T.C.
HALİÇ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

HİBRİT UYGULAMALAR İÇİN SOKET ARAYÜZLERİ VE ANDROİD İMPLEMENTASYONU

BİM599 YÜKSEK LİSANS TEZİ

Hazırlayan
Yusuf DAĞLIOĞLU

Danışman
Yrd.Doç.Dr. Ülviye HACIZADE

Mayıs – 2017

FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRLÜĞÜNE

Bilgisayar Mühendisliği A.B.D. Bilgisayar Mühendisliği Yüksek Lisans öğrencisi Yusuf DAĞLIOĞLU tarafından hazırlanan **"Hibrit Uygulamalar için Soket Arayüzleri ve Android İmplementasyonu"** konulu çalışması jürimizce Yüksek Lisans Tezi olarak kabul edilmiştir.

Tez Savunma Tarihi : 08.05.2017

(Jüri Üyesinin Ünvanı, Adı, Soyadı ve Kurumu): İmzası

Jüri Üyesi : Yrd. Doç Dr. Ulviye HACIZADE
Haliç Üniv. (Danışman)

Jüri Üyesi : Prof. Dr. Mübariz EMİNLİ
Haliç Üniv.

Jüri Üyesi : Yrd. Doç.Dr. Alev MUTLU
Kocaeli Üniv.

Bu tez Enstitü Yönetim Kurulunca belirlenen yukarıdaki jüri üyeleri tarafından uygun görülmüş ve Enstitü Yönetim Kurulunun kararıyla kabul edilmiştir.

Prof. Dr. Oya Oğuz
Fen Bilimleri Enstitüsü Müdür V.

ÖNSÖZ

Eğitimim süresince desteği esirgemeyen, tez çalışmamda büyük katkılarda bulunan Yrd.Doç.Dr. Ülviye Hacızade'ye, beni bugüne getiren aileme sonsuz teşekkürlerimi borç bilirim.

İstanbul 2017 Yusuf DAĞLIOĞLU

İÇİNDEKİLER

Sayfa No.

KISALTMALAR

CSS : Cascading Style Sheets (Basamaklı Stil Şablonları)

HTML : Hyper Text Markup Language (Zengin Metin İşaret Dili)

AJAX : Asynchronous Javascript and XML (Eşzamansız Javascript ve XML)

IP : Internet Protocol

XML : Extensible Markup Language (Genişletilebilir İşaretleme Dili)

HTTP : Hyper-Text Transfer Protocol (Hiper Metin Transfer Protokolü)

TCP : Transmission Control Protocol (İletim Denetimi Protokolü)

UDP : User Datagram Protocol (Kullanıcı Veri Bloğu İletişim Kuralları)

JDK : Java Development Kit (Java Geliştirme Kiti)

ADT : Android Development Tools (Android Geliştirme Araçları)

MQTT : Message Queue Telemetry Transport (Mesaj Dizilerini Uzölçüm İletimi)

SDK : Software Development Kit (Yazılım Geliştirme Kiti)

NPM : Node Package Manager (Node Paket Yöneticisi)

API : Application Programming Interface (Uygulama Programlama Arayüzü)

TABLOLAR

Sayfa No.

ŞEKİLLER

Sayfa No.

GENEL BİLGİLER

Adı ve Soyadı : Yusuf DAĞLIOĞLU

Anabilim Dalı : Bilgisayar Mühendisliği

Programı : Bilgisayar Mühendisliği

Tez Danışmanı : Yrd.Doç.Dr. Ülviye HACIZADE

Tez Türü ve Tarihi : Yüksek Lisans – Mayıs 2017

ÖZET

HİBRİT UYGULAMALAR İÇİN SOKET ARAYÜZLERİ VE ANDROİD İMPLEMENTASYONU

Günümüz ihtiyaçları mobil gibi taşınabilir platformların daha çok kullanılmasını gerektirmektedir. İhtiyaçların bu yönde artması mobil ve diğer taşınabilir platformlar üzerindeki teknolojik çalışmaları hızlandırmıştır. Bu hızlı artış ortaya alışılmışın dışında çok farklı teknolojilerin ve metotların ortaya çıkmasına sebep olmaktadır. Özellikle özgür yazılım toplulukları standartların açık olabilmesi için büyük çaba sarf etmekte ve standartların açık olması teknolojinin gelişimini kat kat daha hızlandırmaktadır. Bu çalışmada; güncel ve platformdan bağımsız yazılım teknolojiler incelenmiş ve web teknolojilerini kullanan (hibrit) uygulamalar için Javascript tarafından bir arayüz (API) aracılığı ile işletim sisteminin ana dilindeki soket ve MQTT protokolüne erişim sağlanmıştır.

GENERAL INFORMATION

Name and Surname : Yusuf DAĞLIOĞLU
Field : Computer Enginnering
Program : Computer Enginnering
Supervisor : Asst.Prof.Dr. Ülviye HACIZADE
Degree Awarded and Date : Master of Science – May 2017

ABSTRACT

SOCKET INTERFACE AND ANDROID IMPLEMENTATION FOR HYBRID APPLICATIONS

Today's needs require portable platforms such as mobile to be used more widely. This requirement has expedited the technological studies on mobile and farther portable platforms, which resulted in the emergence of various unusual technologies and methods. Particularly, the open source software communities strive in order for the standards to be open and that the standards are open tremendously accelerates the development of technology. In this study, actual and platform-independent software technologies are analyzed and offered an access to socket and MQTT of the native language of operating system through Javascript interface (API) for web technologies based (hybrid) applications.

1. GİRİŞ

Çağın gereksinimleri birçok farklı platformun ortaya çıkmasına sebep olmuştur. Platformların artışı, yazılım dünyasında farklı ihtiyaçların doğmasına sebep olmaktadır. Geliştiriciler ve son kullanıcı açısından bakıldığında her platformun avantajları ve dezavantajları görülmektedir.

Son kullanıcı gereksinimlere cevap verirken, tüm platformları göz önünde bulundurmak yazılım geliştiricilerine büyük yük olmaya başlamıştır. Ne kadar da bazı son kullanıcı ürünleri belli platformlara destek verip bu yükü azaltmaya çalışsa da, günümüz teknolojilerindeki değişimler ve yeni buluşlar destek alanını ve geliştirme ortamlarını çok farklı bir biçime sokmaktadır.

Günümüzde özellikle mobil platformlar tercih edilmektedir. Bu durum yazılım geliştiricilerini, pazarın yoğun olduğu bu alanda, çalışmalar ve yeni fikirler ortaya koymalarına sebep olmaktadır. Yeni fikirler yeni yapılanmaların ortaya çıkmasına yol açmaktadır. Birçok farklı yapılanma, yazılım geliştiricilerini oldukça zorlanmaktadır. Uygulamalar büyük oranla internete ihtiyaç duymakta ve bu da haberleşme protokollerinin de kullanımını gerektirmektedir. Soket altyapısı alt seviyeli bir haberleşme protokolü olup, birçok diğer üst seviyeli haberleşme protokollerinin çekirdeğini oluşturmaktadır. Soket altyapısı baz alınarak geliştirilen üst seviyeli protokoller, geliştiriciler tarafından kullanıldıklarında, soketleri direk olarak değil, üst seviyeli protokolleri kullanarak işlem yapmaktadırlar. Bu sebeple soket temelli programlama daha çok yazılım kütüphaneleri geliştirilirken tercih edilmektedir. MQTT protokolü ise günümüzde IoT (Internet of Things – Nesnelerin İnterneti) platformlarında tercih edilmekte olup, platformdan bağımsız bir yapıya sahiptir. Bu bağlamda; bu çalışmada, MQTT ve soketin önemi gözetilerek; web teknolojilerini kullanan mobil uygulamalar (hibrit) için soket ve MQTT kullanımını sağlayan bir mimari geliştirilmiştir.

Bu çalışmada hibrit geliştirme platformları için avantajlar ve dezavantajlar araştırılmış, hibrit uygulama modellerinin önemi ortaya koyulmuştur. Web teknolojileri üzerinde farklı haberleşme protokolleri ve bunların teknolojik altyapıları incelenmiştir. Haberleşme protokollerinin avantajları ve dezavantajları ortaya koyulmuştur. Web teknolojileri üzerine geliştirilen uygulamalar ve uygulamalar için yazılan eklentiler incelemeye alınmıştır. Aynı zamanda bu çalışmada; yazılımcıların soket ve MQTT üzerinden haberleşebilmeleri için bir arayüz yazılmıştır. Bu arayüz, web teknolojilerinin baz aldığı Javascript dilinden ve işletim sisteminin ana dilinden çağrılacak şekilde tasarlanmıştır. Tasarlanan bu arayüzün bellek alanı ihtiyacı incelenmiştir.

2. YAZILIM TEKNOLOJİLERİ

2.1. Farklı Platformlarda Yazılan Uygulamaların Önemi

Tüm platformlarda olduğu gibi, mobil platformların üzerinde yürütülen farklı işletim sistemi sayısı gün geçtikçe artmaktadır. Bu sayının artışı ile mobil platform üzerinde uygulama geliştirme maliyeti ve süreci artış göstermektedir. Mobil platformların pazar payının hızla artışı mobil uygulama yazan geliştiricilerin farklı platformlara destek verme zorunluluğunu arttırmaktadır [1-3]. Farklı işletim sistemlerinin mobil cihazları üzerinde ekran görüntüleri, işletim sistemi marka ve logoları ile Şekil 2.1'de gösterilmiştir [4].

Şekil 2.1. Farklı işletim sistemlerinin mobile üzerinde görüntüleri

2.2. Farklı Platformlarda Yazılım Geliştirme Zorlukları

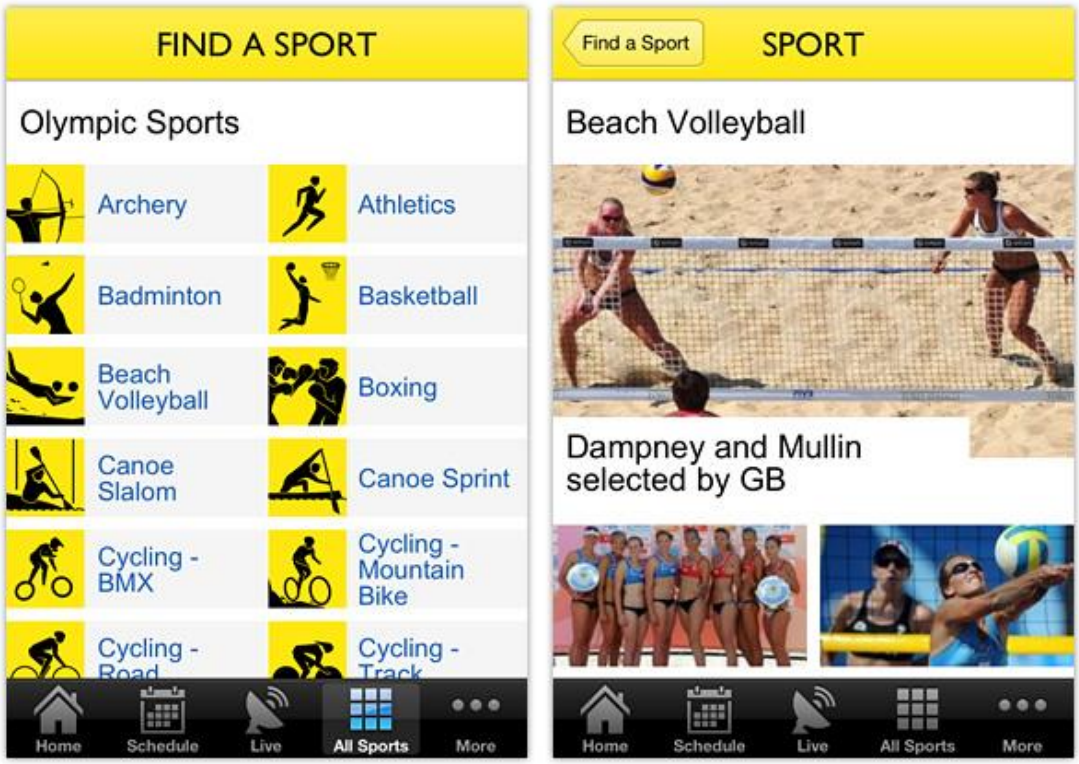
Yazılım geliştiricilerinin birçok işletim sistemine destek verirken yaşadıkları zorlukları azaltmak için birçok konuda, birçok farklı çözüm üretilmektedir. Fakat farklı işletim sistemlerine destek vermek için yazılımcıların karşısına çıkan en büyük zorluk, her platform için farklı dillerle uygulama sıfırdan geliştirmektir. Örneğin

Windows Phone için C# kullanmak ve hatta Visual Studio yazılım geliştirme platformunu kullanmak zorunludur. Android üzerinde çalışan uygulamalar için Java, Apple'ın iOS işletim sistemine bir uygulama yazmak için Objective C kullanmak gerekmektedir. Diğer platformlar da hesaba katıldığında, önemli bir maliyet olduğu görülmektedir [5-7].

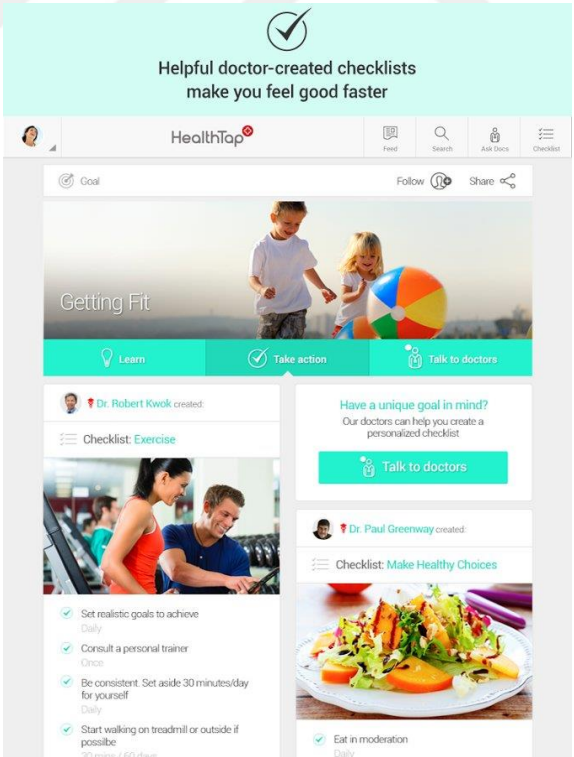
2.3. Hibrit Uygulama Modeli

Farklı işletim sistemlerine destek için harcanan maliyetleri düşürmek için çözüm olarak hibrit uygulama modeli geliştirilmiştir. Hibrit uygulamalar, adından da anlaşıldığı gibi farklı dillerden ve katmanlardan yararlanılarak yazılırlar. Mobil uygulamalarda hibrit uygulamalar kısmen web teknolojilerinden yararlanılırken, kısmen de işletim sisteminin ana dilinden (native language) yararlanılır [8-11]. Hibrit uygulamalara örnek olarak HealthTap ve BBC Olympics uygulamaları verilebilir. Ekran görüntüleri Şekil 2.2, Şekil 2.3, Şekil 2.4 ve Şekil 2.5'te verilmiştir [12, 13].

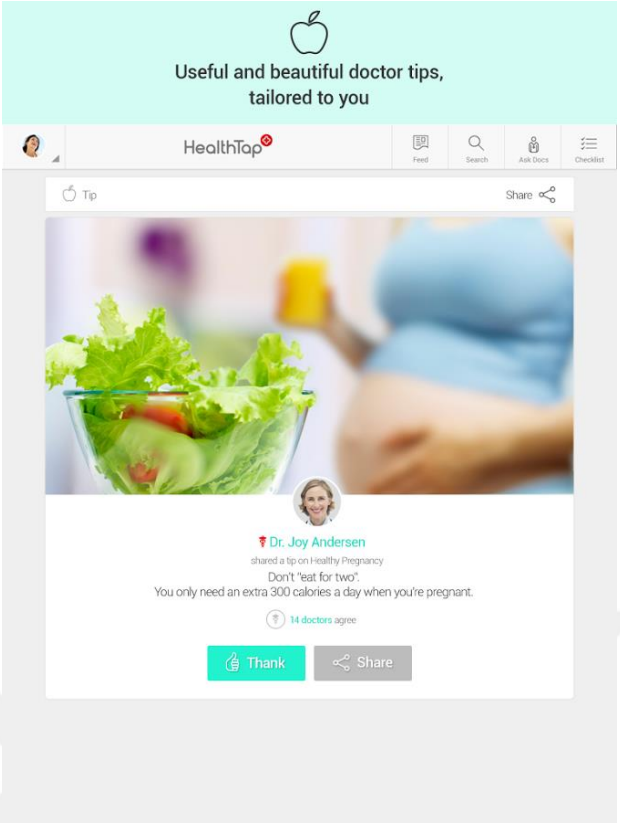
Şekil 2.2. BBC Olympics uygulamasının açılış ekranı ve 'Canlı' menüsü



Şekil 2.3. BBC Olympics uygulamasının 'Tüm Sporlar' ve haber akışı sayfası



Şekil 2.4. HealthTap uygulamasının bilgi akış sayfası



Şekil 2.5. HealthTap uygulamasının profil sayfası

Hibrit uygulamalar çalıştırıldığında ana dilde yazılan uygulama normal olarak başlar. Ana dilde yazılmış uygulama, başlangıçta kendi penceresi dahilinde işletim sisteminin grafik arayüzünde sunduğu web bileşenini (component, view) açmaktadır. Web bileşeni üzerinde, uygulama kendi içine yüklediği web teknolojilerini çalıştırmaya başlamaktadır. Bu web bileşeni çalıştırılırken, sadece bir grafik arayüz bileşeni olarak çalışmakta olduğundan, ana dilde yazılan diğer kodlar ve bileşenlerde çalışmaya devam etmektedir. Bu şekilde uygulama hibrit yapıda çalışmaktadır [8, 9, 14, 15].

Hibrit yapıda çalışan uygulama içerisinde, web bileşeni ile ana dildeki kod üzerinde çalışan kodlar kendi aralarında birbirlerine veri aktarımı yapabilmektedirler. Dolayısı ile web teknolojilerinin kısıtlamaları ana dildeki kod tarafına erişilerek tamamlanabilmektedir. Bu şekilde web teknolojilerin kısıtlamaları aşılabilmektedir [16, 17]. Örnek olarak; web tarayıcılarının verdiği uyarı penceresi yerine, o anda çalışmakta olan işletim sisteminin varsayılan uyarı penceresini kullanmak için

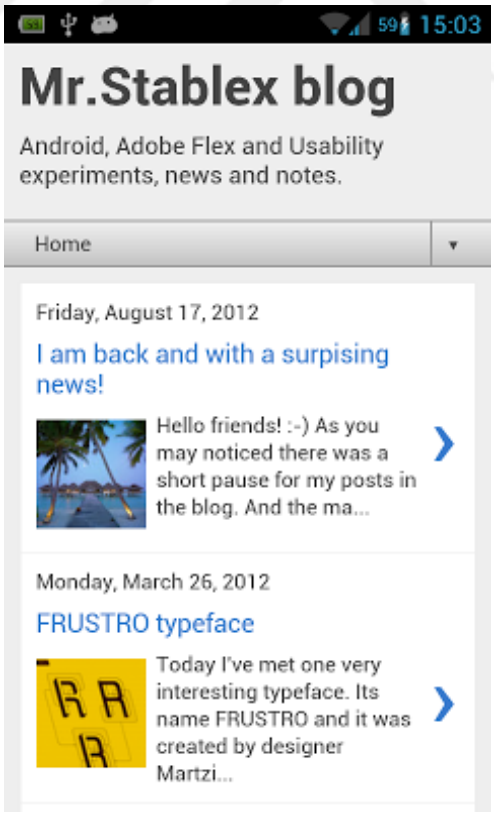
yapılmış bir eklentinin, iOS ve Android üzerinde çalıştığı zamanki ekran görüntü Şekil 2.6'da verilmiştir [18].

