADERS,
params=QUERY_STRING)

DATA_URL = "http://localhost:9000/api/v1/sensors/1/data/filter"
PAYLOAD = "{\"value\": null, \"time\": {\"min\":
\"2016-12-02T23:39:30.0+03:00\", \"max\": \"2016-12-07T23:39:30.0+03:00\"}}"
QUERY_STRING = {"access_token":RESPONSE.content.replace("\"", "")}
RESPONSE = requests.request("POST", DATA_URL, data=PAYLOAD, headers=HEADERS,
params=QUERY_STRING)
print(RESPONSE.text)
```

Program 18. REST-API kullanan etkileşim mekanizması B2
(*rest_06_app_one_time_filter.py)*

6.1.3 Web Ara yüzü

Kullanıcıların platform ile etkileşime geçebilmesi, kimlik doğrulama, erişim hakkı elde etme, sensör ve uygulama kaydı gibi işlenleri yapabilmesi için bir web arayüzü sunulmuştur. Bununla ilgili ekran çıktıları EK-B'de verilmiştir.

6.1.4 Sıkça Sorulan Sorular (SSS)

Bu bölümde projede gerçekleştirilen kullanıcı kılavuzları yer almaktadır.

1. **PCAD platformuna nasıl ulaşabilirim?**
   Platforma ulaşmak için yazılımın kurulduğu alan adını tarayıcıya yazmak yeterlidir. Böylelikle EK-B'de Akka Web Arayüzü altında yer alan bir numaralı örnekte görülecek olan ekran açılacaktır. Eğer yazılım geliştirme veya test etmek için kurulmuşsa, tarayıcıdan localhost:9000 yazarak platforma ulaşılabilir.

2. **Sensörler nasıl sorgulanır?**
   Sensörleri sorgulamak için EK-B'de yer alan 10 numaralı örnek takip edilir. Arama çubuğundan arama yapılabilir. Her bir sensörün nerede yer aldığı görülebilir. Harita üzerinde yakınlaştırma yapılarak tam konumu görüntülenebilir.

3. **Erişim token'nına nasıl ulaşabilirim?**
   Erişim token'nına ulaşmak için EK-B'de yer alan 8 numaralı örnek takip edilir. Açılan tabloda uygulamalara ait olan erişim tokenlerine ulaşılabilir, eğer yanında yer alan mavi renkli butona tıklanarak token kopyalanır. Eğer sensörlere ait olan erişim token'larına ulaşılmak isteniyorsa 17 numaralı talimatlar takip edilir, böylelikle erişim tokenı kopyalanabilir. Erişim tokenları uygulamalara ait olan erişim tokenlarına uygulamayı sistem üzerine kayıt edenler ulaşabilirken, sensörlere ait olan erişim tokenlarına, sensör üzerinde *bilgi* iznine sahip olan kullanıcılar erişebilir.

4. **Sensörler için verilen izinleri nasıl iptal edilir?**
   EK-B'de yer alan 15 ve 16 numaralı örneklerde yer alan ekran görüntülerindeki kırmızı butonlara basılmak suretiyle iptal işlemi yapılabilir.

5. **Rol nasıl oluşturulabilir?**
   EK-B'de yer alan 14 numaralı örneğe bakılabilir.

6. **Kurallar nasıl oluşturulmaktadır?**
   Kurallar bir dosya içerisinde 4.2.1'de tanımlı BNF notasyonu çerçevesinde oluşturulur. Örnek kural, Kural Servisi bölümünde sunulmaktadır.

7. **“application.conf” dosyasında kesinlikle yapmam gereken değişiklikler nelerdir?**
   Bu dosyaya veri tabanı için kullanıcı adı şifre bilgileri mutlaka girilmelidir, bu kesinlikle yapılmalıdır. Aksi takdirde sistem çalışmaz. Diğer parametrelerde istenilirse değişiklik yapılabilir.

8. **PCAD farklı veri tabanlarını destekliyor mu?**
   PCAD’in Node.js ile yapılan gerçekleştirilmesi NoSQL tipindeki doküman veri tabanlarının yanısıra MySQL, Oracle, IBM DB2, PostgreSQL, SQL gibi ilişkisel veri tabanlarını da desteklemektedir. Bunun için veri tabanı bağlantı sürücülerinin kurulumu ve PCAD’a tanıtımı yeterli olmaktadır.

9. **Yükleme ile ilgili sıkıntı oluştu ve PCAD çalışmadı ne yapmam gerekiyor?**
   `./bin/logs` dizininin içinde bulunan `application.log` dosyası oluşan hatayla ilgili bilgi vermektedir. Bu hataya dayanarak düzeltici işlemler yapılmalıdır.

6.2 Node.js Gerçekleştirmesini Devreye Alma ve Dokümantasyon

Bu bülümde Node.js kullanılarak yapılan gerçekleştime ile ilgili devreye alma ve dokümantasyon bilgisi verilmektedir.

6.2.1 Node.js ile gerçekleştirilen sistemin kurulumu

Bu bölümde, İP5’e uygun olarak, Node.js uygulama iskeleti kullanılarak sistemin nasıl kurulacağına yönelik yapılan çalışmalar sunulmaktadır.

Kurulum için gereken pcad.tar.gz ismi dosya ufuk.celikkan@ieu.edu.tr adresine yollanacak bir elektronik posta ile elde edilebilir. Dosya terminalden (komut satırı) dosyanın indirildiği dizine gittikten sonra “`tar -xzvf pcad.tar.gz`” komutu ile arşiv dosyasından çıkarılır. PCAD bilgisayarda kaynak kodu ve yükleme dosyasının olduğu bir dizine çıkarılacaktır. Dizin yapısı aşağıdaki gibi olacaktır:

```
pcad/
  |-- client
  |-- common
  |-- server
  |-- pcad_client
  |-- package.json
  |-- install.sh
  |-- uninstall.sh
```

Sistemin kurulumu sırasında yukarıdan aşağıya doğru bir kurulum yaklaşımı benimsenmiştir. Bu yaklaşıma PCAD'in çalışması için gerekli üst seviye sistem bileşenleri aşağıda sıra ile bilgisayara yüklenmektedir `install.sh` dosyası bu yüklemeler için gerekli depoları sisteme ekleyerek kurulumu tamamlar. Aşağıda belirtilen bileşenler sırası ile kurulacaktır.

1. Node.js
    a. NPM
    b. LoopBack
2. MongoDB
3. Mosquitto

Çıkacak olan "pcad" klasörüne girdikten sonra dizinde bulunan "`install.sh`" ve "`uninstall.sh`" dosyaları çalıştırılabilir hale getirilmelidir. Bunun için şu iki komut çalıştırılmalıdır:

```
$ sudo chmod +x install.sh
$ sudo chmod +x uninstall.sh
```

`install.sh` dosyasının içeriği:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
0C49F3730359A14518585931BC711F9BA15703C6
echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu
xenial/mongodb-org/3.4 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-3.4.list
sudo apt-add-repository  ppa:mosquitto-dev/mosquitto-ppa
sudo apt-get --assume-yes update
sudo apt-get --assume-yes install -y mongodb-org
sudo mkdir /data/
sudo mkdir /data/db
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
sudo apt-get install -y nodejs
sudo apt-get --assume-yes install mosquitto
sudo apt-get --assume-yes install npm
sudo npm --assume-yes install -g strongloop / sudo npm install -g --
unsafe-perm install strongloop
sudo systemctl start mongod.service
sudo npm install
sudo node .
```

`./install.sh` komutu ile yukarıda bulunan işlemler yapılır. Bu komutların açıklamaları ve yükleme talimatları aşağıda anlatılmıştır.

Ubuntu işletim sistemi için yükleme talimatları:

Bir nolu komut `install.sh` dosyasını, 2 nolu komut `uninstall.sh` dosyasını terminal üzerinden çalıştırılabilir duruma getirir.

1. `sudo chmod +x install.sh`
2. `sudo chmod +x uninstall.sh`
3. `./install.sh`

Bu komut ile dosya çalıştırılır ve içinde bulunan komutlar sırasıyla gerçekleşir. Dosyanın içinde aşağıdaki komutlar bulunur.

1. Ubuntu paket yönetim araçları (dpkg ve apt) dağıtıcıların paketleri GPG anahtarlarıyla imzalamalarını istemek suretiyle paketin tutarlılığını ve doğruluğunu garantiler. Bunu sağlamak adına şu komut çalıştırılır.

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
0C49F3730359A14518585931BC711F9BA15703C6
```

Bu komutun çıktısı aşağıdaki gibidir:

```
Executing: /tmp/tmp.DS6CHlRMDK/gpg.1.sh --keyserver
hkp://keyserver.ubuntu.com:80
--recv
0C49F3730359A14518585931BC711F9BA15703C6
gpg: requesting key A15703C6 from hkp server keyserver.ubuntu.com
gpg: key A15703C6: "MongoDB 3.4 Release Signing Key
<packaging@mongodb.com>" not changed
gpg: Total number processed: 1
```

2. Ubuntu 16.04 sürümünü uygun olan yükleme adresi yukarıda ki komut ile “list” dosyası yaratılarak içine eklenir.

```
echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu
xenial/mongodb-org/3.4 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-3.4.list
```

Bu komutun çıktısı aşağıdaki gibidir:

```
deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu
xenial/mongodb-org/3.4 multiverse
```

3. Aşağıdaki komut ile mosquitto yazılımının bulunduğu depo adresi sisteme eklenir

```
sudo apt-add-repository  ppa:mosquitto-dev/mosquitto-ppa
```

Bu komutun çıktısı aşağıdaki gibidir:

```
gpg: keyring `/tmp/tmpz03gcgsm/secring.gpg' created
gpg: keyring `/tmp/tmpz03gcgsm/pubring.gpg' created
gpg: requesting key 262C4500 from hkp server keyserver.ubuntu.com
gpg: /tmp/tmpz03gcgsm/trustdb.gpg: trustdb created
gpg: key 262C4500: public key "Launchpad mosquitto" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
OK
```

4. Şu ana kadar yapılan işlemlerin geçerli olması için apt-get güncellenir. Bu işlem yukarıda yapılan değişiklikler ile paket yöneticisini günceller. Bunun için gerekli komut aşağıdadır:

```
sudo apt-get --assume-yes update
```

Bu komutun çıktısı aşağıdaki gibi olacaktır.

```
Hit:1 http://ppa.launchpad.net/mosquitto-dev/mosquitto-ppa/ubuntu
xenial InRelease
Ign:2 http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4
InRelease
Hit:3 http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4
Release
Hit:5 https://deb.nodesource.com/node_6.x xenial InRelease
Reading package lists... Done
```

5. MongoDB yukarıda eklenen depodan indirilerek, yükleme yapmak için aşağıdaki komut çalıştırılır:

```
sudo apt-get --assume-yes install -y mongodb-org
```

Bu komutun çıktısı aşağıdaki şekilde olacaktır:

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
.... <Silindi>
.... <Silindi>
Setting up mongodb-org-tools (3.4.3) ...
Setting up mongodb-org (3.4.3) ...
```

6. MongoDB'nin veri tabanı dizinlerini oluşturmak için aşağıdaki komutlar çalıştırılır:

   a. `sudo mkdir /data/`

   b. `sudo mkdir /data/db`

7. Node.js depo bilgileri apt-get içine eklemek için aşağıdaki komut çalıştırılır:

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
```

Bu komut çalıştırıldığında aşağıdaki çıktıyı verir:

```
## Installing the NodeSource Node.js v6.x repo...

## Populating apt-get cache...
+ apt-get update
Hit:1 http://ppa.launchpad.net/mosquitto-dev/mosquitto-ppa/ubuntu
xenial InRelease
.... <Silindi>
.... <Silindi>
Hit:5 https://deb.nodesource.com/node_6.x xenial InRelease
Reading package lists... Done
```

```
## Run `apt-get install nodejs` (as root) to install Node.js v6.x
and npm
```

8. Eklenen Node.js depolarından yükleme yakmak için aşağıdaki komut çalıştırılır:

```
sudo apt-get install -y nodejs
```

Bu konut çalıştırıldığında çıktısı aşağıdaki gibi olacaktır:

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no
longer required:
  gyp javascript-common libc-ares2 libev4 libjs-inherits libjs-
jquery
.... <Silindi>
.... <Silindi>
Processing triggers for man-db (2.7.5-1) ...
Setting up nodejs (6.10.2-1nodesource1~xenial1) ...
```

9. Mosquitto yazılımını yukarıda eklene depodan indirerek sisteme kurmak için aşağıdaki komut çalıştırılır:

```
sudo apt-get --assume-yes install mosquitto
```

Çalışırılan konut aşağıdaki çıktıyı verir:

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no
longer required:
.... <Silindi>
.... <Silindi>
Setting up mosquitto (1.4.11-0mosquitto2xenial1) ...
Processing triggers for systemd (229-4ubuntu16) ...

Processing triggers for ureadahead (0.100.0-19) ...
```

10. Bu komut ile npm depolarında bulunan LoopBack yüklenir.

```
sudo npm --assume-yes install -g strongloop / sudo npm install -g -
-unsafe-perm install strongloop
```

Yükleme esnasında bilgisayarınızda eksik bulunan ya da npm de henüz güncellenmemiş kütüphanelerden kaynaklı uyarı mesajı alabilirsiniz. Ancak bunlar yüklemeye ve çalışmaya engel teşkil etmemektedir, başarılı yükleme sonucunda şu şekilde bir çıktı verir.

```
/usr/bin/lb-ng ->
/usr/lib/node modules/strongloop/node modules/loopback-sdk-angular-
cli/bin/lb-ng.js
/usr/bin/slc -> /usr/lib/node_modules/strongloop/bin/slc.js
/usr/lib
```

```
└─┬ strongloop@6.0.3
  └─┬ generator-loopback@1.25.0
    ├─┬ request@2.81.0
    │ ├─┬ har-validator@4.2.1
    │ │ └── ajv@4.11.7
    │ └─┬ http-signature@1.1.1
    │   └── sshpk@1.13.0
    └─┬ yeoman-generator@0.23.4
      ├─┬ dateformat@1.0.12
      │ └─┬ meow@3.7.0
      │   └─┬ normalize-package-data@2.3.6
      │     └── hosted-git-info@2.4.2
      └─┬ download@4.4.3
        └─┬ vinyl-fs@2.4.4
          └─┬ glob-stream@5.3.5
            └─┬ micromatch@2.3.11
              └─┬ arr-diff@2.0.0
                └── arr-flatten@1.0.3
```

11. MongoDB servisini başlatmak için aşağıdaki komut kullanılır.

```
sudo systemctl start mongod.service
```

Bu komut ile MongoDB servisi başlatılır. Servisin başarılı olarak başlatılması durumunda herhangi bir çıktı olmayacaktır.

```
sudo systemctl status mongod.service
```

Yukarıdaki komutla servisin durumu kontrol edilir. Servis çalışır durumda ise aşağıdaki ekran görüntüsüne benzer bir çıktı verir.

```
mongod.service - High-performance, schema-free document-oriented
database
   Loaded: loaded (/lib/systemd/system/mongod.service; disabled;
vendor preset: enabled)
   Active: active (running) since Pzt 2017-04-17 17:34:07 +03; 5s
ago
     Docs: https://docs.mongodb.org/manual
 Main PID: 17328 (mongod)
   CGroup: /system.slice/mongod.service
           └─17328 /usr/bin/mongod --quiet --config
/etc/mongod.conf

Nis 17 17:34:07 aykut-VirtualBox systemd[1]: Started High-
performance, schema-free document-oriented database.
```

12. PCAD'in kütüphane gereksinimleri yükleme adına aşağıda bulunan komut "pcad"dizininin içinde çalıştırılır.

```
sudo npm install
```

13. PCAD bilgisayarınızda başlatmak için aşağıdaki komut kullanılır.

```
node .
```

Bu komut çıktısından sonra PCAD `http://localhost:3000` adresinden erişebilir durumdadır.

Yukarıdaki adımlarda bulunan **--assume-yes** öbeği kullanıcı girişi beklemeden yükleme yapılmasını sağlamak amacıyla kullanılmıştır.

Açık olması gereken portlar

Mosquitto: 1883

PCAD: 3000

MongoDB: 27017

6.2.2 Örnekler

Node.js ile yapılan gerçekleştirilen platform ile iletişim sadece REST-API ile kullanılarak yapılmaktadır. Herhangi bir programlama arayüz kütüphanesi yoktur. REST-API isteklerini göndermeyi destekleyen bir programla dilini kullanmak yeterlidir.

A1, A1, A3, A4 etkileşim mekanizmaları sürekli veri alışverişi gerçekleştirilmesinde kullanılan mekanizmalardır. Java ve MQTT kütüphanesi kullanan bir örnek Program 19, Program 20 ve Program 21 kaynak kodlarında verilmektedir. Program 21'da bir uygulamanın simsar aracılığı ile PCAD veri alması için kullanabileceği Java dilinde yazılmış örnek bir program verilmiştir

```
class SimpleCallback implements MqttCallback {
   // Called when the client lost the connection to the broker
    public void connectionLost(Throwable cause) { }
    public void messageArrived(String topic, MqttMessage message) throws
Exception {
    ObjectMapper mapper = new ObjectMapper();
    Object obj = mapper.readValue(message.getPayload() , Object.class);

        System.out.println("| Topic:" + topic);
        System.out.println("| Message: " + obj);
    }
     //Called when a outgoing publish is complete
    public void deliveryComplete(IMqttDeliveryToken token) { }
}
```

Program 19. MQTT callback fonksiyonları gerçekleştirilmesi

Program 19'de gösterilen "**`SimpleCallback`**" objesi *callback* yapısı ile çalışmaktadır ve simsardan bir mesaj iletildiğinde bu obje tetiklenmektedir. Gelen mesaj ve topic ("kanalID") konsola yazılmaktadır.

```
public static void connect (String url, String chanelID){
  MemoryPersistence persistence = new MemoryPersistence();
  try {
    MqttClient sampleClient = new MqttClient(url, clientId, persistence);
    MqttConnectOptions connOpts = new MqttConnectOptions();
    connOpts.setCleanSession(true);
    sampleClient.connect(connOpts);
    sampleClient.subscribe("#", 1);
    System.out.println("Connected chanelID: " + chanelID);
    MqttMessage message = new MqttMessage(chanelID.getBytes());
    message.setQos(2);
    sampleClient.setCallback(new SimpleCallback());
  } catch(MqttException me){
    me.printStackTrace();
  }
}
```

Program 20. MQTT bağlantı parametrelerinin ayarlanması

Program 20 kaynak kodunda gösterilen "*connect*" metotunda belirlenen parametreler ile simsara bağlantı açılır. Qos (*Quality of Service*) parametresi iletişimin ne şekilde doğrulanacağını belirtir. Bağlantı sağlandıktan sonra Program 19 'da listelenen kodda yer alan "**`SimpleCallback`**" objesi çalışarak veri gelişini beklemeye başlar.

```
try {
  DefaultHttpClient httpClient = new DefaultHttpClient();
  HttpPost postRequest = new HttpPost(
        "http://172.16.12.228:3000/api/sensorData/get_data?access_token=
        ofq8RPCxT3HJCpqY27Kr2PIXV1B5q0WPgEY55EH6FDWTzCCOHbzGoUm2cLFJTBrg");
  StringEntity input = new StringEntity("data={"
     + "\"sensors\": [\"58f7b4a1927d01aafd365614\"],"
     + "\"rules\": {"
     + "    \"58f7b4a1927d01aafd365614\": {"
     + "\"lowerThan\": \"12\","
     + "\"operator\": \"and\","
     + "\"greaterThan\": \"6\""
     + "}"
     + "},"
     + "\"mode\":\"real\""
     + "}"
     + "");
  input.setContentType("application/json");
  postRequest.addHeader("content-type", "application/x-www-form-urlencoded");
  postRequest.setEntity(input);
  HttpResponse response = httpClient.execute(postRequest);
  String json = EntityUtils.toString(response.getEntity(), "UTF-8");
  JSONParser parser = new JSONParser();
  JSONObject json2 = (JSONObject) parser.parse(json);
  JSONObject responseJson = (JSONObject) json2.get("response");
  String chanelID = responseJson.get("chanelID").toString();
  String brokerUrl = responseJson.get("brokerURL").toString();
  brokerUrl = brokerUrl.replace("\\\\", "");
  Connect.connect(brokerUrl,chanelID);
  httpClient.getConnectionManager().shutdown();
} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

JSON yükü

Rules

Bağlantı

Program 21. PCAD'den gerçek zamanlı filtreli veri isteği ve simsara bağlanılması

Program 21 kaynak kodunda, öncelikle bir JSON yükü oluşturulmaktadır. Bu JSON yükü, aşağıda gösterilen kural objesinin JSON formatında kodlanmış şeklidir.

```
{
      "sensors": ["58f7b4a1927d01aafd365614"],
      "rules": {
            "58f7b4a1927d01aafd365614": {
                  "lowerThan": "12",
                  "operator": "and",
                  "greaterThan": "6"
            }
      },
      "mode":"real"
}
```

Daha sonra bu yük ile PCAD'e veri isteği çağrısı yapılmaktadır. PCAD, aşağıda sunulan cevabı döndürmektedir.

```
{
  "response": {
    "chanelID": "58e537fc87ab4e1d1c92ff49",
    "brokerURL": "tcp://PCAD_IP_ADRESS:6000"
  }
}
```

"Bağlantı" şeklinde gösterilen yerde, gelen cevap ayrıştırılır ve Program 20'de gösterilen "`connect`" metotuna ilgili değişkenler verildikten sonra bu metot, simsar ile bir bağlantı açarak ve dinleme konumuna geçmektedir.

