Linköpings Üniversitesi | Veri Koruma Kurumu
Yeni başlayan sınavlar, 30hp | Datateknik
2016 | LIU-IDA/LITH-EX-A--16/050--SE

React Native uygulama geliştirme

– Yerel Android ile React Native arasında bir karşılaştırma

William Danielsson

Yöneten: Anders Fröberg
İncelemeci: Erik Berglund

Linköpings universitet SE–
581 83 Linköping www.liu.se
013-28 10 00 ,

Soyut

Bir mobil uygulama oluşturmak çoğu zaman geliştiricilerin Android için bir tane oluşturmasını gerektirir.
biri iOS için, mobil cihazlar için önde gelen iki işletim sistemi. İki uygulama
aynı düzen ve mantığa sahip olabilir ancak kullanıcı arayüzünün (UI) çeşitli bileşenleri
farklılık gösterir ve uygulamaların iki farklı dilde geliştirilmesi gerekir.
Bu süreç, iki uygulama oluşturmanın zaman alıcı olması ve iki farklı bilgi seti gerektirmesi
nedeniyle ürkütücüdür. Teknikler yaratma girişimleri oldu,
Bu sorunu çözmek için hizmetler veya çerçeveler geliştirildi, ancak bu melezler bunu başaramadı
ortaya çıkan uygulamalara dair yerel bir his sağlamak.

Bu tez, yeni çıkan ve oluşturabilen React Native çerçevesini değerlendirmiştir.
React'ta yazılan kodu derleyerek hem iOS hem de Android uygulamalarını kullanabilirsiniz. Ortaya
çıkan uygulamalar kod paylaşabilir ve benzersiz UI bileşenlerinden oluşur.
her platform. Tez Android'e odaklandı ve mevcut bir Android'i kopyalamaya çalıştı
Kullanıcı deneyimini ve performansını ölçmek amacıyla yapılan uygulama. Sonuç React Native için
şaşırtıcı derecede olumluydu çünkü bazı kullanıcılar iki uygulamayı birbirinden ayıramadı ve
neredeyse tüm kullanıcılar React Native uygulamasını kullanmaktan çekinmedi. Performans
değerlendirmesinde GPU frekansı, CPU yükü, bellek kullanımı ve güç tüketimi ölçüldü. Neredeyse
tüm ölçümler Android uygulaması için bir performans avantajı gösterdi ancak
farklılıklar göze çarpmıyordu.

Genel deneyim, React Native'in mobil uygulamalar için geliştirme sürecini büyük ölçüde
basitleştirebilecek çok ilginç bir çerçeve olduğu yönündedir. Sürece
uygulamanın kendisi çok karmaşık değil, geliştirmesi karmaşık değil ve
çok kısa sürede uygulama oluşturup hem Android hem de iOS'a derlenebiliyor.

Öncelikle tezin tamamında kitaplar, araçlar ve bilgilerle bana yardımcı olan Valtech'e en derin şükranlarımı sunmak isterim. Bana çok yetkin iki danışman Alexander Lindholm ve Tomas Tunström'ü sağladılar, bu da benim fikirlerimden yola çıkmamı ve sonunda harika bir tez sahibi olmamı mümkün kıldı. Ayrıca bu süre zarfında birbirlerine ve bana destek olan ve en yorucu anlarda bile bunu eğlenceli hale getiren Talangprogrammet'teki diğer öğrencilere çok teşekkür ederim.

Ayrıca, son aylarda bana rehberlik eden ve makaleyle ilgili aydınlatıcı yorumlarda bulunan Linköpings Üniversitesi'ndeki sınav görevlisi Erik Berglund'a teşekkür etmek isterim.

Son olarak her zaman yanımda olan ve desteklerini esirgemeyen aileme ve özellikle hayattaki temel motivasyonum olan küçük kardeşime teşekkür etmek isterim.

İçindekiler

Şekil Listesi

1. Giriş

İlk akıllı telefonun piyasaya sürülmesinden bu yana mobil uygulamaların kullanımı ve talebi hızla arttı. Pek çok işletme uygulama üzerinden hizmet vermek suretiyle kurulmuştur ancak bazı işletmeler çağdaş olduklarını kanıtlamak için veya rakiplerinde uygulama olduğu için uygulama istemektedir. Bu, çok sayıda uygulamanın oluşturulmasına yol açtı ancak önemli bir sorun var; uygulamanın hem Android hem de iOS tarafından desteklenmesi gerekiyor. Uygulamanın kendisi aynı olmasına rağmen geliştiricilerin yine de gereksiz zaman ve beceri gerektiren iki uygulama geliştirmesi gerekiyor. 2015 yazında Facebook, React'ta bir uygulama oluşturmak ve ardından onu Android veya iOS'a derlemek için kullanılan React Native adlı bir çerçeve yayınladı.

1.1 Motivasyon

Bu proje ve tez, Yetenek programının bir parçası olarak Valtech İsveç ile işbirliği içinde gerçekleştirildi. Valtech İsveç'in müşterileri için web hizmetleri oluşturması çok yaygındır ve çoğu zaman müşteri de bir mobil uygulama istemektedir. Bu, Val-tech'in biri Android ve diğeri iOS için olmak üzere iki mobil uygulamayla birlikte bir web uygulaması geliştirmesiyle sonuçlanır. Valtech'in farklı alanlardaki geniş uzmanlık yelpazesi nedeniyle müşterilerine sunması ciddi bir sorun olmasa da bu hala bir sorundur. Bir müşteri iOS ve Android için bir web sitesi ve uygulama sahibi olmak istiyorsa, geliştiricilerin hem web geliştirme, iOS için programlama dili olan Swift hem de Android geliştirmede kullanılan Java konusunda bilgi sahibi olması gerekir. Bu büyük olasılıkla üç farklı ekiple sonuçlanacak, müşteri için maliyetli bir kurulum olacak ve Valtech'in tüm bu alanlarda danışmanlara sahip olması gerekecek. Valtech ve diğer geliştiriciler React Native'i kullanmaya başlarsa, üç hizmeti de aşağı yukarı aynı kodla oluşturabilecek tek bir ekibe ihtiyaçları olacak. Bu, geliştirme süresini büyük ölçüde kısaltacaktır ve ayrıca yalnızca React konusunda uzmanlığa sahip geliştiricileri işe almaları gerekecektir.

1.1.1 Valtek

Valtech, yaklaşık 1800 çalışanı bulunan küresel bir BT danışmanlık şirketidir ve 1993 yılında Fransa'da kurulmuştur. Valtech İsveç, Valtech'in İsveç'teki 258 çalışana sahip yan kuruluşudur ve ana ofisi Stokholm'dedir. Valtech, dijital bir ortak olmaya çabalıyor ve

yaratıcılık, teknik ve esnek bir ürün geliştirme stratejisi. Yetenek programı, eğitim içeren bir staj ve yeteneklerin hayatı gerçek bir danışman gibi deneyimlediği 8 haftalık bir proje sunuyor. Bu programın ardından öğrencilerin tezlerini Valtech'te yapmalarına izin verilmektedir.

1.2 Araştırma soruları

React Native'in 2015 yılında duyurulmasından bu yana, geliştiricilerin çerçeveyi kullanarak uygulamalar oluşturmasına yardımcı olmak amacıyla bazı blog gönderileri ve eğitimler çevrimiçi olarak yayınlanıyor. Ancak React Native ile ilgili makalelerin sayısı neredeyse yok denecek kadar az ve mevcut bilgilerin çoğu Facebook'tan geliyor. Bu tezin motivasyonu React Native'i değerlendirmek ve geliştirme, kullanıcı deneyimi ve performans açısından Android ile karşılaştırmaktır; bu da aşağıdaki sorulara yol açmaktadır:

1. Ne kadar zor ve React Native, Android uygulamasından ayırt edilemeyecek bir uygulama geliştirme desteğine sahip mi?
2. React Native'de oluşturulan bir uygulama, Android'deki yerel bir uygulamaya kıyasla ne kadar iyi performans gösteriyor?

İlk sorunun ilk kısmı oluşturulacak uygulamanın hayata geçirilmesi ve geliştirilmesine odaklanıyor. Bunun cevabı sayılarla değil, geliştirme aşamasında elde edilen deneyim ve geliştirme aşaması tamamlandığında ortaya çıkan sonuç ile cevaplanabilir. İkinci bölümde, bir uygulamayı React Native'de çoğaltma yeteneği ve uygulamanın, yerel bir uygulama ile aynı görünüm ve hissi elde etme konusunda ne kadar iyi performans gösterdiği değerlendirilecektir. Bu hususun değerlendirilmesi süreci bölüm 3.3.1'de daha ayrıntılı olarak açıklanmaktadır. İkinci soru, iki uygulamanın tamamlanmasının ardından farklı yönleri ölçülerek yanıtlanacaktır. Performansı ölçmenin birçok yolu olmasına rağmen bu tez, 3.3.2'de bahsedilenlere odaklanacaktır.