Şekil 2.6. İOS ve Android üzerinde ana dilde uyarı penceresi

2.4. Hibrit ve Ana Dil Kullanımındaki Dezavantajlar ve Avantajlar

Hibrit uygulama, hibrit kısmı yürütebilmek için web teknolojilerinden yararlanır. Web teknolojilerini kullanabilmek için onları yürütebilen altyapıyı da kullanması gerekmektedir. Aynı bir web tarayıcısı (Firefox, Chrome, Safari...) gibi arka planda tüm web teknolojilerini (Javascript, CSS (Cascading Style Sheets), HTML (Hyper Text Markup Language) gibi) yürütebilmeli ve yürütülen ana dilde yazılmış uygulama içerisinde bağımsız şekilde kodları derleyebilmelidir [19]. Hibrit uygulamalarda olduğu gibi tarayıcı mekanizmasını barındıran WebView nesnesi (component) ekran görüntüleri Şekil 2.7 ve Şekil 2.8'de verilmiştir [20, 21].

Şekil 2.7. Android üzerinde hibrit hesap makinası



Şekil 2.8. Android üzerinde WebView kullanarak yapılmış pencere

Hibrit uygulama, web teknolojilerini destekleyebilmek için arka planda normalden fazla şekilde işlem yapması ve fazla bellek kullanımı gerekmektedir. Bu noktada hibrit uygulamanın dezavantajı ortaya çıkmaktadır. Sadece ana dille yazılıp çalışan bir uygulama ekstra bellek alanına ihtiyaç duymaz. Oysa hibrit uygulama yazarken, normal uygulamaya ekstradan bir tarayıcı ve bu tarayıcı üzerinde birden fazla sekme açıkmış gibi düşünülmelidir. Bu sebeple gözle görülür önemli bir yavaşlık söz konusu olmaktadır [19].

Hibrit uygulamanın başka bir dezavantajı ise web teknolojilerinin, ana dilde yazılmış uygulamalar kadar hızda yürütülememesidir. Web teknolojileri kısmında yürütülen kodlar (Javascript, HTML, CSS...), derleyiciler tarafından işlendikten sonra işletim sisteminde çalıştırılır. Dolaysıyla, gerçek zamanlı derleme işlemi süreci karşımıza çıkmaktadır. Hibrit uygulamalarda en büyük performans farkını bu faktör meydana getirmektedir. Tüm kodların kütüphanelerin derlenmesi ve çalıştırılması bir yavaşlama meydana getirmektedir. Hibrit yazılım geliştiricileri uygulamanın her geliştirilme noktasında bu faktörü göz önünde bulundurmaları gerekmektedir. Aksi halde son kullanıcıya hitap edilen uygulamalar yazılması yavaş yavaş imkansız hale gelebilmektedir [19].

Hibrit uygulama yazımında karşımıza çıkan bir başka dezavantaj ise kodların her platformda rahat şekilde hata ayıklanamamasıdır (debugging). Ne kadarda bunun için farklı çözümlerde sunulsa da, Javascript kodları ana programlama dili kodları kadar her platformda rahat hata tespit (hata ayıklama, İngilizce: debugging) edilememektedir [19, 22, 23].

Hibrit uygulamalar performans hususlarında dezavantaj meydana getirse de, diğer açılardan yazılımcılara büyük avantajlar sağlayabilmektedir. Hibrit uygulamalar yazılımcının her platform için ayrı ayrı ana dilde kod yazma gereksiniminden kurtarmaktadır. Çünkü web teknolojileri standarttır ve her platformda aynı şekilde çalışır. Dolayısı ile web teknolojilerinde yazılan uygulama farklı sistemlerde aynı şekilde çalışması beklenmektedir. Nasıl ki bir kullanıcı web sayfalarında gezdiğinde, tüm işletim sistemlerinde görüntü ve arka plandaki işleyiş değişmiyor ise, hibrit uygulamalarda yazılan kodlar da değişmemektedir. Sadece ana dilde olan kısmı değişmektedir. Ana dildeki kod çok daha az miktarda olacağından,

maliyet neredeyse sıfıra düşmektedir. Bu hibrit uygulamaların en önemli avantajlarındandır [19].

Her ne kadarda hibrit uygulama teorik olarak web standartlarını kullanıyor olsa da pratikte her zaman istendiği gibi çalışmayabilir. Bazı tarayıcılar güncel web standartlarına gerektiği gibi destek vermemektedirler. Bu da yazılımcının özellikle test sürecini uzatabilir. Zira her platformda uygulamanın testi ve karşılaşılan platforma özgü sorunları çözmek gerekebilir. Web standartları desteklenmediğinde yazılımcının yapabileceği tek şey alternatif çözümlere gitmektir. Fakat kod tüm platformlar için ortak olduğu için yapılacak değişiklikler tüm platformlara uygulanmak zorundadır. Bu da test ve hata giderim sürecini uzatmaktadır [19, 24, 25, 26].

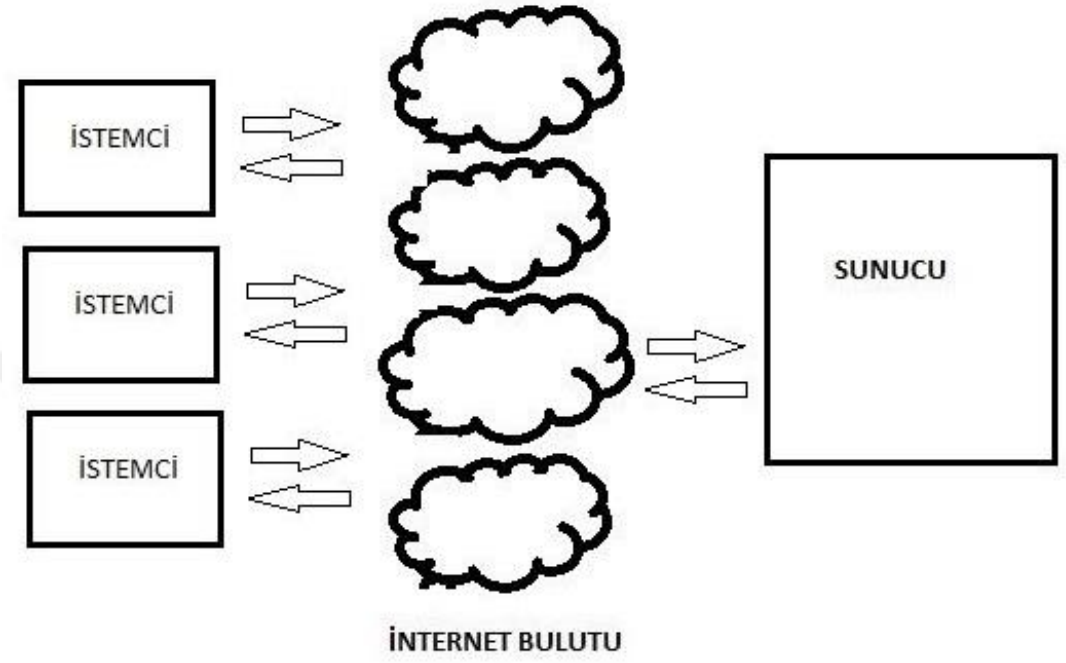
2.5. Cordova

Hibrit uygulama modelinden yararlanabilmek için hibrit altyapısını sunan bir iskeletten (framework) yararlanmak gereklidir. Piyasada birçok alternatif mevcuttur. Fakat özellikle açık kaynak olması ile ön plana çıkan Cordova projesi, mobil platformlar için hibrit uygulama iskeleti sunmaktadır. Eklenti altyapısı aracılığı ile Javascript tarafındaki nesneleri, işletim sisteminin ana diline aktarmamızı sağlamaktadır. Bu şekilde web teknolojilerinin yanı sıra, işletim sisteminin ana dilinden de yararlanılabilmektedir.

2.6. İstemci-Sunucu Mimarisi

İstemci-sunucu mimarisi; birden fazla istemcinin sunucuya istek yaptığı mimaridir. En belirgin örneklerden biri; internet sitelerinin sunucu ile haberleşme mimarisidir. Bir internet sitesine web tarayıcısından gidildiğinde; internet sitesi sayfanın ihtiyaçlarına göre sunucuya istek yapar. Sunucu bu isteklere cevap döner ve web sayfası aldığı cevaba göre sayfada değişiklik yapar. Bu örnekte; web sayfası istemci rolündedir. Sunucu tektir ve sabit bir erişim adresi vardır. Çünkü ancak bu şekilde birden fazla istemci sunucuya bağlanabilecektir. İstemcinin erişim adresi değişken olabilir. Fakat sunucu sadece ona gelen isteklere cevap vereceği için, gelen

isteğin adresine cevap yollamaktadır. Böylece sunucu; tüm istemcilerin adresini, istemcilerin kendisinden almış olur. Bu şekilde karşılıklı haberleşme mimarisi istemci-sunucu mimarisi olarak adlandırılmaktadır. İstemci-sunucu mimari yapısı Şekil 2.9'da verilmiştir.



Şekil 2.9. İstemci-sunucu mimarisi

Şekil 2.9'da görüldüğü gibi birden fazla istemci tek bir sunucuya bağlanabilmektedir.

2.7. Haberleşme Protokolleri

2.7.1. AJAX

Cihazların üzerinde çalışan yazılımlar internet üzerinden diğer cihazlarla (sunucular gibi) iletişim halinde olmaları gerekebilir. Söz konusu iletişim, internet üzerinden birçok farklı protokol ile yapılabilir. Haberleşme var olan internet altyapısı TCP/IP (Transmission Control Protocol / Internet Protocol) standartlarını kullanır. Tüm internet ağı bu protokol ile haberleştiğinden birbirleri arasında iletişim sorunu

yaşamazlar. Tabi bu standart belli katmanlara bölünmüştür. Bu katmanlar yazılımsal ve donanımsal olarak ayrılmaktadır. Yazılımsal katmanlarda iletişim için kullanılan protokoller mevcuttur. Bu protokollerden biri de AJAX'tır (Asynchronous JavaScript and XML). AJAX, iletişim kuracağı IP'ye, XML (Extensible Markup Language) tabanlı HTTP (Hyper-Text Transfer Protocol) isteği göndermektedir [27, 28].

AJAX teknolojisi, web tarayıcılarında Javascript dili üzerinden eş zamansız (asenkron) istek yapılabilmesi için tasarlanmıştır. Bazı durumlarda web sayfaları, sayfa geçişleri yerine kendi içerisinde kısmen değişikliğe uğrar. AJAX, bu değişikliklerin sayfa geçişi olmadan yapılabilmesi için geliştirilmiştir. Javascript üzerinde callback, diğer programlama dillerinde "Handler" gibi özellikler sayesinde AJAX senkron olarak da kullanılabilmektedir [29-31].

AJAX tekniği temelde web sayfalarında tercih edilse de günümüz web servislerinde de önemli ölçüde tercih edilmeye başlanmıştır. Özellikle birçok farklı platformdan çalışan uygulamalar ve web sayfaları ortak olarak AJAX tekniğini kullanarak aynı web servisleri ile haberleşmektedir. Bu şekilde tek web servisi ile tüm platformlara sunucu desteği sağlanabilmektedir [32].

Belirtilenler üzere anlaşıldığı gibi, AJAX birçok amaç için kullanılabilmektedir. Örneğin; Estreamchat isimli web sitesi anında mesajlaşma uygulamalarını web ara yüzüne taşımış bulunmaktadır. EyeOS web sitesi, sanal olarak işletim sistemi kullanımını web ara yüzünden sağlamaktadır. Box.net web sayfası dosya depolama ve paylaşımı hizmeti sunmaktadır. AjaxWhois web sitesi, web domainlerinin hangi firma aracılığı ile satın alındığını sorgulanmaktadır. Bu hizmetlerin tümü arka planda AJAX kullanmaktadır. Her işlem, AJAX isteği formatında sunucu tarafa veri iletip alarak yapılmaktadır. Web ara yüzünde ise, gelen cevaplara göre kullanıcı ara yüzüne yansıtılmaktadır [33-36].

2.7.2. WebSocket

WebSocketler HTML'in 5inci sürümü ile gelen, henüz az kullanılmakta olan yeni bir teknolojidir. Bu teknoloji ile AJAX işlemleri yerine gerçek soketlere benzer bir yapı kullanılmaktadır. AJAX sorgusunda kullanılan XML formatı birden fazla

nitelikten oluşmaktadır. Bu niteliklere her zaman ihtiyaç duyulmaktadır ve her kullanılmayan nitelik internet üzerinden kullanılan veri paketlerinin boyutunu arttırmakta ve bu verilerin işlenme süreci için kullanılan maliyeti arttırmaktadır. Oysa soketlerde karşı taraf ile bağlantı kurulduğunda gönderilen sadece verinin kendisidir. Bu noktada ise soketlerin dezavantajı ortaya çıkmaktadır. Her veri alışverişi düzenlemelerini yazılımcı ayarlamaktadır. Çünkü veri belli bir formatta gelmemektedir [37].

Websocketler, soket yapısına çok benzer şekilde çalışırlar. Fakat websocketler, web sayfalarda kullanılacak şekilde tasarlanmışlardır. WebSocketler sadece web sayfalarında kullanılmamaktadırlar. Fakat güncel HTML standartıdırlar. Websocket'ler normal soket ile haberleşemezler. İstemci soketinden gelecek olan bağlantıyı bekleyen sunucu, websocket'ten gelen bağlantıları karşılayamaz. WebSocket protokolünü destekleyen bir istemci-sunucu (client-server) mimarisi zorunludur. Bu durum, TCP soketi destekleyen bir sunucunun yeniden yazılmasına yada TCP soketlerine verdiği desteğe paralel websocketlere de ayrı hizmet vermek zorunda bırakmaktadır. TCP soketlerinin tabanı çok eskiye dayanmaktadır. Fakat websocketler HTML5 ile gelen yeni bir teknoloji olduğundan henüz sunucuların birçoğu bunu desteklemektedir. Bu da büyük bir iş yükünü sunucu maliyetine yükleyecektir. Fakat soket altyapısını web sayfalarında kullanabilme ihtiyacı çok artmıştır ki; HTML 5 teknolojisi ile böyle bir altyapı oluşturulmuştur ve şimdiden kullanılmaya başlanmıştır [38-42].

2.7.3. Java Applet'ler

JVM işletim sisteminden bağımsız olarak yazılımların yürütülmesini sağlamaktadır. JVM üzerine yazılan bir uygulama, JVM'in anlayacağı bir biçime çevrilir. Uygulama işletim sisteminde yürütülmek istediğinde, aslında Java platformu bu uygulamayı yürütür. Fakat arka planda aslında işletim sistemi Java uygulamasını yürütmektedir. Günümüzde birçok işletim sistemini kendine özgü avantajları olması ve piyasada daha çok pazar edinmeleri sonucunda Java daha çok kullanılmaya başlanmıştır [43-45].

Java altyapısını kullanarak Web tabanlı uygulamalar da geliştirilmeye imkan tanınmıştır. İşletim sistemine kurulduğu gibi, var olan tarayıcılara da Java eklentisi kurulabilmektedir. Bu eklenti üzerinde tarayıcılardan bağımsız şekilde, Java'da çalışacak uygulama yürütülmektedir. Bu şekilde web sayfaları sadece web teknolojilerini değil, aynı zamanda birçok işletim sistemi ana dildeki API'sine de erişebilmesi sağlanmaktadır [46, 47]. Firefox üzerinde çalışan örnek Java Applet uygulaması Şekil 2.10'da verilmiştir [48].

Şekil 2.10. Firefox üzerinde çalışan Java Applet örneği ekran görüntüsü

2.8. Java Koleksiyon Veri Yapısı

Her yazılım yürütüldüğünde birçok veri üzerinde işlem yapar. Bu veriler, veri tabanlarından, web servislerinden, dosyalardan, soketlerden ya da herhangi bir veri kaynağından okunabilir. Fakat bu veriler bir yere kaydedilmediği sürece program akışı boyunca işlenir ya da geçici olarak saklanırlar. Bu veriler program yürütülürken, işletim sisteminin tahsis ettiği bellek alanında saklanırlar ve yürütülen program izin vermedikçe, dışardan hiçbir yazılım bu verilere erişemez. Bu verilerin yönetiminden uygulamanın kendisi sorumludur.

Yazılımlar kendine tahsis edilen bu alanda sakladığı verileri doğru şekilde yönetebilmek için, bu verileri kendi anlayacağı bir biçimde (formatta) tutmalıdır. Yazılımların temel işlevi aslında veri işlemek olduğundan, bu verilerin tutulacağı

formatlar her programın mimari tasarımında belirlenmektedir. Bu kadar çok sık kullanıldıklarından, piyasada genel olarak kabul görmüş mimari tasarımlar mevcuttur. Bu tasarımların diğer programcılar tarafından kullanılabilmesi için birçok programlama dili içerisinde standart olarak kullanıma hazır halde (kütüphane olarak) sunulmaktadır [49].

Her veri yapısının avantajları ve dezavantajları vardır. Bazı veri yapıları verileri belli bir sıra içerisinde tutar, bazıları veriler üzerinde arama yapmayı veya döngü içerisinde kullanımını kolaylaştırır, bazı verilere belli bir anahtar veri üzerinden hızlı erişimi sağlar. Bu gibi birçok özellik mevcuttur. Fakat yazılımın doğası gereği, fazla özellik sunan veri yapıları daha fazla bellek tüketir ya da daha ağır performans ile verileri işler. Bu sebeple programcı, verileri gruplandırarak her veri grubuna uygun veri yapısını kullanmalıdır. Aksi durumda uygulamanın performansı olumsuz yönde etkilenecektir [50-52].

Java programlama dili üzerinde Koleksiyon (Collection) kavramı üzerinde veri yapıları ortak bir soyut kavramda toplanmıştır. Koleksiyon temelde bir arayüzdür ve piyasada en çok kullanılanı List ara yüzüdür. List ara yüzünü implemente ederek oluşturulmuş bir sınıftan oluşturulan nesne kullanılarak, liste oluşturulup kullanılmaya başlanabilir. List arayüzünün iki farklı implementasyonu vardır. Bunlar LinkedList ve ArrayList'tir. Her iki implementasyonun kendine özgü özellikleri vardır. Örneğin; ArrayList kütüphanesi verilere rastgele erişim konusunda daha avantajlıyken, LinkedList veriler arasına veri yerleştirme konusunda daha başarılıdır [49-54].

Java programlama dilinde standart olarak sunulan HashMap veri yapısı mevcuttur. Bu veri yapısı, verileri bir anahtar ile saklamaktadır. Verilere tekrardan erişim için bu anahtarlar kullanılır. Aynı zamanda liste baştan sona dolaşılarak da elemanlara erişilebilir, fakat daha hızlı erişim için baştan sona dönmek çoğu zaman verimsizdir [55, 56].

2.9. Piyasada Bulunan Hibrit Uygulama Eklentileri

Hibrit uygulama altyapısını sunan birçok iskelet vardır. Bu iskeletler arasında açık kaynaklı olması ile geliştiricilerin dikkatini çeken Cordova projesi, işletim sisteminin ana diline veri transferi sağlayan bir eklenti altyapısı sunmaktadır. Cordova projesi için açık kaynaklı birçok eklenti mevcuttur. Bu eklentiler sayesinde, web standartlarının kısıtlamalarından kurtulup, ana dilin özellikleri kullanılmaya çalışılmaktadır [57-59].

2.9.1. Cordova Network Information Plugin

Cordova Network Information Plugin, Cordova için yazılmış internet ağı ile ilgili ihtiyaçları karşılayan bir eklentidir. Eklenti eski "Network Information API" ara yüzünün implementasyonunu sunmaktadır. Eklenti temel olarak işletim sisteminin kullandığı internet ağına çıkış cihazının tipini (Wi-Fi, 3G, 2G…) belirlemekte ve bunu Javascript tarafına sunmaktadır. Hibrit uygulamalar genelde internet ağı tipine göre veri kullanımını gerçekleştirmektedir. Zira her ağın hızına ve kotasına göre son kullanıcıyı etkilemeyecek işlemler yapılmaya gayret edilmektedir. Eklentinin teknik altyapısı incelendiğinde, değerlerin JSON standartlarındaki veri tiplerinin olduğu görülmektedir. Çünkü veri transferini sağlayan eklenti altyapısı Javascript'in kısıtlamaları sebebiyle JSON veri tipleri dışında bir veri kullanımını destekleyememektedir. Dokümantasyonda da görüldüğü gibi tüm platformlardan dönen ağ bilgisi diğer tüm platformlarda dönen ağ bilgisi ile aynı sayıda değil. Bunun sebebi her platformun ana programlama dilinde sunduğu özelliklerin aynı olmamasıdır [60-62].