PCAD-Uygulama Etkileşim Mekanizması A1

A1 mekanizmasında uygulama filtre belirtmeden sensörden doğrudan veri isteminde bulunur. Uygulama tarafından tarafından yapılan isteğin içinde konulması gereken JSON yükü aşağıda verilmiştir.

```
{
     "sensors": [""],
     "mode":"real"
}
```

PCAD-Uygulama Etkileşim Mekanizması A2

A2 mekanizmasında uygulama filtre belirterek sensörden doğrudan veri isteminde bulunur. Uygulama tarafından tarafından yapılan isteğin içinde konulması gereken JSON yükü aşağıda verilmiştir.

```
{
     "sensors": [""],
     "filter": {
          "": {
               "lowerThan": "",
               "operator": "",
               "greaterThan": ""
          }
     },
     "mode":"real"
}
```

PCAD-Uygulama Etkileşim Mekanizması A3

A3 mekanizmasında uygulama filtre tanımlamadan veri tabanından veri isteminde bulunur Uygulama tarafından tarafından yapılan isteğin içinde konulması gereken JSON yükü aşağıda verilmiştir.

```
{
     "sensors": [""],
     "mode":"periodic"
}
```

PCAD-Uygulama Etkileşim Mekanizması A4

A4 mekanizmasında uygulama filtre belirterek veri tabanından veri isteminde bulunur Uygulama tarafından tarafından yapılan isteğin içinde konulması gereken JSON yükü aşağıda verilmiştir.

```
{
     "sensors": [""],
     "filter": {
          "": {
               "lowerThan": "",
               "operator": "",
               "greaterThan": ""
          }
     },
     "mode":" periodic"
}
```

B1 ve **B2** Etkileşim mekanizmaları tek seferlik veri alış verişi yapılan durumlarda kullanılan mekanizmalardır. Bu mekanizmalar REST-API, HTTP istek/yanıt ve JSON kullanılarak gerçekleştirilmiştir.

PCAD-Uygulama Etkileşim Mekanizması B1

B1 mekanizmasında Uygulama filtre belirtmeden bir kereye mahsus veri tabanından veri isteminde bulunur. Uygulama tarafından tarafından yapılan isteğin içinde konulması gereken JSON yükü aşağıda verilmiştir.

```
{
     "sensors": [""],
}
```

Not: Bu istek herhangi bir JSON yükü kullanılmadan GET metotu ile de aşağıdaki şekilde yapılabilmektedir. Burada bulunan parametreler URL formatında kodlanmış olmalıdır.

```
/api/sensorData?filter=%7B%22where%22%20%3A%20%7B%22sensorID%
22%3A%2258f7b4a1927d01aafd365614%22%7D%7D&access_token=ofg8RP
CxT3HJCpqY27Kr2PIXV1B5q0WPgEY55EH6FDWTzCCOHbzGoUm2cLFJTBrg
```

URL formatında kodlanmış parametrelerin açılımı:

```
{"where" : {"sensorID":"58f7b4a1927d01aafd365614"}}&
access_token=ofg8RPCxT3HJCpqY27Kr2PIXV1B5q0WPgEY55EH6FDWTzCCO
HbzGoUm2cLFJTBrg
```

PCAD-Uygulama Etkileşim Mekanizması B2

B2 mekanizmasında Uygulama filtre belirterek bir kereye mahsus veri tabanından veri isteminde bulunur. Uygulama tarafından tarafından yapılan isteğin içinde konulması gereken JSON yükü aşağıda verilmiştir.

```
{
     "sensors": [""],
     "filter": {
          "": {
               "lowerThan": "",
               "operator": "",
               "greaterThan": ""
          }
     }
}
```

6.2.3 Web Ara yüzü

Kullanıcıların platform ile etkileşime geçebilmesi, kimlik doğrulama, erişim hakkı elde etme, sensör ve uygulama kaydı gibi işlenleri yapabilmesi için bir web arayüzü sunulmuştur. Bununla ilgili ekran çıktıları EK-B'de verilmiştir.

7. SONUÇLAR

Projenin tamamlanmasıyla aşağıda sunulan sonuçlar elde edilmiştir:

1. Proje bütünleşik bir yazılım mimarisi çözümü ortaya çıkarmıştır. Bu çözüm, yazılım mimarisinin tüm katmanlarını ve katmanlar arası bağlantıları içermesi açısından oldukça önemlidir. Proje ekibinin akademik ve endüstriyel deneyimine göre, yazılım sektörde yaşanan teknik sorunların önemli bir kısmı birbiriyle uyumsuz çalışan yazılım araçlarının neticesinde ortaya çıkmaktadır. Farklı aygıt ve araçların yazılım geliştirmede kullanımı ise gittikçe artmaktadır. Projede, bir biriden farklı işlevleri olan yazılım katmanlarının ve servislerinin çeşitli haberleşme protokolleri ve araçları kullanmaları neticesinde ortaya çıkan sorunlarla uğraşılmış ve bu sorunlar giderilmiştir. Elde edilen bulgular tatmin edici sonuçlar verirken, proje ekibinin de deneyimini artırmıştır.
2. Proje ile birlikte API Kütüphanesi geliştirilmiştir. Bu kütüphane geliştiricilerin özellikle durum farkında ve nesnelerin internetine yönelik uygulamalarda kullanabilecekleri bir yapıdır. Böylelikle hızlı geliştirme yapabilecekleri ve test süreçlerini kısaltabilecekleri bir geliştirme platformu deneyimlenmiştir.
3. PCAD Platformunun Akka uygulama iskeletinin geliştirilmesinde, durum farkında uygulamaları için var olan Aktör modeli kullanılmıştır. Bu model, Scala, Java ve Phyton dilleri, RESTful web servisleri ve Play Framework geliştirme platformu birlikte kullanılarak gerçekleştirilmiştir. Bu araçlar, web ve IoT endüstriyel uygulamaları için sıkça kullanılan güncel ürünler olup, birlikte çalıştırılmaları projeye ve proje elemanlarına önemli bir katma değer sağlamıştır.
4. PCAD Platformunun Node.Js uygulama iskeleti kullanılarak gerçekleştirilmesi sürecinde, sensörler ile kullanıcılar arasındaki veri haberleşmesinde mesajlaşma protokolü MQTT, bağlantı sırasındaki veri güvenliği için ise HTTP protokolünün kullanılmıştır. Bu durum iki güncel ve sık kullanılan protokolün birlikte kullanılmasını sağlamıştır. Böylelikle projenin gerçekleştirilmesinde önemli bir deneyim elde edilmiştir. Bu deneyim yayınlanmak üzere bildiri olarak gönderilmiş ve hakemler tarafından da kabul edilmiştir.
5. Projede, klasik ilişkisel veri tabanı yönetim sistemi olan MySQL ile geliştirme ve ölçekleme kolaylığı sağlayan ve de farklı ve alternatif bir yaklaşım sunan NoSQL kullanılmıştır. Böylece her iki temel veri tabanı sistemi de yukarıda bahsedilen programlama araçları ve platformlar için uygulanmıştır.
6. Projede geliştirilen API, GUI ve entegrasyon testleri bir kütüphane oluşmasını sağlamıştır. Böylelikle benzer uygulamalarda kullanılabilecek bir yöntem ve de bir

kütüphane oluşturulmuştur. Akka ve Node.js tabanlı iki platform büyük oranda ortak test kütüphanesi ile test edilmiştir. Bu durum yazılım mühendisliğinde zor ve zahmetli olan test konusuna verimli bir çözüm getirmiştir. Benzer yaklaşım ve uygulama başka web ve IoT platformlarında kullanılabilir.

7. Proje ekibi, proje boyunca çeşitli yazılım programlama dili, araç ve gereçlerini öğrenmişler ve cesurca uygulamışlardır. Böylece projenin gerçekleştirilmesinde farklı geliştiricilerin ürünleri birlikte kullanılmıştır. Bu yazılım sektöründe oldukça sık karşılaşılan ve yazılım mimarisinin geliştirilmesi açısından düşünüldüğünde de başlı başına bir sorundur. Bu yaklaşım, projede zaman zaman gecikmelere neden olmuşsa da sonuç itibarıyla birbiriyle uyumlu çalışan, ortak veri tabanı ve API kütüphanelerini kullanan, ortak GUI'lere ve test senaryolarına sahip iki ayrı geliştirmenin yapılması ile sonuçlanmıştır. Bu proje ekibi açısından önemli bir kazanımdır.

Projenin, proje elemanları için kariyerlerinde olumlu etkileri olmuştur ve olmaya da devam edecektir. Yürütücü, araştırıcı ve lisansüstü öğrenciler için yeni bilgilerin öğrenilmesi ve uygulanması önemlidir. Yazılım mühendisliğinin gereksinin analizi, modelleme, mimari, programlama, veri tabanı uygulamaları, test ve dokümantasyon gibi temel alanları ile proje yönetimi, birlikte çalışma, sürüm denetimi ve raporlama gibi tamamlayıcı alanları bir bütün olarak yürütülmüştür. Yapılan literatür taramaları ve okumalar, hazırlanan yayımlar ve sunumlar her açıdan faydalı olmuştur.

Durum Farkında Servis Platform (PCAD) Mimarisi, Tasarımı ve Geliştirilmesi projesi kapsamında yapılan veya hazırlanan yayımlar ve toplantılarda sunulan bildiriler Tablo 21'de sunulmaktadır.

Tablo 21. Proje kapsamında yapılan yayımlar ve toplantılarda sunulan bildiriler

Sıra	Çıktı türü	Yazarlar	Başlık	Yayın yeri	Durumu
1	Konferans	Ufuk Çelikkan, Kaan Kurtel	A Platform for Context-Aware Application Development: PCAD	FEDCSIS, Lodz, Poland https://fedcsis.org/proceedings/2015/pliks/49.pdf	Yayınlandı Indeks: Scopus'da yer alıyor, WoS'da indeksleniyor
2	Seminer	Aykut Güner	The Use of Relational and NoSQL databases and their performance comparison based on a Node.JS implementation of a Context Aware System	Üniversite Semineri	Tamamlandı

3	Seminer	Orkut Karaçalık	Evaluation of Actor Model Formalism in the Implementation of a Context-Aware Platform	Üniversite Semineri	Tamamlandı
4	Makale	Ufuk Çelikkan, Kaan Kurtel	Application of Service-Oriented Context-Aware Architecture to Laundry Management System	Studies in Informatics and Control, 26(2) , 193-202 , 2017. https://sic.ici.ro/volume-26-issue2-2017/ufuk-celikkan/	Yayınlandı Indeks: Scopus'da yer alıyor, WoS'da indeksleniyor.
5	Konfe-rans	Aykut Güner, Ufuk Çelikkan, Kaan Kurtel	A Message Broker Based Architecture for Context Aware IoT Application Development	International Conference on Computer Science and Engineering UBMK'17	Yayınlandı http://ieeexplore.ieee.org/document/8093381/
6	Makale	Ufuk Çelikkan, Kaan Kurtel, Burak Yanığlı	Nesnelerin İnterneti için Sensör Tabanlı Yazılım Uygulaması: Veteriner Tanı Destek Sistemi	Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi	Hakem değerlendir-mesinde

Proje kapsamında elde edilen sonuçlar sektörle paylaşılmıştır ve web sitesi hizmeti hazırlanmıştır (Tablo 22).

Tablo 22. Proje kapsamında yapılan web sitesi ve sektör paylaşımları

Sıra	Paylaşım	Başlık
1	http://deviot.ieu.edu.tr/	Context aware solutions using IoT
2	Probel Yazılım	1. Durum Farkında Servis Platform (PCAD) Mimarisi 2. Akıllı Çöp Sepeti Uygulaması ve Çöp Toplama Sistemi
3	Teta Teknik Tarım Hayvancılık (http://www.te-ta.com.tr/)	1. Nesnelerin interneti için sensör tabanlı yazılım uygulaması: veteriner tanı destek sistemi 2. Nesnelerin interneti için yazılım mimarisi

Durum Farkında Servis Platform (PCAD) Mimarisi, Tasarımı ve Geliştirilmesi projesi başlangıçta konulan hedeflere ulaşmış ve başarı ile tamamlanmıştır.

KAYNAKLAR

Abowd, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R., Pinkerton, M. 1997. "Cyberguide: A mobile context-aware tour guide", Wireless Networks, 3 (5), 421-433.

Ailisto, H., Alahuhta, P., Haataja, V., Kylloenen, V., Lindholm, M. 2002. "Structuring context aware applications: Five-layer model and example case", Proceedings of the Workshop on Concepts and Models for Ubiquitous Computing, Goteborg, Sweden, 2002. http://www.comp.lancs.ac.uk/computing/users/dixa/conf/ubicomp2002-models/pdf/Ailisto-Ubicomp%20Workshop8.pdf. Son erişim tarihi: 9 Eylül 2014.

(AKKA) http:// www.akka.io Son erişim tarihi: 9 Eylül 2014.

Antonic A., Marjanovic M., Skocir P., Zarko I.P. 2015. "Comparison of the cupus middleware and MQTT protocol for smart city services", In 2015 13th International Conference on Telecommunications (ConTEL), 1–8, July 2015.

Babovich Z. B., Protic J., Milutinovic V. 2016. "Web Performance Evaluation for Internet of Things Applications," in IEEE Access, vol. 4, 6974-6992.

Baldauf M., Dustdar S., Rosenberg F. 2007. "A survey on context-aware system", International Journal of Ad Hoc and Ubiquitous Computing, 2(4), 263-277.

Celikkan, U., Kurtel, K. 2017. "Application of Service-Oriented Context-Aware Architecture to Laundry Management System". Studies in Informatics and Control, 26(2), 193-202, June 2017.

Celikkan, U., Kurtel, K. 2013. "A Context-Aware Architecture for the Management of Laundry Business Processes", 14th European Conference on Knowledge Management - ECKM 2013, Kaunas, Lithuania, 159-166.

Collina M, Corazza G. E., Vanelli-Coralli A. 2012. "Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST," 2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC), Sydney, NSW, 36-41.

Dey, A.K. 2001. “Understanding and using context”, Personal Ubiquitous Computing”, 5(1), 4-7.

Dey, A.K., Abowd, G.D. 2000. “Towards a Better Understanding of Context and Context-Awareness”, Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness, ACM Press, New York.

Dhar P., Gupta P. 2016. “Intelligent parking Cloud services based on IoT using MQTT protocol,” 2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT), Pune, 30-34.

Doblander C., Zhang K., Jacobsen H.A.. 2016. “Publish/subscribe for mobile applications using shared dictionary compression”, In 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), 775–776, June 2016.

Domingo M. C. 2012. “An overview of the internet of things for people with disabilities”, Journal of Network and Computer Applications, 35 (2), 584-596.

Ferraiolo, D., Cugini, J., Kuhn, R. 1995 “Role-based access control (RBAC): Features and motivations,” in Proc. of the 11th Annual Conf. OnComputer Security Applications (New Orleans, LA, Dec. 11-15), 241–248.

Fox G. C, Kamburugamuve S. Hartman R.D. 2012. “Architecture and measured characteristics of a cloud based internet of things,” 2012 International Conference on Collaboration Technologies and Systems (CTS), Denver, CO, 6-12.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1994. Elements of Reusable Object-Oriented Software, Prentice Hall, 1994.

Gruber, T.R. 2004. “A translation approach to portable ontology specifications”, Knowledge Acquisition, 5(2), 199-221.

Gu, T., Pung, H.K., Zhang D.Q. 2004.”A service-oriented middleware for building context-aware services”, Journal of Network And Computer Applications, 28 (1), 1-18.

Hewitt, C., Bishop, P., Steiger, R. 1973. “Universal Modular ACTOR Formalism for Artificial Intelligence" in Proceedings of the 1973 International Joint Conference on Artificial Intelligence, 235–246.

Hewitt, C. 2012. “Actor Model of Computation: Scalable Robust Information Systems”. Tech. rep. arxiv:1008.1459. url: http://arxiv.org/abs/1008.1459v24.

HiveMQ. http://www.hivemq.com/ Son erişim tarihi: 8 Aralık 2017.

Hong, J.Y., Suh, E.H., Kim, S.J. 2009. ”Context-aware systems: A literature review and classification”, Expert System with Applictions, 36(4), 8509-8522.

Hristova, A. 2009. “Conceptualization and Design of a Context-aware Platform for User-centric Applications”, Master's Thesis, Department of Internetworking, Norwegian University of Technology.

Kang D.H. et al. 2017. “Room Temperature Control and Fire Alarm/Suppression IoT Service Using MQTT on AWS”, 2017 International Conference on Platform Technology and Service (PlatCon), Busan, 1-5.

Karagiannis V., Chatzimisios P., Vázquez-Gallego F., Alonso-Zarate J. 2015. “A survey on application layer protocols for the internet of things,” Transaction on IoT and Cloud Computing, vol. 3, no.1, pp. 11-17.

Kayal P., Perros H. 2017. “A comparison of IoT application layer protocols through a smart parking implementation,” 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), Paris, 331-336.

Li Y., Manoharan S. 2013. “A performance comparison of sql and nosql databases”, in Proc. of IEEE PACRIM 2013, Victoria, Canada, August 2013.

Lu, Y., Yu, H. 2010. “A Flexible Architecture for RFID Based Laundry Management Systems”, Wireless Communications Networking and Mobile Computing (WiCOM), 6th International Conference, 1-4.

Lu, Y., Zhang, W., Qin, Z., Meng, Y., Yu, H. 2010. “DeftRFID: A lightweight and distributed RFID middleware”, Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Sixth International Conference, 181-186.

Merezeanu, D., Vasilescu, G., & Dobrescu, R. 2016. “Context-aware Control Platform for Sensor Network Integration in IoT and Cloud”. Studies in Informatics and Control, 25(4), 489-498.

Miraoui, M., Tadj, C., Amar, B.C. 2008. “Architectural Survey of Context-Aware Systems in Pervasive Computing Environment”, Ubiquitous Computing and Communication Journal, 3 (3), 68-76.

MongoDB. https://www.mongodb.com/nosql-explained Son erişim tarihi: 8 Aralık 2017.

Mosquitto. https://mosquitto.org/ Son erişim tarihi: 8 Aralık 2017.

MQTT. http://mqtt.org. Son erişim tarihi: 8 Aralık 2017.

MQTT-IBM. International Business Machines Corporation (IBM) Eurotech. “MQTT V3.1 Protocol Specification”, http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/MQTT_V3.1_Protocol_Specific.pdf. Son erişim tarihi: 8 Aralık 2017

Munõz, M.A., Gonzalez, V.M., Rodriguez, M., Fa vela, J. 2003. “Supporting context-aware collaboration in a hospital: an ethnographic informed design”, Proceedings of Workshop on Artificial Intelligence, Information Access, and Mobile Computing 9th International Workshop on Groupware, CRIWG, Grenoble, France, 330–334.

Nayak A., Poriya A., Poojary D. 2013. “Type of NOSQL Databases and its Comparison with Relational Databases”, International Journal of Applied Information Systems (IJAIS), 5(4), 16-19.

Node.js. https://nodejs.org/en/ Son erişim tarihi: 8 Aralık 2017.

MySQL. https://www.mysql.com/ Son erişim tarihi: 8 Aralık 2017.

Noor, M. Z. H., Razak, M. H. A., Saaid, M. F., Ali, M. S. A. M., Zolkapli, M. 2012. “Design and development of smart basket system for resource optimization”, Control and System Graduate Research Colloquium (ICSGRC), 338-342.

NoSQL. http://nosql-database.org/ Son erişim tarihi: 8 Aralık 2017.

Osathanunkul K., Hantrakul K., Pramokchon P., Khoenkaw P., Tantitharanukul N. 2017. “Configurable automatic smart urinal flusher based on MQTT protocol,” 2017 Int. Conf. on Digital Arts, Media and Technology (ICDAMT), Chiang Mai, 58-61.

Papazoglou M.P., Traverso P., Dustdar S., Leymann F. 2008. “Service-Oriented Computing: A Research Roadmap”, International Journal of Cooperative Information Systems, 17 (2), 223-255, (doi: 10.1142/s0218843008001816).

PAYPAL. https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/ 8 Aralık 2017.

Perttunen, M., Riekki, J., Lassila, O. 2009. “Context representation and reasoning in pervasive computing: a review”, International Journal of Multimedia and Ubiquitous Engineering, 4(4), 1–28.

Play Framework. https://www.playframework.com/. Son erişim tarihi: 8 Aralık 2017.

Pimentel,V., Nickerson, B.G., 2012. “Communicating and Displaying RealTime Data with WebSocket”, IEEE Internet Computing, 16(4), 45-53.

Rehman, K., Stajano, F., Coulouris, G. 2007. “An architecture for interactive context-aware applications”, Pervasive Computing, 6(1), 73-80, (doi. 10.1109/MPRV.2007.5).

Stallings S., (2012) Operating Systems: Internals and Design Principles (7. Basım) Prentice Hall.

Strang, T., Linnhoff-Popien, C. 2004. “A Context Modeling Survey”, First International Workshop on Advanced Context Modelling, Reasoning and Management at UbiComp.

Sung, M., Gips, J., Eagle, N., Madan, A., Caneel, R., Devaul, R., et al. 2005. “Mobile-IT Education (MIT. EDU): m-Learning applications for classroom settings”, Journal of Computer Assisted Learning, 21(3), 229–237.

Tajima, N., Tsukada, K., Siio, I. 2011. “AwareHanger: Context-aware hanger for detecting the status of laundry”, Pervasive, San Francisco, CA, USA.

The Institute of Electrical and Electronics Engineers, Inc. IEEE Std 829-2008 IEEE Standard for Software and System Test Documentation, 2008.

Toma C., Popa M. 2014. “IoT – Internet of Things Architecture for Context Aware Sensors Data Processing in Waste Management Solution”. Journal of Mobile, Embedded and Distributed Systems, 6(4).

Truong H-L, Dustdar, S. 2009. “A survey on context-aware web service systems”, International Journal of Web Information Systems, 5(1), 5–31.

Van, N.T., Lee, S.J., Lee, C.W., Eom, K. H., Jung, K.K. 2012. “An implementation of Laundry Management System based on RFID hanger and wireless sensor network”, Ubiquitous and Future Networks (ICUFN), 2012 Fourth International Conference, 490-493, (doi: 10.1109/ICUFN.2012.6261758).

VerneMQ. https://vernemq.com/ Son erişim tarihi: 8 Aralık 2017.

Yassein M. B., Shatnawi M. Q., Al-zoubi D. 2016. “Application layer protocols for the Internet of Things: A survey,” 2016 Int. Conf. on Engineering & MIS (ICEMIS), 1-4.