1.3 Amaç

Bu tezin amacı React Native çerçevesini ve çerçeve tarafından oluşturulan uygulamaların yerlilerle ne kadar iyi uyum sağladığını değerlendirmektir. Tez, React Native'de bir uygulama geliştirerek, geliştirme potansiyelini ve yeni başlayanlar için kendi uygulamalarını oluşturmanın basitliğini değerlendirecektir. Ayrıca bu makale, bir React Native uygulamasının performansını Android'deki benzer bir uygulamayla karşılaştırarak analiz edecektir. Bu iki hususun sonucu, React Native'in yatırım yapmaya değer bir çerçeve olup olmadığı veya Native yaklaşımda yazma uygulamasının yerini alamayacak olup olmadığı konusunda sağlam ve adil bir sonuca varılmasını sağlayacaktır.

1.4 Sınırlamalar

Bir çerçevenin tam bir değerlendirmesini yapmak, özellikle hem Android hem de iOS'ta derlenebilen React Native ile büyük bir görevdir. Sınırlı süre nedeniyle, React Native'deki iOS desteği şu anda daha eksiksiz olmasına rağmen, Android için geliştirme konusundaki önceki bilgilerden dolayı bu tez yalnızca Android'in karşılaştırılmasına odaklanacaktır. Ayrıca bir çerçevenin nasıl değerlendirilebileceğine dair birçok yön vardır; bunlardan bazıları 1.5'te belirtilmiştir, ancak bu tez 1.2'de listelenen iki soruya odaklanacaktır. Bu makale, mevcut bir Android uygulamasını yeniden oluşturmaya çalışarak React Native kullanarak bir uygulama geliştirmenin olanaklarını araştıracaktır. Android'de yeni bir uygulama geliştirmek için zaman harcamak yerine, halihazırda var olan bir uygulama elde edilecek. Elde edilen uygulama basit olacak, yeniden oluşturulabilecek makul boyutta olacak ve performansı optimize edilmeyecektir.

adil bir sonuç almak. Ayrıca iki uygulama performans açısından karşılaştırılacaktır. Bir uygulamanın performansını ölçmenin sayısız yolu vardır ancak bu tez, GPU frekansını, CPU yükünü, bellek kullanımını ve güç tüketimini Arnesson'a[2] benzer şekilde inceleyecektir.

1.5 İlgili Çalışma

Daha önce de belirtildiği gibi, çerçevenin neredeyse bir yıl önce yayınlanmasına rağmen React Native ile ilgili akademik raporların sayısı mevcut değil. Yeni bir çerçeve olduğu için konuyla ilgili kitap sıkıntısı olacağı anlaşılabilir ve çoğunlukla dokümantasyon ve blog yazıları aracılığıyla React Native ile ilgili eğitimler ve bilgiler edinilebilir. Ancak React Native'in kendisi ile ilgili daha önce yapılmış bir çalışma olmasa da hibrit framework değerlendirmenin benzerlikleri ve değerlendirmenin nasıl yapılacağı ile ilgili birçok makale bulunmaktadır.

Arnesson 2015 yılında bir makale yayınladı[2] ve yerel Android'i Codename One ve PhoneGap adlı iki hibrit çerçeveyle karşılaştırdı. Arnesson, üç farklı yöntemi kullanarak benzer bir uygulama oluşturdu ve üç uygulamanın performansını değerlendirdi. Uygulama farklı sıralama algoritmaları uyguladı, bir veritabanından yazıp okudu, bir liste oluşturdu ve bu listeyi sıraladı ve son olarak kullanıcının konumunu belirlemek için GPS'i kullandı. Tüm bu işlevler etkinliklere bölünmüştü ve Arnesson, uygulamanın farklı yönlerini ölçmek için PowerTutor[38] ve Trepn Profiler 5.1 araçlarını kullandı. Arnesson, CPU yükünü, bellek kullanımını, uygulama boyutunu, enerji tüketimini ve yürütme süresini ölçerek kapsamlı bir test gerçekleştirebildi ve adil ve eksiksiz bir sonuç elde etti.

Dahası, çerçeveleri karşılaştırmanın başka yolları da vardır ve Sommer, FURPS modelini kullanarak farklı çerçeveleri yerel çerçevelerle karşılaştırmıştır[14]. Somer, çerçevenin oluşturduğu uygulamayı ve çerçevenin kendisini beş özellik üzerinden değerlendirdi; İşlevsellik, Kullanılabilirlik, Güvenilirlik, Performans ve Desteklenebilirlik. Her özelliğe 1'den 5'e kadar bir derecelendirme verildi ve çerçevenin ortalama bir puan almasıyla sonuçlandı. Bu, bir çerçeveyi yalnızca performanstan daha fazlasına göre değerlendirmenin etkili bir yolu olsa da, değerler Sommer'in kendisi tarafından verilmektedir ve bu da taraflı bir sonuca yol açabilir.[33]

Mobil Sistemler için Hibrit ve Yerel Uygulamaların Deneysel Karşılaştırmasında[19] Seung-Ho Lim, bir sosyal ağ hizmeti oluşturarak ve kullanıcı arayüzüne ve cihaz özelliklerinin verimli kullanımına odaklanarak yerel bir uygulamayı bir web uygulamasıyla karşılaştırır. Kullanıcı arayüzünü değerlendirmek için Lim, kullanıcı arayüzünü oluşturmanın yanıt süresini ve verileri elde etmek ve yüklemek için kaç ağ işleminin yapıldığını araştırır. Ancak bir web uygulamasının kameraya veya pil seviyesine erişememesi nedeniyle Lim, hibrit versiyonun işlevselliğini bile kazanamaması nedeniyle cihazın yeteneklerinin ne kadar verimli olduğunu değerlendiremedi.

Son olarak Johansson ve Andersson makalelerinde farklı çerçeveleri birbirleriyle karşılaştırmaktadırlar[15]. Çerçeveler PhoneGap, Unity3D, GameMaker ve Qt'dir ve karşılaştırma, pil tüketimi ve 10 milyon kez döngü için gereken süre değerlendirilerek yapılır. Güç tüketimini ölçen araçlar açıklanmadı ancak pil tüketimini gözlemlemek için yalnızca telefonun kendi istatistiklerini gözlemledikleri görülüyor. Döngüyü gerçekleştirme süresi, işlemin manuel olarak zamanlanmasıyla yapıldı. Pil tüketimini ve performansını ölçmenin bu yolu oldukça eski modadır çünkü telefonun istatistikleri doğru bir sonuç vermeyebilir ve bir işlemin manuel olarak zamanlanması da mümkün olmayabilir.

1.6 Rapor Yapısı

Bölüm 1 tezin girişini içermekte olup, tezin motivasyonunu ve ilgili konularda benzer çalışmaların mevcut olduğunu anlatmaktadır. Bölüm 2, bu tezdeki alanlara ilişkin göreceli teoriden oluşmaktadır. Android hakkında daha spesifik ayrıntılar ve teori sağlayacak ve

bölüm 2.1'de gelişimini, bölüm 2.2'de platformlar arası geliştirmenin mevcut durumunu ve 2.3'te React'ın ne olduğunu ve nasıl kullanıldığını açıklayın. Ayrıca bölüm 2.4'te React Native'in nasıl çalıştığına dair gerekli ayrıntılar sağlanacaktır. Daha sonra bölüm 2.5, React Native uygulamasında kullanılan veritabanı çözümü olan Realm'i açıklamaktadır.

Ayrıca 3. bölümde ön çalışma ve çoğaltma için kullanılan uygulama anlatılarak çalışmanın nasıl yürütüldüğü anlatılacaktır. Bölüm 3.2'de React Native'deki uygulamanın nasıl geliştirildiği anlatılacaktır. Uygulamadaki temellerden kod örnekleri ve bazı sorunların nasıl çözüldüğü yer alacak. Bölüm 3.3, uygulamayı değerlendirmeye yönelik farklı yaklaşımları açıklayacaktır ve bölüm 3.4, bu tezde kullanılan araçları içermektedir. Daha sonra testlerden elde edilen verileri analiz etme prosedürü bölüm 3.5'te açıklanmaktadır.