2.9.2. Cordova Plugin Console

Cordova-plugin-console eklentisi Cordova için Android, iOS ve Windows platformları için implemente edilmiş bir pakettir. Apache Vakfı tarafından dağıtılan paket, konsol çıktısını kullanılmasına olanak tanır [63].

Uygulamanın kaynak kodları incelendiğinde Android iplmementasyonunun ana dilde kodları olmadığı görülmektedir [64]. Bu durum; Android'in webview'inin Javascript'te konsol çıktısını yakaladığını göstermektedir. Her platform kendi başına farklı kararlar aldığından, hibrit uygulamalar için ana dilden sağlanacak API'lerin birçok durumu göz önünde bulundurması gerektiği görülmektedir. Ubuntu üzerindeki kodlar incelendiğinde; header dosyası tanımında log atacak fonksiyonun imzası ve C++ kodunda bu fonksiyonun implementasyonu yapıldığı görülmektedir [65].

2.9.3. Cordova Plugin Globalization

Cordova-plugin-globalization eklentisi BlackBerry, Firefox OS içinde olmak üzere birçok platforma implemente edilmiş durumdadır. Apache Vakfı tarafından paketlenen eklenti, uygulamaların farklı dillerde dağıtılmasını kolaylaştırmaktadır[66].

Uygulama implementasyonları arasında tarayıcı implementasyonu görülmektedir. Tarayıcı implementasyonu Cordova'nın güncel sürümlerinde gelen yeni bir özelliktir. Cordova tarayıcı implementasyonu paketlendiğinde, gerçek bir tarayıcıda çalıştırılmak üzere bir dizin yaratmaktadır. Bu dizin herhangi bir web tarayıcısı ile açıldığında, gerçek bir tarayıcı ile uygulamanın açılmasını sağlamaktadır. Bu şekilde, yazılımcı ürününü tarayıcı üzerinde test edebilmesini sağlamaktadır. Bunun yapılması sebebi her platformlarda testinin özellikle zaman açısından maliyetli olmasıdır. Zira neredeyse her platform kendi geliştirme araçlarını sunmaktadır. Tarayıcı implementasyonu üzerinde uygulama, diğer platformları simüle edecek Javascript kodları kullanmaktadır. İlgili eklentinin bir özelliği simüle edecek kodların nasıl çalışacağına implementasyonu yazan yazılımcı karar vermektedir. Bu simüle eden kodlar sadece Javascript ile yapılabilmektedir. Zira ana dile hiçbir şekilde erişilmemekte ve sadece tarayıcı üzerinde çalışmaktadır.

2.10. Dağıtık sistemlerde TCP kullanımı

Günümüzde yazılım teknolojileri dağıtık sunucu mimarileri sıkça kullanmaya başlamıştır. Bunun temel sebeplerinden biri; günümüz yazılımlarının/altyapıların

dağıtık mimarilere izin vermesidir. Örneğin; bilgisayar kullanımının artması ile günümüzde karşımıza çıkan "büyük veri" ayıklama işlemleri doğal olarak dağıtık sunucularda paralel yapılması beklenmektedir [67, 68].

'A Smart TCP Socket for Distributed Computing' başlıklı çalışmada [69]; dağıtık sistemler için bir TCP soket arayüzü tasarlanmıştır. Normal koşullarda bir programcı, dağıtık olan sunucular arasından TCP ile haberleşerek, bu sunucuların yoğunluk analizini yapıp, hangi sunucuların uygun olduğunu bulup, o sunucular üzerinden kısmı olarak işlem gerçekleştirmesi gerekmektedir. 'A Smart TCP Socket for Distributed Computing' başlıklı çalışmada geliştirilen arayüz aracılığı ile; programcı istediği miktarı TCP soket fonksiyonlarına parametre olarak geçmektedir. Geliştirilen arayüz, tüm sunucuların uygunluğunu TCP aracılığı ile toplamaktadır. Uygun olan sunucuları; yine geliştirilen bir karar mekanizması (algoritma) üzerinden geçirerek seçmektedir. Seçilen uygun sunucuları, programcının kullanması için aracılık sağlamış olmaktadır. Sunucular üzerinde yoğunluğu tespit etmek için Cisco Trust Agent ve Windows API'lerinden yararlanmaktadır. Sunucular ile haberleşmede soket arayüzleri kendi içerisinde kendi dilini kullanmaktadır. Mesajlar karşıya gönderilirken belli formatlarda gönderilmektedirler [69].

3. HİBRİT UYGULAMALARDA SOKET VE MQTT KULLANIMI

3.1. Hibrit Uygulamalarda Ana Dil kullanımı

Hibrit uygulamaların socket altyapısından yararlanmaları özellikle internet ağının daha az kullanımı ve soket kullanan sunucular ile haberleşebilmesi için çok önemlidir. Günümüz ihtiyaçları web platformlarında kullanılmak üzere Websocket altyapısını ortaya çıkarmış olsa da, ana dildeki özelliklerden de yararlanabilmelidir. Her ne kadar hibrit uygulamalar web teknolojileri kullanılarak geliştirilse de, son kullanıcı tarafından ana dilde çalışan uygulama olarak değerlendirilmektedir. Bu sebeple ana dilin özelliklerden yararlanmak, hem son kullanıcı hem de geliştiriciler için çoğu zaman büyük avantaj sağlamaktadır [70].

3.2. Geliştirme Ortamı

Eklentinin kullanılacağı ortamda birçok farklı teknoloji yer almaktadır. Benzer şekilde; eklentinin geliştirilmesi durumunda (yeni fonksiyonlar eklenmesi gibi) benzer birçok teknolojinin geliştirme platformunda yüklü olması gerekmektedir.

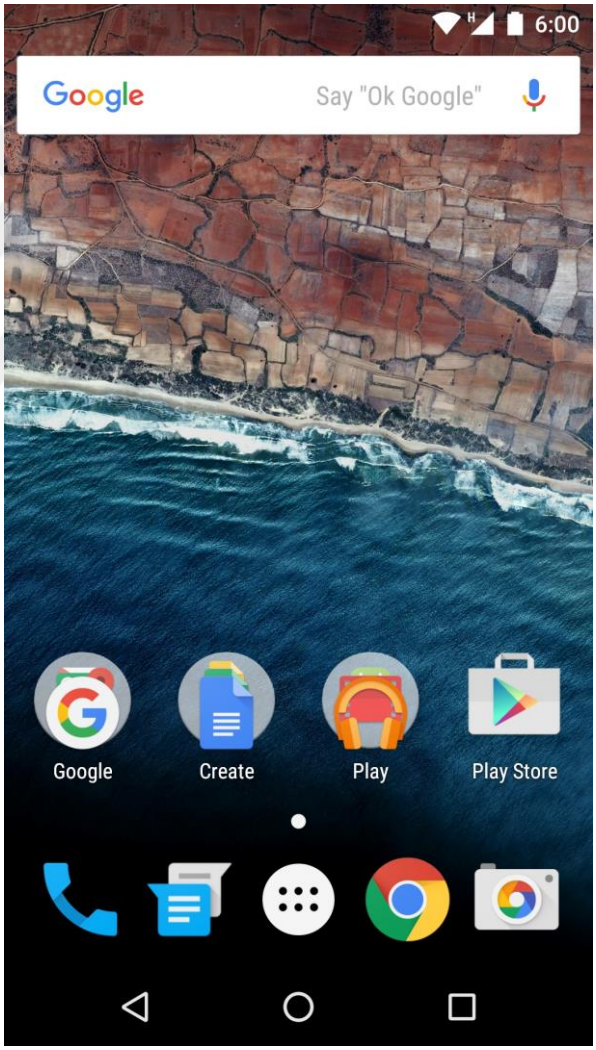
3.2.1. JDK

Bu çalışmada kullanılan Android SDK, MQTT sunucusu, MQTT istemcisi ve Eclipse geliştirme uygulamaları, açık kaynaklı Java üzerinde yürütülmektedir. Eclipse üzerindeki derleme işlemleri için JDK'ya ihtiyaç duyulmaktadır. Bu sebeple Java platformunun geliştirme sürümü olan JDK sistemde kurulu olması gerekmektedir. Geliştirme platformu üzerinde farklı Java sürümleri çalıştırılabilir. Bu şekilde kurulumsuz olarak JDK çalıştırılabilir duruma getirilmiştir. Bu sebeple; daha önce kurulu herhangi bir Windows makinenin "Program Files" dizini içerisinden

JDK dizini kopyalanarak çalışma ortamı için ayrılan dizine kopyalanmalıdır. Bu çalışmada JDK'nın 1.8.0_66 64 bit sürümü kullanılmıştır.

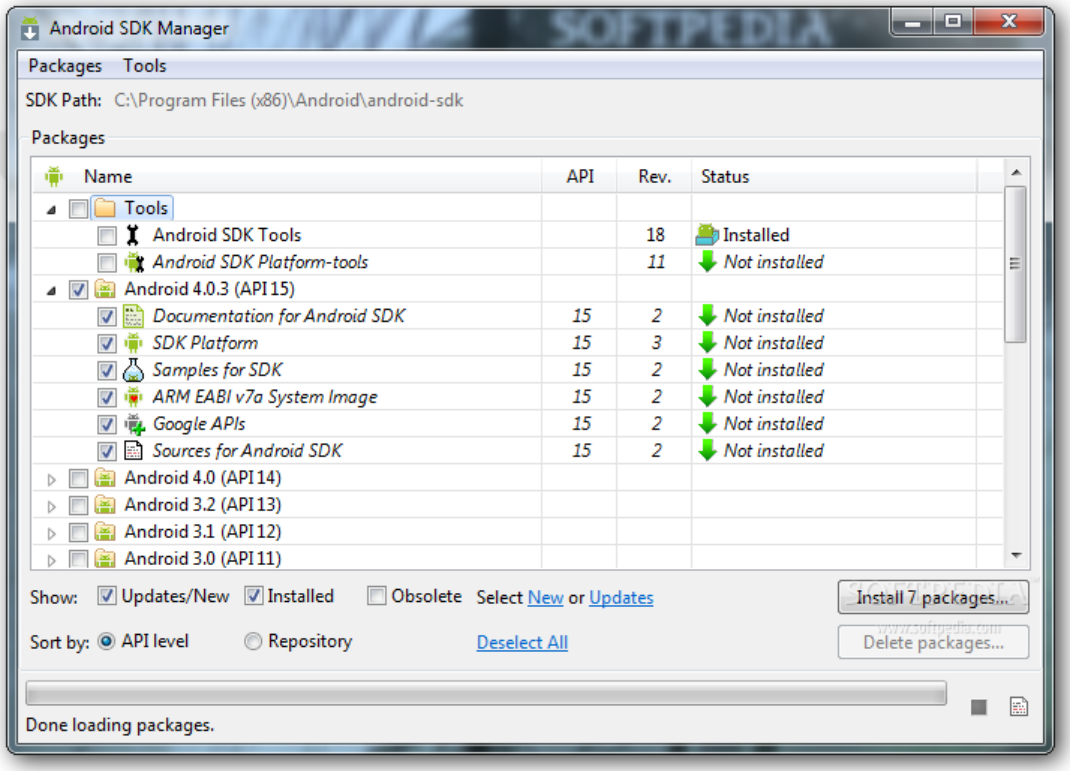
3.2.2. Android SDK

Android, Google tarafından geliştirilen Linux çekirdeği üzerine inşa edilmiş mobil (tablet ve birçok taşınabilir cihaz) işletim sistemidir. Büyük bir kısmı açık kaynaklı olsa da, bazı kısımlar Google tarafından kapalı tutulmaktadır. Google, Android üzerinde çalıştırılan uygulamalar üzerinde aldığı reklamlar ile gelir elde etmektedir. Android 6'ıncı sürümüne ait ekran görüntüsü Şekil 3.1'de verilmiştir.



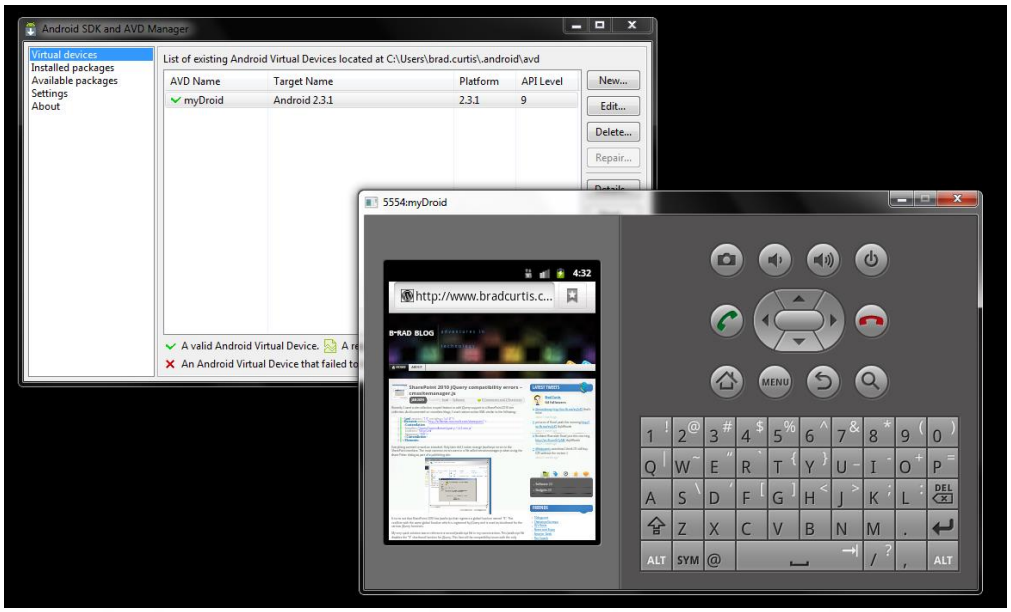
Şekil 3.1. Android 6 başlangıç ekranı

Android üzerinde yürütülmek üzere yazılan uygulamalar için geliştirme ortamı olarak Android SDK kullanılmaktadır. Android üzerine birçok yazılım dilinde uygulamalar geliştirilebilmektedir. Fakat pazarda en çok Java dili ile yazılan uygulamalar bulunmaktadır. Android geliştirme ortamı tüm işletim sistemleri üzerinde çalışabilmektedir. Android SDK içerisinde her Android sürümü için ayrı ayrı geliştirme paketleri mevcuttur. Her sürüm altında, o sürümün desteklediği çeşitli sanal makineler, dökümantasyon, örnek kodlar gibi kurulum seçenekleri Şekil 3.2'de görüldüğü gibi mevcuttur.



Şekil 3.2. Android SDK kurulum seçenekleri

Android SDK kendi içerisinde geliştirme ortamı için gerekli tüm platformu bulundurmaktadır. Emülatörlerin çalıştırılması ve yönetimi için sanal makine yöneticisi Şekil 3.3'te olduğu gibi mevcuttur.



Şekil 3.3. Android sanal makine yöneticisi

Android simülatörleri ve derleme gibi geliştirme aşamaları için Android SDK gerekmektedir. Windows geliştirme platformu üzerinden SDK'yı taşınabilir şekilde kullanabilmek için http://developer.android.com/sdk/index.html#Other sayfası üzerinden "android-sdk_r24.4.1-windows.zip" paketi indirilmelidir. Paket indirildikten sonra, zip içerisinden çıkarılmalı ve çalışma ortamı için yaratılan dizine taşınmalıdır. Android SDK, Java'ya ihtiyaç duymaktadır. Java işletim sistemine kurulmadığı için, SDK direk olarak Java'nın bulunduğu dizini algılayamayacaktır. Bu sebeple işletim sisteminde global olarak tüm kullanıcılar için yada sadece kullanıcı seviyesinde yeni path tanımlaması yapılması gerekmektedir. Path listesine yeni değişken olarak "JAVA_HOME" ve bu değişkene, çalışma dizini altına kopyalanan JDK dizininin değeri verilmelidir.

3.2.3. Eclipse ve Android Entegrasyonu

Geliştirme ortamında IDE olarak Eclipse kullanılmıştır. Bu çalışmada Eclipse 4.5.1 Java Edition 64 bit kullanılmıştır. Eclipse'in çalışma dizinine indirilmesi yeterli olacaktır. Eclipse varsayılan olarak kurulumsuz çalışmaktadır. Geliştirme ortamında JDK işletim sistemi seviyesinde kurulu olmadığından, Eclipse'e JDK dizinini tanıtmak gerekmektedir. Bunun için Eclipse dizini altında bulunan "eclipse.ini"

dosyası içerisine “-vm” satırı ve hemen altındaki satıra JDK’nın bulunduğu dizinin altındaki “bin” klasörünün adresi yazılmalıdır. Yazılması gereken bu parametreler, “–vmargs” satırından hemen önce bulunmalıdır.

Eclipse üzerinde Android geliştirme ortamının entegre olabilmesi için birçok eklenti mevcuttur. Google’ın resmi olarak geliştirdiği Android Development Tools (ADT) eklentisine destek resmi olarak kesilmiştir. Zira Google resmi olarak geliştirme ortamını, Eclipse’den Intellij IDEA taşımıştır. Açık kaynağa destek veren topluluklar ADT’yi kopyalayarak geliştirmelere “Andmore” ismini kullanarak devam etmektedirler. Bu çalışmada Andmore eklentisi Eclipse’e kurulmuştur.

3.2.4. Node JS, NPM ve Cordova

Cordova, hibrit uygulama geliştirmek için geliştirme ortamı sunmaktadır. Cordova altyapısını kullanabilmek için, Javascript ile uygulama geliştirmeyi sağlayan NodeJS sisteme kurulmalıdır. Bu çalışmada NodeJS’in 4.2.3 LTS 64 bit sürümü kullanılmıştır. Taşınabilir sürümü için https://nodejs.org/en/download/ sayfasından “Windows Binary (.exe) 64 bit” indirilmesi ve çalışma dizinine taşınması yeterli olacaktır.

NodeJS üzerinde Cordova gibi modül kurulumu yapabilmek için paket yöneticisine ihtiyaç vardır. Paket yöneticisi NPM, http://nodejs.org/dist/npm/ sayfasından 1.4.9 sürümü kurulmuştur. NPM indirildiğinde çalışma dizinine taşınmalıdır. NPM, NodeJS ile çalışmaktadır Dolayısı ile NPM, NodeJS’in bulunduğu dizini bilmesi gerekmektedir. Bu sebeple sistemde zaten varolan “PATH” değişkenine NodeJS’in bulunduğu dizin verilmelidir.