EK-A: KAVRAMLAR SÖZLÜĞÜ

Abone: Platformun uygulama ile olan veri akışını düzenlemek üzere kullanılan yapılardır. Bu yapıda Uygulamalar, PCAD platformunun abonesi olarak yayılan mesajı kullanmaktadır. (*Subscriber*)

Aktör: İçinde durum (*state*) bilgisini barındıran ve bu bilgiyi işleyen bir iş parçacığı ünitesine sahip birimdir. (*Actor*)

Aktör modeli: Koşutzamanlı (*concurrent*) ve dağıtık sistemlerin ihtiyaçlarını karşılamak için geliştirilmiş bir modeldir. (*Actor model*)

Ayrıştırıcı: Bir tümceyi, dilin biçimbilgisi, sözdizimi, gerekirse anlam bilgisi kurallarına göre, basamak basamak işlevsel öğelerine ayırmak amacını güden bir algoritma ya da bilgisayar izlencesi. (*Parser*)

Kullanıcı: PCAD platformunu REST API yordamıyla istek yaparak kullanan kişilere denir. Bunlar geçerli URL adreslerine, doğru şekilde istek yapmak suretiyle platformdan cevap alırlar. (*User*)

Kural: Uygulama tarafından PCAD platformundan istenilen verilere ilişkin seçim kriterleridir.

Kural işleyici: İçinde veriler de olduğu halde, kuralların doğrulanması ve çalıştırılması görevini yürütür. (*Rule handler*)

Kural objesi: Kural dosyası ve kural ait meta verileri içeren objedir.

Kural servisi: Uygulama veya kullanıcı tarafından oluşturulan kuralların çalıştırıldığı servistir.

Planlayıcı: Planlayıcı periyodik veri isteklerinde zaman planına uygun şekilde iş parçacığı oluşturmaktadır *(Scheduler).*

Simsar: Mesajı yayan birimlerle (publisher), bu mesajı kullanmak isteyen aboneler (subscriber) arasında verinin iletilmesini sağlayan yapılardır. Projede MQTT protokol¨u ile kullanılmaktadır. (*Broker*)

Uygulama: PCAD platformunu veri kaynağı olarak kullanıp, bunları son kullanıcıya sunan yazılım.

Uygulama geliştiricisi: PCAD platformundan faydalanarak, uygulama geliştiren kişilere denir. Platformun REST API veya sunulan kütüphanelerini kullanarak son kullanıcıya yönelik uygulama geliştirirler. (*Application developer*)

Uygulama kullanıcısı: Uygulama geliştiricilerinin hazırlamış olduğu yazılım veya uygulamaları kullanan kişilere denir. Platformla dolaylı olarak iletişime geçer. Önceki iki kullanıcı tipi gibi doğrudan haberleşme imkânı yoktur. (*Application user*)

Yayıncı: Yayıncı: Platformun, sensörler veya veri tabanı ile olan veri akışını düzenlemek üzere kullanılan yapılardır. Bu yapı, gelen verileri abone öğerine göndermekle sorumludur. (*Publisher*)

JSON Token: İçinde JSON formatında veriler içeren ve web üzerinden veri kaynağına erişmek üzere tanımlanan bie tür jeton veya bilettir. (*JSON Token*)

JSON Yükü: JSON Yükü, kullanıcının isteğine bağlı olarak içine ne konulacağına karar verilen (örneğin kullanıcıya ait kimlik ve kullanıcı yetkileri gibi) JSON token yapısının bölümlerinden biridir. (*JSON Payload*)

Yorumlayıcı: Girdi olarak verilen deyim ya da yordamları, herhangi bir amaç izlence üretecek biçimde derlemeksizin, doğrudan uygulayan bir izlence. (*Interpreter*)

EK-B: PCAD GERÇEKLEŞTİRMESİ: GELİŞTİRME ARAÇLARININ KURULUMU VE WEB ARA YÜZLERİ

AKKA Geliştirme Sürümü Kurulumu

Platform üzerinde geliştirme yapmak isteyenler için geliştirme araçlarının kurulum adımları aşağıda verilmiştir.

Gereksinimler:

- Java 8 - http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Scala 2.11.8 - https://www.scala-lang.org/download/all.html
- sbt 0.13 - http://www.scala-sbt.org/download.html
- mysql 5.7 - https://dev.mysql.com/downloads/
- Lightbend Activator - https://www.lightbend.com/activator/download

Bu kurulum için ilk olarak yukarıda belirtilen linklerde yer alanlar indirilip kurulmalıdır. Kurulduktan sonra aşağıda yazılan komutlar ile yükleme kontrol edilmelidir. Java ve MySQL kurulumları bir önceki bölümde geçen komutlar ile kontrol edilebilir.

```
$ scala -version
Scala code runner version 2.11.x -- Copyright 2002-2013,
LAMP/EPFL
$ sbt sbtVersion
0.13.x
```

Lightbend Activator dosya olarak indirilip, içerisindeki yürütülebilir dosya ile çalıştırılacaktır bunun için herhangi bir komut ile kontrol etme gereği yoktur. Bu kurulum için ilk olarak yukarıda belirtilen linklerde yer alanlar indirilip kurulmalıdır. Daha sonra pcad-1.0-dev isimli zip dosyasından çıkarılmalıdır. Daha sonra aşağıda yazan betik komut satırından çalıştırılmalıdır.

```
$ /path/to/activator ui
```

Bundan sonraki adımda tarayıcı üzerinden http://localhost:8888 açılmalıdır. Sağ tarafta yer alan menüde proje dosyasının bulunduğu dizine gidilip, bu dosya açılmalıdır. Böylelikle proje daha sonra kolaylıkla açılabilecektir. Eklenen proje açılarak bir süre işlemlerin bitmesi beklenmelidir. Sol taraftaki menüde sırasıyla LEARN, DEVELOP, DELIVER ana seçenekleri yer almaktadır. Tutorial kısmında proje ile ilgili bilgiler yer almaktadır. Build kısmında Compile

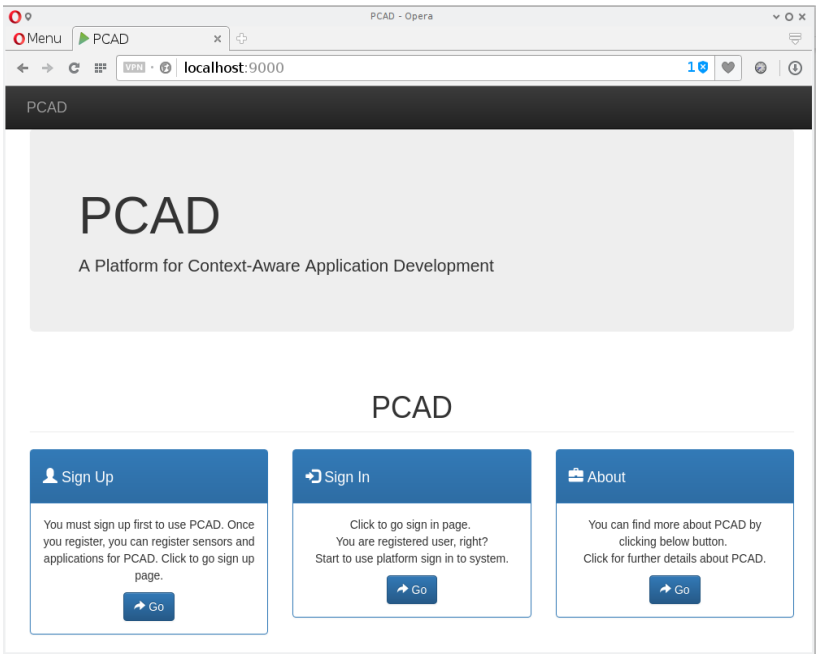
butonuyla projenin derlenmesi yapılmaktadır. Compile on file change aktifleştirerek, proje dosyalarında herhangi bir değişiklik yapıldığında kendiliğinden yeniden derlenmesi sağlanabilir. Code bölümünde proje dosyaları görüntülenebilir. Bu kısımda programlama yaparken bir hata yapıldığında bir hata yapılırsa bunun gösterimi de yapılmaktadır. Run kısmındaki butonla sistem çalışır hale getirilebilir. Run on build yine Compile on build ile benzer işleve sahiptir. Test bölümü de testlerin çalıştırılması için ayrılmıştır.

Geliştirme için IDE olarak Intellij IDEA tavsiye edilmektedir. Scala eklentisi kurularak IDE hazır hale getirilebilir. Son olarak da proje dosyaları IDE'de içeri aktarılarak, geliştirmeye başlanabilir.

AKKA Web Ara yüzü

Bu bölümde AKKA kullanılarak yapılan gerçekleştime ile Web Arayüzü verilmektedir.

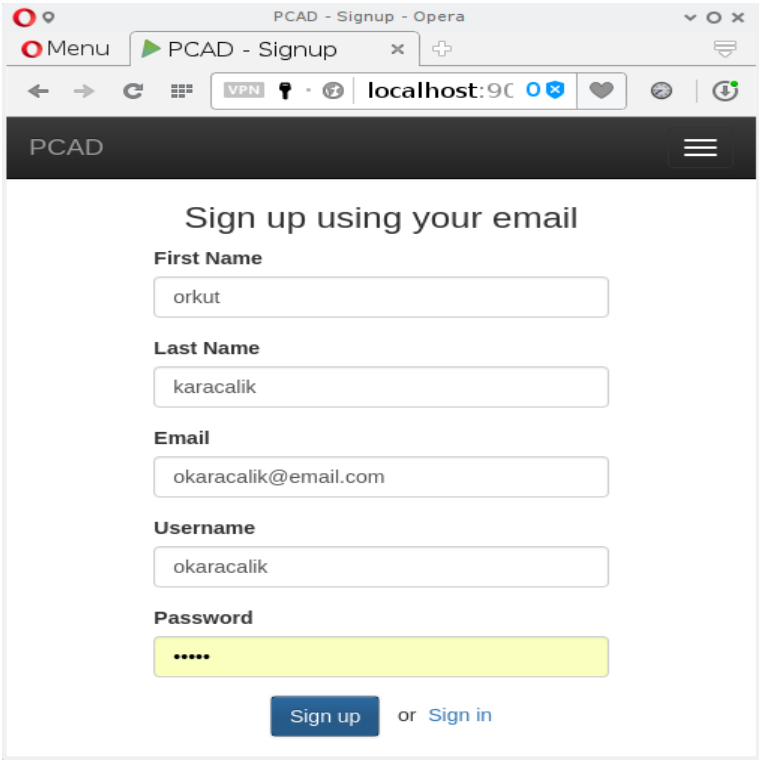
1. **Web arayüzünün açılması:** Kullanıcılar tarayıcıdan http://localhost:9000'e girerek web arayüzünün açılmasını sağlayabilirler. Şekil B1'de gösterilen ana sayfa tarayıcıda açılacaktır.



Şekil B1. Ana sayfa

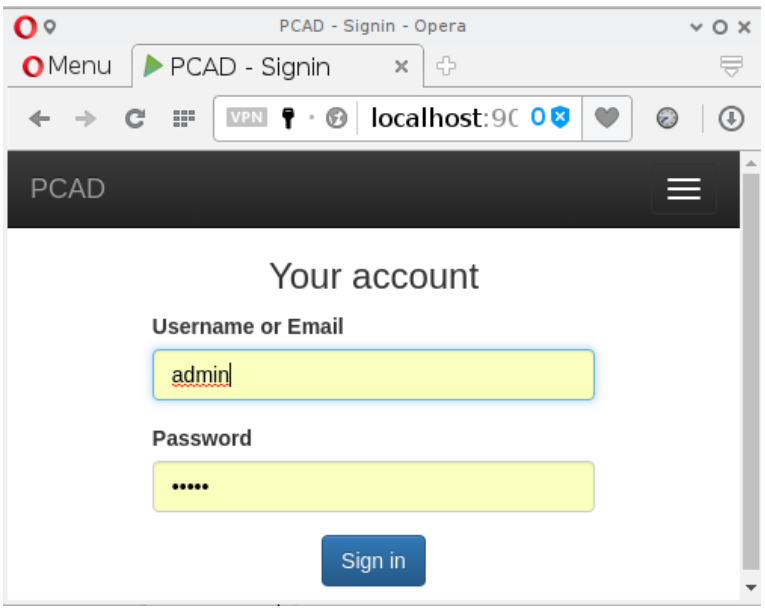
2. **Yeni kullanıcı kaydı:** Web arayüzünü açtıktan sonra üstte yer alan kısımda "Sign Up" butonuna tıklayarak, kullanıcı kayıt formu açılır. Şekil B2'de bu ekran verilmiştir. Formdaki alanlar doldurularak kayıt işlemi gerçekleştirilir. Admin olarak kayıt olmak için

kullanıcı ismini “admin” olarak girmek gerekmektedir. Sistemde bir adet admin tipinde kullanıcı bulunabilir. Bu kullanıcı sadece kullanıcı ismi “admin” olmak şartıyla aynı formdan yaratılabilir.



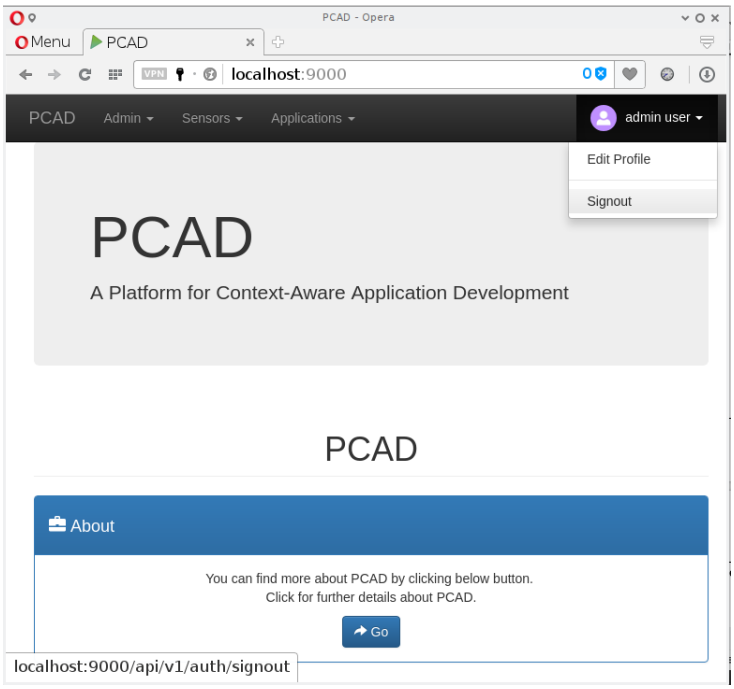
Şekil B2. Kullanıcı kayıt ekranı

3. **Kullanıcı girişi: Web arayüzünü açtıktan sonra üstte yer alan kısımda “Sign In”** butonun tıklayarak, kullanıcı giriş formu Şekil B3’de görülebileceği gibi açılır. Formdaki alanlar doldurularak giriş yapılır.



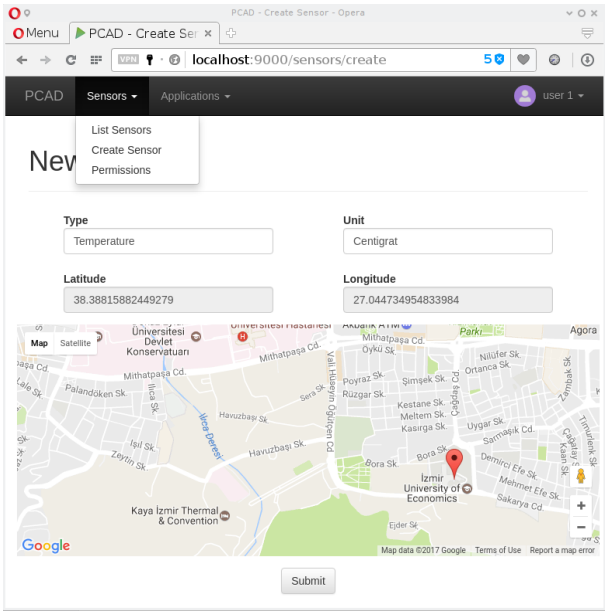
Şekil B3. Kullanıcı girişi

4. **Kullanıcı çıkışı:** Kullanıcı girişi yapıldıktan sonra sağ üst kısımda kullanıcı ismi üzerinde tıklandığında bir menü açılacaktır ve Şekil B4'teki görünecek olan "Sign Out" butonuna basılarak çıkış yapılabilir.



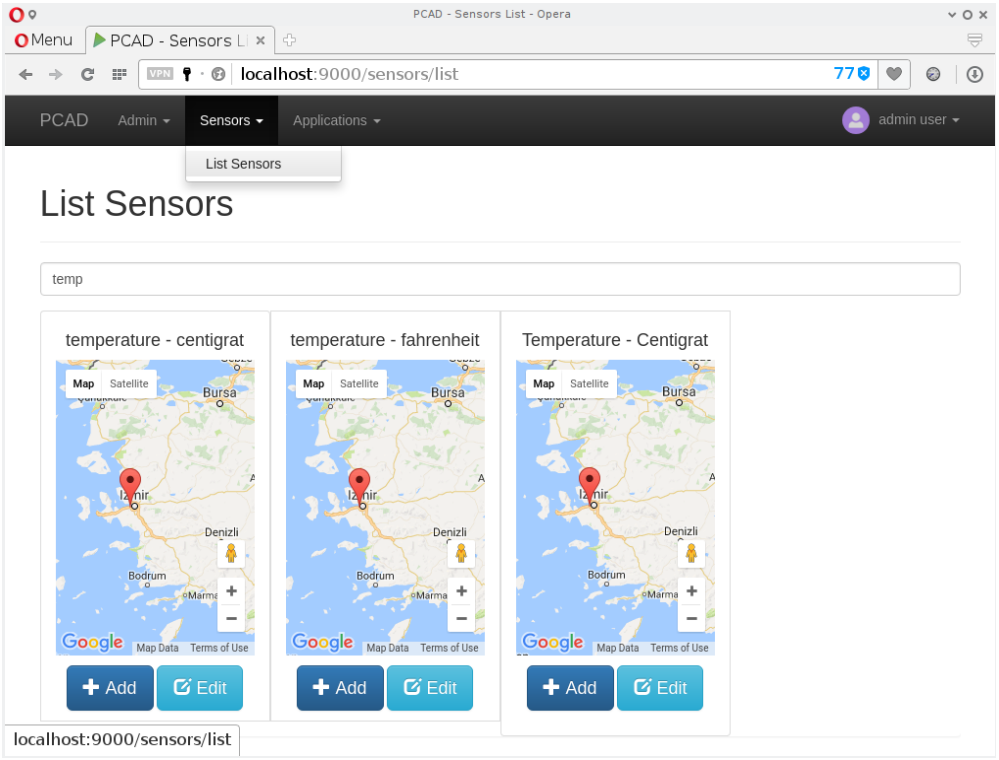
Şekil B4. Kullanıcı çıkışı

5. **Sensör kaydı:** Sisteme yeni sensörler eklemek için, normal kullanıcılar giriş yaptıktan sonra üst menüde yer alan "Sensors" butonuna tıklayıp, açılan listede "Create Sensor" seçeneğine tıklar. Şekil B5'deki gibi bir sayfa açılır. Açılan formda sensörün tipi ve ölçme birimi alanlarını doldurur, sonrasında harita üzerinden sensörün bulunduğu yer işaretler "Submit" butonuna tıklayıp işlemi tamamlar.



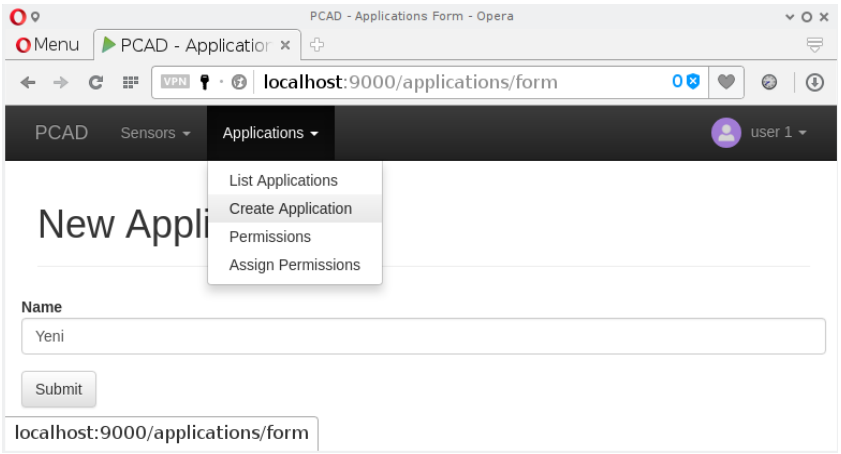
Şekil B5. Sensör kaydı

6. **Sensör güncellemesi:** Sistemde sensör güncellemelerini “Admin” yapabilir. Bunun için “Admin”, sisteme giriş yaptıktan sonra, üst menüde yer alan “Sensors” butonuna tıklayıp, açılan listede “List Sensors” seçeneğine tıklar. Şekil B6’da görülebileceği üzere arama yapılabilir. Burada güncellenmek istenen sensör için “Edit” butonuna tıklanmak suretiyle sensör güncelleme formu açar. Bu formdan istenilen değişiklikler yapılır.



Şekil B6. Sensör güncellemesi

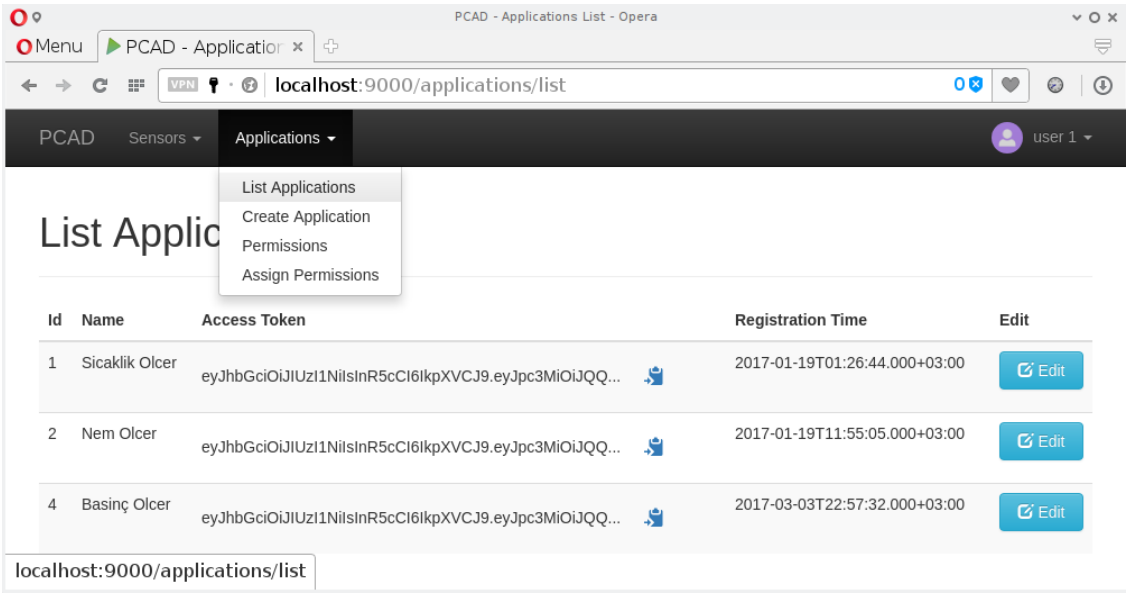
7. **Uygulama kaydı:** Sisteme yeni uygulamalar eklemek için, normal kullanıcılar giriş yaptıktan sonra üst menüde yer alan “Applications” butonuna tıklayıp, açılan listede “Create Application” seçeneğine tıklar. Şekil B7’de görünen formu doldurulup işlem tamamlanır.