Bölüm 4'te replikasyona ilişkin test sonuçları gösterilmekte ve ardından uygulamaların performansa ilişkin karşılaştırması sunulmaktadır. Son olarak, 5. bölümde sonuçlar ve geliştirme deneyimi tartışılmaktadır ve 6. bölümde bu tezin sonucu ve ileri çalışmalar için öneriler yer almaktadır.

2 Teori

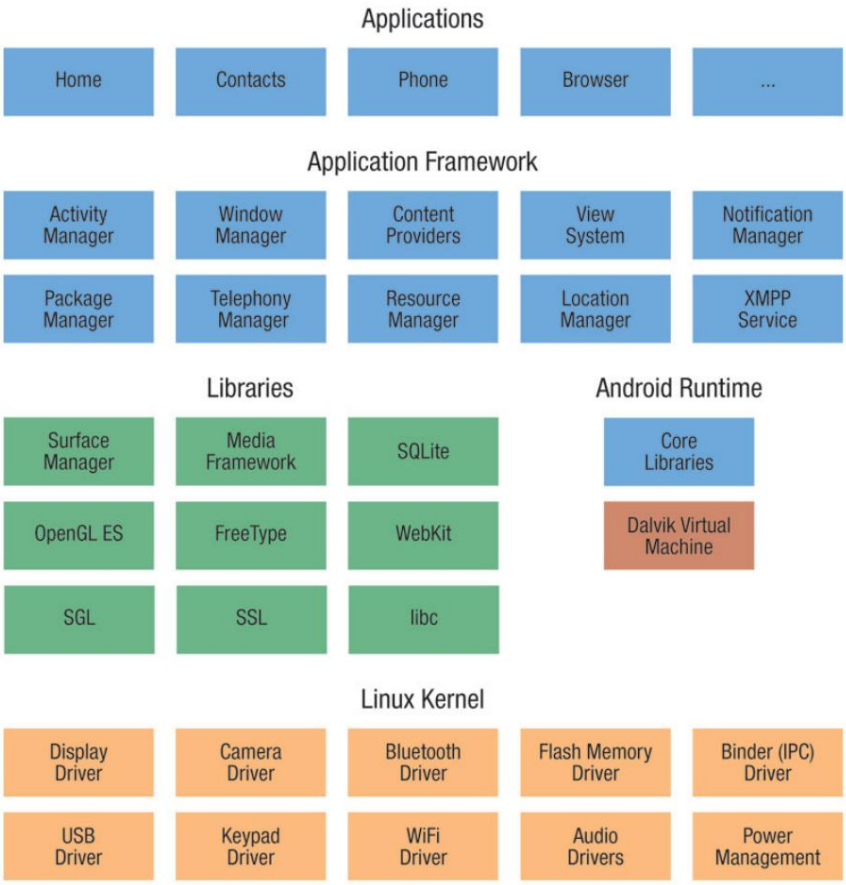
2.1 Android

Android, değiştirilmiş tek bir Linux çekirdeğini temel alan ve Google'a ait olan açık kaynaklı bir işletim sistemidir. Google, Open Hand-set Alliance (OHA) ile birlikte geliştirmeden sorumludur.[24] 2005 yılında Android Inc. tarafından geliştirilmeye başlandığında işletim sisteminin başlangıçta dijital kameralar için bir platform olduğu düşünülüyordu. Ancak Google şirketi satın aldı ve üç yıl sonra Android çalıştıran ilk akıllı telefon satıldı. 2015 yılında mobil pazarın yaklaşık %83'ü Android kullanıyor ve her ay 1,5 milyar1 uygulama Android kullanıcıları tarafından indiriliyor.[31]

2.1.1 Mimarlık

Android işletim sistemi bir yazılım bileşenleri yığınıdır ve şekil 2.1'de gösterildiği gibi Android mimarisi dört katmandan oluşur: Linux çekirdeği, Kitaplıklar ve Android çalışma zamanı, Uygulama çerçevesi ve Uygulamalar.

1http://www.idc.com/prodserv/smartphone-os-market-share.jsp



Şekil 2.1: Android Mimarisi[36]

Linux çekirdeği

Android, telefonda çalışacak şekilde değiştirilmiş bir Linux 2.6 çekirdeğini temel alır. Birçok işletim sistemi gibi dahili depolama, internet protokolü, süreç yönetimi, cihaz yönetimi ve diğer temel hizmetler gibi işlevler sunar. Android işletim sistemi bu katmanda cihazın donanımıyla etkileşime girer ve modüller ve sürücüler çoğunlukla C programlama dillerinde yazılır.[31][21][32]

Kitaplıklar ve Android çalışma zamanı

Linux çekirdek katmanının üstünde yerel kütüphaneler ve Android çalışma zamanı modülü bulunur. Yerel kütüphaneler uygulamaların kullandığı kütüphanelerdir ve uygulamaları Android'in sunduğu özelliklere bağlayan önemli bir bağlantıdır. Kitaplıklar C veya C++ dilinde yazılmıştır ve telefondaki farklı veri veya ses çeşitlerine erişmeyi sağlayan bir Java arayüzü aracılığıyla çağrılır.[31][24][3]

Android çalışma zamanı, bir dizi çekirdek kitaplığın ve ART'ın (Android Çalışma Zamanı) birleşimidir ve Dalvik Sanal Makinesinin yerini almıştır. Java Çekirdek Kitaplığı en temel işlevleri sağlar ve ART, uygulamaları çalıştırmak için kullanılır. Dalvik VM, değiştirilmiş bir Java Sanal Makinesiydi ve düşük güç ve bellek gereksinimlerine sahip olduğundan cep telefonlarında çalışacak şekilde kayıt tabanlı ve optimize edilmişti. Üstelik DVM, uygulamaların makinenin kendi örneğinde çalışmasına izin verdi. Bu, DVM'nin izolasyon, bellek yönetimi ve iş parçacığı desteği sağlayan sanal makinenin birden fazla örneğinin eşzamanlı olarak oluşturulmasına olanak sağlaması nedeniyle güvenilir bir sistem üretti. Dalvik Sanal Ma-

chine, .class ve .jar dosyalarının dönüşümü olan .dex dosyalarını çalıştırdı ve
daha düşük kaynaklar. Ancak Android 5.0'da DVM'nin yerini tamamen ART aldı.
artık yönetilen çalışma zamanıdır ve özellikle Android için oluşturulmuştur.[35] SANAT dönüşümleri
bytecode'u yerel talimatlara göre derler ve ardından çalışma zamanı ortamında yürütülür. Dalvik ile
aynı .dex dosyalarını kullanır ancak .odex dosyalarını Yürütülebilir ve
Bağlanabilir Format (ELF) yürütülebilir dosyaları. Bu ELF'ler, uygulama derlendiğinde yürütülen tek
dosyalardır; bu da farklı uygulama yürütme ek yüklerinin ortadan kaldırılmasıyla sonuçlanır.
ART, tüm uygulamayı makineye derleyen önceden hazırlanmış (AOH) derlemeyi kullanır
Uygulama yüklendiğinde kod. Bu, diğer şeylerin yanı sıra yürütme verimliliğini artırır,
bellek ayırma ve güç tüketimini azaltır.[31][21][1]

Uygulama çerçevesi

Uygulama çerçevesi, birleştirilmiş yerel kütüphanelerin soyutlamalarını sunar
Dalvik'in yetenekleriyle. Ayrıca uygulama programlama arayüzlerini sağlar
(API'ler) ve Java sınıfları biçimindeki diğer üst düzey hizmetler. Önemli bir blok
uygulama çerçevesi katmanı, uygulamaların yaşam döngüsünü yöneten Etkinlik Yöneticisidir.[31][21]

Uygulamalar

Uygulamalar katmanı mimaride en üst katmandır ve uygulamaların yer aldığı katmandır. Uygulamalar
önceden yüklenebilir ve temel işlevleri sağlayabilir
telefon görüşmeleri yapmak veya internette gezinmek için telefonun ancak uygulama katmanının da
indirilenleri veya geliştirilmekte olanları yönetir. Bu uygulamalar Java'da yazılır ve daha sonra akıllı
telefona kurulmak üzere makine kodu halinde derlenir.[31][3][32]

2.1.2 Geliştirme

Aşağıdaki gibi entegre bir geliştirme ortamında (IDE) yeni bir Android projesi oluşturmak:
Android Studio veya Eclipse, Android SDK'nın geliştiricinin işi yapılandırmasına yardımcı olduğu ve
Android geliştiricileri için zorunlu olduğu temel bir uygulama sağlar. Android SDK'sı
Geliştiricinin ihtiyaç duyduğu tüm paketleri, uygulama çerçevesini ve sınıf kitaplıklarını içeren Java
Android Kitaplığı adı verilen Java programlama dilinden yararlanır.
Android uygulaması oluşturmak için sipariş verin. Sözdizimi orijinal Java ile aynıdır.
işlenenlere, yinelemelere ve seçimlere gelir ancak bazı belirli Android sınıfları vardır ve
Activity ve View Class gibi paketler.[32][22][3]