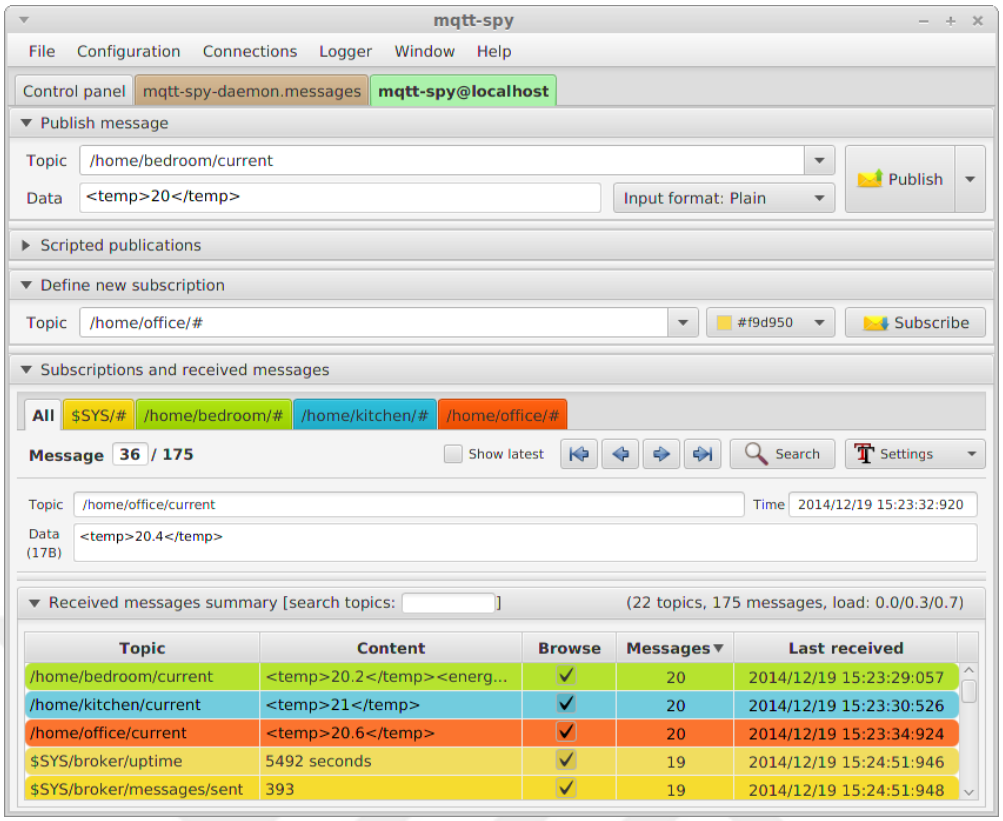
Cordova modülün kurulması için komut satırından NPM’in bulunduğu satıra gidip, “npm.cmd install cordova” ile NPM’in Cordova modülünü kurması sağlanmalıdır. Komut satırından kurulan varsayılan sürüm yerine istenen herhangi bir sürüm kurulabilir. Bu çalışmada kurulan Cordova sürümü 5.4.1’dir.

3.2.5. Moquette ve MQTT Spy

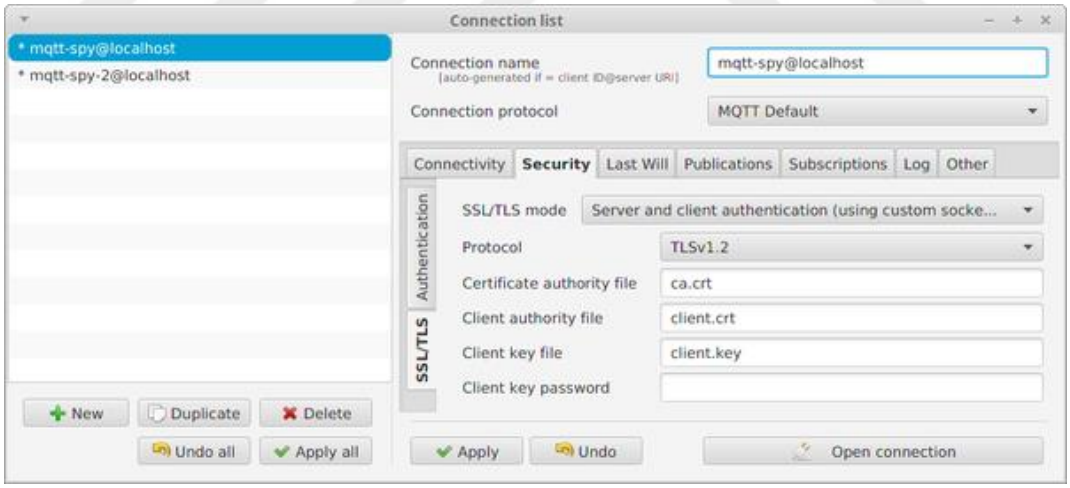
Cihazlar arası haberleşme ve mobil cihazlardan toplu bilgi toplama amaçlı geliştirilen MQTT protokolü için Broker olarak adlandırılan sunuculara ihtiyaç duyulmaktadır. Bu sunucular istemci taraflardan gelen giden bilgilere aracılık etmektedirler. MQTT protokolünü destekleyen birçok broker bulunmaktadır. Bu çalışmada MQTT haberleşmesi için sunucu olarak, açık kaynaklı Moquette tercih edilmiştir. Moquette Java platformu üzerinde çalıştırıldığından birçok işletim sisteminde çalışmaktadır. Moquette, basit konfigürasyona sahip ve sistem kaynaklarını diğer sunuculara nazaran düşük seviyede tüketmekte olduğundan tercih edilmektedir. Moquette aynı zamanda uygulamaların içinde gömülü olarakta yürütülebilmektedir.

Moquette Java platformu üzerinde yürütüldüğünden çalışma ortamımızda bulunan JDK dizinini "JAVA_HOME" değerini path'e eklememiz gerekmektedir. Daha önceden JDK kurulumunda bu işlemi yaptığımızdan, tekrarlanmasına gerek yoktur.

MQTT sunucusu üzerinde birkaç istemci simüle edebilmek gerekebilir. Bu durumlar için açık kaynaklı MQTT Spy uygulaması kullanılmaktadır. MQTT Spy uygulaması Java üzerinde yürütülmektedir. Bu yüzden; çalışma dizinindeki JDK içerisindeki bin klasöründeki "java.exe" ile birlikte yürütülmesi gerekmektedir. MQTT Spy, mesaj takibi yapmayı sağlayan penceresi ve bağlantıları oluşturmak için sunduğu konfigürasyon penceresi sırası ile Şekil 3.4 ve Şekil 3.5'te verilmiştir.



Şekil 3.4. MQTT Spy mesaj takibi penceresi



Şekil 3.5. MQTT Spy bağlantı konfigürasyon sayfası

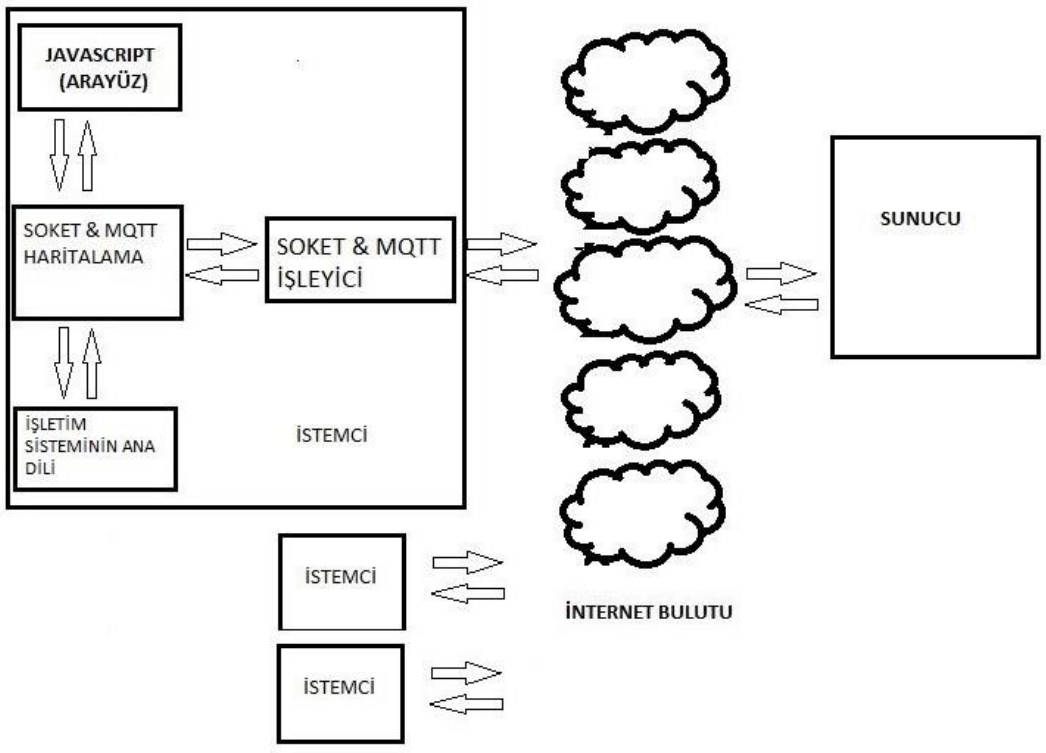
3.3. Eklenti Altyapısı Mimarisi

Eklenti kendi içerisinde soket ve MQTT bağlantılarını yönetmekte ve yazılım geliştiriciye bu bağlantılara erişimi sağlamaktadır. Temel olarak; Javascript ve ana

dil arasında JSON formatında yapılmaktadır. Bu sebeple mimari JSON veri tipleri üzerinden yönetilmektedir. Javascript tarafında sunulan aynı API ana dilde de sunulduğu için ana dilde de JSON formatında API'ler hazırlanmıştır.

3.3.1. Genel Mimari

Hibrit uygulamaların ana dil üzerinde TCP soketi veya MQTT Broker'ı kullanabilmeleri için, işletim sisteminin soket ve MQTT altyapısını kullanan bir ara yüz tasarlanmıştır. Bu arayüzde Javascript tarafından çağrılacak fonksiyonlar bulunmaktadır. Javascript tarafında kullanılacak bu fonksiyonlar, hibrit uygulama geliştirme ortamındaki eklenti altyapısını kullanarak, ana dil tarafındaki fonksiyonlar ile haberleşmektedirler [71]. Bu tezde geliştirilen sistemin genel mimari yapısı Şekil 3.6'de verilmiştir.



Şekil 3.6. Geliştirilen sistemin genel mimari yapısı

Mimaride görüldüğü gibi Javascript tarafından ana dilin soket ve MQTT altyapısına erişebilecek bir ara yüz tasarlanmıştır. Bu ara yüz, ana dil tarafına

erişirken, gerçek soket veya MQTT birim bileşenine direk olarak erişemeyeceğinden, bir tekil referans ile erişebilecektir. Bu erişimi yönetebilmek için ayrı bir haritalama görevi yapan bir mekanizma yaratılmıştır. Bu mekanizma aynı zamanda, sadece veya kısmen ana dilde yazılan ve soket ya da Broker birimlerine erişmek isteyen kısımlara da destek verecek şekilde tasarlanmıştır. Bu sebeple ana dil tarafı içinde erişim arayüzü tasarlanmıştır. Bu API Şekil 3.6'da "Soket ve MQTT işleyici" olarak gösterilmektedir. Hem Javascript, hem de ana dil tarafından yönetilecek olan protokol birimleri, birbirleri arasında çakışma olmayacak şekilde tasarlanmıştır. TCP soket işlemleri programlama dilinden bağımsız olarak tasarlanmıştır. Kendi içlerinde gönderilecek olan farklı veri tipleri belli kalıplar çerçevesinde tutulmuştur. Zira bu kalıplara uyulmazsa ve programlama dilinin standart paketlerinde kullanılan fonksiyonlardan yararlanılırsa, sunucu tarafı ile tüm platformlardan iletişime geçilemez. Bu sebeple; bu çalışmada veri transferi için kullanılan fonksiyonlar dışında, daha farklı veri tipler yollanmak istendiğinde (bu bir nesne olabilir), sunucu ve istemci tarafında iletişimini belirleyen fonksiyonlar da tasarlanmalıdır.

3.3.2. Hata Durumları Yönetimi

Ağ üzerinde veri transferi esnasında hatalar veya gecikmeler meydana gelebilir. Aynı zamanda cihaz üzerinde soket kullanımını kısmen ya da tam olarak kısıtlayıcı güvenlik önlemleri alınmış olabilir. Bu sebeple eklenti tasarımında hata tespit mekanizması bulunmalıdır. Bu mekanizma olabilecek herhangi bir hataya karşı hem Javascript tarafını hem de ana dil tarafı uygun şekilde uyarabilmelidir [72, 73]. Hata tespit mekanizması çalışır durumda olan uygulamayı hiçbir şekilde durdurmadan, programlama dili üzerindeki hata modellerini kullanarak, gerekli noktalarda hatalar fırlatılmalıdır. Javascript'in doğası gereği, hata modelleri ana dilden Javascript tarafına aktarılamazlar. Bu sebeple ana dilden Javascript tarafına ek parametreler dönerek, uygulamaya hata olduğunu bildirmeli ve hatanın detayları yine aynı şekilde ek parametrelerle aktarılmalıdır.

3.3.3. MQTT ve Soket Bağlantılarının Yönetimi

Uygulama çalışır durumdayken kullanılacak soket sayısı veya MQTT Broker sayısı programcının isteğine ve programın akışına bağlı olarak değişkenlik gösterebilir. Soket nesneleri ve MQTT Broker'ları birbirinden bağımsız çalışmalıdırlar. Aynı zamanda hem Javascript tarafından hem de işletim sisteminin ana dilinden erişilebilecek durumda olmalıdırlar. Programcı geliştirilecek soket altyapısını kullanırken direk olarak bu internet erişim düğümlerine erişmek yerine, sunulan arayüz aracılığı ile bu birimler üzerinde işlem yapacak ve sadece istisnai durumlar için bu düğümleri direk olarak çağırıp üzerinde işlem yapabilecek şekilde mimari tasarlanmıştır. Javascript tarafında ise; bu birimlere hiçbir şekilde direk olarak erişim teknik olarak mümkün olamayacağından, Javascript tarafında sadece arayüz kullanımı mümkündür. Burada belirtilen arayüz, kendi içerisinde tüm düğümlere kolayca erişilebilir şekilde tasarlanmıştır. Düğümler tekil bir anahtar ile erişilebilir şekilde tasarlanmıştır. Her düğüm tekil bir anahtar ile temsil edilip saklanmıştır. Herhangi bir düğüm üzerinde işlem yapılmak istendiğinde, bu tekil anahtar üzerinden işlem yapılmalıdır. Yeni bir düğüm oluşturulduğunda ise oluşturulan düğümün tekil anahtarı, düğümü oluşturan program bloğuna döndürülmelidir. Bu şekilde yazılım geliştirici, programın akışına göre aynı düğüme tekrar erişebilir durumda olacaktır. Düğüm anahtarları kaybolduğu durumda, anahtarı kaybolan düğümlere erişim ancak ve ancak tüm açık düğümler döngü içerisinde dolaşılarak ulaşılabilir durumda olacaklardır. Fakat bu performans açısından olumsuz etki yaratacaktır. Bu yüzden programcının düğümlere sadece tekil anahtarlar üzerinden erişmesi daha uygun olacaktır. Fakat istisnai durumlar için bu düğümler üzerinde belli kriterleri sağlayan düğümleri çağırabilecek fonksiyonlar hazırlanabilir. Bu fonksiyonlar ek olarak tüm soketler üzerinde kapama gibi ortak işlevler gerçekleştirebilirler.

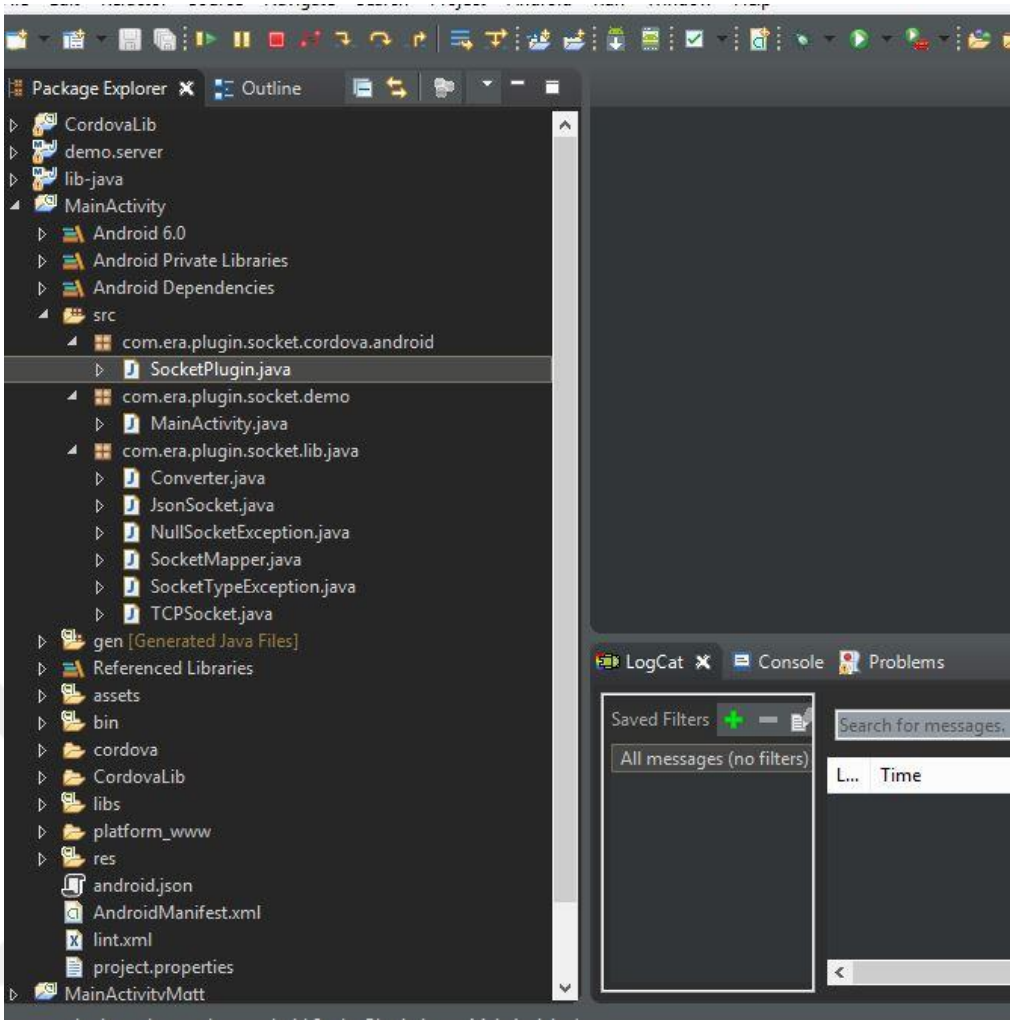
3.3.4. Soketler Arası Veri Transferi

Programın ihtiyaçları gereği, sunucu ve istemci arasında veri paylaşımı birden çok farklı veri tipi ile gerçekleşebilmektedir. Soket üzerinden gönderilecek veri tipi ve bu verinin kendisi ayrı ayrı bağlı olunan sokete transfer edilir. Bu veriler byte

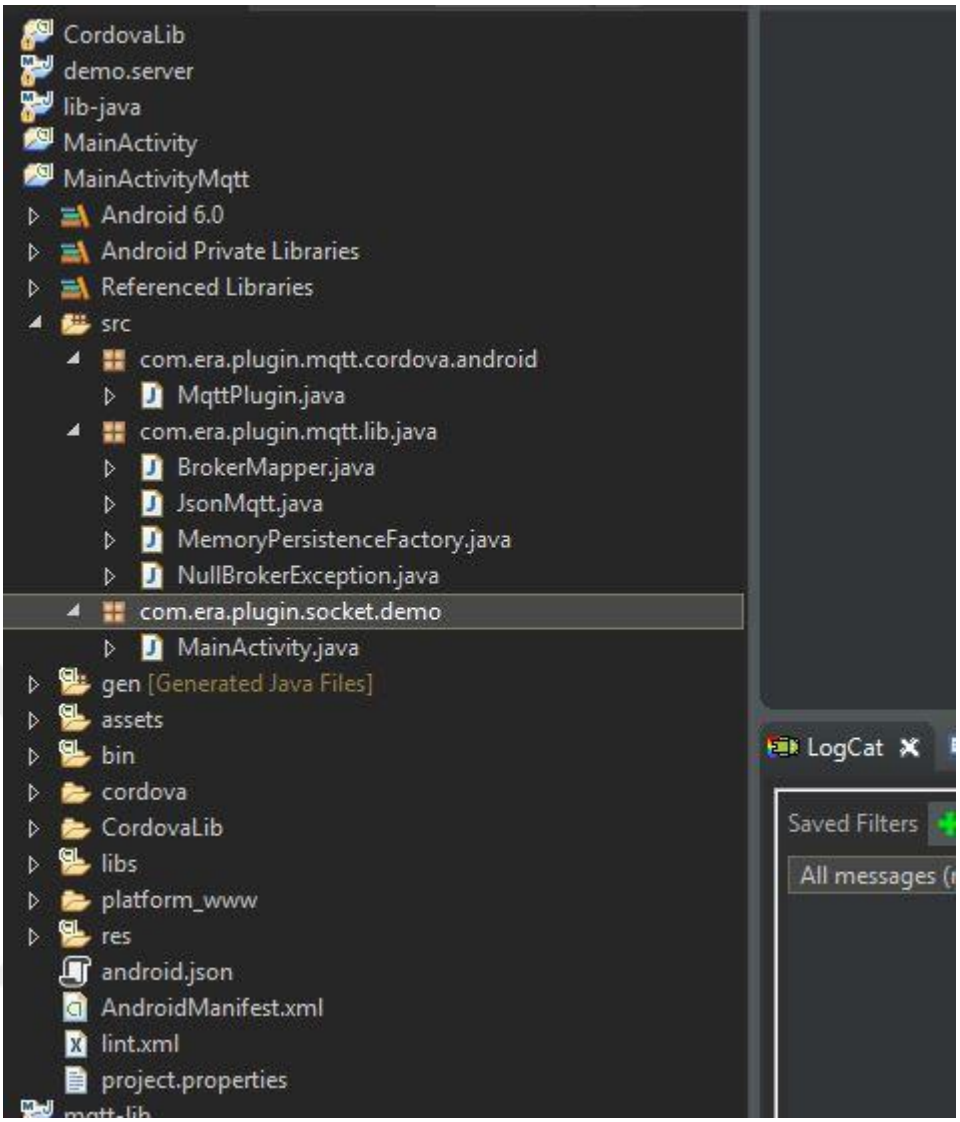
bazında gönderildiği için karşı tarafın gelecek olan bilgilenin formatını önceden biliyor olması gerekir. Hem istemci hem sunucu tarafta aynı programlama dilinin aynı kütüphanesinden yararlanıldığında, veri transferlerinde kullanılan veri tipleri kütüphanenin belirlediği şekilde gerçekleştiğinden programcı verileri byte bazından düşünmeden kullanabilir. Fakat başka kütüphanelerden yararlanması durumunda tüm kod üzerinde baştan sona değişiklik gerektirebilir. Hibrit uygulamalar platformdan bağımsız olduğundan bu tarz bir yapı tercih edilmemelidir. Zira farklı platformların ana dilindeki soket yapılarından, farklı dilde yazılmış bir sunucu ile iletişime geçmek gerekebilir. Bu sebeple soket üzerinden gerçekleşen veri transferleri için özel bir arayüz hazırlanmalı ve bu arayüz aracılığı ile haberleşmeler gerçekleştirilmelidir. Bu arayüzler hem sunucu hem istemci tarafta sorunsuzca çalışabilmeli ve mümkün olduğunca platformlardan bağımsız çalışacak şekilde tasarlanmalıdır.