Şekil B7. Uygulama kaydı

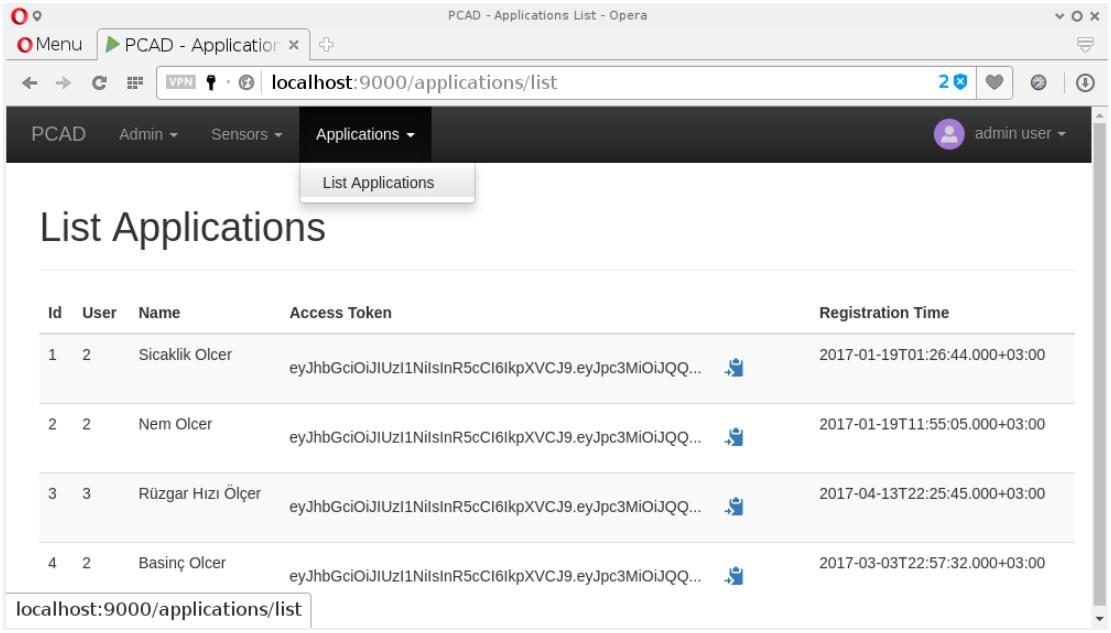
8. Uygulamaların listelenmesi:

a. **Normal Kullanıcılar** giriş yapıldıktan sonra üst menüde yer alan “Applications” butonuna tıklayıp, açılan listede “List Applications” seçeneğine tıklanmalıdır. Şekil B8’de sunulduğu gibi açılan sayfada kullanıcıya ait uygulamalar gösterilecektir.



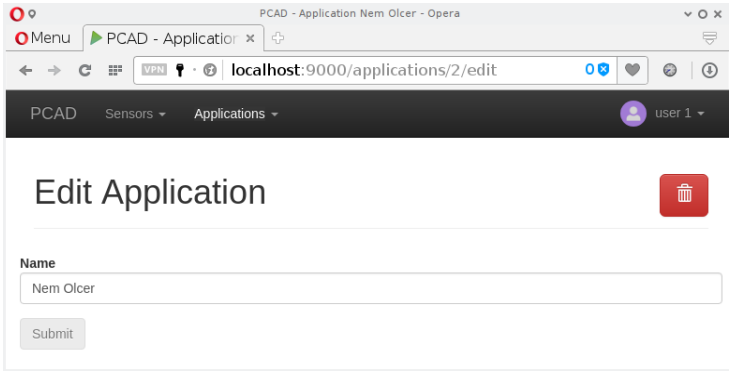
Şekil B8. Uygulamaların listelenmesi

b. **Admin** kullanıcısı aynı şekilde listeleme yapacaktır ve sistemde kayıtlı olan bütün uygulamaları görüntüleyebileceklerdir (Şekil B9).



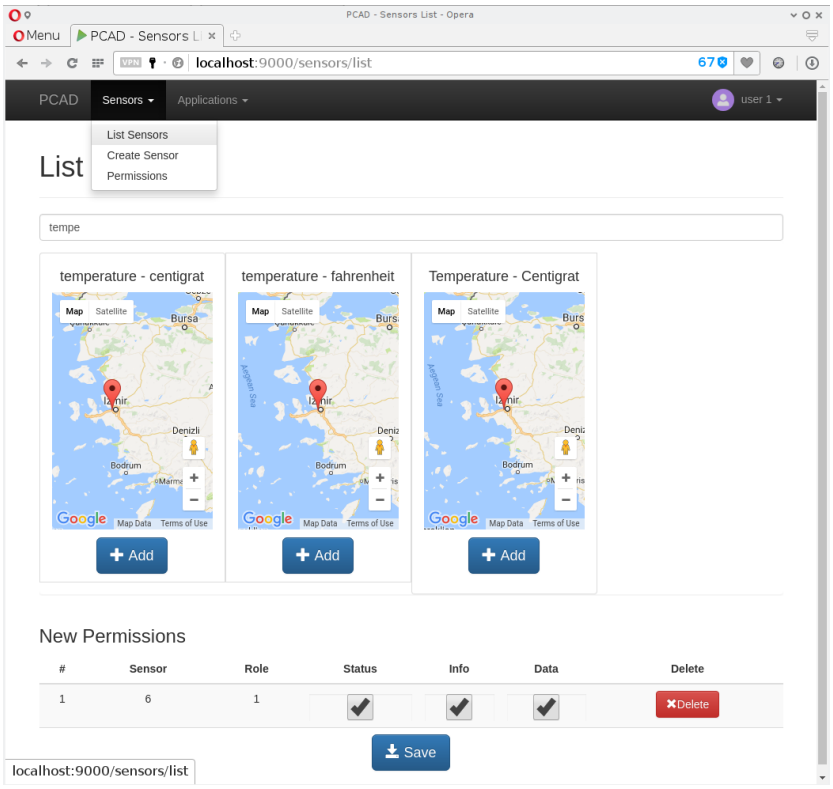
Şekil B9. Uygulama listelenmesi

9. **Uygulama güncellemesi:** Kullanıcılar, kendi uygulamalarını listesini açarak her bir uygulamanın karşısında yer alan “Edit” butonuna uygulama güncelleme formunu açmalıdırlar. Şekil B10’da gibi sunulan formda uygulama ile ilgili istedikleri değişiklikleri yaparak işlemi tamamlayabilirler.



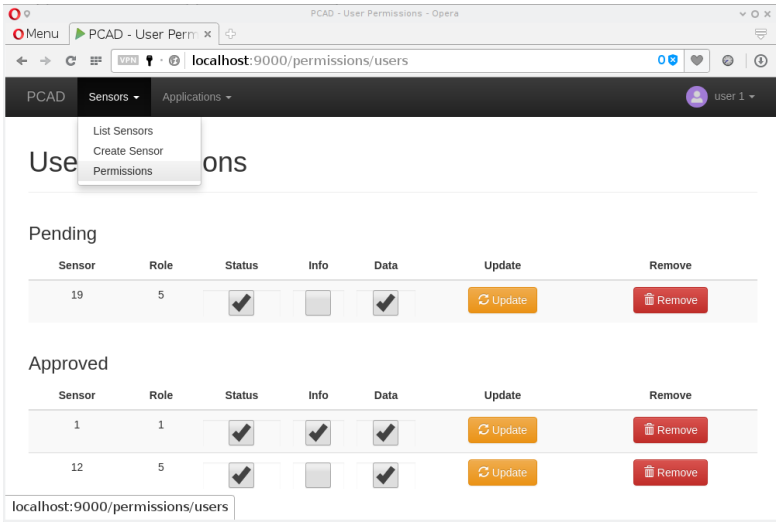
Şekil B10. Uygulama güncelleme

10. **Sensör izin isteği:** Normal kullanıcılar sisteme giriş yaptıktan sonra, üst menüde yer alan “Sensors” butonuna tıklayıp, açılan listede “List Sensors” seçeneğine tıklar. Arama çubuğundan arama yapıp, filtreleme yaparlar (Şekil B11). Listede her sensör için “Add” butonu yer almaktadır. İzin istenilen sensör için bu butona basılır. Bu sensörler alt kısımda yer alacak olan, listeye eklenecektir. Bu listede istenilen izin tipleri işaretlenip, gönderilerek işlem tamamlanır.



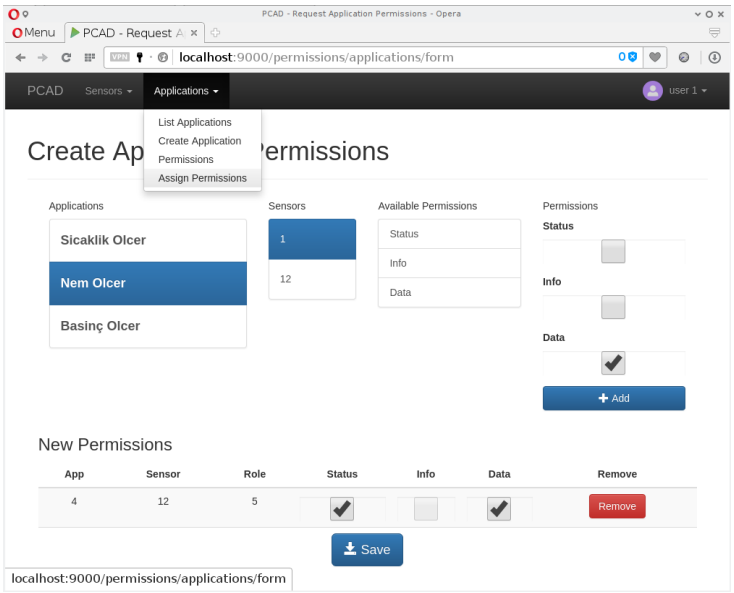
Şekil B11. Sensör izin isteme

11. **Sensör izni güncellemesi:** Kullanıcılar sisteme giriş yaptıktan sonra, üst menüde yer alan “Sensors” butonuna tıklayıp, Şekil B12’deki gibi açılan listede “Permissions” seçeneğine tıklar. Burada beklemede olan ve onaylanan izinler görünebilir. Her bir izin için kendisine ayrılan satırda “Update” ve “Remove” butonları yer almaktadır. İstenilen izin değişiklikleri yapılarak karşısında “Update” butonuna tıklayarak, güncelleme yapılabilir. “Remove” butonuna tıklanırsa izin silinecektir.



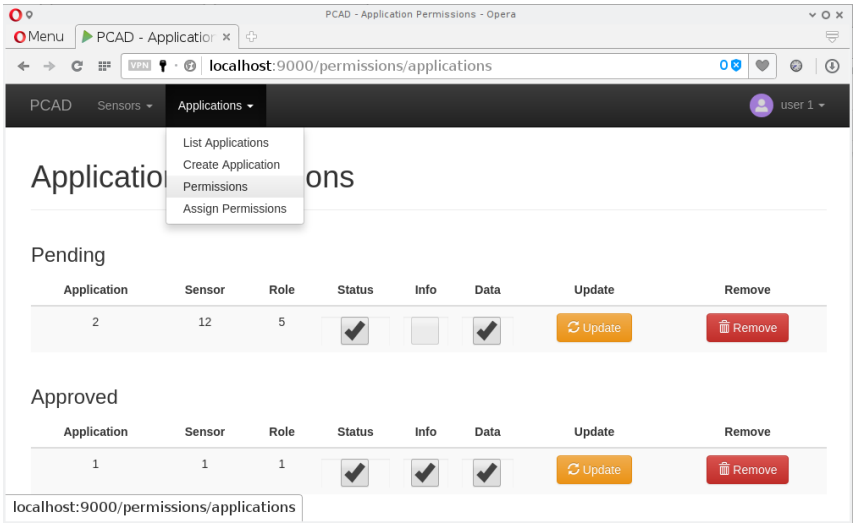
Şekil B12. Sensör izin güncelleme

12. **Uygulama sensör izni ataması:** Kullanıcılar sisteme giriş yaptıktan sonra, üst menüde yer alan “Applications” butonuna tıklayıp, Şekil B13’deki gibi açılan listede “Assign Permissions” seçeneğine tıklamalıdır. Sırasıyla uygulama ve sensör seçimi yapılmalıdır. Son olarak izin tipleri seçilip “Add” butonuna basılmalıdır. İzin bir tabloya eklenecektir. Birden fazla izin bu şekilde tabloya eklenebilir. “Submit” butonuna basılarak izin atamaları yapılır.



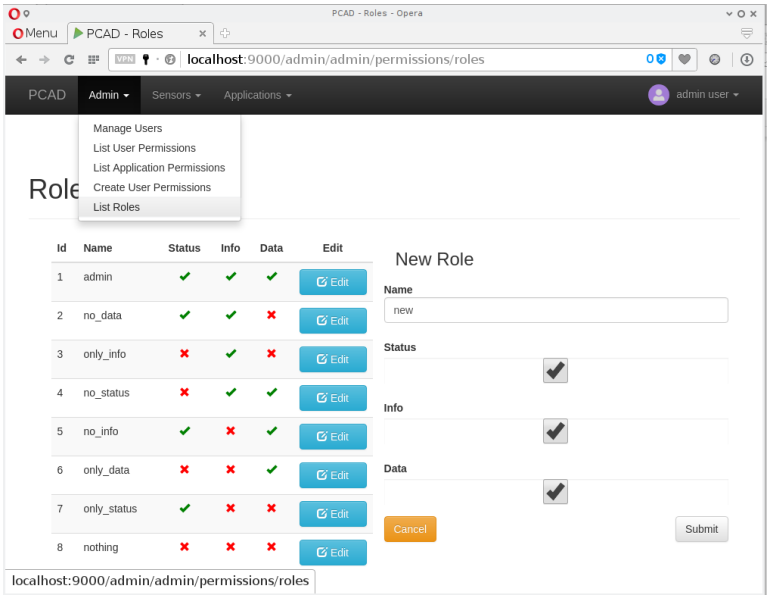
Şekil B13. Uygulama izin atama

13. **Uygulama sensör izni güncellemesi:** Kullanıcılar sisteme giriş yaptıktan sonra, üst menüde yer alan “Sensors” butonuna tıklayıp, Şekil B14’deki gibi açılan listede “Permissions” seçeneğine tıklamalıdır. Burada beklemede olan ve onaylanan izinler görünebilir. Her bir izin için kendisine ayrılan satırda “Update” ve “Remove” butonları yer almaktadır. İstenilen izin değişiklikleri yapılarak karşısında “Update” butonuna tıklayarak, güncelleme yapılabilir. Eğer “Remove” butonuna tıklanırsa izin tamamen silinecektir.



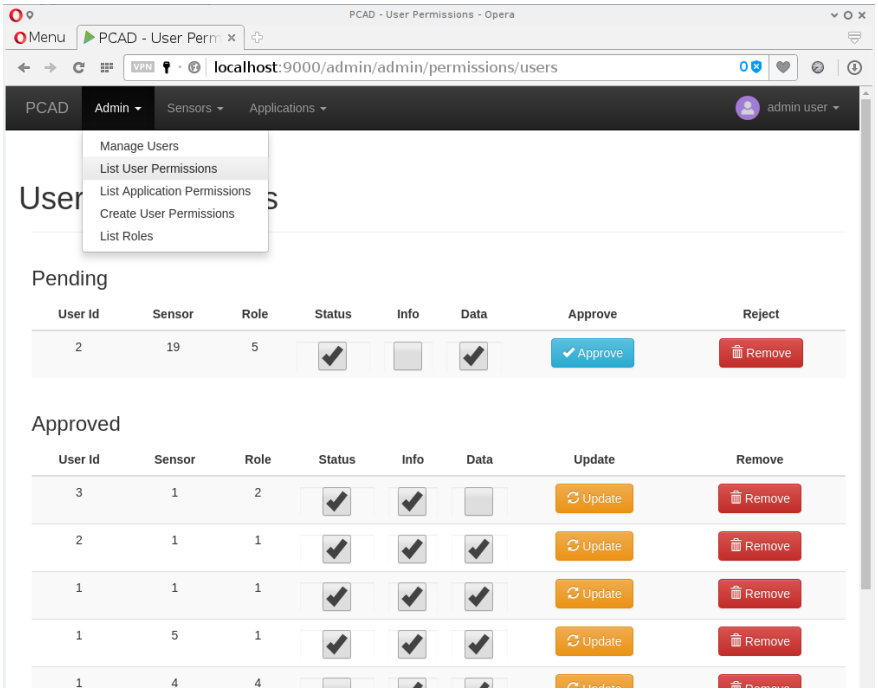
Şekil B14. Uygulama izin güncelleme

14. **Rol Yaratılması/Listelenmesi/Güncellenmesi:** “Admin” kullanıcısı sisteme giriş yaptıktan sonra, üst menüde yer alan “Admin” butonuna tıklayıp, Şekil B15’deki gibi açılan listede “List Roles” seçeneğine tıklamalıdır. Burada sol kısımda var olan roller listelenmektedir. Satır karşısındaki “Edit” butonuna tıklanarak güncelleme yapılabilir ya da sağ kısımdaki butona basılarak yeni rol formu açılabilir, sonra istenen değişiklikler yapılarak işlem tamamlanır.



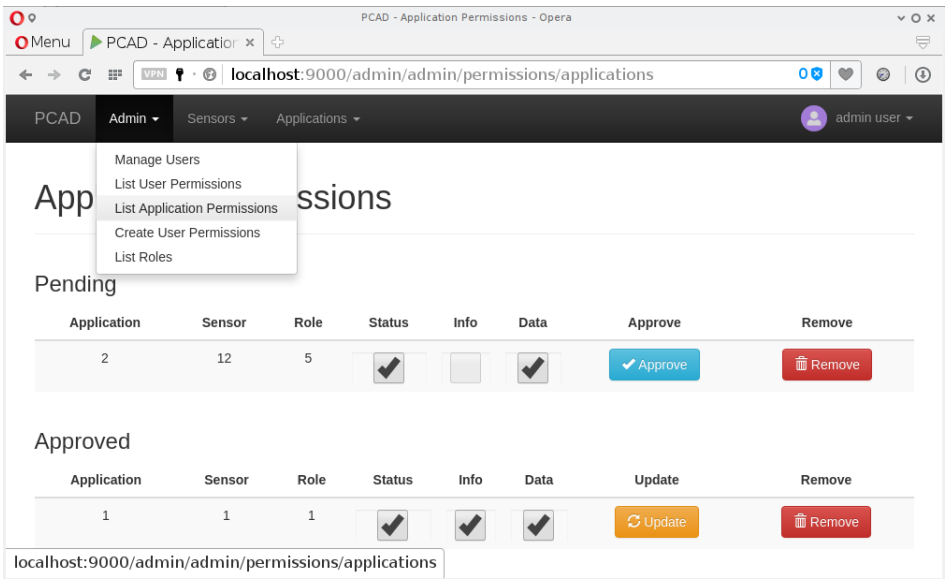
Şekil B15. Roller

15. **Kullanıcı izinleri Listeleme/Onaylama/Reddetme/Geri Alma:** Admin kullanıcısı, normal kullanıcılar tarafından talep edilen izinleri onaylar veya reddeder. Üst menüden, "Admin" ve ardından "List User Permissions" seçeneğine tıklar. Şekil B16'daki gibi açılan tablodan onaylama ve reddetme işlemlerini yapabilir. Ayrıca, var olan izinler üzerinde de değişiklikler yapılabilir.



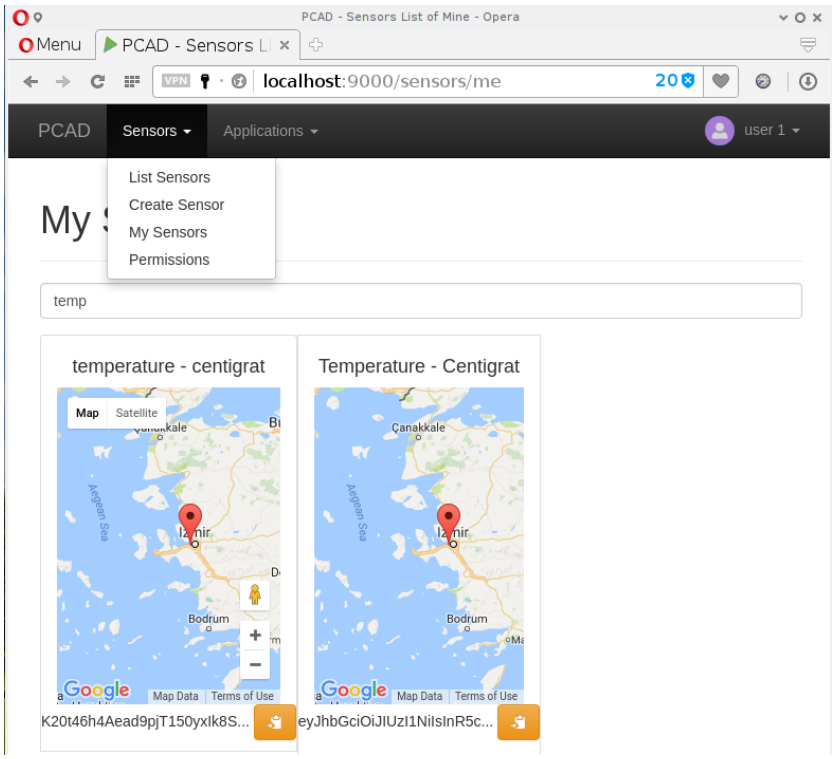
Şekil B16. Kullanıcı-Sensör izinleri

16. **Uygulama İzinleri Listeleme / Onaylama / Reddetme / Geri Alma:** Normal kullanıcılar tarafından, uygulamalara atanan izinler de kullanıcı izinleri gibi aynı işleme tabi tutulur (Şekil B17).