Bir Android projesi oluşturulduğunda geliştirici, Android SDK'yı ve geliştirme ortamını kullanarak
Java kodunu derleyebilir. Bu bir başvuruyla sonuçlanacak
kod bileşiklerinin sıkıştırılmış bir koleksiyonu olan bir Android Paketinden (APK) oluşur
Uygulama gerçek veya sanal bir cihaza kurulabilir ve çalıştırılabilir. Her
Android sürümü, yapılandırılabilen farklı bir Android Sanal Aygıtına (AVD) sahiptir ve
IDE'de başlatıldı. AVD, belirli akıllı telefon işletim sistemini içeren bir emülatördür ve
geliştiricinin bir Android cihazına sahip olmaması durumunda kullanışlıdır.[32][22][24]

Android'deki kullanıcı arayüzünün tasarımı ve uygulanması, yerel Java kullanıcı arayüzü ile benzer
kullanıcı arayüzü bileşenlerini ve kavramlarını kullanır ancak bazı farklılıklar vardır. Tüm kullanıcı arayüzü
Android'deki bileşenler Görünümler üzerine kuruludur ve uygulamalardaki kullanıcı arayüzü, kullanıcı
arayüzünü temsil eden Etkinliklerden oluşur. Etkinlik birçok Java bileşeni içerir ve
stillendirme, HTML kullanarak bir web sayfası tasarlarken olduğu gibi XML yoluyla yapılır
ve CSS.[19][24]

2.2 Platformlar Arası Geliştirme

Son yıllarda platformlar arası uygulama oluşturmaya yönelik çözüm sağlamak amacıyla farklı teknikler ve çerçeveler ortaya çıktı. Amaç, erişilebilen veya farklı işletim sistemlerine dağıtılabilen, homojen ve doğal bir his sağlayan tek bir hizmet geliştirmektir.

Popüler tekniklerden biri, mobil cihazların tarayıcıları aracılığıyla erişebileceği web üzerinde bir uygulama olan duyarlı bir web sitesi oluşturmaktır. HTML5 ve CSS3'teki özelliklerin popüler ön uç kütüphanesi Bootstrap[5] ile birlikte kullanılmasıyla web uygulaması, bilgisayar veya mobil cihazla kullanımı kolay bir kullanıcı arayüzüne sahip olacaktır. Kullanıcı arayüzü, bileşenlerin boyutları ve düzeni, erişim sağlayan cihazın ekranının yüksekliği ve genişliğine göre şekillendirildiğinden duyarlıdır. Bu, bir bilgisayar tarafından erişildiğinde sıradan bir web sitesi gibi bir kullanıcı arayüzüne sahip bir web uygulamasıyla sonuçlanır, ancak mobil bir cihazdan erişildiğinde bir uygulamaya daha çok benzer.[11] [17]

Bir alternatif, herhangi bir işletim sistemi tarafından kullanılabilecek bir uygulama oluşturmak için hibrit bir çerçeve kullanmaktır. Bu yaklaşım, uygulama web teknikleri kullanılarak oluşturulduğundan ancak WebView[7] kullanılarak yerel bir uygulama olarak yürütüldüğünden, işlendiğinden ve görüntülendiğinden, web ve yerel geliştirmeyi kullanmanın bir kombinasyonudur. Cihazın yetenekleri, hibrit uygulamanın cihazın işlevlerine ve özelliklerine erişmesine olanak tanıyan bir JavaScript Uygulama Programlama Arayüzü olarak bir soyutlama katmanı tarafından ortaya çıkarılır. Ancak hibrit uygulamaların geliştirme maliyeti native uygulamalara göre oldukça düşük olsa da hibritler aynı native kullanıcı deneyimini sağlayamamakta ve bu nedenle de native geliştirmenin yerini alma konusunda başarılı olamamaktadır.[16][15]

2.3 Tepki

Bazen React.js olarak da anılan React, geliştirme topluluğunun arayüzler oluşturmasına yardımcı olmak amacıyla Facebook tarafından geliştirilen ve 2013 yılında açık kaynak olarak piyasaya sürülen bir JavaScript çerçevesidir. Facebook, zamanla değişen verilere sahip karmaşık kullanıcı arayüzleriyle ilgili sorunlarını çözebilecek bir çerçeveye ihtiyaç duyuyordu. React, geliştirme paradigması model-görünüm-denetleyicinin (MVC) "görünüm" kısmını sağlar ve MVC'de V'ye hizmet eden bir çerçeve olarak React'ın yalnızca müşteri tarafında çalıştığına inanılabilir. Bununla birlikte, sunucu tarafında da oluşturulabilir ve bu da iki taraf arasında birlikte çalışabilir bir iletişim sağlar. Facebook'ta mühendis olan Tom Occhino şunları söyledi: "React, zorunlu bir API'yi bildirimsel bir API ile tamamlıyor. React'ın gerçek gücü, size nasıl kod yazdırdığından kaynaklanmaktadır".[26] Bildirimsel programlama stili ne yapılacağını açıklar ancak nasıl yapılması gerektiğini açıklamaz, bu da daha az kodla sonuçlanırken, zorunlu programlama stili bunun nasıl yapılacağını açıklar.[12] [10]

2.3.1 Sanal DOM

Bir web sayfası bir web tarayıcısına yüklendiğinde, o web sayfasını içeren bir Belge Nesne Modeli (DOM) oluşturulur. DOM, ağaç yapısı biçiminde bir temsildir ve geçerli web sayfasının yapısını ve durumunu HTML öğelerini kullanarak görüntüler. Web sayfasında bir eylem gerçekleştirildiğinde, örneğin kullanıcı başka bir sayfaya gittiğinde veya uygulama bir sunucudan veri aldığında, yeni sayfa ya yeniden oluşturulur ya da DOM'un içeriği JavaScript tarafından manipüle edilir. İkincisi, tek sayfalık bir uygulama (SPA) kullanıldığında yapılır, ancak DOM manipülasyonu pahalıdır ve bir mantra haline gelmiştir. Kod tabanının bakımı zorlaşacak ve hız için optimize edilemedikleri için mutasyonlar yavaşlayacak. Uygulamaların kullanıcı arayüzü durumu ile veri modelinin durumu arasındaki senkronizasyonu almak için iki yönlü bir veri bağlama kullanılabilir. Bu, Knockout.js ve iOS tarafından kullanılan anahtar-değer gözlemleme kullanılarak ve diğeri ise Google'ın Angular çerçevesinin kullandığı kirli kontrol kullanılarak gerçekleştirilebilir. Bu, hangi verilerin değiştiğine bakmak ve DOM'u güncel tutmak için zorunlu olarak değişiklikler yapmak amacıyla bir bağlantı işlevi gerektirecektir.[10] [27] [18]

React, iki yönlü veri bağlama yerine Sanal DOM adı verilen bir konsepti kullanır. Sanal DOM, geçerli duruma göre DOM'daki düğümlerin alt ağaçlarını seçici olarak oluşturur, bu da sayfayı güncel tutmak için gereken minimum miktarda manipülasyonla sonuçlanır.
Sanal DOM, gerçek DOM'un bir temsilidir ve JavaScript ile DOM'u reaktifmiş gibi ele alma yeteneğini veren bir soyutlamadır. React, uygulamanın durumunu dahili olarak saklayacak ve DOM manipülasyonunu yalnızca durum değiştiğinde gerçekleştirecektir. Veri modelinin durumu değiştiğinde sanal DOM ve React, kullanıcı arayüzünü sanal bir DOM temsiline yeniden dönüştürecektir. Daha sonra React, farklılıkları ve neyin manipüle edilmesi gerektiğini belirlemek için iki sanal DOM'u birbiriyle karşılaştıracaktır. Bu farklılıklar bundan sonra güncellenir ve gerçek DOM'a dönüştürülür. Bu sürece uzlaşma denir ve tamamı bileşenlerin oluşturulmasından kaynaklanır.
Bileşen ilk kez başlatıldığında render yöntemi çağrılır ve bir işaretleme dizisi üretilip DOM'a enjekte edilir. Veriler değiştiğinde render yöntemi bir kez daha çağrılır. Geri dönüş değerinin önceki ve yeni çağrıdan farkı, uygulanması gereken minimum değişiklik kümesidir.[10][27]