3.4. MQTT ve Soket API Kullanımı için Entegrasyon

Bu çalışmada geliştirilen eklenti altyapısını, geliştirilecek hibrit mobil uygulamada kullanabilmek için bazı entegrasyon işlemleri gereklidir. Soket altyapısından yararlanabilmek için öncelikle oluşturulan Android projesindeki anadilde yazılan Java kodlarının kopyalanması gerekmektedir. Paket isimleri Şekil 3.7'de görüldüğü gibi olmalıdır. Benzer şekilde MQTT altyapısı için ise Şekil 3.8'teki gibi dosyalar kopyalanmalıdır.

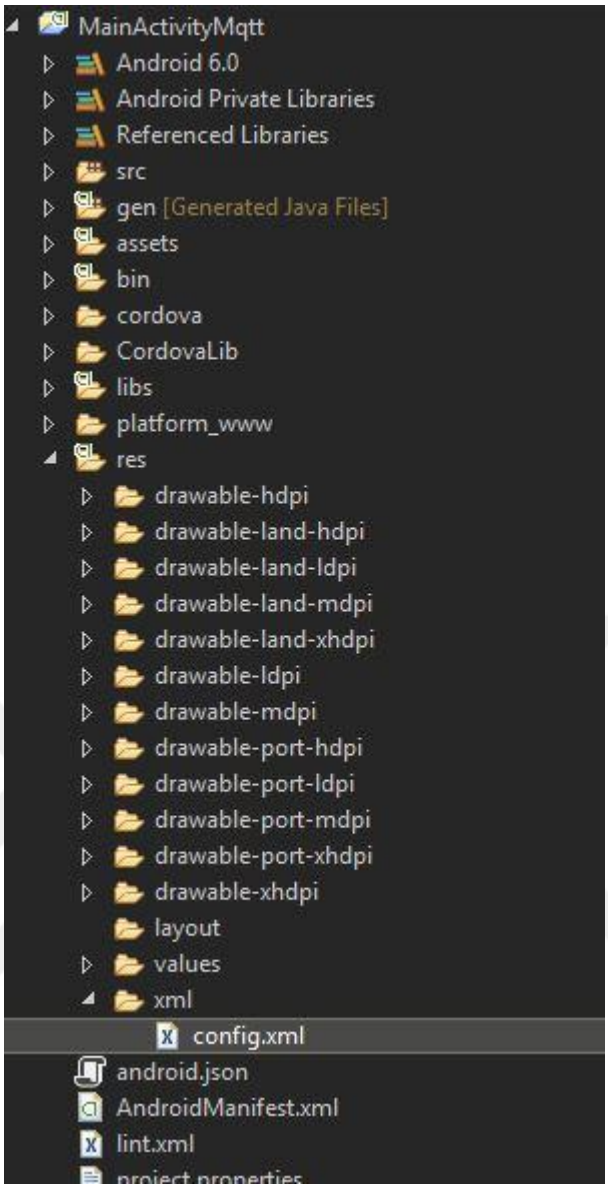


Şekil 3.7. Soket eklentisi için Android Java sınıfları dizin ağaç yapısı



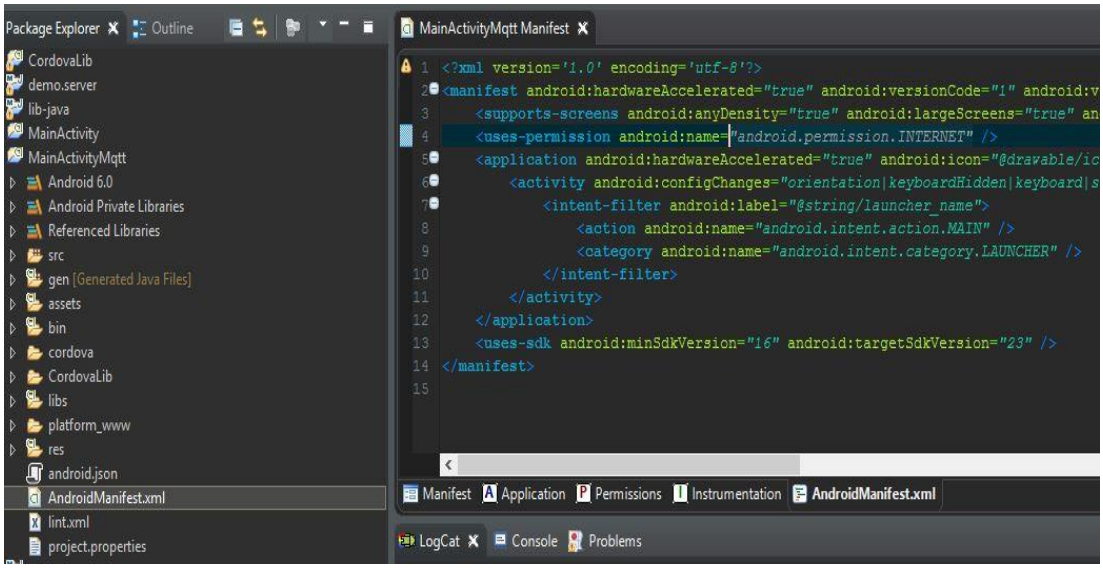
Şekil 3.8. MQTT Eklentisi için Android Java sınıfları dizin ağaç yapısı

Daha sonra Javascript tarafından Java tarafına erişebilmek için gerekli konfigürasyon ayarlarının olduğu dosyayı almak gerekmektedir. Tüm eklentiler için ortak olarak belirlenen config.xml dosyası Şekil 3.9'te olduğu gibi /res/xml dizini altına kopyalanmalıdır. Şekil 3.9'daki dizin yapısının, eklentiyi kullanacak uygulamada aynı olması şarttır.



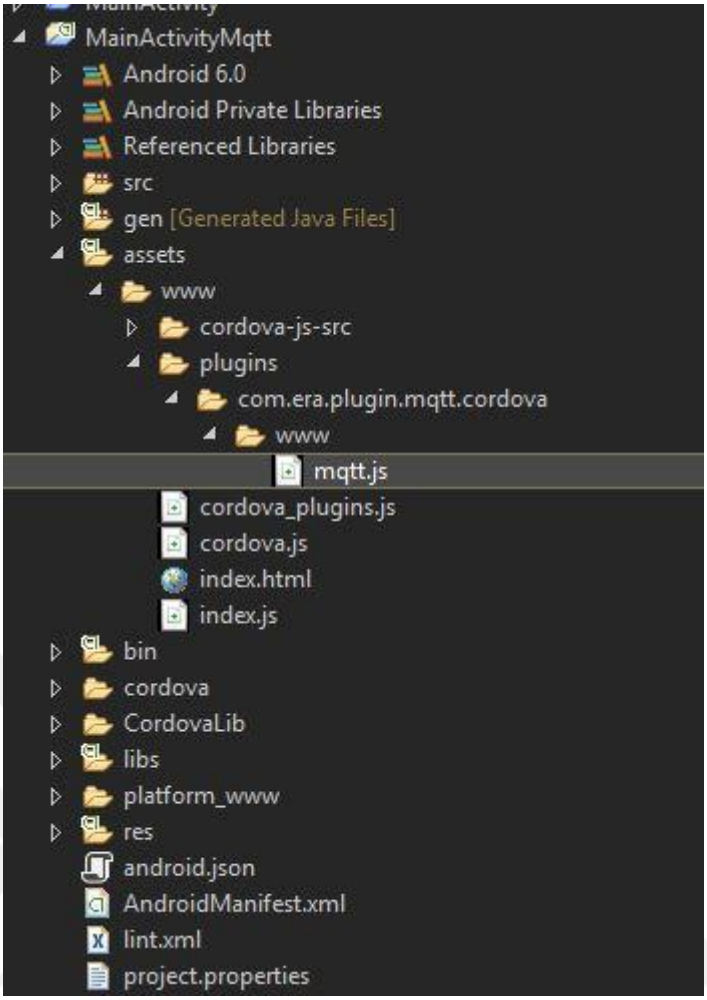
Şekil 3.9. Eklentiler için gerekli konfigürasyon dosyası

Kullanılacak Android uygulamasının internet yetkisinin olması gerekmektedir. Android üzerinde yetkiler manifest.xml dosyası içerisinde belirtilmektedir. İnternet yetkisi Şekil 3.10'teki gibi verilmiş olmalıdır.



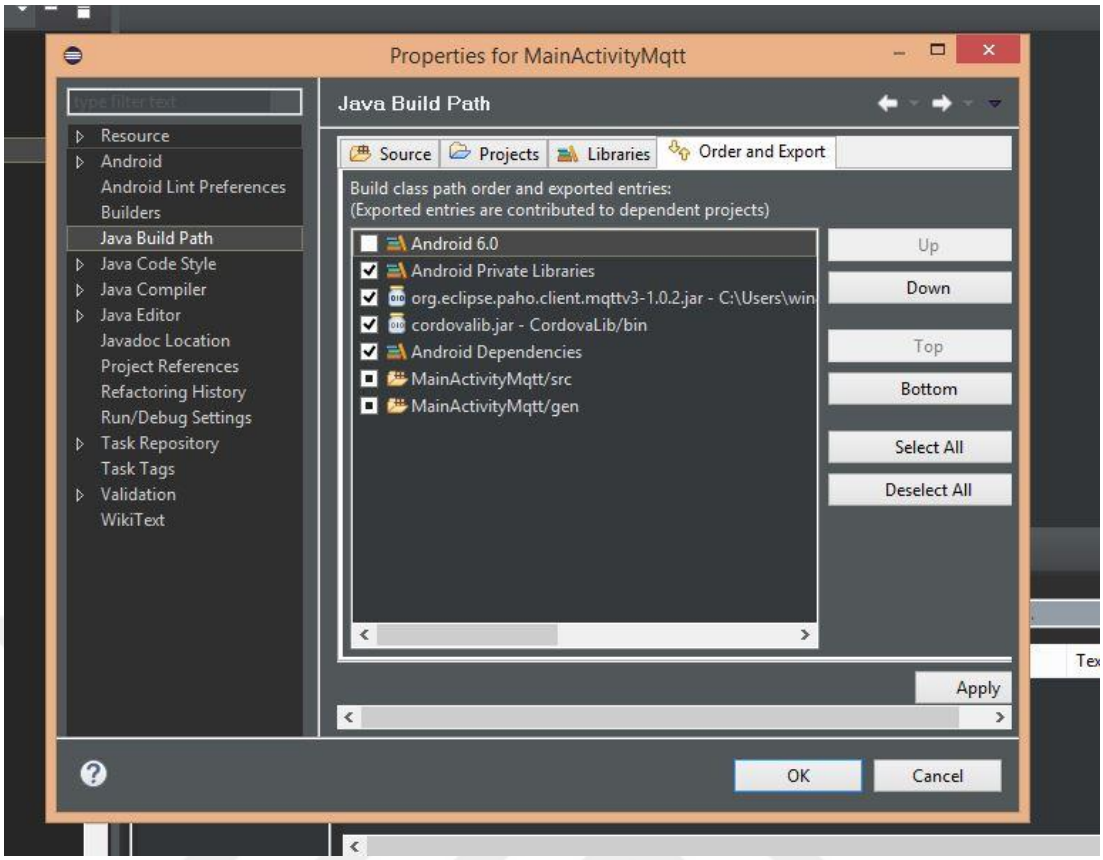
Şekil 3.10. Android üzerinde internet yetkisi için gerekli konfigürasyon dosyası

Android yetkileri tamamlandıktan sonra, Javascript tarafından anadile erişilebiliyor olunmaktadır. Bu sebeple Javascript kodlarının proje içerisine aktarılması gerekmektedir. Şekil 3.11'da görüldüğü gibi Javascript kodları /assets/www altına kopyalanmalıdır.



Şekil 3.11. Android projesi içerisindeki Javascript dosyaları

MQTT eklentisi Eclipse tarafından geliştirilen “Paho” isimli kütüphaneyi baz alarak çalışmaktadır. Bu yüzden; Şekil 3.12’deki gibi “Paho” kütüphanesini Android projesinin bağımlılıklarına eklememiz gerekmektedir. “Paho” kütüphanesi dışındaki bağımlılıklar Android Cordova projesinde varsayılan olarak gelen bağımlılıklardır.



Şekil 3.12. Android projesi üzerindeki kütüphane bağımlılıkları

Bu kodların proje içerisinde bulunması ile birlikte, Javascript tarafından soket ve MQTT erişimini sağlayan bir yapı ortaya çıkmış olacaktır.

3.5. Geliştirilen Fonksiyonlar

3.5.1. Javascript'ten Kullanılabilir Soket Eklenti Fonksiyonları

Soket eklentisi altyapısı için aşağıdaki fonksiyonlar Javascript tarafından çağrılabilir şekilde geliştirilmiştir. Bu şekilde yazılım geliştirici soket protokolüne erişebilir duruma getirilmiştir.

newSocket (successCallback, errorCallback, ipAddr, portNo) : Yeni soket açmak için kullanılır. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, karşı tarafın IP adresini ve soket numarasını almaktadır.

closeAll (successCallback, errorCallback) : Tüm açık soketleri kapatmak için kullanılır. Burada kapatılacak tüm soketler eklenti API'si kullanılarak açılmış soketlerdir.

close (successCallback, errorCallback, socketKey) : Soketi kapatmak için kullanılır. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu ve kapatılacak soketin anahtarını almaktadır.

sendInt (successCallback, errorCallback, socketKey, val) : Karşı tarafa Integer tipinde bir değer göndermek için kullanılır. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, kullanılacak soketin anahtarını ve gönderilecek değeri alır.

receiveInt (successCallback, errorCallback, socketKey) : Bağlantı kurulmuş taraftan Integer tipinde bir değer alabilmek için kullanılır. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, kullanılacak soketin anahtarını alır.

sendString (successCallback, errorCallback, socketKey, val) : Karşı tarafa String tipinde bir değer göndermek için kullanılır. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, kullanılacak soketin anahtarını ve gönderilecek değeri alır.

receiveString (successCallback, errorCallback, socketKey) : Bağlantı kurulmuş taraftan String tipinde bir değer alabilmek için kullanılır. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, kullanılacak soketin anahtarını alır.

sendBoolean (successCallback, errorCallback, socketKey, val) : Karşı tarafa Boolean tipinde bir değer göndermek için kullanılır. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, kullanılacak soketin anahtarını ve gönderilecek değeri alır.

receiveBoolean (successCallback, errorCallback, socketKey) : Bağlantı kurulmuş taraftan Boolean tipinde bir değer alabilmek için kullanılır. Parametre

olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, kullanılacak soketin anahtarını alır.

sendLong (successCallback, errorCallback, socketKey) : Karşı tarafa Long tipinde bir değer göndermek için kullanılır. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, kullanılacak soketin anahtarını ve gönderilecek değeri alır.

receiveLong (successCallback, errorCallback, socketKey) : Bağlantı kurulmuş taraftan Integer tipinde bir değer alabilmek için kullanılır. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, kullanılacak soketin anahtarını alır.

sendDouble (successCallback, errorCallback, socketKey) : Karşı tarafa Double tipinde bir değer göndermek için kullanılır. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, kullanılacak soketin anahtarını ve gönderilecek değeri alır.

receiveDouble (successCallback, errorCallback, socketKey) : Bağlantı kurulmuş taraftan Double tipinde bir değer alabilmek için kullanılır. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, kullanılacak soketin anahtarını alır.

3.5.2. Javascript'ten Kullanılabilir MQTT Eklentisi Fonksiyonları

MQTT eklentisi altyapısı için aşağıdaki fonksiyonlar Javascript tarafından çağrılabilir şekilde geliştirilmiştir. Bu şekilde son kullanıcı MQTT protokolüne erişebilir duruma getirilmiştir.

newBroker (successCallback, errorCallback, serverURI, clientId) : Broker ile bağlantıya geçilmesini sağlar. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, Broker'ın bulunduğu sunucu IP adresini ve istemcinin ismini alır.

sendString (successCallback, errorCallback, brokerKey, val, topic) : Broker aracılığı ile String tipinde mesaj iletimi yapar. Parametre olarak; başarılı olma

durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, Broker'ın anahtarını, gönderilecek değeri ve hangi başlık altında mesaj iletimi yapılacağını alır.

getString (successCallback, errorCallback, brokerKey, topic) : Broker aracılığı ile String tipinde mesaj alımını yapar. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, Broker'ın anahtarını ve hangi başlık altında mesaj iletimi yapıldığını alır.

subscribe (successCallback, errorCallback, brokerKey, topic) : Broker üzerinde hangi başlığa abone olunacağını belirler. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu, Broker'ın anahtarını ve hangi başlığa abone olunacağını alır.

closeAll (successCallback, errorCallback) : Bağlantıda olunan tüm Broker'larla bağlantıyı keser. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu alır.

closeSafely (successCallback, errorCallback, brokerKey) : Broker'larla bağlantıyı keser. Parametre olarak; başarılı olma durumunda çağrılacak fonksiyonu, hata olması durumunda çağrılacak fonksiyonu alır.