Şekil B17. Uygulama-Sensör izinleri

17. **Sensör Tokenı Listeleme:** Normal kullanıcılar, Şekil B18'de gösterildiği gibi açılan menüde "My Sensors" seçeneğine tıklarlar. Sonrasında açılan her bir bloğun altında yer alan turuncu butona basmak suretiyle sensöre ait olan tokeni kopyalayabilirler.

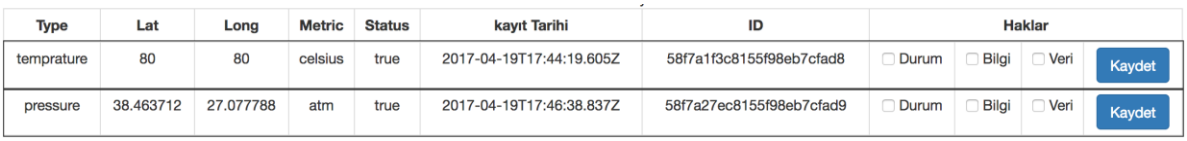


Şekil B18. Sensör token'ları

NODE.js Web Ara yüzü

Kullanıcıların platform ile etkileşime geçebilmesi, kimlik doğrulama, erişim hakkı elde etme, sensör ve uygulama kaydı gibi işlenleri yapabilmesi için bir web arayüzü sunulmuştur.

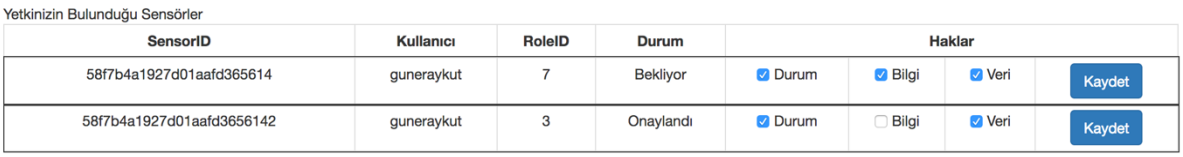
1. **Sensör listeme ve izin ekleme:** Sistemde yer alan sensörler http://localhost:3000/sensors/list.html adresinde listelenirler. Şekil B19'da gösterildiği üzere Sensörlerin durum/bilgi/veri değerlerine erişim hakkı istemek için "haklar" kolonunda yer alan haklardan istenenler seçildikten sonra "kaydet" butonuna basılır. Bu sayfa her yeniden yüklenişinde haklar kolonu ilgili sensör için login olan kullanıcının hakkı olsa dahi boş gelir. Yeni yapılan talep doğrultusunda eski haklar sistemden silinir ve yeni talep kaydedilir.



Type	Lat	Long	Metric	Status	kayıt Tarihi	ID	Haklar			
temprature	80	80	celsius	true	2017-04-19T17:44:19.605Z	58f7a1f3c8155f98eb7cfad8	Durum	Bilgi	Veri	Kaydet
pressure	38.463712	27.077788	atm	true	2017-04-19T17:46:38.837Z	58f7a27ec8155f98eb7cfad9	Durum	Bilgi	Veri	Kaydet

Şekil B19. Sensör listeleme ve izin ekleme

2. **Sensör izinleri listeleme ve düzenleme**: http://localhost:3000/sensors/grandet.html adresinde bulunan sayfada kullanıcının erişim yetkisi ya da talebi olduğu sensörler gösterilmektedir. Durum kolonunda erişim talebinin onay durumu gösterilmektedir. Durum sütununda "onaylandı" ya da "bekliyor" şeklinde iki farklı durum yer alabilmektedir. "bekliyor" durumu bu talebin "admin" tarafından henüz onaylanmadığını ifade etmektedir. "haklar" sütununda hangi haklarının talep edildiği/olduğu listelenir. Kullanıcı bu sayfa da haklarını değiştirebilir. Hak değişikliği yaptığı zaman durum sahası "bekliyor" durumuna dönecektir. Şekil B20 kullanılan ekranı göstermektedir.

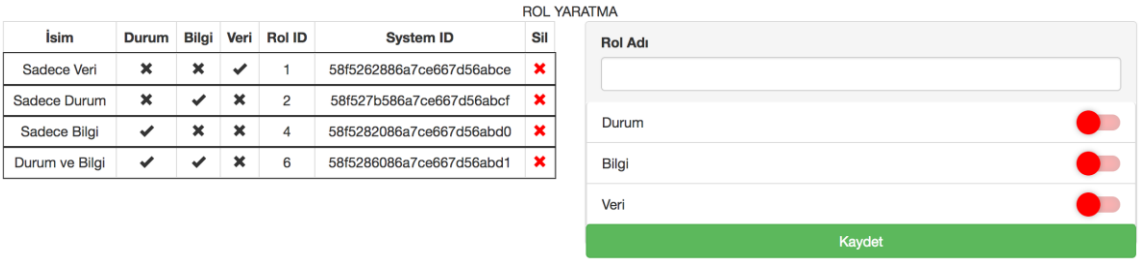


Yetkinizin Bulunduğu Sensörler

SensorID	Kullanıcı	RoleID	Durum	Haklar			
58f7b4a1927d01aafd365614	guneraykut	7	Bekliyor	Durum	Bilgi	Veri	Kaydet
58f7b4a1927d01aafd3656142	guneraykut	3	Onaylandı	Durum	Bilgi	Veri	Kaydet

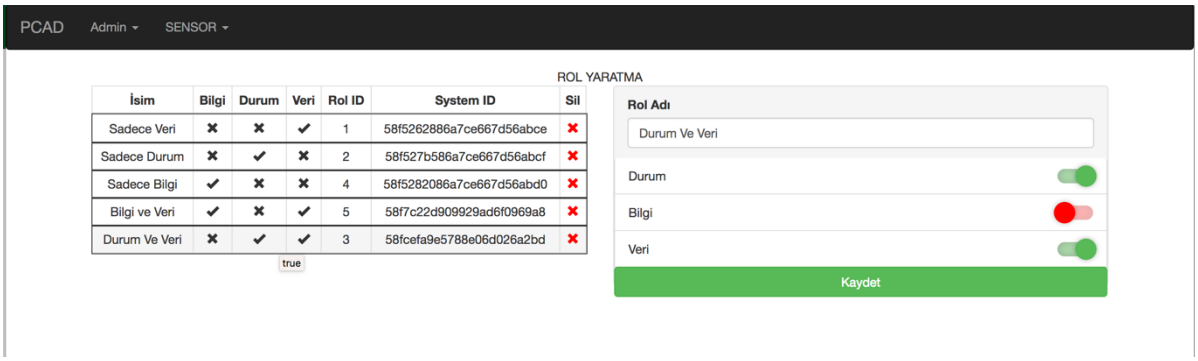
Şekil B20. Sensör izinleri listeleme ve düzenleme

3. **Admin: Role Ekleme/Silme:** http://localhost:3000/admin/create_role.html URL'sinde bulunan Şekil B21'de gösterilen ekranda sistemde var olan roller sol tarafta yeni rol ekleme sağ tarafta yer almaktadır. Sistem üzerinde aynı rol adına ve aynı rol yetkilerine sahip sadece bir adet rol bulunabilir. Örneğin yukarıdaki ekran görüntüsünde bulunan "Sadece Durum" isimli ve sadece sensörün bilgisine erişim yetkisi sağlayan rol tekrar yaratılamaz. Sil kolonunun altında bulunan "x" butonuna basıldığında ilgili satırdaki rol bilgisi sistemden silinmektedir.



Şekil B21. Rol ekleme ve silme

4. **Admin: Rol Düzenleme:** Şekil B22'de listelenen rollerden biri tıklandığında sağ tarafta bulunan yapı bu rol bilgileri ile dolar. Gerekli güncellemeler sağlandıktan sonra kaydet butonuna basıldığında rol düzenlenmiş olur.



Şekil B22. Rol düzenleme

5. **Admin: Kullanıcı Hakları değiştirme:** http://localhost:3000/admin/user_permissions.html URL'sinde bulunan bu admin kullanıcısının ekranında kullanıcıların talep etmiş oldukları izinler ve durumları yer almaktadır. Durum sahasında "onaylandı" ya da "bekliyor" şeklinde iki farklı durum yer alabilmektedir. "bekliyor" durumu bu talebin "admin" tarafından henüz onaylanmadığını ifade etmektedir. "haklar" sütununda hangi haklarının talep edildiği/olduğu listelenir. "admin" kullanıcısı hakları değiştirip ya da değiştirmeden "kaydet" butonuna bastığında haklar sisteme kaydedilir ve durum sahası "onaylandı" olarak değişir (Şekil B23).

Şekil B23. Kullanıcı hakları değiştirme

EK-C
API TESTLERİNE AİT TEST SENARYOLARI

Test - 01

Amaç	Platforma kullanıcı kaydı yapmak.
Metot	POST
URL	/api/v1/auth/signup
Sorgu Parametreleri	-
Yük (JSON format)	{first_name: String, last_name: String, email: String, company: String, username: String, password: String}

▷ *Test-Senaryosu-01.1*

Girdi : Sorgu	-
Girdi : Yük	{first_name: "jon", last_name: "crow", email: "joncrow@mail.com", company: "Raven Industries", username: "jon", password: "pwd"}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{message: "User successfully registered."}

▷ *Test-Senaryosu-01.2*

Girdi : Sorgu	-
Girdi : Yük	{first_name: "jon", last_name: "crow"}
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 02

Amaç	Kullanıcının platforma giriş yapmasını sağlamak.
Metot	POST
URL	/api/v1/auth/signin
Sorgu Parametreleri	access_token
Yük (JSON format)	{username:String,password:String}

▷ *Test-Senaryosu-02.1*

Girdi : Sorgu	-
Girdi : Yük	{username: TrueUsername, password: TruePassword}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: Number, first_name: String, last_name: String, email: String, company: String, username: String, access_token: String, registration_time: Timestamp, valid: Boolean}

▷ *Test-Senaryosu-02.2*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{AUTHORIZED_ACCESS_TOKEN}

▷ *Test-Senaryosu-02.3*

Girdi : Sorgu	-
Girdi : Yük	{username: TrueUsername, password: FalsePassword}
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-02.4*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-02.5*

Girdi : Sorgu	-
Girdi : Yük	-
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 03

Amaç	Kullanıcının platformdan çıkış yapmasını sağlamak.
Metot	GET
URL	/api/v1/auth/signout
Sorgu Parametreleri	access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-03.1*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{message:"User successfully logged out"}

▷ *Test-Senaryosu-03.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-03.3*

Girdi : Sorgu	-
Girdi : Yük	-
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 04

Amaç	Kullanıcıları listelemek.
Metot	GET
URL	/api/v1/users
Sorgu Parametreleri	access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-04.1*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: Number, first_name: String, last_name: String, email: String, company: String, username: String, access_token: String, registration_time: Timestamp, valid: Boolean}

▷ *Test-Senaryosu-04.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 05

Amaç	Kullanıcı bilgilerini görüntülemek.
Metot	GET
URL	/api/v1/users/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-05.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: Number, first_name: String, last_name: String, email: String, company: String, username: String, access_token: String, registration_time: Timestamp, valid: Boolean}

▷ *Test-Senaryosu-05.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 06

Amaç	Kullanıcı bilgilerini güncellemek.
Metot	PUT
URL	/api/v1/users/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	{id: Number, first_name: String, last_name: String, email: String, company: String, username: String, access_token: String, registration_time: Timestamp, valid: Boolean}

▷ *Test-Senaryosu-06.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{id: 1, first_name: "New name", last_name: "New last name", email: "jon@email.com", company: "my company", username: "jonny"}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, first_name: "New name", last_name: "New last name", email: "jon@email.com", company: "my company", username: "jonny", access_token: "TOKEN", registration_time: "TIME", valid: "FALSE"}

▷ *Test-Senaryosu-06.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-06.3*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{first_name: "jon", last_name: "crow", email: "jon@email.com"}
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 07

Amaç	Sensörleri listelemek.
Metot	GET
URL	/api/v1/sensors
Sorgu Parametreleri	access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-07.1*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, type: "temperature", company: "some company", unit: "centigrat", latitude: "36", longitude: "45", access_token: "TOKEN", valid: "TRUE"}

▷ *Test-Senaryosu-07.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 08

Amaç	Yeni sensör oluşturmak.
Metot	POST
URL	/api/v1/sensors
Sorgu Parametreleri	access_token
Yük (JSON format)	{type: String, company: String, unit: String, latitude: String, longitude: String}

▷ *Test-Senaryosu-08.1*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{type: "temperature", company: "some company", unit: "centigrat", latitude: "36", longitude: "45"}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, type: "temperature", company: "some company", unit: "centigrat", latitude: "36", longitude: "45", access_token: "TOKEN", valid: "TRUE"}

▷ *Test-Senaryosu-08.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-08.3*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 09

Amaç	Sensör bilgilerini görüntülemek.
Metot	GET
URL	/api/v1/sensors/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-09.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, type: "temperature", company: "some company", unit: "centigrat", latitude: "36", longitude: "45", access_token: "TOKEN", valid: "TRUE"}

▷ *Test-Senaryosu-09.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 10

Amaç	Sensör bilgilerini güncellemek.
Metot	PUT
URL	/api/v1/sensors/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	{id: Number, type: String, company: String, unit: String, latitude: String, longitude: String}

▷ *Test-Senaryosu-10.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{id: 1, type: "temperature", company: "other company", unit: "centigrat", latitude: "38.8", longitude: "43.4"}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, type: "temperature", company: "other company", unit: "centigrat", latitude: "38.8", longitude: "43.4", access_token: "TOKEN", valid: "TRUE"}

▷ *Test-Senaryosu-10.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-10.3*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 11

Amaç	Sensör bilgilerini silmek.
Metot	DELETE
URL	/api/v1/sensors/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-11.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, type: "temperature", company: "some company", unit: "centigrat", latitude: "36", longitude: "45", access_token: "TOKEN", valid: "TRUE"}

▷ *Test-Senaryosu-11.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 12

Amaç	Uygulamaları listelemek
Metot	GET
URL	/api/v1/applications
Sorgu Parametreleri	access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-12.1*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, user: 1, name: "my app", registration_time: "TIME", access_token: "TOKEN", valid: "TRUE"}

▷ *Test-Senaryosu-12.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 13

Amaç	Yeni uygulama oluşturmak.
Metot	POST
URL	/api/v1/applications
Sorgu Parametreleri	access_token
Yük (JSON format)	{user: Number, name: String}

▷ *Test-Senaryosu-13.1*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{user: 1, name: "my app"}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, user: 1, name: "my app", registration_time: "TIME", access_token: "TOKEN", valid: "TRUE"}

▷ *Test-Senaryosu-13.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-13.3*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 14

Amaç	Uygulama bilgilerini görüntülemek.
Metot	GET
URL	/api/v1/applications/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-14.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, user: 1, name: "my app", registration_time: "TIME", access_token: "TOKEN", valid: "TRUE"}

▷ *Test-Senaryosu-14.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 15

Amaç	Uygulama bilgilerini güncellemek.
Metot	PUT
URL	/api/v1/applications/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	{id: Number, user: Number, name: String, registration_time: Timesamp}

▷ *Test-Senaryosu-15.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{id: 1, user: 1, name: "my updated app"}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, user: 1, name: "my updated app", registration_time: "TIME", access_token: "TOKEN", valid: "TRUE"}

▷ *Test-Senaryosu-15.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-15.3*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 16

Amaç	Uygulama bilgilerini silmek.
Metot	DELETE
URL	/api/v1/applications/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-16.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, user: 1, name: "my app", registration_time: "TIME", access_token: "TOKEN", valid: "TRUE"}

▷ *Test-Senaryosu-16.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 17

Amaç	Kullanıcı izinlerini listelemek.
Metot	GET
URL	/api/v1/permissions/users
Sorgu Parametreleri	access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-17.1*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, user: 1, sensor: 1, role: 1, valid: true}

▷ *Test-Senaryosu-17.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 18

Amaç	Yeni kullanıcı izni oluşturmak.
Metot	POST
URL	/api/v1/permissions/users
Sorgu Parametreleri	access_token
Yük (JSON format)	{user: Number, sensor: Number, role: Number}

▷ *Test-Senaryosu-18.1*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{user: 1, sensor: 1, role: 1}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, user: 1, sensor: 1, role: 1, valid: true}

▷ *Test-Senaryosu-18.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-18.3*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 19

Amaç	Kullanıcı izni bilgilerini görüntülemek.
Metot	GET
URL	/api/v1/permissions/users/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-19.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, user: 1, sensor: 1, role: 1, valid: true}

▷ *Test-Senaryosu-19.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 20

Amaç	Kullanıcı izni bilgilerini güncellemek.
Metot	PUT
URL	/api/v1/permissions/users/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	{id: Number, user: Number, sensor: Number, role: Number}

▷ *Test-Senaryosu-20.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{id: 1, user: 1, sensor: 1, role: 3}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, user: 1, sensor: 1, role: 3, valid: true}

▷ *Test-Senaryosu-20.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-20.3*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 21

Amaç	Kullanıcı izni bilgilerini silmek.
Metot	DELETE
URL	/api/v1/permissions/users/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-21.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, user: 1, sensor: 1, role: 1, valid: true}

▷ *Test-Senaryosu-21.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 22

Amaç	Uygulama İzinlerini listelemek.
Metot	GET
URL	/api/v1/permissions/applications
Sorgu Parametreleri	access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-22.1*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, application: 1, sensor: 1, role: 1, valid: true}

▷ *Test-Senaryosu-22.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 23

Amaç	Yeni uygulama izni oluşturmak.
Metot	POST
URL	/api/v1/permissions/applications
Sorgu Parametreleri	access_token
Yük (JSON format)	{application: Number, sensor: Number, role: Number}

▷ *Test-Senaryosu-23.1*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{application: 1, sensor: 1, role: 1}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, application: 1, sensor: 1, role: 1, valid: true}

▷ *Test-Senaryosu-23.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-23.3*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 24

Amaç	Uygulama izni bilgilerini görüntülemek.
Metot	GET
URL	/api/v1/permissions/applications/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-24.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, application: 1, sensor: 1, role: 1, valid: true}

▷ *Test-Senaryosu-24.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 25

Amaç	Uygulama izni bilgilerini güncellemek.
Metot	PUT
URL	/api/v1/permissions/applications/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	{id: Number, application: Number, sensor: Number, role: Number}

▷ *Test-Senaryosu-25.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{id: 1, application: 1, sensor: 1, role: 2}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, application: 1, sensor: 1, role: 1, valid: true}

▷ *Test-Senaryosu-25.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-25.3*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 26

Amaç	Uygulama izni bilgilerini silmek.
Metot	DELETE
URL	/api/v1/permissions/applications/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-26.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, application: 1, sensor: 1, role: 1, valid: true}

▷ *Test-Senaryosu-26.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 27

Amaç	Rollerı listelemek
Metot	GET
URL	/api/v1/roles
Sorgu Parametreleri	access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-27.1*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, name: "all", status: true, info: true, data: true}

▷ *Test-Senaryosu-27.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 28

Amaç	Yeni rol oluşturmak.
Metot	POST
URL	/api/v1/roles
Sorgu Parametreleri	access_token
Yük (JSON format)	{name: String, status: Boolean, info: Boolean, data: Boolean}

▷ *Test-Senaryosu-28.1*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{name: "all", status: true, info: true, data: true}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, name: "all", status: true, info: true, data: true}

▷ *Test-Senaryosu-28.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-28.3*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 29

Amaç	Rol bilgilerini görüntülemek.
Metot	GET
URL	/api/v1/roles/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-29.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, name: "all", status: true, info: true, data: true}

▷ *Test-Senaryosu-29.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 30

Amaç	Rol bilgilerini güncellemek.
Metot	PUT
URL	/api/v1/roles/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	{id: Number, name: String, status: Boolean, info: Boolean, data: Boolean}

▷ *Test-Senaryosu-30.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{id: 1, name: "admin", status: true, info: true, data: true}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, name: "admin", status: true, info: true, data: true}

▷ *Test-Senaryosu-30.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 31

Amaç	Rol bilgilerini silmek.
Metot	DELETE
URL	/api/v1/roles/:id
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-31.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, name: "all", status: true, info: true, data: true}

▷ *Test-Senaryosu-31.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 32

Amaç	Sensörün aktif olup olmadığını görüntülemek.
Metot	GET
URL	/api/v1/sensors/:id/status
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-32.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	`{status: true}`

▷ *Test-Senaryosu-32.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	`{message: "Unauthorized Access."}`

Test - 33

Amaç	Sensöre ait bilgileri görüntülemek.
Metot	GET
URL	/api/v1/sensors/:id/info
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-33.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	`{id: 1, type: "temperature", company: "some company", unit: "centigrat", latitude: "36", longitude: "45", access_token: "TOKEN", valid: "TRUE"}`

▷ *Test-Senaryosu-33.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	`{message: "Unauthorized Access."}`

Test - 34

Amaç	Sensöre ait en son veriyi görüntülemek.
Metot	GET
URL	/api/v1/sensors/:id/data
Sorgu Parametreleri	id, access_token
Yük (JSON format)	-

▷ *Test-Senaryosu-34.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, sensor: 1, value: 27.3}

▷ *Test-Senaryosu-34.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

Test - 35

Amaç	Sensöre ait verinin platfoma gönderilmesini sağlamak.
Metot	POST
URL	/api/v1/sensors/:id/data
Sorgu Parametreleri	id, access_token
Yük (JSON format)	{sensor: Number, time: Timestamp, value: Float}

▷ *Test-Senaryosu-35.1*

Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{sensor:1, time: "TIME", value: 23.4}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{1}

▷ *Test-Senaryosu-35.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

▷ *Test-Senaryosu-35.3*

Girdi : Sorgu	access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	400
Çıktı : Cevap İçeriği	{message: "Bad Request."}

Test - 36

Amaç	Sensöre ait verileri filtreleyip görüntülenmesini sağlamak.
Metot	POST
URL	/api/v1/sensors/:id/data
Sorgu Parametreleri	id, access_token
Yük (JSON format)	{value: {min: Number, max: Filter}, time: {min: DateTime, max: DateTime}}