2.3.2 Bileşenler ve JSX React

genellikle bir kullanıcı arayüzü kitaplığı olarak görülür ve etkileşimli, yeniden kullanılabilen ve durum bilgisi olan kullanıcı arayüzü bileşenlerinin oluşturulmasını hızlandırır. Bu bileşenler React'ın yapı taşlarıdır ve özellik ve durum parametrelerine sahip işlevlere benzerler; bunlar daha sonra açıklanacaktır. Bileşenlerin oluşturulması düz JavaScript veya JSX kullanılarak yapılabilir. JSX, XML tabanlı nesne gösterimi için bir JavaScript sözdizimi uzantısıdır ve okunması kolay büyük ağaçlar oluşturma avantajına sahiptir. Ayrıca React, dizeler halinde HTML etiketleri veya sınıflar biçiminde React bileşenleri oluşturabilir.[27]
Bir bileşeni web sayfasına dönüştürmenin sözdizimi basittir:

```
var divStyle = {renk: mavi};

React.render( <h1
    style={divStyle}>Merhaba dünya!</h1>, document.getElementById('myDiv') );
```

Render yönteminin ilk argümanı render edilmesi gereken bileşen, ikinci argümanı ise hangi DOM düğümüne enjekte edilmesi gerektiğidir. Yukarıdaki kod, mavi metin renginde h1 türünde bir başlık oluşturacak ve bunu "myDiv" kimliğine sahip bölüme ekleyecektir. Ancak, daha sonra oluşturulabilecek özel bir bileşenin ilk olarak oluşturulması için createClass adı verilen başka bir yöntem kullanılabilir. Ayrıca bileşenlere props adı verilen öznitelikler eklenebilir ve dinamik verilerin sunulması amacıyla render yönteminde kullanılabilir.[27]

```
var MyComponent = React.createClass({
    render: function(){ return
        ( <h1>Selamlar,
            {this.props.name}!</h1> );


} });
```

Bu bileşen artık render yöntemi çağrılarak DOM'da oluşturulabilir:

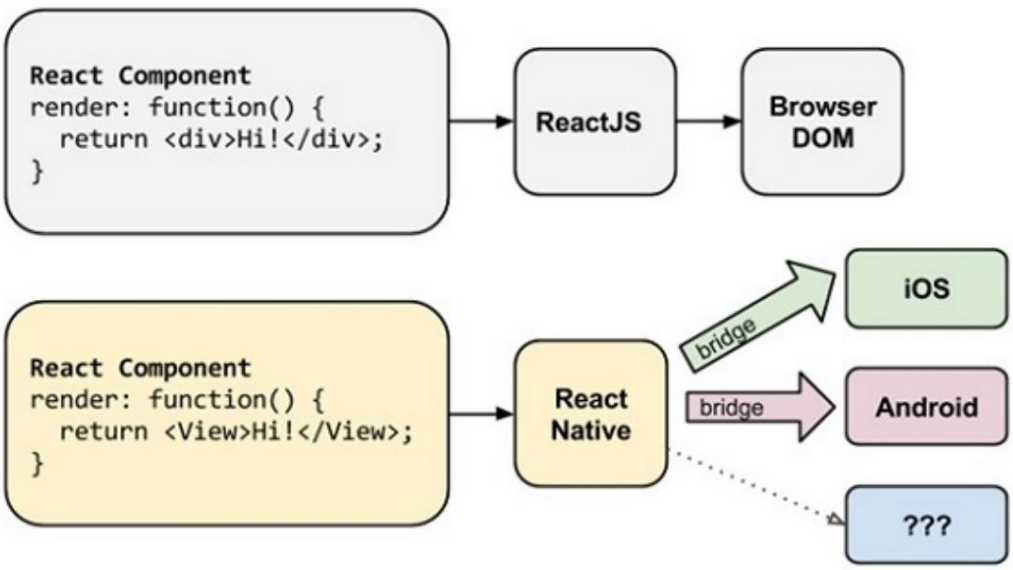
```
React.render(
    <MyComponent name="Kullanıcı"/>,
    document.getElementById('myDiv') );
```

Durumlar, React'in etkileşiminin temelidir ve verileri bileşenler halinde birleştirmek için kullanılır. Bu, verileri daha sonra yeniden oluşturulan bir bileşene aktaran setState yöntemi kullanılarak yapılır.[10][27]

2.4 Yerel Tepki

2015'teki React.js konferansında Facebook, yeni çerçevesi React Native'i tanıttı.
mobil uygulamaların oluşturulma biçiminde devrim yaratacağını düşündükleri çerçeve. Ne zaman
React Native piyasaya sürüldü, yalnızca iOS için destek vardı ancak o zamandan beri destek
Android eklendi ve genişlemeye devam ediyor. Facebook daha açık kaynak olmaya başladı ve React Native
için seçtikleri yaklaşım bu. Kaynak olmasa da
Henüz tamamen açık olmasına rağmen Facebook bunu başarmaya çalışıyor ve topluluğun
çerçevenin iyileştirilmesine katkıda bulunacaktır.

React Native'in temel amacı basittir; bir geliştiricinin mobil uygulama oluşturmak için bilgiye ihtiyaç duymaması veya gereksiz zaman harcaması gerekmemelidir, çünkü en azından
Hem iOS hem de Android'i desteklemek için iki uygulamanın geliştirilmesi gerekiyor. Farklı platformlar
farklı görünüm, his ve yeteneklere sahip olduğundan bir uygulama olamaz.
tüm işletim sistemlerinde homojendir. Ancak farklılık gösteren grafiksel arayüz olduğundan, geliştirme
aynı dili temel alabilir ancak grafiklere sahip olabilir.
hedeflenen platforma bağlı olarak farklı şekilde oluşturulabilir ve gerçek yerel bileşenler olabilir.
Facebook, React Native'in neyle ilgili olduğunu açıklayan bu yaklaşımı "bir kez öğren, her yere yaz" olarak adlandırıyor. React Native'in teknolojisi bölüm 2.3'te açıklanan React'a dayanmaktadır.
ve React'ın avantajları, onu yerel uygulamalara uygulayan çerçeveye aktarılır. Tarayıcıda React'ı çalıştırmak ve görülebilen div'leri ve metinleri işlemek yerine
bölüm 2.3.2'de, React Native, yerleşik bir JavaScriptCore (iOS) veya V8 örneğinde çalışır
(Android) uygulamaların içinde yer alır ve daha yüksek düzeydeki platforma özgü bileşenlere dönüştürülür.
JavaScript bileşenleri, iOS veya tarafından desteklenen bir dizi yerleşik temel öğe kullanılarak bildirilir.
Android bileşenleri.[25][37]



Şekil 2.2: React ve React Native'de İşleme [9]

Geliştirme topluluğuna hibrit bir çözüm sağlamak için birçok girişimde bulunuldu.
çerçevesine sahiptir ancak hiçbiri React Native tarafından sunulan özelliklerin tamamına sahip değildir. çoğunun aksine
Diğer mobil platformlar arası geliştirme yaklaşımlarına göre React Native hibrit bir çözüm değildir.
Phonegap, Titanium ve Ionic'ten farklı olarak React Native, Web'in içinde görüntü oluşturmaya dayanmaz.

HTML ve CSS'yi görüntüler veya taklit etmeye çalışır. Düzen ayrı bir iş parçacığı üzerinde gerçekleştirilir, bu da ana iş parçacığının animasyonları rahatsız edilmeden gerçekleştirebilmesini sağlar. React Native'deki bir uygulama WebView'ları kullanmadığından, yerel yanıt verme özelliğine sahip uygulamalar oluşturmak mümkündür. Üstelik tek bir kod tabanını tüm platformlara itmek yerine, ortak bir kod tabanı yazıp platformlara dağıtılıyor ve hedeflenen her platform için yalnızca grafiksel öğeler ve platforma özgü bileşenler olarak bazı bölümler ayrı ayrı yazılıyor.[25] [28]

2.4.1 React Native'in özü

Daha önce belirtildiği gibi React Native, React Native bileşenlerini Android için gerçek yerel Görünümlere veya iOS için UI Görünümlerine dönüştürebilir. Bu, React Native'in Android için Java'da veya iOS için Objective-C'de oluşturma API'lerini çağırmasını sağlayan "köprü" olarak bilinen soyutlama katmanı nedeniyle mümkündür. Ayrıca çerçeve, JavaScript arayüzünü ortaya çıkararak uygulamanın pil veya konum gibi platforma özgü özelliklere erişmesine olanak tanır. Bölüm 2.4, React Native'de kullanılan farklı iş parçacıklarından bahsetmişti ve aslında React Native'in dayandığı üç ana iş parçacığı vardır: düzenin işlendiği gölge kuyruğu; kullanıcı arayüzü oluşturmanın gerçekleştirildiği ana iş parçacığı; betiklerin çalıştığı JavaScript iş parçacığı. Bu iş parçacıkları, uygulamadaki farklı olayların işlenmesinden sorumludur.[20][8]

2.4.2 Özellikler

React Native'in birkaç özelliği makalede daha önce kısaca tanıtılmıştı ancak React Native'in, onu geliştirme topluluğu için daha da çekici kılan çeşitli özellikleri var. Yerel kullanıcı arayüzü bileşenlerinin oluşturulması ve oluşturulmasına ilişkin açıklamalar daha sonra bölüm 2.4.3'te açıklanacaktır.