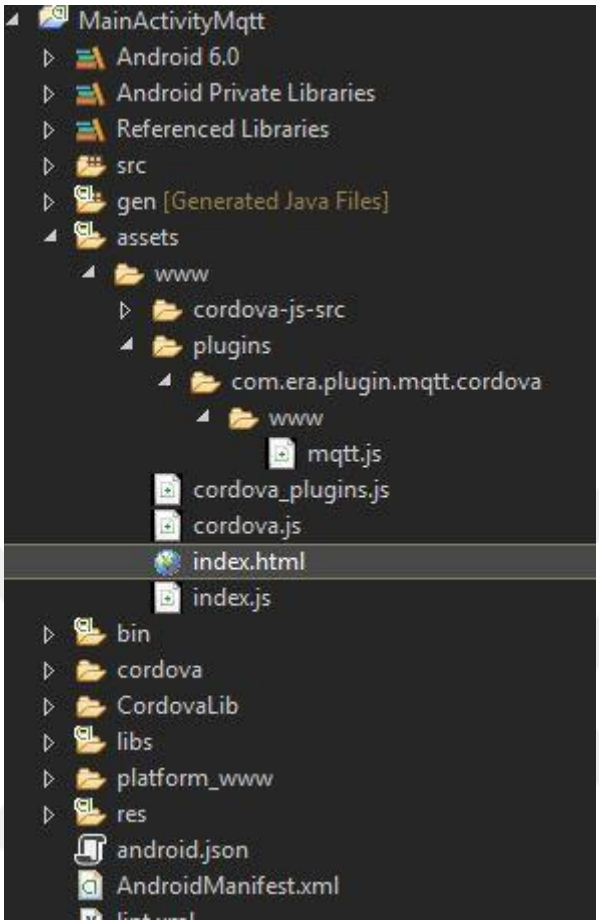
3.6. Eklenti Kullanımı

Entegrasyon yapılmış bir Android projesinden Javascript üzerinden soket ve MQTT protokolleri kullanılabilmektedir. Entegrasyon; cordova eklenti altyapısının kullanımını sağlamaktadır.

3.6.1. MQTT Eklentisi için Örnek Kullanım ve Bellek Alanı İhtiyacı

MQTT Eklentisi için entegrasyonu yapılmış bir Android projesi içerisinde Javascript tarafından MQTT protokolü üzerinden direk olarak haberleşme sağlanabilir. Entegrasyon yapılmış Android projesinde, Şekil 3.13'da görüldüğü gibi web dosyaları "assets/www" içerisinde yer almaktadır. Bu dizindeki web dosyaları,

uygulama başlatıldığında varsayılan olarak çalıştırılacaktır. Bu sebeple Javascript tarafından MQTT erişimi için gerekli kodlar buraya yerleştirilmiştir.

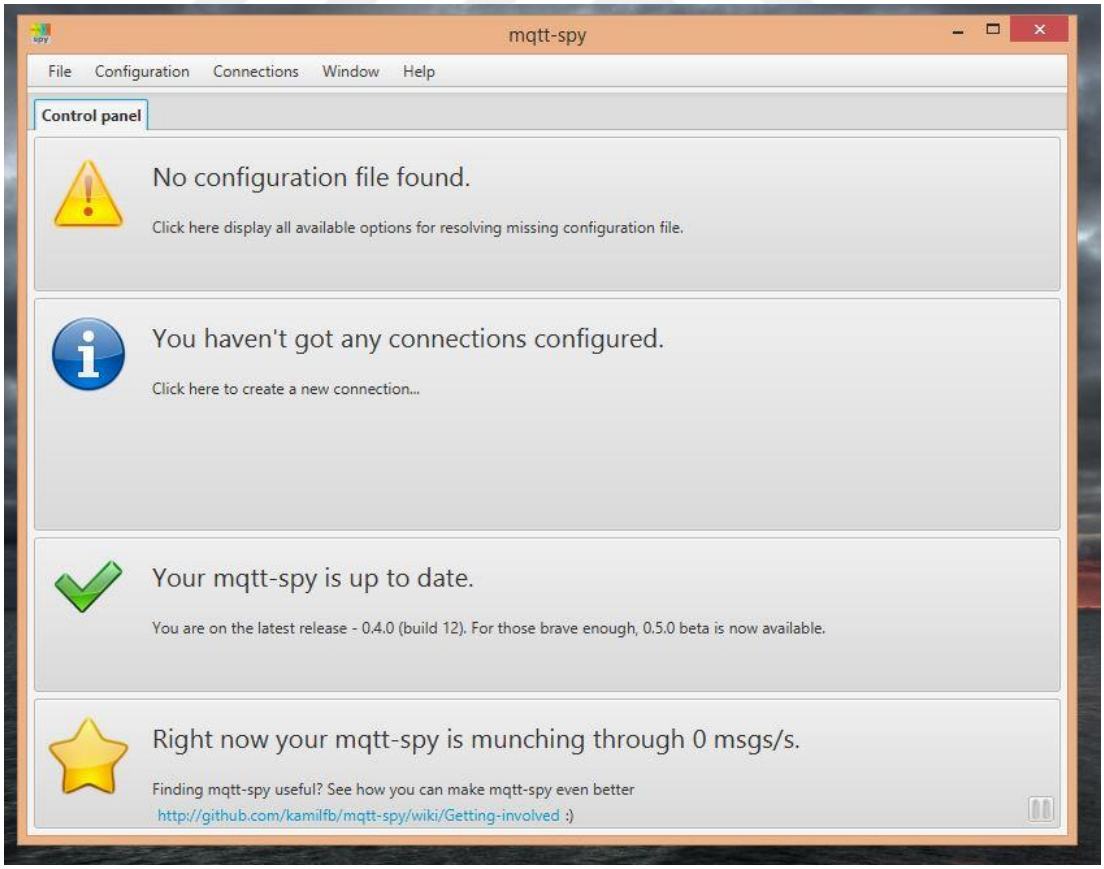


Şekil 3.13. Android projesinde değiştirilecek dosyalar dizini

Javascript tarafından yapılacak istek durumunda MQTT Broker'ının hazırda beklemesi gerekmektedir. MQTT Broker'ı olarak çalışma ortamındaki Moquetto uygulamasının içerisindeki "bin" dizini içerisindeki "moquetto.bat" dosyasını çalıştırarak başlatabilir. Sunucu Şekil 3.14'deki gibi başlamaktadır. MQTT sunucusu ile iletişime geçmek için ikinci bir istemci daha olması ve istemci üzerinden izleme işlemi yapılabilmesi amaçlı, çalışma ortamında bulunan MQTT Spy uygulamasının başlatılması gerekmektedir. MQTT Spy uygulamasını Java üzerinde başlatabilmek için, çalışma ortamında bulunan taşınabilir Java dizini altındaki "bin" dizini içerisindeki Java uygulamasına "–jar" ve ek olarak yine çalışma ortamındaki MQTT Spy dizini altındaki "mqtt-spy-0.4.0-jar-with-dependencies.jar" dosyasını vermek

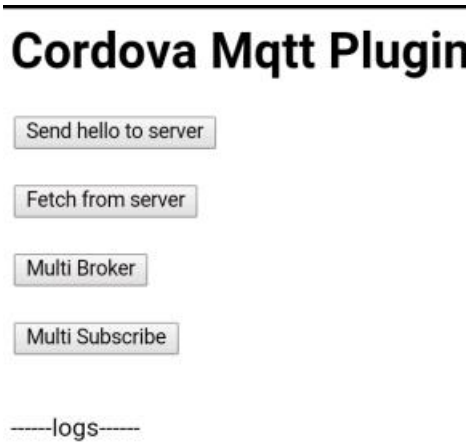
gerekmektedir. Hata alınmadığı takdirde MQTT Spy grafik arayüzü Şekil 3.15'deki gibi açılacaktır.

Şekil 3.14. MQTT Sunucusu komut satırı açılış ekranı



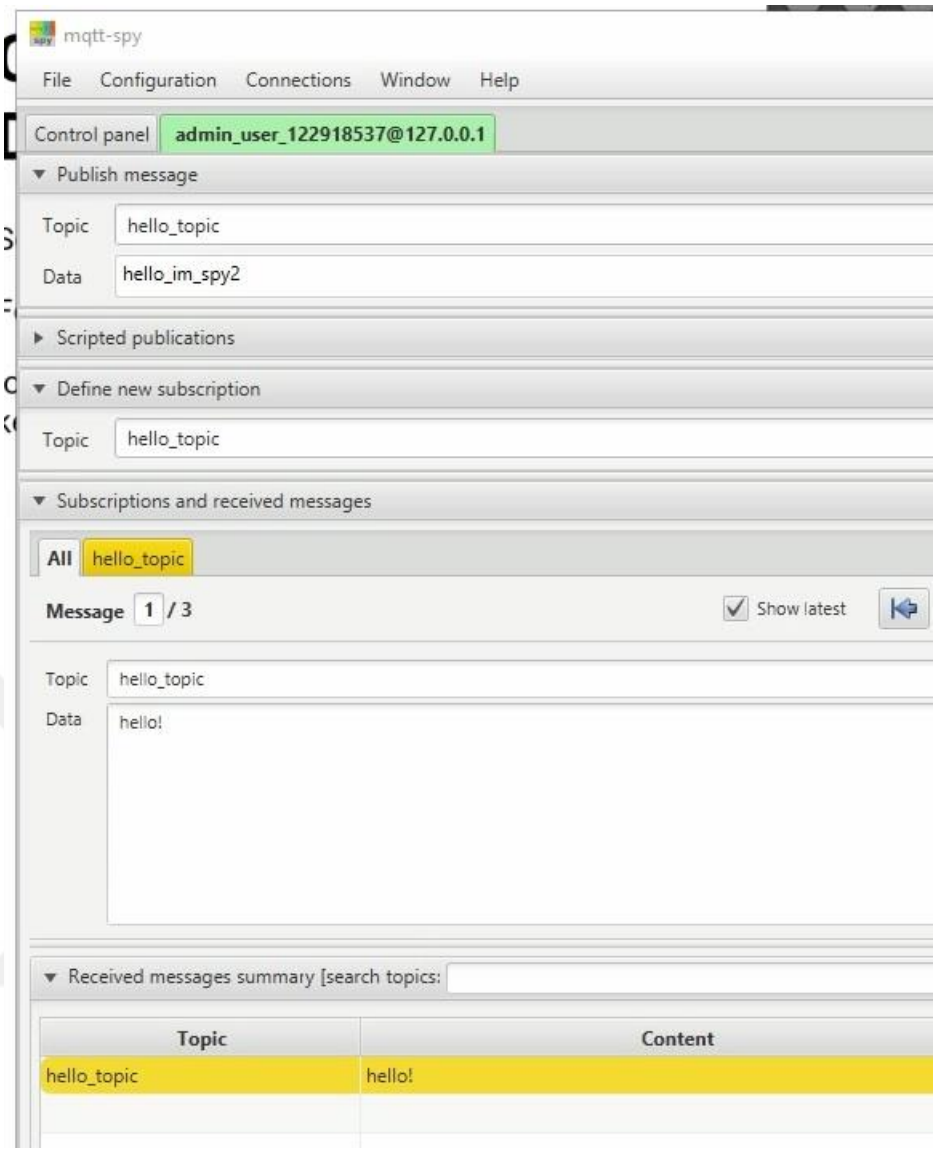
Şekil 3.15. MQTT Spy grafik arayüz açılış ekranı

Javascript tarafında MQTT üzerinden işlem yapabilmek için Şekil 3.16'deki ekran görüntüsüne sahip uygulama hazırlanmıştır.



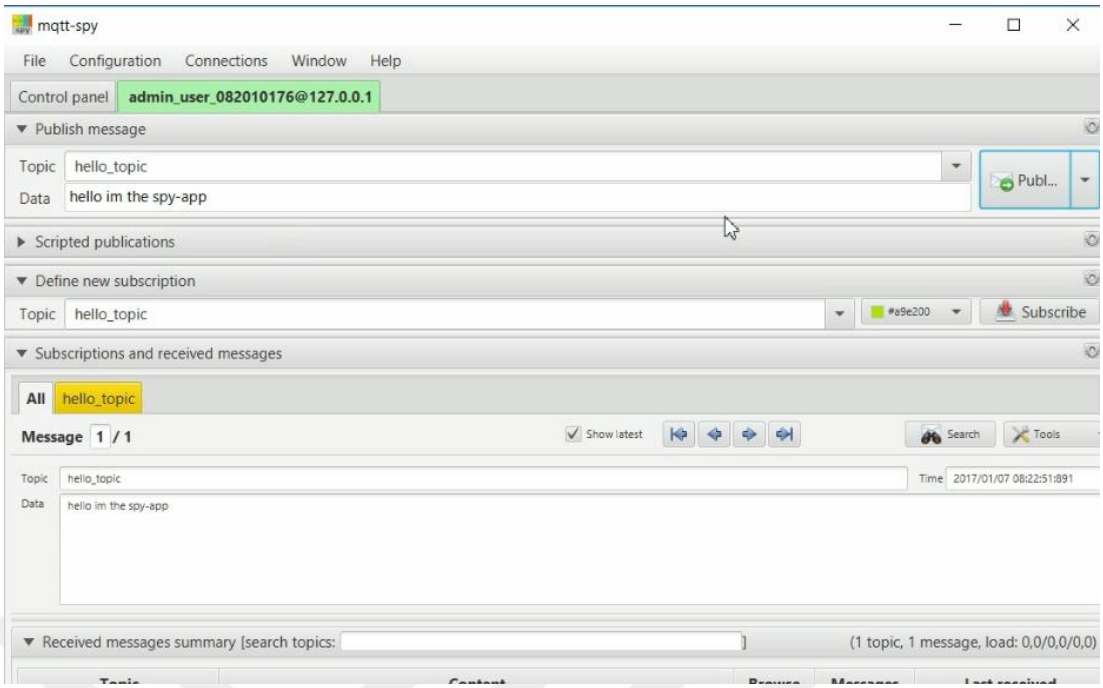
Şekil 3.16. MQTT işlemleri için uygulanın ekran görüntüsü

Javascript tarafında MQTT Broker'a veri yollamak için, ilk aksiyon alınan fonksiyonun içerisinde era.newBroker fonksiyonu çağrılmıştır. Bu fonksiyona, Broker'a başlanma durumunda çağrılacak callback fonksiyonu, hata alınma durumunda çağrılacak callback fonksiyonu, sunucunun IP adresi ve istemcinin kimliği gönderilmektedir. Hata alma durumunda, hata mesajı callback fonksiyonu parametrelerinden alınıp ekrana yansıtılmaktadır. Başarılı olma durumunda, callback fonksiyonu Broker için bir anahtar dönecektir. Bu anahtar kullanılarak, callback fonksiyonunun devamında era.subscribe fonksiyonu çağrılmıştır. Subscribe fonksiyonu MQTT sunucusuna abone olma işlemi gerçekleştirilmiş olacaktır. Subscribe fonksiyonu Broker için gerekli anahtarı, başarılı abone olma işlemi sonrası için callback fonksiyonunu, hata alma durumunda çalıştırılacak callback fonksiyonunu, Broker anahtarını ve abone olunacak başlığı parametre olarak almaktadır. Subscribe fonksiyonunun başarılı olma durumunda; era.sendString fonksiyonu çağrılmaktadır. Bu şekilde uygulama çalıştırıldığında, aksiyonu alacak button'a tıklandığında, broker'a mesaj gönderilmiş ve MQTT Spy'a mesaj Şekil 3.17'deki gibi görünecektir.



Şekil 3.17. MQTT Spy mesaj okuduğundaki ekran görüntüsü

MQTT Spy üzerinden mesaj broker'a iletildiğinde ise; Android uygulaması bu mesajı algılayacaktır. MQTT Spy uygulamasında Şekil 3.18'de görüldüğü gibi mesajın iletilmesi gerekmektedir.



Şekil 3.18. MQTT Spy üzerinden mesaj iletimi

Uygulama hibrit modelde çalıştığı için performans önemli bir konu halini almaktadır. Bellek kullanımını ölçmek için ana dildeki "android.os" isimli paketlerden yararlanılmıştır. Uygulama hiçbir işlem yapmıyorken Şekil 3.19'de görüldüğü gibi 51 MB civarında kullandığı görülmektedir. Bu 51 MB bellek içerisinde Cordova iskeletinin kullandığı hafıza alanı da mevcuttur.

```
I/MEMORY(2384): App Memory: Pss=51.47 MB, Private=12.24 MB, Shared=40.24 MB
I/MEMORY(2384): App Memory: Pss=51.47 MB, Private=12.24 MB, Shared=40.24 MB
I/MEMORY(2384): App Memory: Pss=51.47 MB, Private=12.24 MB, Shared=40.24 MB
```

Şekil 3.19. Boş uygulama bellek kullanımı

Arkaplanda uygulamanın belleği Java objeleri ile doldurulduğunda Şekil 3.20'de görüldüğü gibi belleğin 1 MB aralıklarla arttığı gözlemlenmektedir. Bu işlem bellek kullanımı göstergelerinin doğruluğunu kanıtlamak için yapılmıştır.

```
I/MEMORY(2781): App Memory: Pss=55.88 MB, Private=16.61 MB, Shared=40.23 MB
I/MEMORY(2781): App Memory: Pss=56.64 MB, Private=17.37 MB, Shared=40.23 MB
I/MEMORY(2781): App Memory: Pss=56.46 MB, Private=17.18 MB, Shared=40.23 MB
```

Şekil 3.20. Java objeleri ile doldurulan belleğin artışı

10 adet broker aracılığı ile subscribe işlemi yapıldığında belleğin Şekil 3.21'deki gibi 57 MB civarına çıktığı görülmektedir.

```
I/MEMORY(2862): App Memory: Pss=57.77 MB, Private=17.32 MB, Shared=40.11 MB
I/MEMORY(2862): App Memory: Pss=57.77 MB, Private=17.32 MB, Shared=40.11 MB
I/MEMORY(2862): App Memory: Pss=57.77 MB, Private=17.32 MB, Shared=40.11 MB
```

Şekil 3.21. 10 adet broker bağlantısı sonucunda bellek kullanımı

100 adet broker aracılığı ile subscribe işlemi yapıldığında belleğin Şekil 3.22'deki gibi 91 MB civarına çıktığı görülmektedir.

```
I/MEMORY(2862): App Memory: Pss=90.77 MB, Private=50.38 MB, Shared=40.12 MB
I/MEMORY(2862): App Memory: Pss=91.24 MB, Private=50.69 MB, Shared=40.12 MB
I/MEMORY(2862): App Memory: Pss=91.24 MB, Private=50.69 MB, Shared=40.12 MB
```

Şekil 3.22. 100 adet broker bağlantısı işlemi yapıldığında bellek kullanımı

Tek broker üzerinden 10 adet subscribe işlemi yapıldığında, bellek kullanımının Şekil 3.23'de görüldüğü gibi 55 MB civarında olduğu görülmüştür.

```
I/MEMORY(2862): App Memory: Pss=54.60 MB, Private=14.20 MB, Shared=40.11 MB
I/MEMORY(2862): App Memory: Pss=54.60 MB, Private=14.20 MB, Shared=40.11 MB
I/MEMORY(2862): App Memory: Pss=54.61 MB, Private=14.20 MB, Shared=40.11 MB
```

Şekil 3.23. 10 adet subscribe işlemi sonrası bellek kullanımı

Tek broker üzerinden 100 adet subscribe işlemi yapıldığında, bellek kullanımının Şekil 3.24'de görüldüğü gibi 56 MB civarında olduğu görülmüştür.

```
I/MEMORY(2862): App Memory: Pss=56.39 MB, Private=15.91 MB, Shared=40.11 MB
I/MEMORY(2862): App Memory: Pss=56.40 MB, Private=15.91 MB, Shared=40.11 MB
I/MEMORY(2862): App Memory: Pss=56.40 MB, Private=15.91 MB, Shared=40.11 MB
```

Şekil 3.24. 100 adet subscribe işlemi sonrası bellek kullanımı

Yukarıdaki testler aynı emülatör üzerinde gerçekleştirilmiştir. Bu başlıktaki aşağıdaki testler ise farklı emülatörlerde gerçekleştirilmiştir. Bu sebeple; yeni emülatörde uygulama ilk başlatıldığındaki bellek kullanımı tekrar incelenmiştir. Bellek kullanımı Şekil 3.25'te verilmiştir.

```
04-15 14:31:21.600: I/MEMORY(2384): App Memory: Pss=56.31 MB, Private=22.95 MB, Shared=29.40 MB
04-15 14:31:26.611: I/MEMORY(2384): App Memory: Pss=56.32 MB, Private=22.96 MB, Shared=29.40 MB
04-15 14:31:31.621: I/MEMORY(2384): App Memory: Pss=56.32 MB, Private=22.96 MB, Shared=29.40 MB
```

Şekil 3.25. Farklı emülatörde işlem yapmamış uygulamanın bellek kullanımı

Uygulama üzerinde 10 adet 13 karakterlik bir string gönderildiğinde bellek kullanımı Şekil 3.26'da verilmiştir.

```
04-15 14:37:32.772: I/MEMORY(2497): App Memory: Pss=57.47 MB, Private=14.49 MB, Shared=40.11 MB
04-15 14:37:37.823: I/MEMORY(2497): App Memory: Pss=57.49 MB, Private=14.50 MB, Shared=40.11 MB
04-15 14:37:42.849: I/MEMORY(2497): App Memory: Pss=57.49 MB, Private=14.50 MB, Shared=40.11 MB
```

Şekil 3.26. 10 adet string göndermiş uygulamadaki bellek kullanımı

Uygulama üzerinde 100 adet 13 karakterlik string gönderildiğinde bellek kullanımı Şekil 3.27'de verilmiştir.

```
04-15 14:40:42.002: I/MEMORY(2672): App Memory: Pss=60.25 MB, Private=16.91 MB, Shared=40.09 MB
04-15 14:40:47.014: I/MEMORY(2672): App Memory: Pss=60.00 MB, Private=16.62 MB, Shared=40.09 MB
04-15 14:40:52.059: I/MEMORY(2672): App Memory: Pss=60.99 MB, Private=16.59 MB, Shared=40.09 MB
```

Şekil 3.27. 100 adet string göndermiş uygulamadaki bellek kullanımı

MQTT testleri sonrasında tüm işlemlerin bellek gereksinimleri özetlenerek Tablo 3.1 ve Tablo 3.2'de verilmiştir. Testler iki farklı emülatör üzerinden yapıldığı için farklı tablolarda gruplandırılmıştır.