▷ *Test-Senaryosu-36.1*

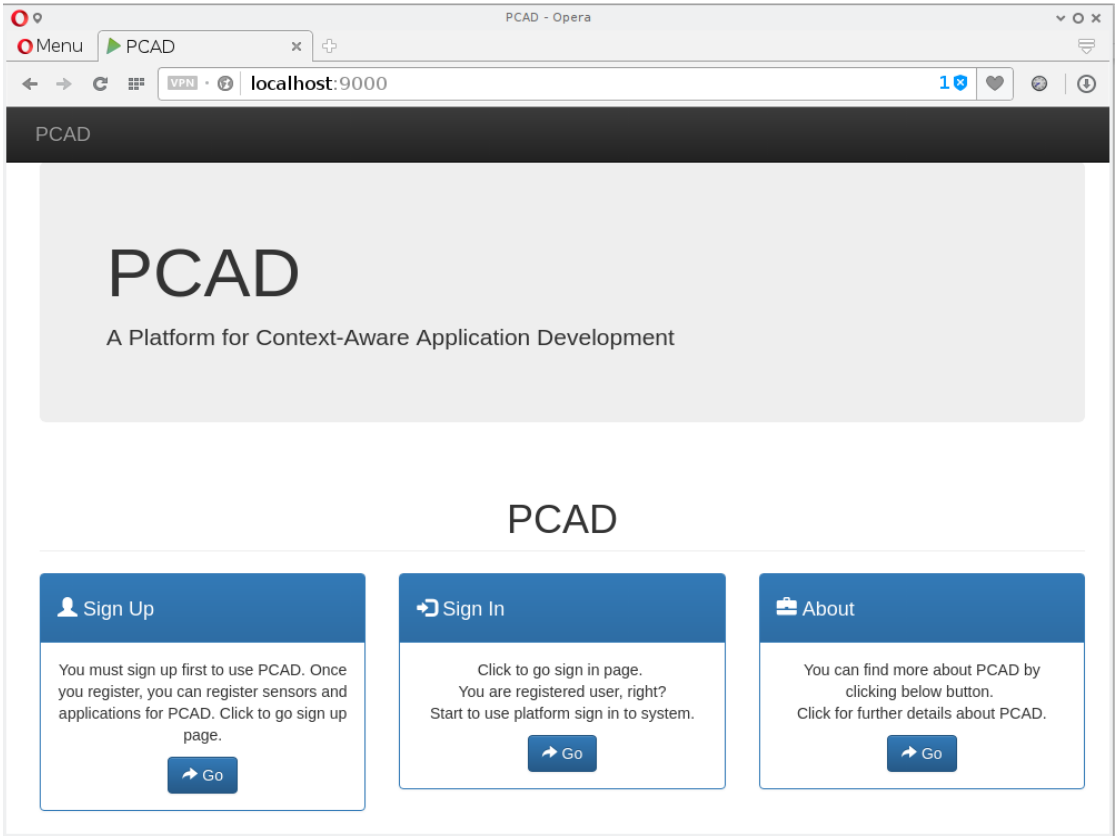
Girdi : Sorgu	id:1, access_token=TRUE_ACCESS_TOKEN
Girdi : Yük	{value: {min: 27, max: 28}}
Çıktı : Cevap Kodu	200
Çıktı : Cevap İçeriği	{id: 1, sensor: 1, value: 27.3}

▷ *Test-Senaryosu-36.2*

Girdi : Sorgu	access_token=WRONG_ACCESS_TOKEN
Girdi : Yük	-
Çıktı : Cevap Kodu	404
Çıktı : Cevap İçeriği	{message: "Unauthorized Access."}

EK-D
GUI TESTLERİNE AİT TEST SENARYOLARI

Test - 01 - Anasayfa



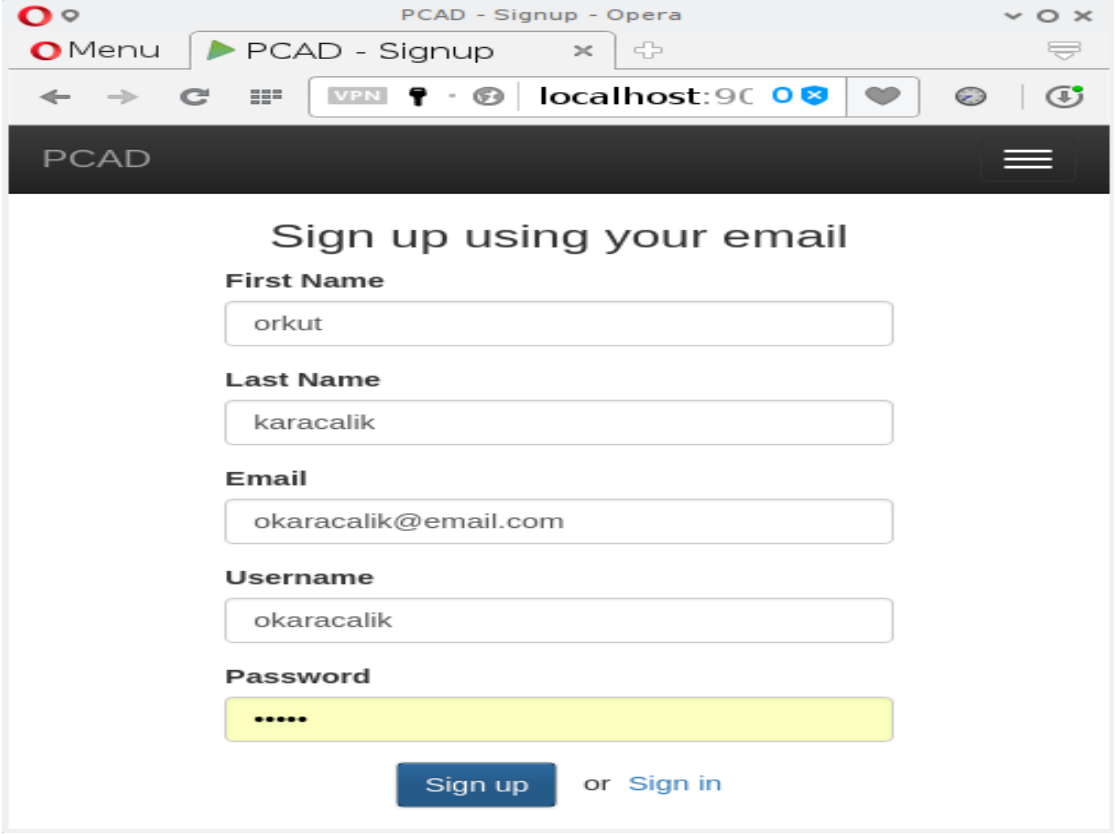
TS-01.1 Üst menü çubuğundaki ”PCAD” butonu tıklandığında ana sayfaya gitmeli

TS-01.2 ”Sign Up” başlıklı kutusundaki butona tıklandığında kullanıcı kaydı formunun açılmalı

TS-01.3 ”Sign In” başlıklı kutusundaki butona tıklandığında kullanıcı giriş formunun açılmalı

TS-01.4 ”About” başlıklı kutusundaki butona tıklandığında *deviot.ieu.edu.tr* sayfasına gitmeli

Test - 02 - Kullanıcı kaydı



TS-02.1 Bütün alanlar dolu olmalı

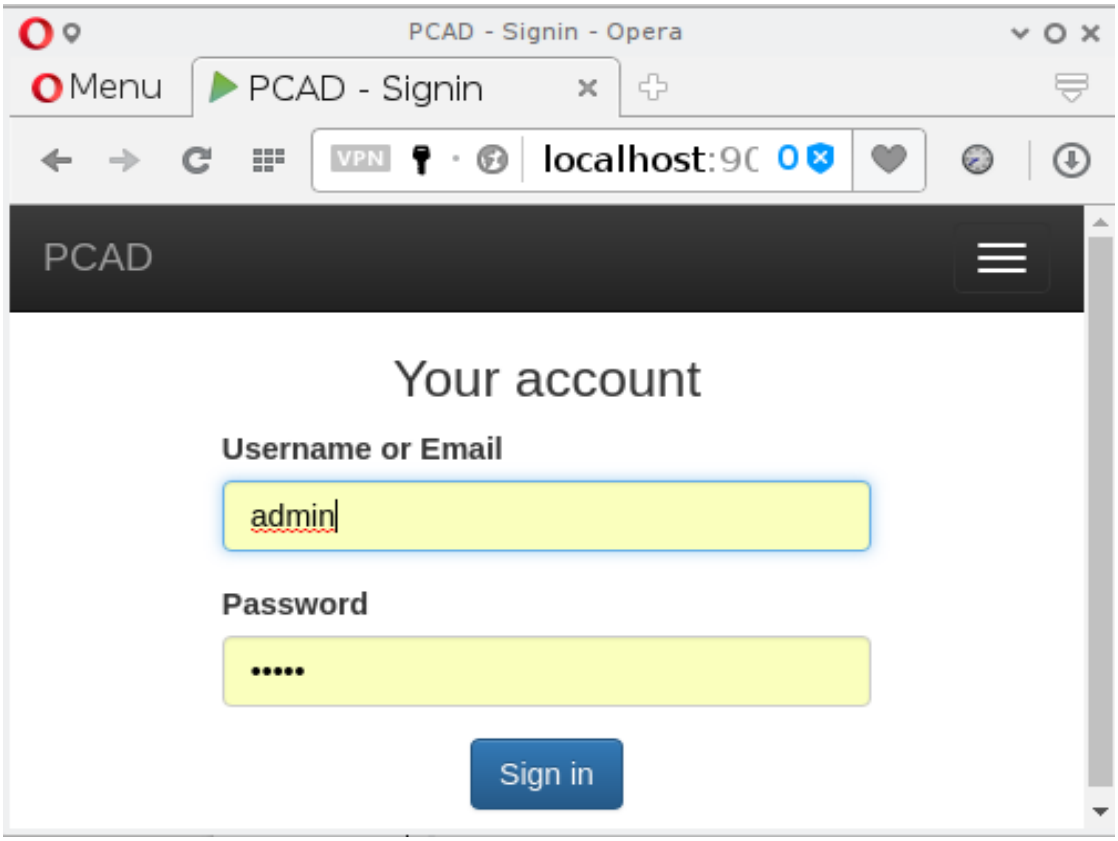
TS-02.2 ”email” alanı girdisi geçerli olmalı

TS-02.3 ”username” alanı girdisi özgün olmalı

TS-02.4 ”Sign Up” butonuna tıklandığında başarılı veya başarısız sonuç gösterilmeli

TS-02.5 ”Sign In” butonuna tıklandığında kullanıcı giriş formunun açılmalı

Test - 03 - Kullanıcı girişi

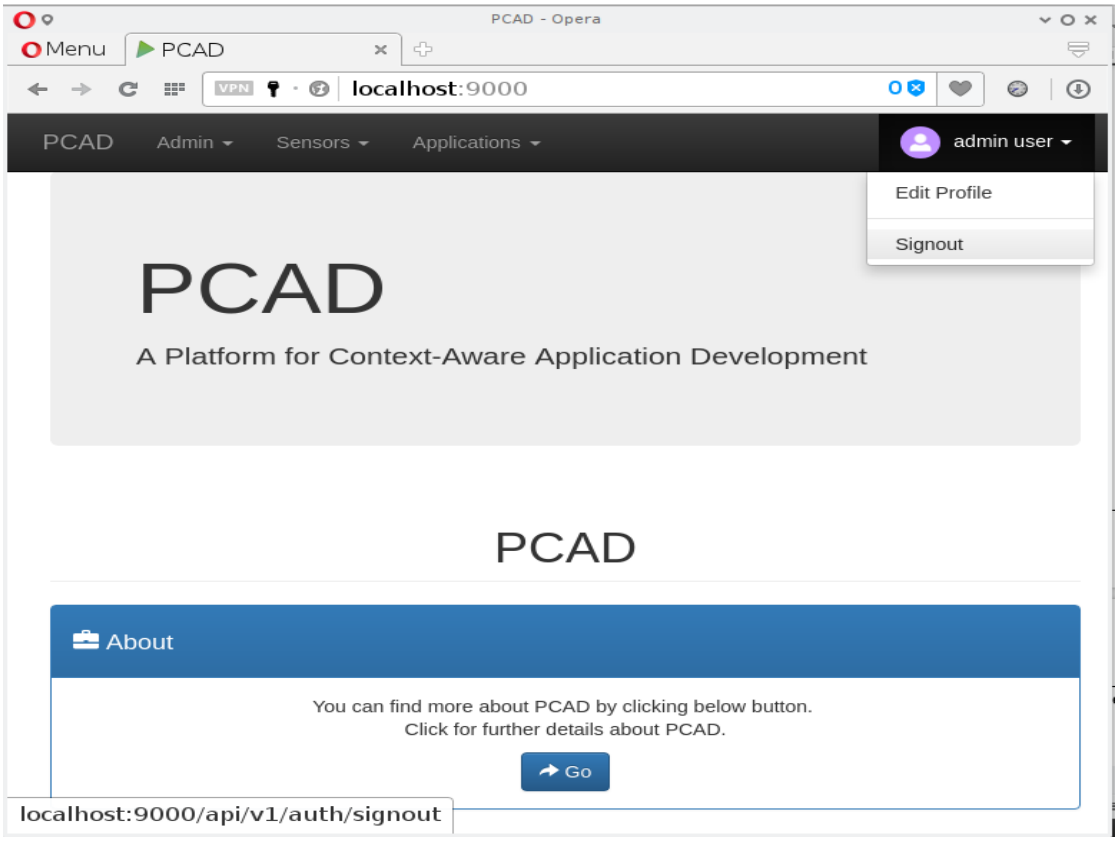


TS-03.1 Bütün alanlar dolu olmalı

TS-03.2 ”Sign In” butonuna tıklandığında sonuç başarılı ise sisteme giriş yapmalı

TS-03.3 ”Sign In” butonuna tıklandığında sonuç hatalı ise uyarı mesajı göstermeli

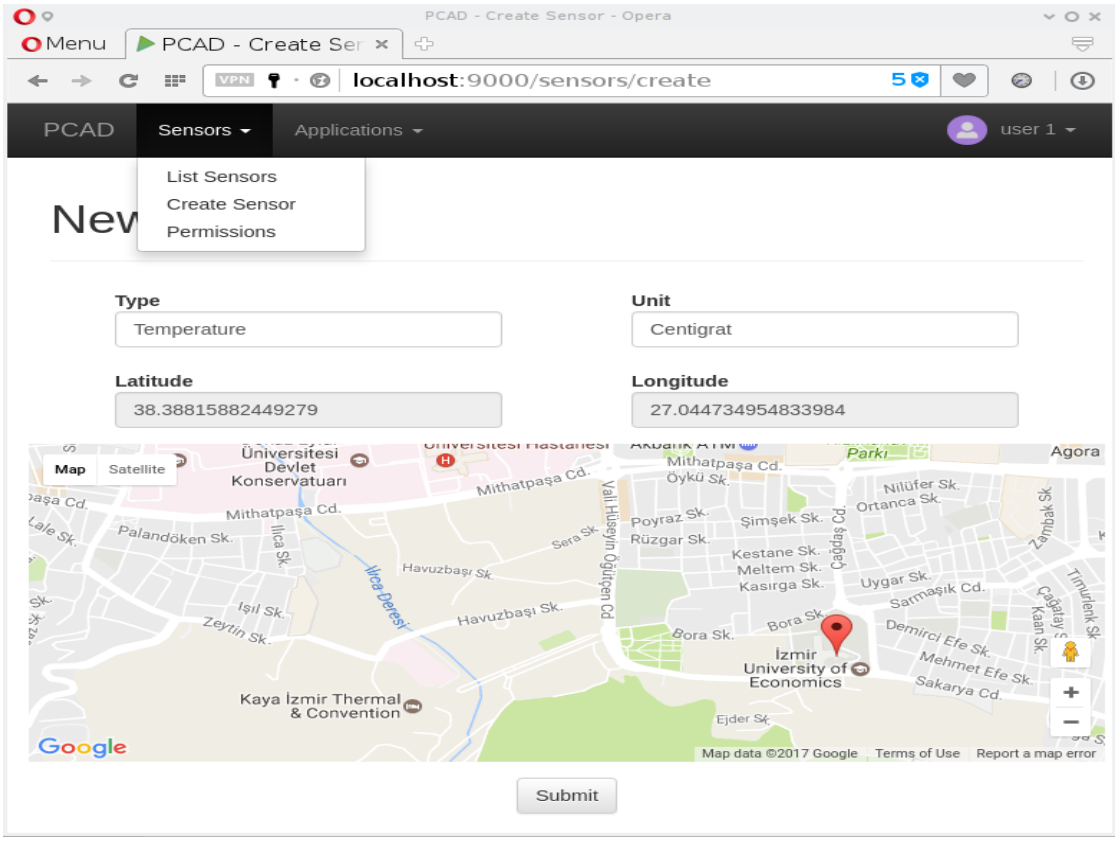
Test - 04 - Kullanıcı çıkışı



TS-04.1 Üst menü çubuğundaki kullanıcı ismine tıklandığında menü açılmalı

TS-04.2 ”Sign Out” butonuna tıklandığında anasayfaya dönüş yapmalı

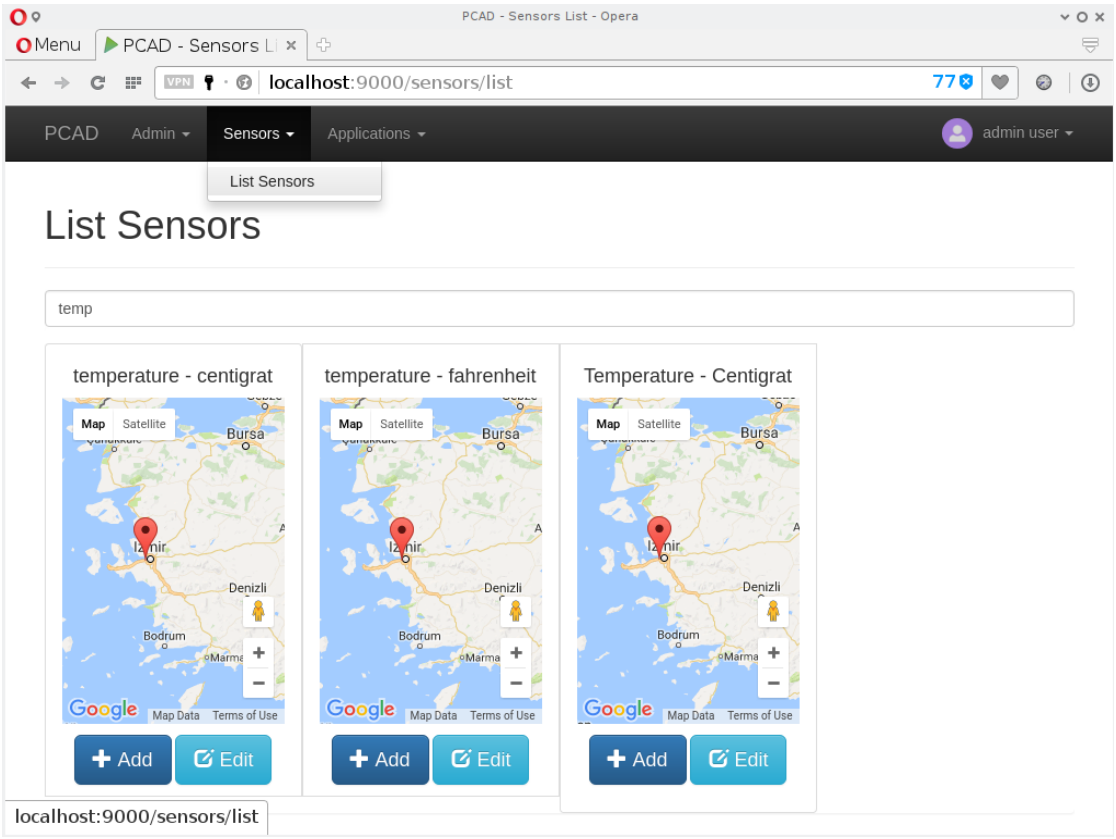
Test - 05 - Sensör kaydı



TS-05.1 Bütün alanlar dolu olmalı

TS-05.2 ”Submit” butonuna tıklandığında sensör listesini göstermeli

Test - 06 - Sensör güncellemesi

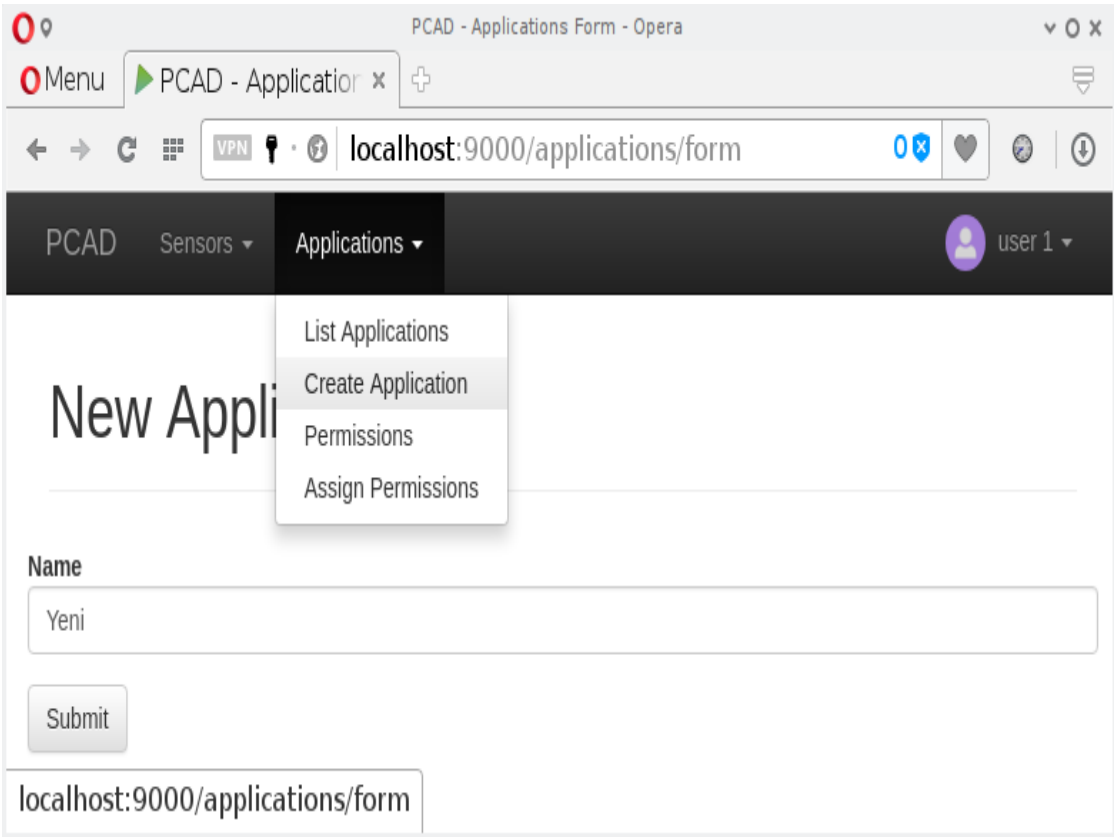


TS-06.1 Admin tipindeki kullanıcı sensörleri listelediğinde her bir sensör için ”Edit” butonu tıklanabilir olmalı