Her şeyden önce, React Native, uygulamadaki JavaScript kodu ile yerel platform arasındaki işlemlerin eşzamansız yürütülmesini destekler ancak aynı zamanda yerel modüllerde iş parçacığına da izin verir. Bu, kullanıcı arayüzünü engellemeden arka planda birçok farklı işlemin gerçekleştirilmesine olanak tanır. Ayrıca geliştiricinin uygulamayı çalıştırırken örneğin Chrome Geliştirici Araçlarını kullanarak kodda hata ayıklamasına da olanak tanır.

İkinci olarak React Native, dokunmaları üst düzey özelliklere sahip karmaşık bir görünümde ele alan bir sistem uygulayarak ekran etkileşimini yönetir. Hareket tanıma, mobil bir cihazda web'e göre daha gelişmiş olduğundan, kaydırma, dokunma, kaydırma vb. gibi dokunmayla yorumlanabilecek birçok farklı eylem vardır. Üstelik birden fazla dokunuş aynı anda gerçekleştirilebiliyor. Kullanıcılar, bir web uygulaması ile yerel bir uygulama arasındaki farkı fark eder; çünkü her dokunuş, yayınlandığında ne olacağını görüntülemeli ve kullanıcı, parmağını uzağa sürükleyerek eylemi iptal edebilmelidir. React Native, ek yapılandırma gerektirmeden kaydırmalar ve diğer öğelerle düzgün bir şekilde asimile olan soyut bir Touchable ve TouchableHighlight uygulaması ekleyerek bu sorunu çözmüştür.

Ayrıca, React'a benzer şekilde React Native, yapıyı veya kodu karmaşıklaştırmadan bileşenleri oluşturmak ve stillendirmek için bölüm 2.3.2'de açıklanan JSX'i ve satır içi stillendirmeyi kullanır. React'ta stil bir nesne olarak belirtilir ve bölüm 2.3.2'deki kodda da görülebileceği gibi HTML etiketinin içindeki öğeye eklenir. Ancak React Native, web'den Flexbox adlı bir düzen modelini kullanır. Flexbox, ortak kullanıcı arayüzü düzenleri oluşturma sürecini basitleştirir ve StyleSheet soyutlaması, bileşen satır içi ile birlikte düzeni ve stili bildirme olanağı sunar. Flexbox'ın kullanımı şekil 2.6'da görülebilir.

Son olarak React Native, bölüm 2.3 ve 2.3.2'de açıklanan bileşenlerin yapı taşları olduğu React'tan ilham almıştır. Uygulama yerel modüller ve kullanıcı arayüzü bileşenleri tarafından oluşturulduğundan benzer yaklaşım React Native için de geçerlidir. React Native, ortak platform API'leri ve yerel kullanıcı arayüzü bileşenleri için desteğin yanı sıra, ihtiyaç duyulması halinde özel yerel modüller ve görünümler oluşturma desteğine de sahiptir.[28] [8] [9]

2.4.3 Geliştirme

React Native'i yüklemek için Homebrew'un kurulması gerekir ve daha sonra Node.js'yi yüklemek için kullanılır. Geliştirici, Node'a ek olan düğüm paketi yöneticisini (npm) kullanarak React Native'i komut satırıyla kurabilir. Android'de bir React Native uygulamasını çalıştırmak için, Java JDK ve Android SDK ile birlikte Android SDK Build-Tools 23.0.1'in bilgisayara yüklenmesi gerekir. React Native yüklendiğinde yeni bir React Native uygulaması oluşturulabilir ve tekrar yeni bir komut girilerek çalıştırılabilir. Komutların adımları şekil 2.3'te görülmektedir.[28]

```
$ /usr/bin/Ruby -e "$(curl -fsSL https://
raw.githubusercontent.com/Homebrew/install/master/install)"

$ demleme kurulum düğümü

$ npm install -g tepki-yerel-cli

$ react-native init TestProject

$ cd TestProjesi

$ tepki-yerel çalıştırma-android
```

Şekil 2.3: React Native projesi kurma ve oluşturma komutları

Yeni oluşturulan TestProject projesi şunları içerir:

- package.json - Proje veya proje bağımlılığı için ilgili meta verileri içeren bir dosya inkarlar.
- node_modules/ - Projeyi kontrol eden bağımlılıkları ve CLI aracını (node_modules/react-native/local-cli/cli.js) içerir. Bu komut dosyası, kazan plakası kodunu birleştiren başka bir yardımcı komut dosyası node_modules/react-native/init.sh'yi çalıştırır.
- index.android.js - Projenin React Native ana dosyası.
- android/ - Android'e özgü kodu içerir ve index.android.js'deki kodla birlikte çalıştırılarak React Native uygulamasını oluşturur.
- iOS/ ve index.ios.js - Yukarıda belirtilen iki dosya ve klasörün iOS sürümü.

Belirtildiği gibi index.android.js dosyası kodun ana dosyasıdır ve baştan itibaren temel işlevleri içerir. Öncelikle ithalatlar listelenir. Bu, uygulamanın React Native olduğunu belirtir ve gereken içe aktarmaları bileşenler, stil sayfaları, resimler vb. olarak listeler. Bu içe aktarmalar daha sonra uygulamada oluşturulabilecek bileşenleri oluşturmak için kullanılabilir. Geliştiricinin daha fazla içe aktarmaya ihtiyacı varsa bunlar içe aktarma nesnesine eklenebilir.

```
React'ı içe aktar, {
    Uygulama Kayıt Defteri,
    Bileşen,
    Stil Sayfası,
    resim,
    Metin,
    Görüş
} 'tepki-yerel'den;
```

Şekil 2.4: React Native'de kullanılan içe aktarma nesnesi örneği

Daha önce de belirtildiği gibi, bir React Native uygulaması, oluşturmadan sorumlu olan ve kullanıcıyı otomatik olarak güncelleyen JavaScript nesneleri biçimindeki React bileşenleri kullanılarak oluşturulur.

arayüz. Oluşturulan ilk bileşen, uygulamanın ilk ekranı olan özel sınıf bileşenidir. Bu bileşen, JSX'te yazılmış düzeni içeren oluşturma işlevini içerir. Şekil 2.5'teki kodda, kapsayıcı olarak bir Görünüm oluşturulur ve iki metin öğesini ve bir görüntüyü kapsar. Metinlerin içeriği ve görüntünün kaynağı MOCKED_DATA dizisindeki nesneden alınır.

```
var MOCKED_DATA = [{ name:
    'William', doğum tarihi:
    '1992', resimler:
    {thumbnail: 'http://goo.gl/W06A0l'} }];


class TestProject extends Bileşen { render() { var data =
    MOCKED_DATA[0];
        return ( <View style={styles.container}>


                <Text style={styles.nameText}>{data.name</Text> <Text style={styles.bornText}
                >{data.born</Text> <Resim stili={styles.image}

                    kaynak={{uri: data.images.thumbnail}} />
            </View> );

    }
}
```

Şekil 2.5: React Native'de bileşen oluşturulması

Yukarıdaki kodda bileşenler, Flexbox kullanan bir React.StyleSheet örneğindeki JSON nesnesine atıfta bulunan bir stil özelliğine sahiptir. Şekil 2.5'teki bileşenlerin stili şekil 2.6'da gösterilmektedir. Kap, aynı kaptaki diğer esnek öğelere göre öğenin uzunluğunu belirten flex özelliğine sahiptir. Üstelik alt öğelerini merkeze hizalayarak çok soluk mavi bir arka plan rengi veriyor. Metinlere farklı yazı tipi boyutları, renkler ve kenar boşlukları verilir ve son olarak görsele belirli bir yükseklik ve genişlik verilir.

```
const stilleri = StyleSheet.create({ kapsayıcı: { flex: 1,
    justifyContent:
        'merkez',
        alignItems: 'merkez', arka planRenk:
        '#F5FCFF', }, nameText: { fontSize:
        20, kenar boşluğu: 10, },bornText : { renk:

    '#333333', kenar
        boşluğuBottom: 5, },
        resim: { genişlik:

    53, yükseklik:
        81, }, });
```