Tablo 3.1. MQTT testlerinde bellek kullanım miktarları (Emülatör-1)

İŞLEM	KULLANILAN TOPLAM BELLEK (MB)
Uygulama Boşta	51
10 adet Broker üzerinden subscribe işlemi	57
100 adet Broker üzerinden subscribe işlemi	91
Tek adet broker üzerinden 10 subscribe işlemi	55

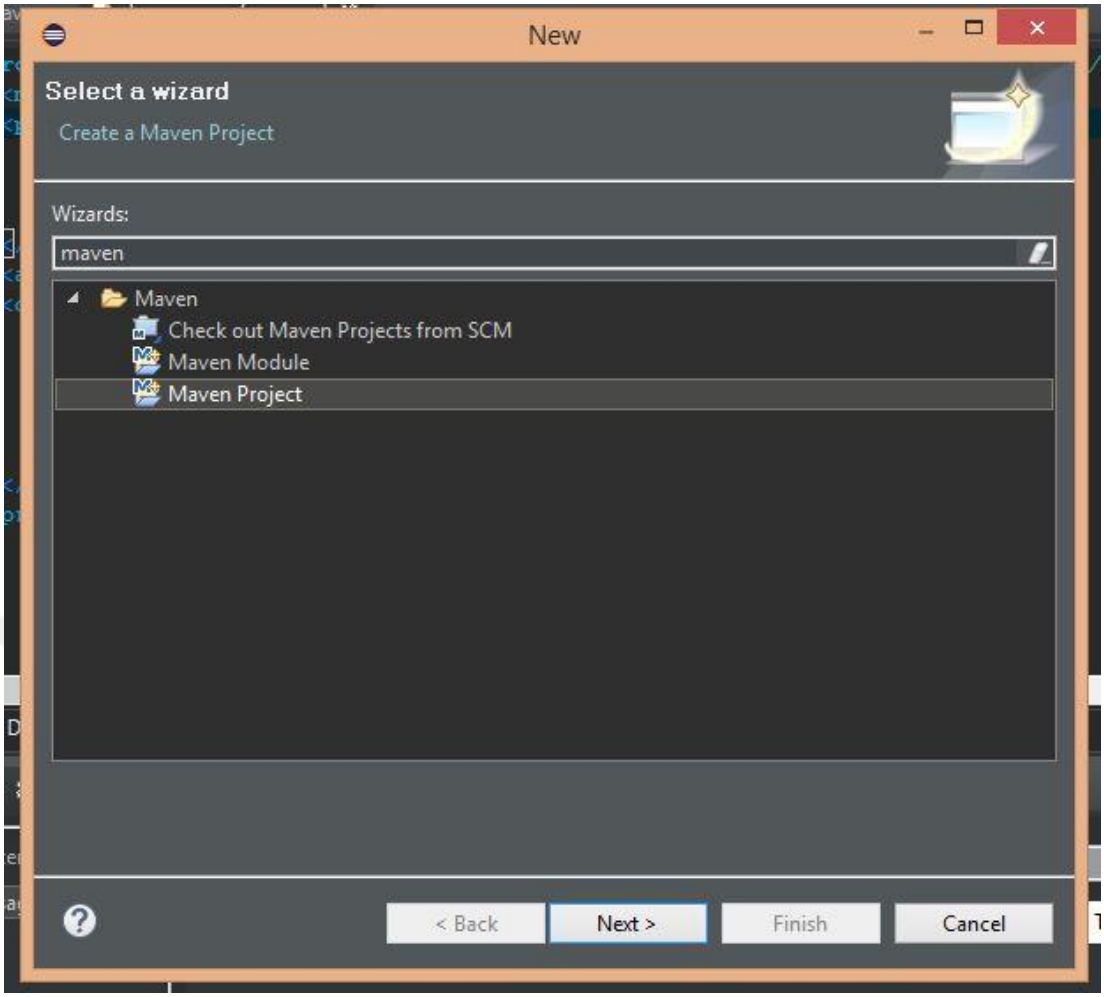
Tek adet broker üzerinden 100 subscribe işlemi	56

Tablo 3.2. MQTT testlerinde bellek kullanım miktarları (Emülatör-2)

İŞLEM	KULLANILAN TOPLAM BELLEK (MB)
Uygulama Boşta	56
10 adet String transferi işlemi	57
100 adet String transferi işlemi	61

3.6.2. TCP Soket Eklentisi için Örnek Kullanım ve Bellek Alanı İhtiyacı

Entegrasyonu yapılmış bir Android projesi üzerinde TCP soket eklentisi kullanımı için, Android cihazın iletişime geçeceği sunucu yazılmıştır. Eklentinin iletişime geçmesi için, Eclipse üzerinde yeni bir Java uygulaması Maven altyapısı ile Şekil 3.28'de olduğu gibi açılmıştır. Proje çalışma ortamı root isimli projenin altında olacağı için Maven Module olarak yaratılmıştır.

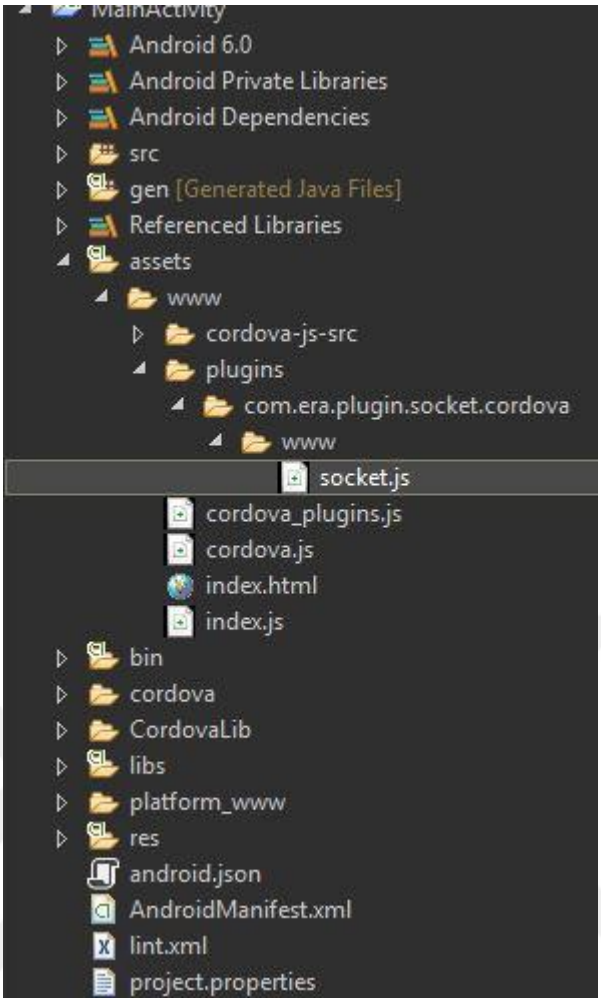


Şekil 3.28. Sunucu için yeni Maven projesi tanımlama ekranı

Demo sunucusu kütüphane olarak TCP soket sınıflarını kullanacağı için, Maven bağımlılıklarına, pom.xml dosyası üzerinden TCP sınıfları eklenmiştir.

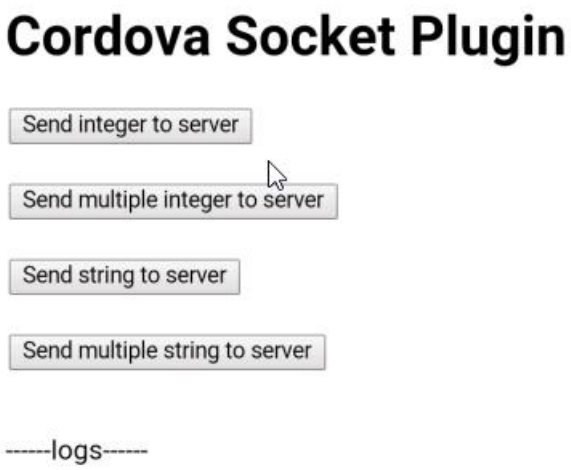
Sunucu üzerinde ana işlem sürekli olarak, bir istemciden bağlantı bekler durumda ayarlanmıştır. Bağlantıyı bekleyen sunucu, istemcilerden bir bağlantı geldiği zaman farklı bir işlem başlatarak, istemcinin göndereceği veriyi paralelden alıp test amaçlı konsolda göstermektedir. Burada gelecek verinin tipi (String, integer gibi) hem sunucu hem de istemci tarafta önceden belirlenmiş olmalıdır. Aksi durumda sunucu hata fırlatmayabilir ve sonsuz döngülere girer yada programın akışı yanlış çalışır.

Entegrasyonu yapılmış Android projesinde, Şekil 3.29'da olduğu gibi platforms\android\assets\www dizini altında web dosyaları yer almaktadır.



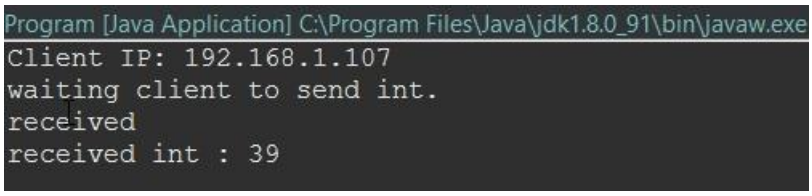
Şekil 3.29. Cordova projesinde web dosyaları dizini

Cordova uygulaması başladığında, bu dizinde varsayılan olarak index.html dosyası çağrıldığından buraya Javascript tarafından sunucuya veri gönderilecek kodlar eklenmiştir. Android uygulamasının arayüzü Şekil 3.30'de verilmiştir.



Şekil 3.30. Soket işlemleri için Android uygulamasının arayüz ekranı

Sunucu tarafta beklemekte olan sokete bağlanmak için yeni soket yaratmak gerekmektedir. Javascript tarafında, istenilen aksiyon üzerine, era.newSocket fonksiyonu çağrılmıştır. Yeni soket açacak bu fonksiyona, başarılı olma durumunda çalıştıracağı callback fonksiyonu, hata alma durumunda alacağı callback fonksiyonu, sunucunun IP adresi ve beklemekte olan portu verilmiştir. Hata alma durumunda callback fonksiyonunda, fonksiyon çağrılınca geçilen parametre içerisindeki hata mesajı ekranda gösterilmiştir. Başarılı olma durumunda, callback fonksiyonuna geçilen parametreler arasında yeni soketin tekil anahtarı yer almaktadır. Bu anahtar kullanılarak istenilen soket üzerinde işlem çağrılabilmektedir. Başarılı olma durumunda çağrılan callback fonksiyonunun içerisinde era.sendInt fonksiyonu çağrılmaktadır. Bu fonksiyon parametre olarak işlem yapılacak soket anahtarını, başarılı gönderme işlemi sonucunda çalıştırılacak callback fonksiyonunu, başarısız gönderme durumunda çalıştırılacak callback fonksiyonunu ve soket üzerinden gönderilecek integer değerini almaktadır. Burada başarısız olma durumunda hata mesajı ekrana yansıtılırken, başarılı olma durumunda ise, başarılı gönderildiğine dair bir mesaj gösterilmektedir. Sunucu tarafında ise hazırda bekleyen işlem bir integer beklemekte ve gelecek olan integer değerini ekranda göstermektedir. Integer gönderme işlemi gerçekleştiğinde, Integer değerini alan sunucu tarafta ekran log'lar Şekil 3.31'daki gibidir.



Şekil 3.31. Integer değerini alan sunucu uygulamadaki log'lar

Uygulama ilk açıldığında bellek kullanımı Şekil 3.32'da görüldüğü gibi 54 MB civarındadır. Bu 53 MB bellek içerisinde Cordova iskeletinin kullandığı hafıza alanı da mevcuttur.

```
I/MEMORY(2862): App Memory: Pss=53.90 MB, Private=13.89 MB, Shared=40.14 MB
I/MEMORY(2862): App Memory: Pss=53.90 MB, Private=13.89 MB, Shared=40.14 MB
I/MEMORY(2862): App Memory: Pss=53.90 MB, Private=13.89 MB, Shared=40.14 MB
```

Şekil 3.32. İşlem başlatmamış Android uygulaması için bellek kullanımı

Uygulama üzerinden aynı soket üzerinden 10 adet integer yollandığı zaman bellek kullanımı Şekil 3.33'de görüldüğü gibi hala yaklaşık 54 MB'lerdedir.

```
I/MEMORY(2862): App Memory: Pss=54.18 MB, Private=14.17 MB, Shared=40.14 MB
I/MEMORY(2862): App Memory: Pss=54.18 MB, Private=14.17 MB, Shared=40.14 MB
I/MEMORY(2862): App Memory: Pss=54.18 MB, Private=14.17 MB, Shared=40.14 MB
```

Şekil 3.33. 10 adet integer transferi sonrası bellek kullanımı

Android uygulamasında tek soket üzerinden 10 adet 13 karakter boyutunda String gönderiminde bellek kullanımı Şekil 3.34'de görüldüğü gibi 54 MB civarındadır.

```
I/MEMORY(2862): App Memory: Pss=54.25 MB, Private=14.25 MB, Shared=40.14 MB
I/MEMORY(2862): App Memory: Pss=54.25 MB, Private=14.25 MB, Shared=40.14 MB
I/MEMORY(2862): App Memory: Pss=54.25 MB, Private=14.25 MB, Shared=40.14 MB
```

Şekil 3.34. 10 adet string transferi sonrası bellek kullanımı

Android uygulamasından tek soket üzerinden 100 adet integer değeri sunucuya yollandığında, bellek kullanımının Şekil 3.35'deki görüldüğü gibi 54 MB civarındadır.

```
I/MEMORY(2862): App Memory: Pss=54.18 MB, Private=14.48 MB, Shared=40.18 MB
I/MEMORY(2862): App Memory: Pss=54.18 MB, Private=14.48 MB, Shared=40.18 MB
I/MEMORY(2862): App Memory: Pss=54.19 MB, Private=14.48 MB, Shared=40.18 MB
```

Şekil 3.35. 100 adet integer transferi sonrası bellek kullanımı

Android uygulamasından tek soket üzerinden 100 adet 13 karakter boyutunda string değeri sunucuya yollandığında, bellek kullanımının Şekil 3.36'deki görüldüğü gibi 54 MB civarındadır.

```
I/MEMORY(2862): App Memory: Pss=54.41 MB, Private=14.37 MB, Shared=40.18 MB
I/MEMORY(2862): App Memory: Pss=54.41 MB, Private=14.37 MB, Shared=40.18 MB
I/MEMORY(2862): App Memory: Pss=54.41 MB, Private=14.37 MB, Shared=40.18 MB
```

Şekil 3.36. 100 adet string transferi sonrası bellek kullanımı

Uzun karakter sayısına sahip string değerleri gönderilmek istenildiğinde Cordova eklenti altyapısının limitine takılmaktadır. Javascript tarafından 9000 karakter uzunluğunda String gönderildiğinde uygulamada hata alınmadığı fakat sunucuda boş karakterlerin algılandığı görülmektedir. Android uygulamasının adımları izlendiğinde, Javascript'teki String değerinin Cordova eklenti altyapısının ana dildeki String dönüştürme işleminde bu sorunun başladığı görülmektedir. Bu çalışmada hazırlanan implementasyon dışında bir çok farklı işletim sistemine de implementasyon yazılabileceğinden; Javascript tarafından her zaman kısa boyutlarda string değerleri kullanılması doğru bir yöntem olacaktır. Çünkü Javascript tarafından ortak yazılan kodlar her işletim sistemini aynı şekilde desteklemesi gereklidir. Yapılan testler sonucu 200 karakterinde desteklendiği görülmüştür. Bu sebeple 200 karakterden fazla gönderilecek string değerlerinin sunucuya ayrı ayrı atılması önerilmektedir.

Yukarıdaki tüm bellek hesaplamaları aynı Android emülatörü üzerinden test edilmiştir ve gönderilen veriler aynı soket üzerinden gönderilmiştir. Farklı soket açılması durumundaki artışı tespit edebilmek için yeni bir emülatör üzerinde testler yapılmıştır. Bu testlerde; sunucuya farklı soket üzerinden veri yollanmıştır. Yeni emülatör üzerinde uygulamada hiçbir işlem yapılmıyorken bellek kullanımı yaklaşık 53 MB olup Şekil 3.37'de verilmiştir.

```
03-25 11:32:33.227: I/MEMORY(2797): App Memory: Pss=53.16 MB, Private=13.23 MB, Shared=40.16 MB
03-25 11:32:37.373: I/MEMORY(2797): App Memory: Pss=53.17 MB, Private=13.23 MB, Shared=40.16 MB
03-25 11:32:38.257: I/MEMORY(2797): App Memory: Pss=53.17 MB, Private=13.23 MB, Shared=40.16 MB
```

Şekil 3.37. İşlem başlatmamış Android uygulaması bellek kullanımı

Android uygulaması üzerinde tek adet soket açıp bir integer gönderildiği durumda bellek kullanımı Şekil 3.38'de görüldüğü gibi 53 MB civarındadır.

```
03-25 11:41:30.564: I/MEMORY(2797): App Memory: Pss=53.33 MB, Private=13.38 MB, Shared=40.15 MB
03-25 11:41:31.089: I/MEMORY(2797): App Memory: Pss=53.33 MB, Private=13.38 MB, Shared=40.15 MB
03-25 11:41:31.765: I/MEMORY(2797): App Memory: Pss=53.33 MB, Private=13.38 MB, Shared=40.15 MB
```

Şekil 3.38. Bir adet integer gönderme durumunda bellek kullanımı

Android uygulaması üzerinde üç adet soket açılıp sunucuya integer değeri gönderildiği zaman bellek kullanımı Şekil 3.39'da görüldüğü gibi 53 MB civarındadır.

```
03-25 11:34:49.012: I/MEMORY(2797): App Memory: Pss=53.34 MB, Private=13.39 MB, Shared=40.15 MB
03-25 11:34:53.232: I/MEMORY(2797): App Memory: Pss=53.35 MB, Private=13.40 MB, Shared=40.15 MB
03-25 11:34:54.060: I/MEMORY(2797): App Memory: Pss=53.35 MB, Private=13.40 MB, Shared=40.15 MB
```

Şekil 3.39. Farklı soketler üzerinden işlem yapıldıktan sonra bellek kullanımı

Son 2 işlemde görüldüğü gibi farklı soketler açılarak, bu soketler üzerinden veri gönderildiğinde bellek kullanım ihtiyacı farkının 1 MB'nin altında olduğu görülmektedir.