TS-06.2 Kullanıcı *Admin* tipnde değilse ”Edit” buton tıklanamaz olmalı

TS-06.3 ”Edit” butonu tıklandığında sensör güncelleme formuna açılmalı

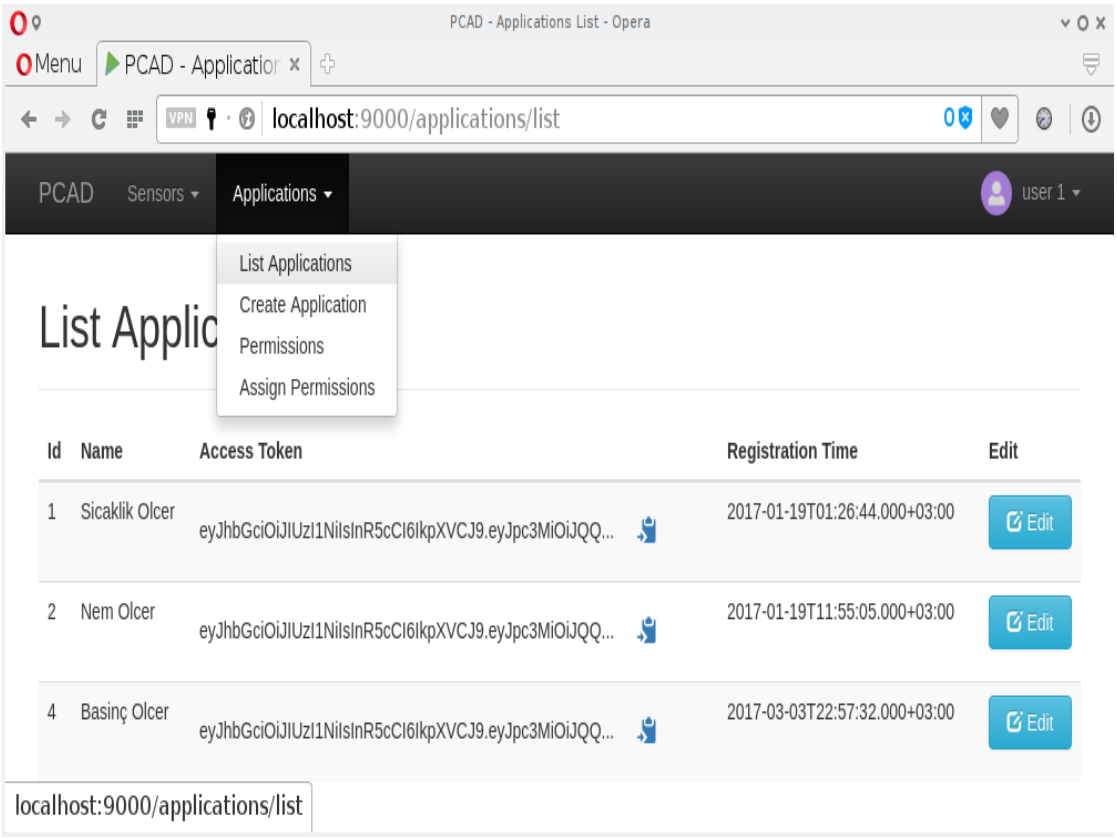
Test - 07 - Uygulama kaydı



TS-07.1 Bütün alanlar dolu olmalı

TS-07.2 ”Submit” butonuna tıklandığında uygulama listeleme sayfasına yönlendirilmeli

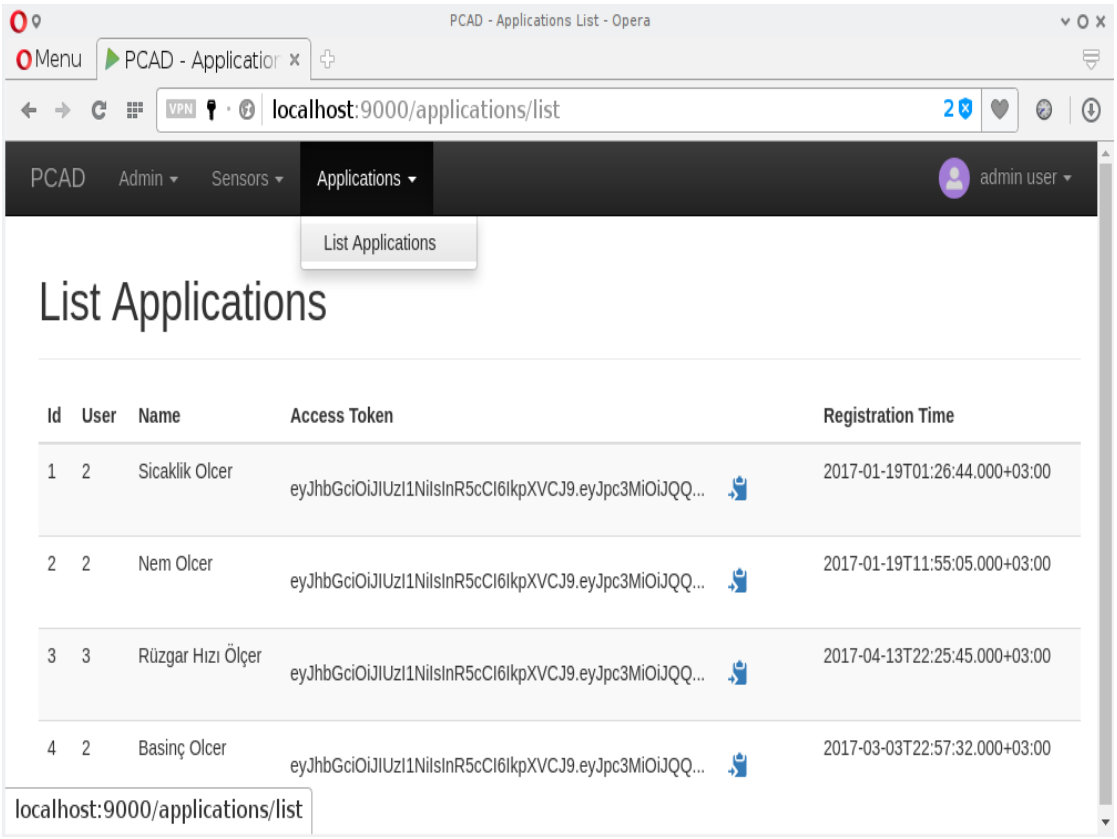
Test - 08 - Uygulama listesi



TS-08.1 Sadece kullanıcıya ait olan uygulamalar listelenmeli

TS-08.2 ”Clipboard” butonuna tıklandığında token kopyalanmalı

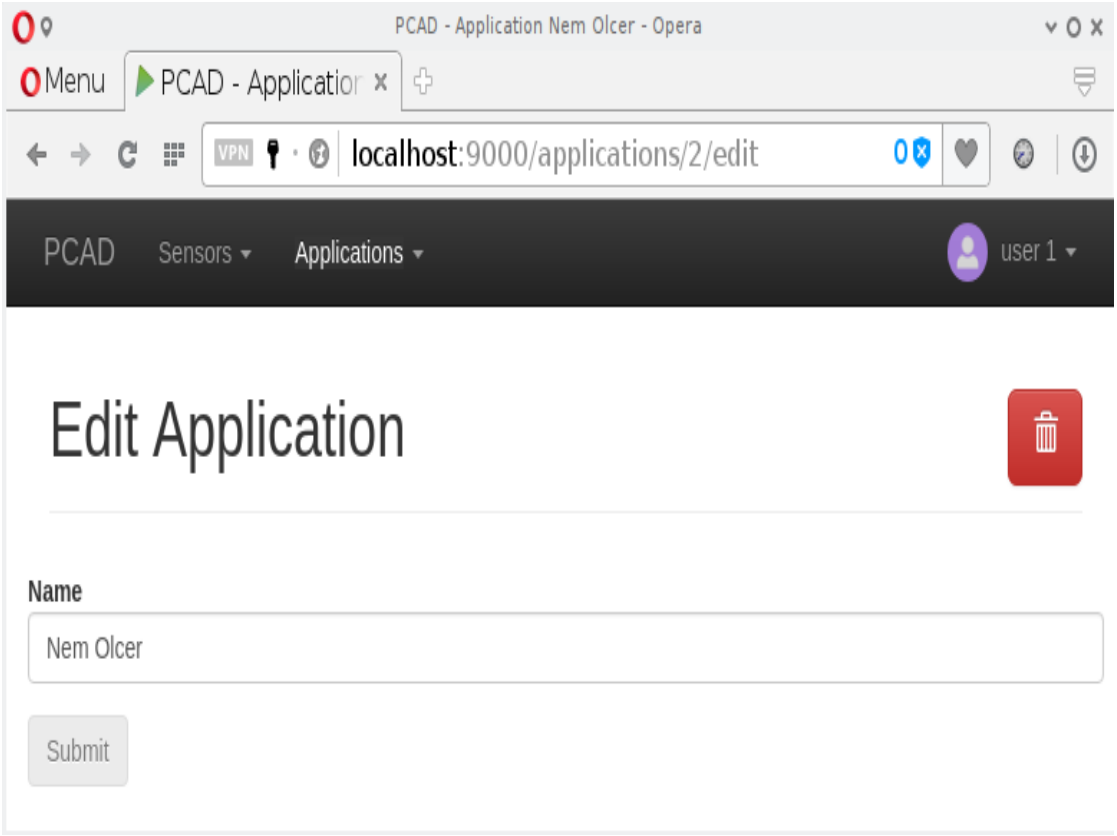
Test - 09 - Admin için uygulama listesi



TS-09.1 Kullanıcı *Admin* tipinde ise bütün uygulamalar listelenmeli

TS-09.2 ”Clipboard” butonuna tıklandığında token kopyalanmalı

Test - 10 - Uygulama güncellemesi

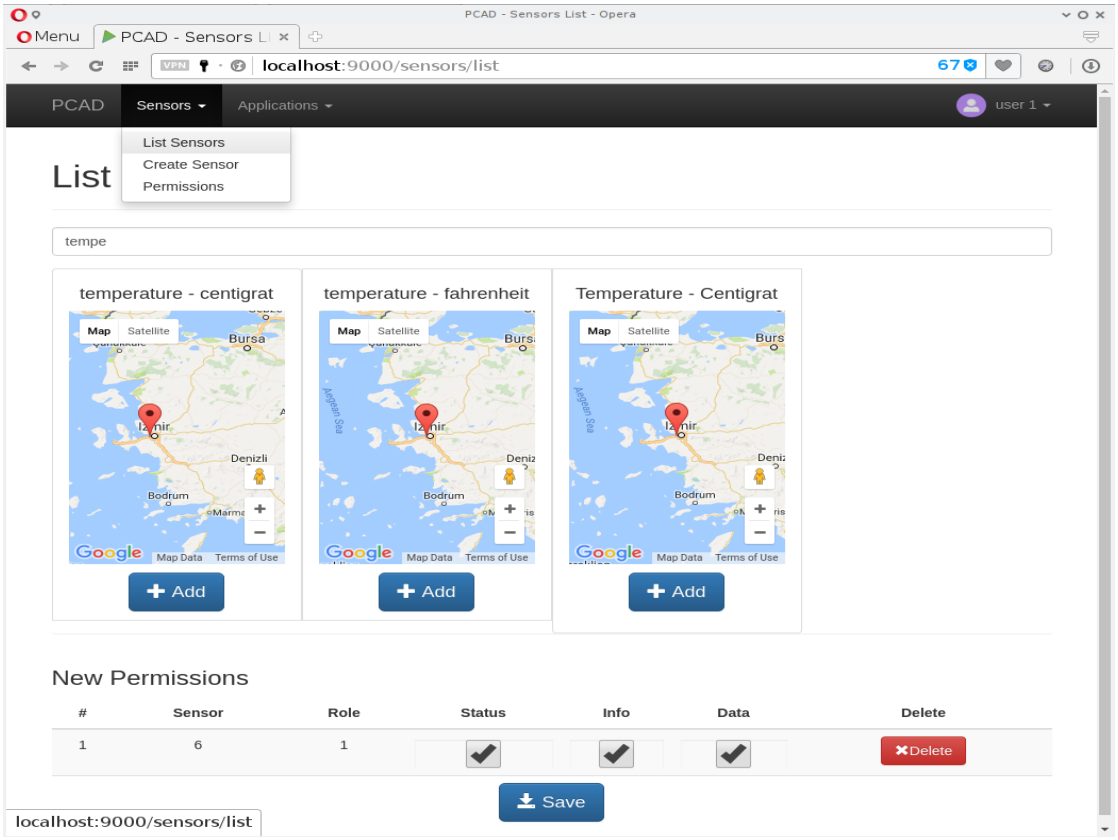


TS-10.1 Bütün alanlar dolu olmalı

TS-10.2 ”Submit” butonuna tıklandığın uygulama listeleme sayfasına yönlendirilmeli

TS-10.3 Çöp kutusu ikonlu butona tıklandığında uygulama listeleme sayfasına yönlendirilmeli

Test - 11 - Kullanıcı izni görüntüleme



TS-11.1 Sensörler listelenmeli

TS-11.2 Sensörler listelenmiyorsa uyarı mesaj görüntülenmeli

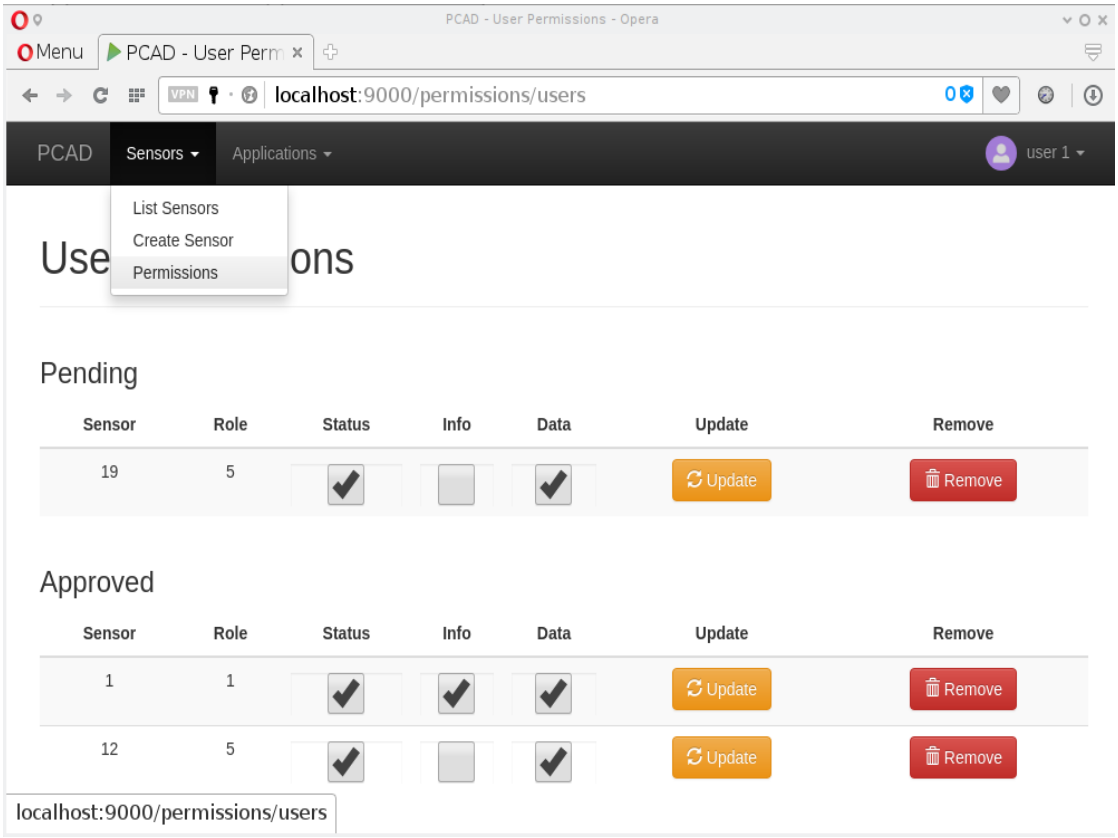
TS-11.3 ”Add” butonu tıklandığında ”New Permissions” isimli tabloya eklenmeli

TS-11.4 Tabloadaki her bir satırda yer alan ”checkbox” tıklanabilir olmalı

TS-11.5 Tabloadaki her bir satırda yer alan ”Delete” butonuna tıklandığında tablodaki o satır çıkartılmalı

TS-11.6 ”Save” butonuna tıklandığında sensör izinleri sayfası açılmalı

Test - 12 - Kullanıcı izni güncellemesi



TS-12.1 ”Pending” isimli tabloda sensör izinleri varsa gösterilmeli

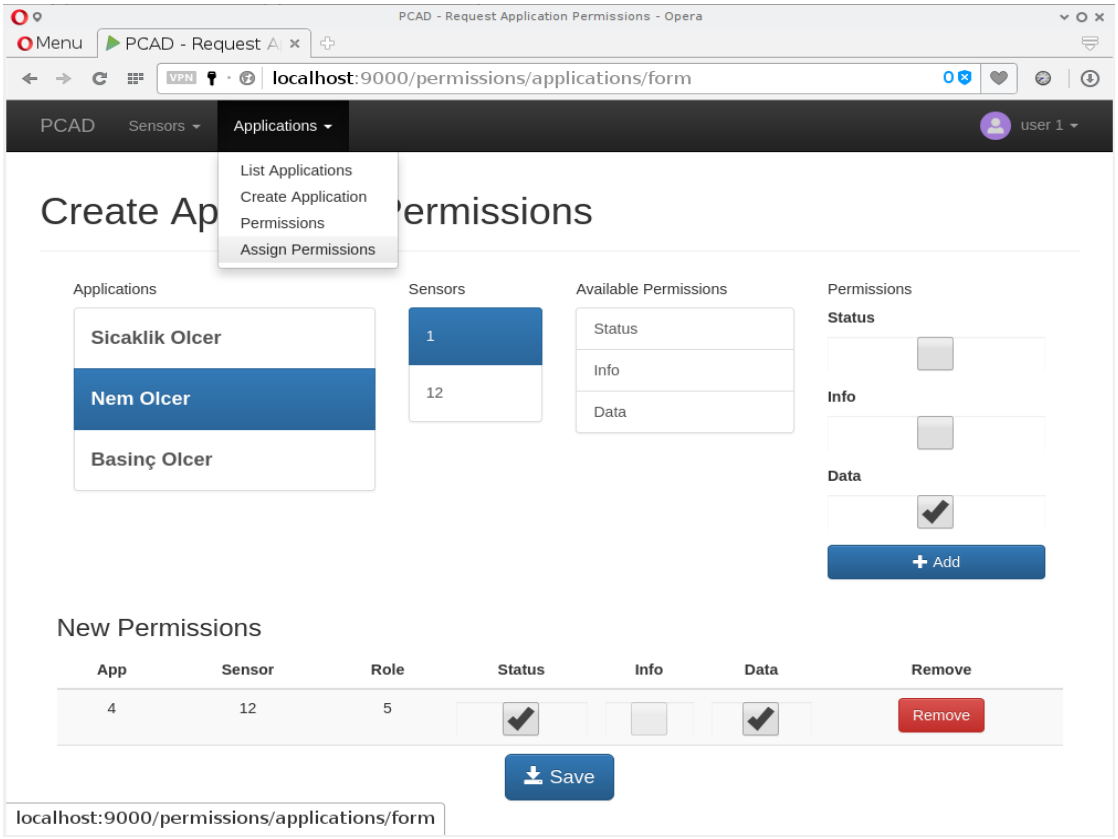
TS-12.2 ”Approved” isimli tabloda sensör izinleri varsa gösterilmeli

TS-12.3 Her bir satırda yer alan ”Remove” butonuna tıklandığında tablodan çıkartılmalı

TS-12.4 ”Pending” tablosunda ”Update” butonuna tıklandığında güncelleme mesajı gösterilmeli

TS-12.5 ”Approved” tablosunda ”Update” butonuna tıklandığında güncelleme mesajı gösterilmeli ve izin üst tabloya aktarılmalı

Test - 13 - Uygulama izinleri



TS-13.1 ”Applications” başlığı altında kullanıcıya ait uygulamalar listelenmeli

TS-13.2 ”Sensors” başlığı altında kullanıcının izini olan sensörler listelenmeli

TS-13.3 ”Permissions” başlığı altında ”Status”, ”Info”, ”Data” isimli ”checkbox”lar tıklanabilir olmalı