Şekil 2.6: Flexbox kullanılarak bileşen stilini içeren JSON nesnesi

Son olarak React Native için yeni oluşturulan bileşenin uygulama başlatılırken render edilmesi gerektiğini belirtmek için bileşen AppReg-istry.registerComponent fonksiyonuna kaydedilir. Bu, ECMAScript 2015 (ES6) spesifikasyonuyla ve olmadan şekil 2.7'de gösterilmektedir.

```
// ES6
AppRegistry.registerComponent('TestProject', () => TestProject);

// Geleneksel
AppRegistry.registerComponent('TestProject', function() {
    TestProject'i döndür; });
```

Şekil 2.7: Uygulama için bileşenin kaydedilmesi

Şekil 2.3'teki komutla kod çalıştırıldığında uygulama derlenecek ve APK cihaza yüklenecektir. Yukarıdaki kodun sonucu şekil 2.8'de gösterilmektedir ve bir değişiklik yapıldığında geliştirici, uygulamadaki güncellemeleri almak için cihazı sallayıp JavaScript'i yeniden yükleyebilir.

Şekil 2.8: Bölüm 2.4.3'teki kodun sonucu

2.5 Bölge

Realm, özellikle mobil uygulamalar için tasarlanmış, platformlar arası bir veritabanı çözümüdür.
Veritabanının kullanımı basit, çok hafif ve SQLite veya Core Data'dan daha hızlı olduğu söyleniyor. Realm beş farklı ortamda kullanılabiliyor ve bunlardan biri React Native.

Realm React Native, verileri anında modellemek, depolamak ve sorgulamak için özel olarak JavaScript için tasarlanmıştır ve canlı nesnelerle reaktif uygulama geliştirmeyi mümkün kılmak için tasarlanmıştır. Şemalar tanımlanır ve başlatma sırasında Realm'e iletilir ve bir nesnenin veri modelini içerir. Nesnenin bir adı ve bir ad, bir tür ve bir türden oluşan bir dizi özelliği vardır.

Varsayılan değer için isteğe bağlı alan. Kullanılacak şemanın adını ve depolanacak verileri parametre olarak alan realm.create yöntemi çağrılarak yeni bir nesne oluşturulabilir. realm.objects yöntemi, nesneleri almak için çağrılır ve elde edilecek şemanın adını gerektirir. Ayrıca sorgulanan nesne üzerinde istenilen nesnelerin elde edilmesi için filtreleme yöntemi kullanılmaktadır. Şemaların oluşturulması bölüm 3.2.5'te şekil 3.1'de görülmektedir.[29]

2.6 Anket değerlendirmesi

Bir sistemi kullanıcı deneyimi yoluyla değerlendirmek, sistemin kullanıcı bakış açısından ne kadar iyi performans gösterdiğine ilişkin bilgi edinmek için sıklıkla kullanılan bir yaklaşımdır. Kullanılabilirlik değerlendirmesi verimliliğe odaklanır ve sistemin günlüğe kaydedilmesi ve hatalar ve tıklama sayıları şeklinde veriler elde edilmesiyle değerlendirilir; kullanıcı deneyimi ise kullanıcının sistem hakkında ne hissettiğine dair bilgi sağlar. Bir sistemi değerlendirmek için çeşitli yaklaşımlar vardır. Sistemin gerçek hayatta kullanımını incelemek için kullanıcının keşif testi yaptığı, yani kullanıcının rehberlik olmadan sistemi keşfetmesine izin verildiği ve iletkenin gözlemci olarak hareket ettiği saha çalışmaları yapılabilir. Ayrıca anketler sistem değerlendirmesi için popüler bir yaklaşımdır ve diğer yöntemleri tamamlayıcı olarak kullanılabilir.

Anket, kullanıcılardan veri toplamanın hızlı ve basit bir yoludur ve kullanıcılar genellikle anonim olabildikleri için yanıtlar konusunda daha dürüst olurlar. Üstelik anketin çevrimiçi olarak yapılması genellikle daha geniş bir yayılma ve dolayısıyla daha fazla yanıtla sonuçlanır. Kullanıcı tarafından gerçekleştirilen keşif testleri, sistem hakkında sorular içeren bir anketle birleştirilerek, kullanıcının genel olarak sisteme ilişkin görüşleri kolaylıkla elde edilebilir. Ancak demografik ve katılımcı sayısı anketin önemli bir parçasıdır, çünkü sonuçlar ve çıkarılacak sonuç üzerinde çok büyük bir etkiye sahiptir. Genellikle hedef grup veya uzman değerlendirmesi olmak üzere iki farklı kullanıcı grubu kullanılmaktadır. Hedef grup, tamamlandığında sistemi kullanma olasılığı yüksek olan kullanıcılardan oluşur. Bunlar, son kullanıcıların düşüncelerini yansıtan gerçekçi görüşler sağlar ancak çoğu zaman daha fazla zaman alır ve pahalıdır. Bu nedenle, kullanıcı deneyimine ilişkin erken dönemde eleştirel bir bakış elde etmek ve daha sonra son kullanıcılardan gelecek olumsuz değerlendirme miktarını en aza indirmek için süreçte uzman değerlendirmesinden yararlanılabilir. Uzman değerlendirmesinin kullanılması genellikle sistem hakkında daha sert bir görüş sağlar ve anketten alınan yanıtlar değerlendirilirken dikkate alınması gerekir. Sonuç üzerinde bir etkisi olabileceği ve ilginç sonuçlar sağlayabileceği için, yürütücünün kullanıcıları analiz etmesi ve onların bilgilerini ve diğer ilgili geçmişlerini incelemesi önemlidir. Ayrıca, katılımcı sayısı önemlidir çünkü düşük bir sayı yanlış sonuçlara varılmasına neden olabilir, ancak daha önce de belirtildiği gibi anket, basitliği ve uzaktan yanıtlanabilme özelliği nedeniyle genellikle daha yüksek düzeyde yanıt alınmasına yardımcı olur.[30] [34]

3 öntem

Bu makalenin amaçları, desteğin mevcut durumunu ve React Native'i geliştirmenin basitliğini değerlendirmektir. Ayrıca React Native'de native uygulama ile aynı görünüm ve hisse sahip bir uygulama oluşturma olasılıkları araştırılacak ve bu iki uygulamanın performansı karşılaştırılacaktır.

3.1 Ön çalışma

Hizmet geliştirmek bilgi gerektirir. İster programlama dili, ister ortam, ister genel iş akışı olsun, geliştirici yeni bir proje oluştururken sağlam bir zemine ihtiyaç duyar. Bu makalenin ön çalışması, daha sonra React Native kullanarak bir uygulama oluşturabilmek için bu tür bilgileri elde etmeye çalıştı. Bölüm 2.4'teki referanslardan da görülebileceği gibi React Native'i ve geliştirmenin nasıl yapıldığını anlatan yayınlanmış bir makale bulunmamaktadır. React Native'i anlamanın kaynakları çok sayıda blog yazısı, çevrimiçi eğitimler ve bir uygulama oluşturmak ve mevcut bir Android uygulamasını kopyalamak için yeterli desteği sağlayan belgelerdi.

Çoğaltma işlemini gerçekleştirebilmek için bir Android uygulamasının edinilmesi gerekiyordu. Mevcut bir uygulamayı kopyalamanın nedeni bölüm 1.4'te açıklanmaktadır ve ana argüman, geliştirme süresinden tasarruf sağlaması ve odak noktasını React Native'de geliştirmeye kaydırması ve daha fazla işlevselliğe sahip bir uygulama oluşturabilmesidir. Ücretsiz ve açık kaynak (FOSS) olan ve çeşitli web sitelerinden erişilebilen birçok Android uygulaması vardır. F-Droid, FOSS uygulamalarının web sitesi şeklinde bir deposudur ve yeterli işlevsellik içerdiği görülen uygulamalar incelenerek Budget Watch1 seçilmiştir.

3.1.1 Bütçe İzleme

Budget Watch, kullanıcının bütçelerini yönetmesine yardımcı olan ve Branden Archer tarafından oluşturulan bir FOSS uygulamasıdır. Kullanıcı yiyecek, giyim vb. için farklı bütçeler oluşturup günlük işlemlerini kayıt altına alabilmektedir. İşlem farklı bütçelere atanmıştır ve farklı türde bilgiler içerir, hatta kullanıcının fotoğraf çekme olanağı da vardır.