Soket testleri sonrasında tüm işlemlerin bellek gereksinimleri özetlenerek Tablo 3.3 ve Tablo 3.4'te verilmiştir. Testler iki farklı emülatör üzerinden yapıldığı için farklı tablolarda gruplandırılmıştır.

Tablo 3.3. Soket testlerinde bellek kullanım miktarları (Emülatör-1)

İŞLEM	KULLANILAN TOPLAM BELLEK (MB)
Uygulama Boşta	53
Tek soket üzerinden 10 adet İnteger transferi	54

Tek soket üzerinden 10 String transferi	54
Tek soket üzerinden 100 İnteger transferi	54
Tek soket üzerinden 10 String transferi	54

Tablo 3.4. Soket testlerinde bellek kullanım miktarları (Emülatör-2)

İŞLEM	KULLANILAN TOPLAM BELLEK (MB)
Uygulama Boşta	53
Tek soket üzerinden İnteger transferi	53
3 soket üzerinden İnteger transferi	53

Farklı veri tiplerinde veri transferleri soket eklentisinde fazla bellek kullanımı ihtiyacı gerektirmemektedir. Çünkü transfer edilen bilgi, transfer işlemi sonrası serbest bırakılmaktadır.

MQTT ve soket eklentisinin Java ve Javascript'teki yönetimi için kullanılan mimariler birbirleri ile aynıdır. İki eklentide de Java üzerindeki bağlantı objeleri HashMap veri yapısında tutulmakta ve anahtar ile Javascript tarafından çağrılmaktadır. Fakat MQTT ile karşılaştırıldığında, soket eklentisinin daha az bellek kapasitesi ihtiyacı gerektirdiği görülmektedir. Bunun temel sebeplerinden biri MQTT için "Paho" kütüphanesinden yararlanılmış olmasıdır. Paho kütüphanesi ihtiyaçları gereği fazla bellek kullanımı gerektirmektedir. Testlerin yapıldığı emülatör Android 5.0.1 sürümünü kullanmaktadır. Farklı Android sürümleri üzerinde farklı sonuçlar alınabilir.

4. SONUÇ

Özellikle özgür yazılım geliştirici topluluklarının varlığı ve çağımızın ihtiyaçları yazılım teknolojilerini hızla değiştirmektedir. Çağımız ihtiyaçları sistemleri karmaşık (kompleks) yapıdan arındırıp, basit fakat daha esnek, hızlı geliştirilebilir, anlaşılır bir hale getirmektedir. Bu çalışma ile güncel teknolojiler incelenmiş, bu teknolojiler üzerinde avantajlar ve dezavantajlar ortaya konmuştur.

Bu çalışmada var olan hibrit uygulama geliştirme teknolojilerin araştırılmasının yanı sıra, güncel teknolojiler kullanılarak; platform bağımsız çalışmayı sağlayan hibrit uygulamalarda soket ve MQTT protokolünün kullanımını sağlayacak yapının oluşturulması sağlanmıştır. Bu çalışmada da görüldüğü gibi; yeni teknolojilerin esnekliği, üzerinde değişiklik yapmayı daha kolay hale getirdiği gibi, bu değişikliklerinde daha istikrarlı (stabil) çalışabilmesi sağlanmaktadır.

Günümüzde IoT (Internet of Things) bazlı teknolojiler büyük ilgi görmektedir. Zira çağımızın ihtiyaçları, kullandığımız birçok cihazda teknoloji ile içiçe olmasını gerektirmektedir. IoT cihazlar, dağınık ve birçok farklı türde platformlarla iletişimde olacağından MQTT protokollünü yoğunlukla kullanmaktadır. Hibrit uygulamalar, IoT cihazları ve diğer MQTT protokollünü kullanan cihazlar ile iletişimde olabilmek için bu çalışmada geliştirilen API'den yararlanabilmektedirler.

İnternetin erişim altyapısında en alt seviyeli iletişim kanalı olan soketler günümüzde birçok protokolün altyapısını oluşturduğu gibi, ek protokol kullanmadan da iletişimi sağlayabilmektedir. Soket kullanımı, ağ üzerinde taşınan verinin daha düşük boyutlarda olmasını sağlamaktadır. Bu avantajdan yararlanmak isteyen hibrit uygulamalar bu çalışmada geliştirilen arayüz (API) üzerinden iletişim sağlayabilmektedirler.

Tasarlanan yapı, örnek bir Android uygulaması üzerinde çalıştırılmıştır. Android uygulaması üzerinde kullanılan fonksiyonların bellek kullanımı

incelenmiştir. İncelemeler sonucunda; soket eklentisi aracılığı ile haberleşmenin, MQTT eklentisi ile haberleşmeye göre daha az bellek kullanımı gerektiği görülmüştür. Soketler aracılığı ile sunucu ile veri alışverişi yapılırken, verilerin farklı soketler üzerinden yollanması yada aynı soket üzerinden verilerin yollanması arasında bellek kullanımı açısından fark görülmemiştir. MQTT sunucuları için farklı implementasyonları kullanıldığından, farklı sonuçlar doğurabileceği göz önünde bulundurulmalıdır. Bu çalışmada testler sadece Moquette sunucu üzerinden yapılmıştır.

Tezde incelenen konu üzerinden yapılabilecek bir sonraki çalışmalarda; hibrit eklentisi için kurulabilir paket yönetimi sağlanabilir. Yazılımcı, Cordova'nın merkezi paket yönetimi aracılığı ile entegrasyonu yapabilir olacaktır. Bu şekilde; geliştirilecek yazılımın sürüm yükseltmeleri de daha başarılı şekilde yapılabilecektir.

Tezde araştırılan konuyla ilgili sonraki çalışmalarda; soket ve MQTT uygulama kullanıcı arayüzüne yeni fonksiyonlar eklenebilir ve geliştirilecek yazılıma ek birçok seçenek sunulabilir. Bu sunulan fonksiyonlar aynı şekilde native uygulama dili tarafında da seçenekleri sunabilir ve benzer API kullanılabilir.

Bu çalışmada tasarlanan yapının birçok platform için implementasyonu yapılabilir. Bu şekilde kullanıcı hibrit uygulama modelinin verimliliğinden daha çok platform altında çalışması ile yararlanabilecektir. İmplemente edilecek fonksiyonların ek özellikler sunması sağlanması için API kendi içerisinde bir eklenti altyapı sunması ve bu altyapı ile soket ve MQTT protokolleri içinin işleyişlerini değiştirebilmesi sağlanabilir.

Cordova veya diğer iskelet yapılar için tarayıcı bazında simülasyon gerçekleştirmek için arayüz desteklenebilir. Zira Cordova'nın güncel sürümlerinde belirtilen tarayıcı (browser) platformu simülasyon için kullanılmaktadır ve test işlemleri oradan yapılmaktadır. API desteklediği kadarı ile simülasyona izin verilebilir.

5. KAYNAKLAR

[1] Gartner Araştırma Yayınları. (2014). Erişim Tarihi: 23.11.2014, http://www.gartner.com/newsroom/id/2665715

[2] Canalys Araştırma Yayınları. (2007). Erişim Tarihi: 23.11.2014, http://www.canalys.com/newsroom/64-million-smart-phones-shipped-worldwide-2006

[3] Huffington Post UK. (2014). Erişim Tarihi: 23.11.2014, http://www.huffingtonpost.co.uk/2014/01/28/billion-smartphones_n_4679082.html

[4] Haroon Q R (2011). 'An Introduction To Modern Mobile Operating Systems'. Erişim Tarihi: 23.11.2014, http://www.addictivetips.com/mobile/an-introduction-to-modern-mobile-operating-systems/

[5] Ivan Njunjic (2012). "Development Techniques for Android Platform Mobile Device Application". Eastern Michigan University, Information Systems, Master of Science.

[6] Mikko Gronoff (2013). "Open Source Mazeball Game for Windows Phone". Oulu University of Applied Sciences. Information Technology. Master's thesis.

[7] Longyi Qi (2012). "A static code analysis tool for Objective-C and related languages". University of Houston, Computer Science, Master of Science.

[8] Sunee W (2010). 'The study of web application development versus native application development on iphone'. San Diego State University, Computer Science, Master of Science.

[9] Antti S (2013). 'Web and Native Technologies in Mobile Application Development'. Degree Programme of Computer Science and Engineering, Master's Thesis.

[10] PhoneGap. Erişim Tarihi: 23.11.2014, http://phonegap.com/app/healthtap-find-doctors-and-free-answers/

[11] PhoneGap. Erişim Tarihi: 23.11.2014, http://phonegap.com/app/bbc-olympics/

[12] Killian B (2012). ‘BBC launches Olympics iPhone app with 24 live streams’. Erişim Tarihi: 23.11.2014, http://www.macsessed.com/posts/bbc-launches-olympics-iphone-app-with-24-live-streams/

[13] Healttap. Tarihi: 23.11.2014, https://play.google.com/store/apps/details?id=com.healthtap.userhtexpress&hl=tr

[14] Samuel C (2013). ‘Developing a cross-platform mobile application: my shopping list’. Turku university of applied sciences, Information Technology, Bachelor’s Thesis.

[15] Sanna Ottka (2015). “Comparison of mobile application development tools for multi-platform industrial applications”. Aalto University, Computer Science and Engineering, Master’s Thesis.

[16] Andrea Sanchez Blanco (2016). “Development of Hybrid Mobile Apps Using Ionic framework”. Mikkeli University of Applied Sciences. Information Technology. Thesis.

[17] Pavel Fidransky (2015). “Mobile client for collecting sport activity statistics”. University of West Bohemia, Computer Science and Engineering, Thesis.

[18] Christian Karasiewicz (2013). ‘Apache Cordova and IBM Worklight: A partnership for building hybrid apps’. IBM Worklight Developer Works. Erişim Tarihi: 23.11.2014, https://www.ibm.com/developerworks/community/blogs/mobileblog/entry/apache_cordova_and_ibm_worklight_a_partnership_for_building_hybrid_apps?lang=en

[19] Oleksii R (2013). 'Comparing HTML5 Based Apps With Native Apps'. Linnaeus University, Department of Computer Science, Master Degree.

[20] Frank A (2010). 'Using XML and JSON with Android, Part 2: Deliver hybrid Android applications with JSON'. IBM Developer Works, Technical Topics.

[21] Alexander N (2012). 'Android: WebView tips & tricks'. Erişim Tarihi: 25.11.2014, http://vision-apps.blogspot.com.tr/2012/08/android-webview-tips-tricks.html

[22] Rakesh Mahato (2016). “Hybrid Mobile Application Development”. Helsinki Metropolia University of Applied Sciences, Engineering Department, Thesis.

[23] Jari Pohjanen (2014). “Cross Platform Application Development With Html5 For Ios And Android Operating Systems”, Oulu University of Applied Sciences, Information Technology, Master’s thesis.

[24] Xiaoyu Zhuang (2012). “Interaction between web browsers and script engines”. Uppsala University, Department of Information Technology, Thesis.

[25] Maximiliano F. 'HTML5 compatibility on mobile and tablet browsers with testing on real devices’. Erişim Tarihi: 23.11.2014, http://mobilehtml5.org/

[26] Frank O ve Adrian B (2014). 'The Mobile Web should just work for everyone', Microsoft Developer Network. Erişim Tarihi: 23.11.2014, http://blogs.msdn.com/b/ie/archive/2014/07/31/the-mobile-web-should-just-work-for-everyone.aspx

[27] Nicholas C. Zakas & Jeremy McPeak & Joe Fawcett (2007). (2nd Edition). “Professional Ajax”. ISBN: 978-0-470-10949-6.

[28] Dave Crane & Bear Bibeault & Jord Sonneveld (2007). “Ajax in Practice”. Manning Publications. ISBN 1-932394-99-0.

[29] Ali Mesbah (2009). “Analysis and Testing of Ajax-based Single-page Web Applications”. Delft University of Technology, Institute for Programming research and Algorithmics, Thesis. ISBN: 978-90-79982-02-8

[30] Thomas Nagele (2015). “Client-side performance profiling of JavaScript for web applications”. Radboud University, Computer Science, Master Thesis.

[31] Anne van Kesteren, Mozilla, Julian Aubourg, Creative Area, Jungkee Song, Samsung Electronics, Hallvord R. M. Steen (2014). ‘XMLhttpRequest Level 1'. World Wide Web Consortium.

[32] Leonard Richardson & Sam Ruby (2007). “RESTful Web Services”. O’Reilly Media. ISBN: 978-0-596-52926-0

[33] Estreamchat.com. Erişim Tarihi: 23.11.2014, http://www.estreamchat.com/

[34] EyeOs. Erişim Tarihi: 23.11.2014, http://www.eyeos.com/technology/architecture

[35] Ajaxwhois.org. Erişim Tarihi: 23.11.2014, http://www.ajaxwhois.org/

[36] Box.com. Erişim Tarihi: 23.11.2014, https://www.box.com/en_GB/home/

[37] Y.D.C.N. op’t Roodt, BICT (2006). 'The effect of Ajax on performance and usability in web environments', VU University Amsterdam, Software Engineering, Thesis.

[38] Danny C (2013). ‘Java WebSocket Programming’. (1.Baskı). McGraw-Hill Osborne, SBN-13: 978-0071827195, ISBN-10: 0071827196

[39] Günay Mert Karadoğan (2013). “Evaluating WebSocket and WebRTC in the Context of a Mobile Internet of Things Gateway”. KTH Information and Communication Technology, KTH Royal Institute of Technology, Master of Science Thesis.

[40] Oyvind Raasholm Tangen (2015). “Real-Time Web with WebSocket”. University of Oslo, Department of Informatics, Master's thesis.

[41] Google Inc & Ian F (2011). ‘The WebSocket Protocol’. Internet Engineering Task Force (IETF), ISSN: 2070-1721.

[42] Afshin Aresh. “Real-Time Voice Chat Realized in JavaScript”. Lulea University of Technology, Department of Computer Science, Electrical and Space Engineering, Master Thesis.

[43] Oleg Petsjonkin (2012), “Migration of Native Android Applications to HTML5”. University of Tartu, Institute of Computer Science, Master Thesis.

[44] Arman Farahzadeh (2013). “A Solution for a Common Front-End File System for Desktop and Smart Device Platforms”. Lulea University of Technology, Computer Science and Engineering, Master of Science, Thesis.

[45] James G & Bill J & Guy S & Gilad B & Alex B (2014). ‘Specification: JSR-337 Java SE 8 Release Contents’. Oracle America.

[46] Herbert Schildt (2007). “Java: The Complete Reference, Seventh Edition”. McGraw-Hill Companies. ISBN: 978-0-07-163177-8, MHID: 0-07-163177-1.

[47] Elizabeth Sugar Boese. An Introduction to Programming with Java Applets. Jones and Bartlett Publishers, Inc; 3rd Revised edition edition (24 April 2009). ISBN-10: 0763754609, ISBN-13: 978-0763754600.

[48] Chua Hock-Chuan (2012). “Applets & Web Start Applications - Java Rich Internet Applications (RIA)”. Nanyang Technological University, Java Programming Tutorial.

[49] Daniel W S (2010). ‘Preserving Unique References in Java Lists’. Virginia Polytechnic Institute and State University. Master of Science In Computer Science.

[50] Ahmed Aziz Khalifa (2014). “Generic Ownership Types for Java and the Collections Framework”. Victoria University of Wellington, Computer Science, Doctor of Philosophy Thesis.

[51] Haggai E (2012). 'A Study of Data Structures with a Deep Heap Shape'. Israel Institute of Technology, Computer Science, Master of Science.

[52] Daniel W S (2010). 'Preserving Unique References in Java Lists'. Virginia Polytechnic Institute and State University, Computer Science, Master of Science.

[53] Duane A. Bailey (2007). “Java Structures: Data Structures in Java for the Principled Programmer”. McGraw-Hill. ISBN 0071121633, 978007112163.

[54] Samir Hasan (2016). “Energy Profiles of Java Collections Classes”. Auburn University, Master of Science Thesis.

[55] Petr Tomanek (2016). “Java Collections Performance”. Czech Technical University, Department of Computer science, Bachelor’s thesis.

[56] Google, Erişim Tarihi: 10.12.2014, http://developer.android.com/reference/java/util/HashMap.html

[57] Apache. Erişim Tarihi: 23.11.2014, http://plugins.cordova.io/#/

[58] Apache (2004). Erişim Tarihi: 23.11.2014, http://www.apache.org/licenses/LICENSE-2.0

[59] Apache. Erişim Tarihi: 23.11.2014, https://git-wip-us.apache.org/repos/asf

[60] Cordova. Erişim Tarihi: 23.11.2014, http://plugins.cordova.io/#/package/org.apache.cordova.network-information

[61] Suresh C & Research In Motion (RIM) & Robin B & Robineko (2011). ‘The Network Information API’. The World Wide Web Consortium (W3C).

[62] Apache. Erişim Tarihi: 23.11.2014, https://git-wip-us.apache.org/repos/asf?p=cordova-plugin-network-information.git;a=tree

[63] GitHub. Erişim Tarihi: 21.01.2017, https://github.com/apache/cordova-plugin-console

[64] GitHub. Erişim Tarihi: 21.01.2017, https://github.com/apache/cordova-plugin-console/tree/master/src

[65] GitHub. Erişim Tarihi: 21.01.2017, https://github.com/apache/cordova-plugin-console/tree/master/src/ubuntu

[66] NPM Inc. Erişim Tarihi: 21.01.2017, https://www.npmjs.com/package/cordova-plugin-globalization

[67] Cong Xu (2012), ‘TCP/IP Implementation of Hadoop Acceleration’, Auburn University, Graduate Faculty, Master of Science.

[68] Baodong Jia (2010), ‘Data Acquisition in Hadoop System’, University of Stavanger, Computer Science, Master of Science.

[69] Shao Tao (2004), ‘A Smart TCP Socket for Distributed Computing’, National University of Singapore, School Of Computing, Bachelor of Science.

[70] Mozilla Developer Network. Erişim Tarihi: 23.11.2014, https://developer.mozilla.org/en-US/docs/WebSockets/Writing_WebSocket_client_applications

[71] Teemu Kontio (2015). “Cross-platform mobile development with PhoneGap”. JAMK University of Applied Sciences, Software Engineering, Bachelor’s thesis.

[72] Lars P & Mike S. ‘Visual Basic Programmer's Guide to the .Net Framework Class Library’. (1.Baskı). ISBN-10: 0-672-32232-3, ISBN-13: 978-0-672-32232-7

[73] Björn A. ‘Explanations of common Java exceptions’. Erişim Tarihi: 25.11.2014, http://rymden.nu/exceptions.html#ConnectException

6. ÖZGEÇMİŞ

1989 yılında İstanbul'da doğdu. İlk, orta ve lise öğrenimini İstanbul'da tamamladı. Lisansını Haliç Üniversitesi bilgisayar mühendisliği bölümünde yaptı. Lisans sonrası iş hayatına yazılım sektöründe devam etti. Yüksek lisans eğitimine Haliç Üniversitesi'nde Bilgisayar Mühendisliği bölümünde devam etmektedir.

HİBRİT UYGULAMALAR İÇİN SOKET ARAYÜZLERİ VE ANDROİD İMPLEMENTASYONU

ORIJINALLIK RAPORU

%4	%3	%1	%3
BENZERLIK ENDEKSI	İNTERNET KAYNAKLARI	YAYINLAR	ÖĞRENCI ÖDEVLERI

BIRINCIL KAYNAKLAR

1	Submitted to TechKnowledge Turkey Öğrenci Ödevi	%2
2	issuu.com İnternet Kaynağı	%1
3	Submitted to Haliç Üniversitesi Öğrenci Ödevi	%1
4	documents.tips İnternet Kaynağı	<%1
5	www.ibk.org.tr İnternet Kaynağı	<%1
6	docplayer.biz.tr İnternet Kaynağı	<%1
7	acikerisim.selcuk.edu.tr:8080 İnternet Kaynağı	<%1
8	cygm.meb.gov.tr İnternet Kaynağı	<%1
9	sawa-zen.com İnternet Kaynağı	<%1

Görüldü

Yrd. Doç. Dr. Ülviye Hacızade

12.05.2017