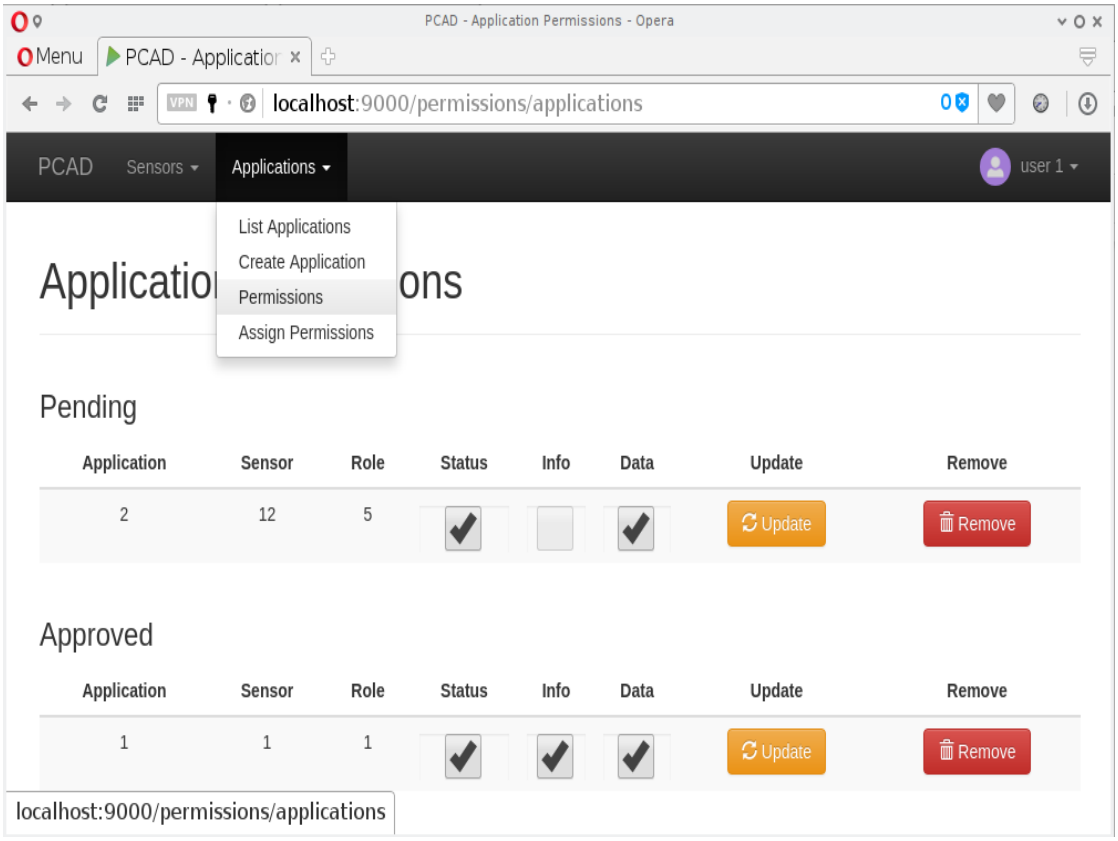
TS-13.4 ”Add” butonun tıklandığında ”New Permissions” isimli tabloya ekleme yapmalı

TS-13.5 Tablonun her bir satırında yer alan ”checkbox”lar tıklanabilir olmalı

TS-13.6 Tablonun her bir satırında yer alan ”Remove” butonuna tıklandığında tablodan çıkartılmalı

TS-13.7 ”Save” butonuna tıklandığında uygulama izinleri sayfası açılmalı

Test - 14 - Uygulama izni güncellemesi



TS-14.1 ”Pending” isimli tabloda sensör izinleri varsa gösterilmeli

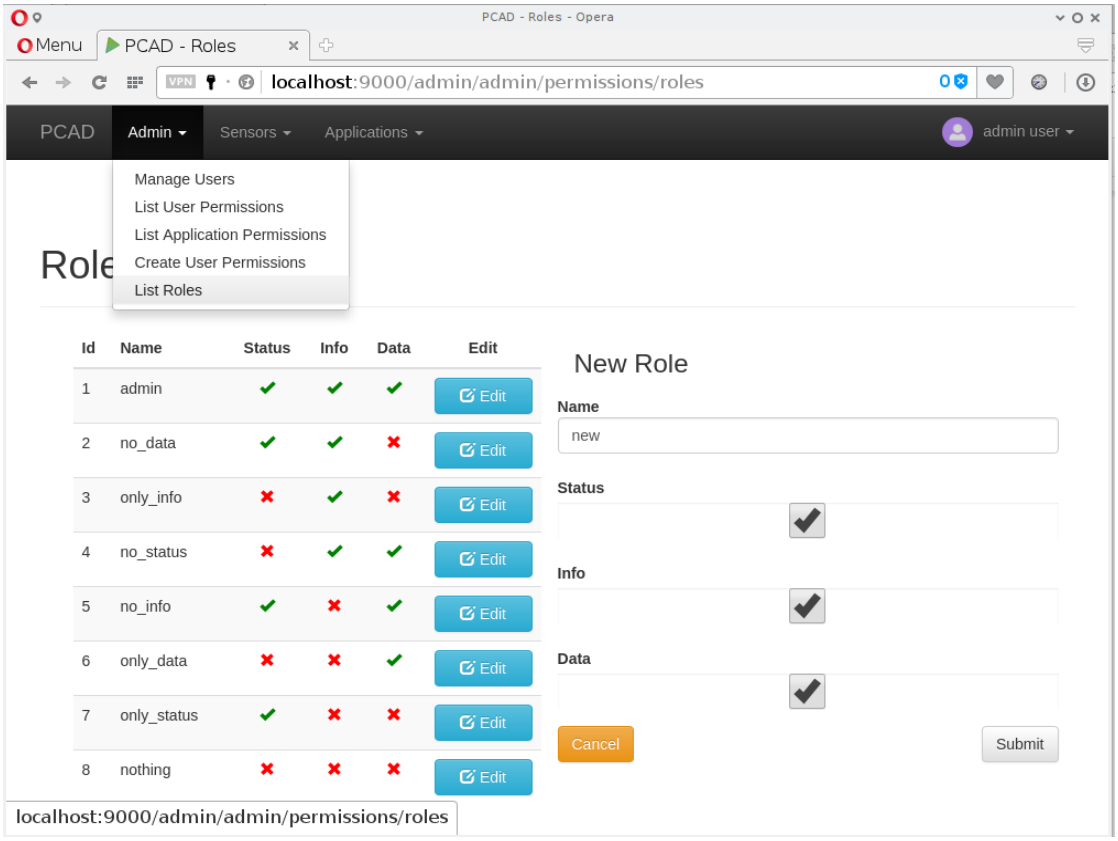
TS-14.2 ”Approved” isimli tabloda sensör izinleri varsa gösterilmeli

TS-14.3 Her bir satırda yer alan ”Remove” butonuna tıklandığında tablodan çıkartılmalı

TS-14.4 ”Pending” tablosunda ”Update” butonuna tıklandığında güncelleme mesajı gösterilmeli

TS-14.5 ”Approved” tablosunda ”Update” butonuna tıklandığında güncelleme mesajı gösterilmeli ve izin üst tabloya aktarılmalı

Test - 15 - Roller ile ilgili işlemler



TS-15.1 Roller, eğer varsa, sol yarım dilimlik sayfada listelenmeli

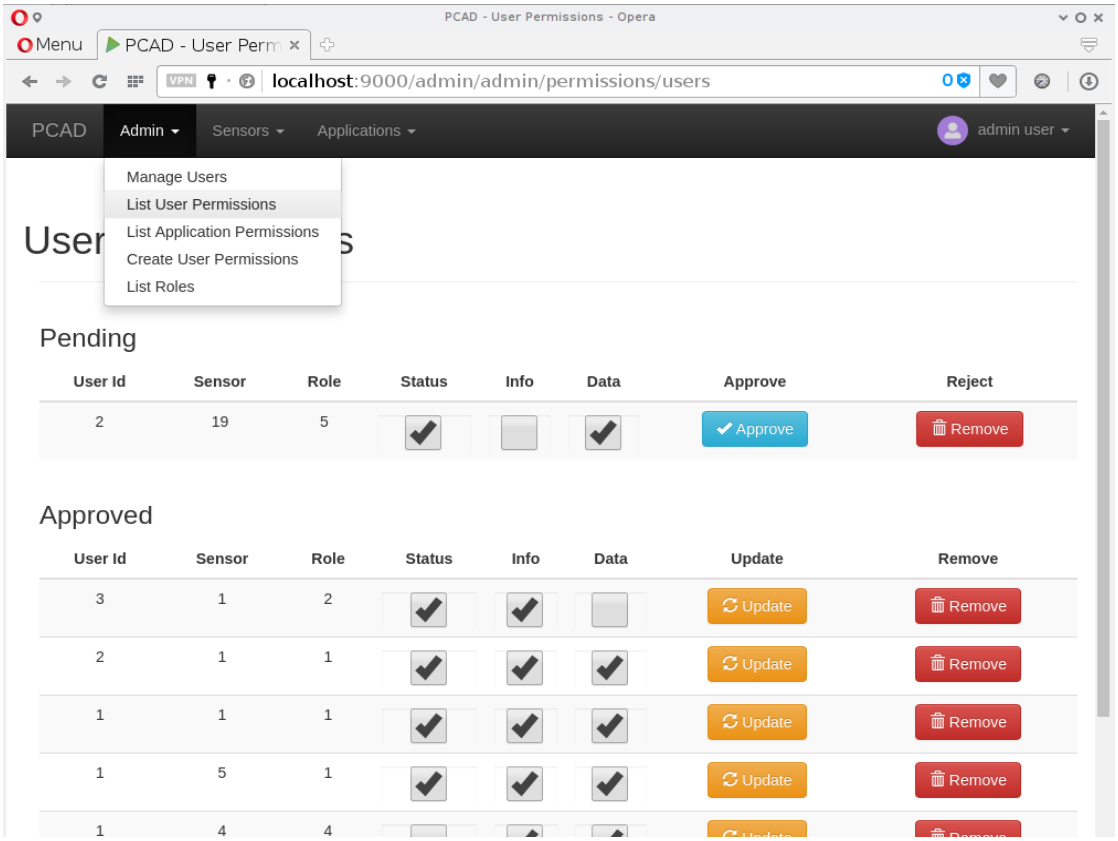
TS-15.2 Sağ yarım dilimlik sayfada yeni rol yaratma butonuna tıklandığında boş rol formu çıkmalı

TS-15.3 Sol bölümde, her bir satırda yer alan ”Edit” butonuna tıklandığında, seçilen rol sağ yarım sayfalık bölümde form içerisinde gösterilmeli

TS-15.4 Formda yer alan ”Cancel” butonuna tıklandığında başlangıç durumuna geri dönmeli

TS-15.5 ”Submit butonuna tıklandığında güncelleme veya yeni kayıt işlemini tamamlayarak başlangıç durumuna dönüş yapmalı

Test - 16 - Kullanıcı izinleri onaylama/reddetme/güncelleme



TS-16.1 ”Pending” isimli tabloda sensör izinleri varsa gösterilmeli

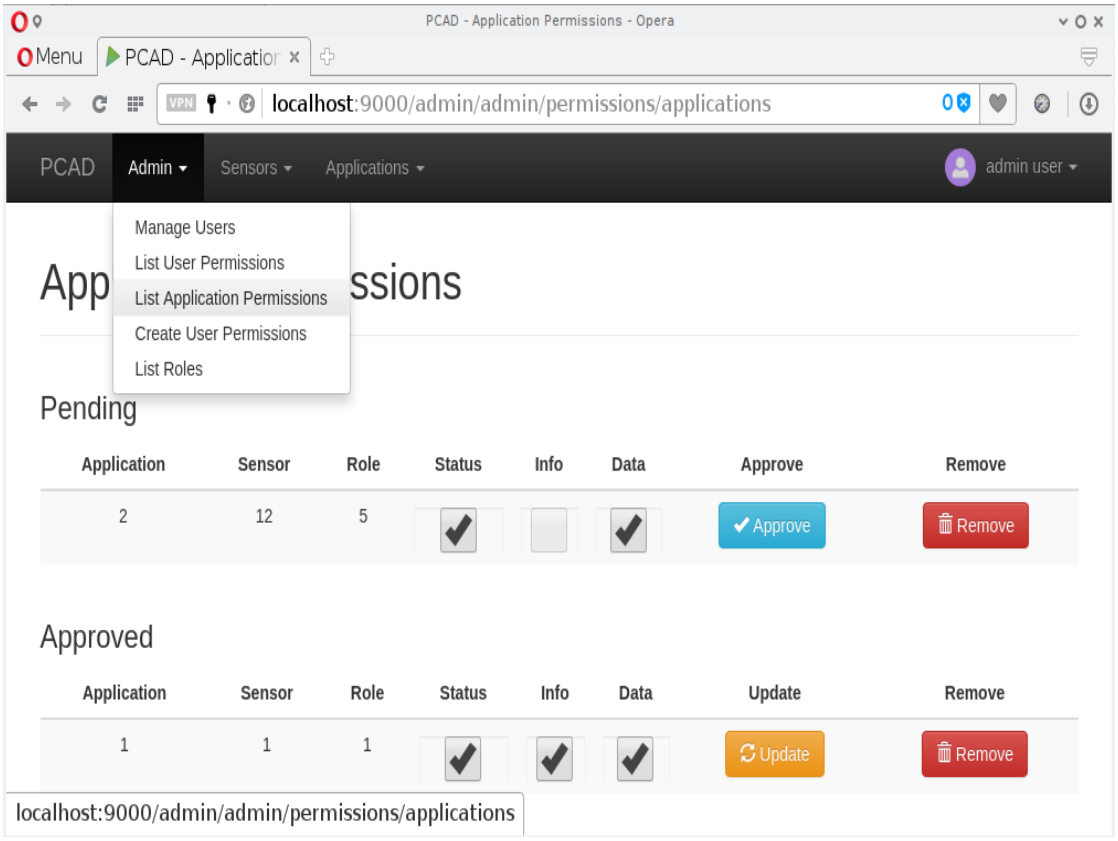
TS-16.2 ”Approved” isimli tabloda sensör izinleri varsa gösterilmeli

TS-16.3 ”Pending” tablosunda ”Approve” butonuna tıklandığında, seçilen izni ”Approved” tablosuna taşımalı

TS-16.4 ”Approved” tablosunda ”Update” butonuna tıklandığında güncelleme mesajı gösterilmeli

TS-16.5 Her bir satırda yer alan ”Remove” butonuna tıklandığında tablodan çıkartılmalı

Test - 17 - Uygulama izinleri onaylama/reddetme/güncelleme



TS-17.1 ”Pending” isimli tabloda sensör izinleri varsa gösterilmeli

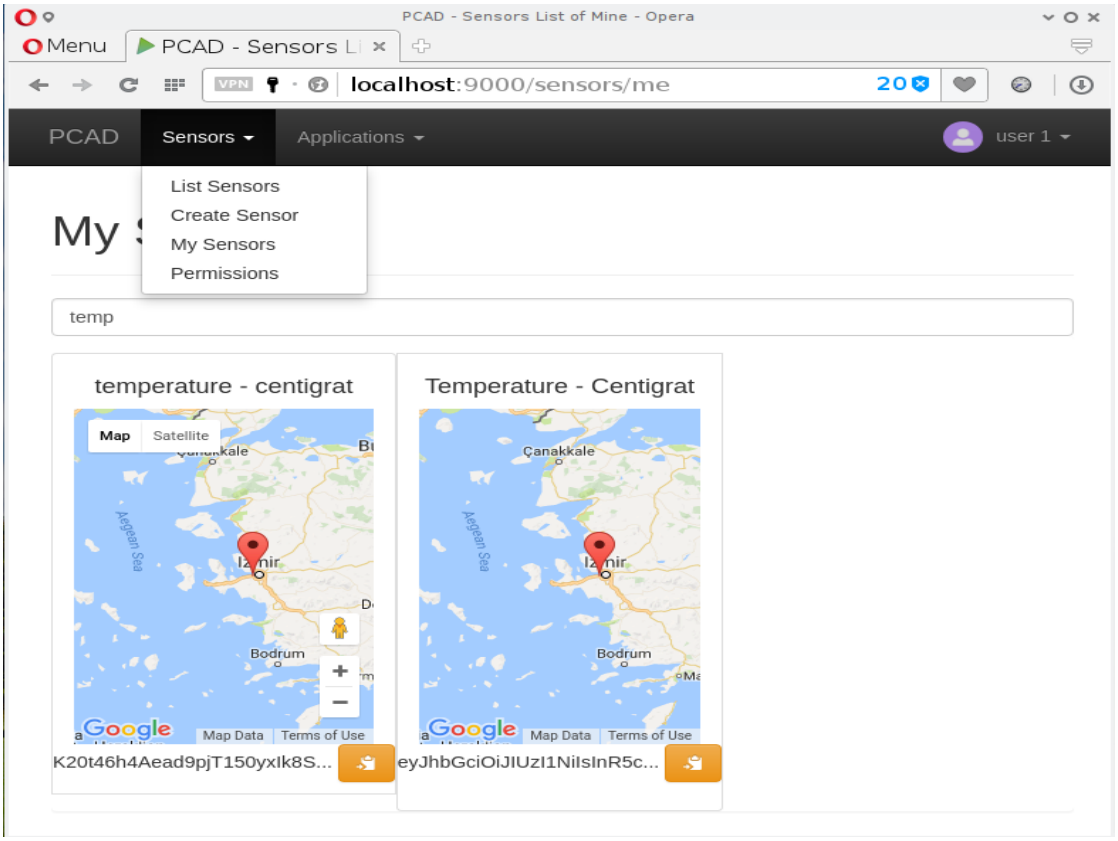
TS-17.2 ”Approved” isimli tabloda sensör izinleri varsa gösterilmeli

TS-17.3 ”Pending” tablosunda ”Approve” butonuna tıklandığında, seçilen izni ”Approved” tablosuna taşımalı

TS-17.4 ”Approved” tablosunda ”Update” butonuna tıklandığında güncelleme mesajı gösterilmeli

TS-17.5 Her bir satırda yer alan ”Remove” butonuna tıklandığında tablodan çıkartılmalı

Test - 18 - Kullanıcının izinli olduğu sensörleri görüntüleme



TS-18.1 Sensörler listelenmeli

TS-18.2 ”Clipboard” butonuna tıklandığında token kopyalanmalı

TÜBİTAK
PROJE ÖZET BİLGİ FORMU

Proje Yürütücüsü:	Yrd. Doç. Dr. UFUK ÇELİKKAN
Proje No:	114E938
Proje Başlığı:	Durum Farkında Servis Platform (Casp) Mimarisi, Tasarımı Ve Geliştirilmesi
Proje Türü:	3001 - Başlangıç AR-GE
Proje Süresi:	24
Araştırmacılar:	KAAN KURTEL
Danışmanlar:	
Projenin Yürütüldüğü Kuruluş ve Adresi:	İZMİR EKONOMİ Ü. MÜHENDİSLİK VE BİLGİSAYAR BİLİMLERİ F. YAZILIM MÜHENDİSLİĞİ B.
Projenin Başlangıç ve Bitiş Tarihleri:	01/05/2015 - 01/11/2017
Onaylanan Bütçe:	99000.0
Harcanan Bütçe:	80259.8
Öz:	Modern bilişim ve iletişim teknolojileri, çok geniş bir çevrede etkili olan bilgisayar ağları ve uygulamaları aracılığı ile çeşitli kaynaklardan verileri toplamakta ve işlemektedir. Nesnelerin interneti olarak da tanımlanan bu kavram, çeşitli haberleşme protokolleri sayesinde bilgi paylaşan ve haberleşen karmaşık bir ağ oluşturmuş cihazlar sistemidir. Bu sistem karmaşık bir yapı göstermekte, farklı yapısal özelliklerde ve değişmeye meyilli fazla sayıda yazılım ve donanım ürününü içinde barındırmaktadır. Nesnelerin interneti kavramını en fazla destekleyen olgu ise, akıllı nesneler olarak tanımlanan sensör, kamera, mikro yongalar ve RFID teknolojileridir. Bu projede, akıllı nesneleri kullanan durum-farkında uygulamalar için genel bir altyapı ve mimari çözüm geliştirilmiştir. İşletim sistemlerinden esinlenen servis tabanlı ara yazılım katmanı içeren mimari sisteminin tasarımında kullanılmıştır. Sensörler, cihazlar, servisler ve uygulamalar birbirinden ayrılmıştır. Projenin temel amacı, sistem yöneticilerinin yeni gereksinimlerden kaynaklanan yeni durum farkındalık bilgileri ile çeşitli servisler arasında, mevcut bilgi sistemine mimaride bir değişiklik yapmadan, gelen verileri uygulamalar ile sorunsuz ve kolaylıkla entegre eden bir yapı geliştirilmesidir. Projede Durum Farkında Uygulama Geliştirme Platformu-PCAD olarak adlandırılan bir platform ve kullanıcı geliştiricileri için bir arayüz kütüphanesi geliştirilmiştir. Projede iki farklı geliştirme iskeleti kullanarak iki alternatif geliştirme yapılmıştır. Bunlardan ilkinde aktör tabanlı bir formalizm olan AKKA iskeleti ile Scala dili ve MySQL ilişkisel veri tabanı kullanılmıştır. İkinci gerçekleştirmede ise Node.js iskeleti, MQTT simsar mimarisi ve NoSQL veri tabanı kullanılmıştır. Her iki gerçekleştirmede de, yazılım uygulamalarının kullanması için RESTful kullanan bir ara yüz sunmaktadır. Kullanıcılar ayrıca bir web arayüzü aracılığı ile veri alabilmekte, uygulamalar ve sensör yazılımlarını kayıt ettirebilmekte, kimlik doğrulayabilme ve erişim haklarını belirleyebilmektedir. Ayrıca AKKA gerçekleştirilmesinde PYTHON dilinde programlama arayüzü kütüphanesi de sunulmaktadır. Her iki geliştirme iskeleti de yazılım mühendisliği esaslarına göre test edilmiş, sistemlerin kurulma ve kullanım kılavuzları hazırlanmıştır. Projede yapılan bu çalışmaların akıllı nesneleri kullanan durum-farkında uygulamalar konusunda yeni projeler yapılmasına önayak olması beklenmektedir. Proje, ekipte yer alan kişilerin akademik ve araştırma becerilerini geliştirmelerine de katkı sağlamıştır.
Anahtar Kelimeler:	Durum Farkında Uygulama Geliştirme Platformu, Yazılım Mimarisi, Uygulama İskeleti, Algılayıcılar
Fikri Ürün Bildirim Formu Sunuldu Mu?:	Hayır

Projeden Yapılan Yayınlar:	1- A Platform for Context-Aware Application Development: PCAD (Bildiri - Uluslararası Bildiri - Sözlü Sunum), 2- Application of Service-Oriented Context-Aware Architecture to Laundry Management System (Makale - İndeksli Makale), 3- A message broker based architecture for context aware IoT application development (Bildiri - Uluslararası Bildiri - Sözlü Sunum),