1https://f-droid.org/repository/browse/?fdfilter=Budget+watch&fdid=protect. bütçe izleme

bir gider veya gelire ilişkin makbuz. Bu, kullanıcının her ay giderleri ve gelirleri hakkında genel bir bakış elde etmesine olanak tanır ve bütçelerini takip etmelerine yardımcı olur.

Uygulamanın tamamının ekran görüntüleri Ek A'da görülebilir ve uygulamanın başlangıç sahnesi iki öğe içerir. İlk öğe bütçelere atıfta bulunur ve bir çanta simgesi, "Bütçeler" yazan bir metin ve altında "Bütçeleri oluşturun ve yönetin" adlı daha küçük bir metinden oluşur. İşlemlere karşılık gelen öğe, el sıkışma şeklinde bir simgeye, "İşlemler" metnine ve "İşlemleri ve gelirleri girin" metnine sahiptir. Bütçeler öğesine basıldığında kullanıcı, bütçelerin listesini, içinde bulunulan ayın tarihini içeren bir sahneye yönlendirilir ve gezinme çubuğunda yeni bir bütçe eklemek için bir simge bulunur. Liste öğeleri bütçenin adını içerir ve altında şu ana kadar bütçenin maksimum değerinin ne kadarına ulaşıldığını gösteren bir ilerleme çubuğu bulunur. İlerleme çubuğunun sağ tarafında bu değer, mevcut değer ve maksimum değerle birlikte daha doğru bir şekilde görüntülenir. Artı işareti şeklindeki simgeye basmak, kullanıcıyı bütçe ekleme sahnesine yönlendirir. Bu sahne, bütçe türü için "Marketler" yer tutucusuna sahip bir giriş alanından oluşur. Aşağıda o bütçeye ait maksimum harcama miktarının yer aldığı ve sadece rakamların hariç tutulduğu bir giriş alanı daha bulunmaktadır.

Son olarak iki düğme var. İlk buton bütçe eklemeyi iptal edip listeye geri yönlendirirken, diğeri bütçeyi kaydedip (iki alan boş değilse) tekrar yeni bütçenin göründüğü listeye yönlendirir.

Başlangıç sahnesinde işlem öğesine basıldığında işlemlere yönelik bir sahne görüntülenir. Sahne, biri harcamalar ve diğeri gelirler için olmak üzere iki sayfalık işlemlerin bulunması dışında bütçelere benzer. Gezinme çubuğunun altında, iki işlem türüyle ilişkili olan iki sekmeden oluşan bir menü bulunur. Basıldığında uygulama, o türdeki işlemi gösteren bir listeden oluşan sayfayı görüntüler. Kullanıcı ayrıca parmağını sola ve sağa kaydırarak iki sayfa arasında geçiş yapabilir. İşlemlerin liste öğeleri, işlemin adını, değerini, işleme atanan bütçenin adını, işlemin çekilmiş bir görüntüsü varsa bir makbuz simgesini ve son olarak işlemin tarihini içerir. Bir işlem ekleme sahnesi şunları içerir:

- Ad - İşlemin adı için bir giriş alanı.
- Bütçe - İşleme ilişkin bütçe için seçilebilir bir liste.
- Hesap - İşlem için kullanılan hesaba ilişkin isteğe bağlı alan.
- Değer - İşlemin değeri.
- Not - Kullanıcı not eklemek isterse isteğe bağlı bir alandır.
- Tarih - İşlemin gerçekleştiği tarih.
- Makbuz - Kamerayı açan ve kullanıcının faturanın fotoğrafını çekmesine olanak tanıyan bir düğme fiş.
- Düğmeler - Tekrar yönlendiren işlemi iptal etmek veya kaydetmek için iki düğme işlemlerin listesi.

Kullanıcı ayrıca bütçe listesindeki bir bütçeye basarak işlem listesini görüntüleyebilir. Bu, işlem sahnesine yönlendirir ancak yalnızca o bütçeyle ilişkili işlemleri görüntüler.

Son olarak kullanıcı bütçeleri veya işlemleri düzenleyebilir ve kaldırabilir. Bir öğeyi düzenleme seçeneği, öğeye daha uzun süre basıldığında görüntülenir. Bu eylem, "Düzenle" yazan bir açılır pencere görüntüler ve basıldığında öğenin düzenlenmesi için bir sahne açar. Sahne, bir bütçe veya işlem eklemeyle aynıdır ancak giriş alanlarında basılan öğenin değerleri bulunur. Öğeyi silmek için gezinme çubuğunda, basıldığında öğeyi silen ve önceki sahneye yönlendiren çöp kutusu biçiminde bir simge bulunur.

3.2 Uygulama

Çoğaltma uygulaması seçilip alındığında, geliştirme işlemi gerçekleştirilebilir.
başlayın. Ek A'da bulunan Budget Watch ekran görüntüleri basıldı
ve son ürünü görselleştirmek için telefonu kapattı. Orijinal Android uygulamasının XML dosyaları
incelenerek dolgu, kenar boşluğu ve diğer stil özellikleri gibi değerler elde edilebilir.
elde edildi ve ekran görüntülerinde not edildi.

İlk başta React Native'i geliştirmek için ortamın kurulması gerekiyordu. React Native[28]
dokümantasyonu bu aşamayı kapsar ve aşağıda açıklanan farklı alt sayfalara sahiptir.
gelişen bilgisayarda farklı işletim sistemleriyle ortamın iki farklı platform için nasıl yapılandırılacağını
ve fiziksel bir cihazın nasıl kullanılacağını veya sanal bir cihazın nasıl kurulacağını ayrıntılarıyla
anlatın. "Başlarken" bölümünde sunulan adımları takip ederek,
ihtiyaç duyulan farklı paketler ve motorlar kurularak ortam oluşturuldu.
tamamlanmış. Ayrıca uygulamanın geliştirilmesine başlamadan önce bir Github
sürümlerini takip etmek amacıyla kaynak kodu için bir depo (3.4'te açıklanmıştır) oluşturuldu.
kod.

3.2.1 Gezinme

Bütçe İzlemenin beş ana bileşeni vardır: "Başlangıç", "Bütçeler", "İşlemler", "Bütçe Ekle"
ve "İşlem Ekle". İlk olarak, navigasyon ve yönlendirme bu beş bileşen için ayarlandı, sonuçta beş boş
sahne ortaya çıktı, ancak navigasyon çubuğunda farklı başlıklar vardı.
navigasyon, sahneler olarak yorumlanan rota nesneleri sağlayarak uygulamadaki sahneler arasındaki
geçişi yöneten Navigator bileşeni kullanılarak etkinleştirilir
ve renderScene işlevi tarafından başlatıldı. Gezinme bileşeni düğüme sahip olabilir
navigasyonBar ekli, tüm sahne geçişleri boyunca kalıcı olacak ve
Navigasyon çubuğunun farklı düzenlerini Naviga-tionBarRouteMapper adlı başka bir bileşen
aracılığıyla yönetin. NavigasyonBarRouteMapper, Navigator'a kullanıcıyı nasıl tasarlayacağını söyler
Her sahnede Gezinme Çubuğunun arayüzü. Bu, bir rotadan her yeni sahne oluşturulduğunda
haritacının çağrıldığı ve buna bağlı olarak gezinme çubuğunun düzenini döndürdüğü anlamına gelir.
oluşturulacak rota üzerinde.[28] [23]

3.2.2 Başlangıç sahnesi

İkinci olarak, "Bütçeler" ve "İşlemler" adlı iki butonun görsel ve metinlerini içeren iki görünüm
uygulanarak ve incelenerek başlangıç ekranının tasarımı eklendi.
daha önce bahsedilen ekran görüntülerine, sahnenin stili kolayca eklendi. İçin
kullanıcının düğmelere basarak Bütçeler veya İşlemler sahnesine gitmesine izin verin
Başlangıç sahnesinde, her düğmenin çevresine TouchableNativeFeedback sarmalayıcısı eklendi. Ne zaman
aşağı basıldığında, bu sarmalayıcının içindeki görünümün arka plan rengi bir geçiş yapabilir.
Kullanıcıdan gelen basınca bağlı olarak dalgalanma etkisi ve farklı olaylar çağrılır.[28] Ne zaman
Kullanıcı Bütçeler düğmesine bastığında, onPress özelliği, bütçeyi iten bir işlevi çağırmak için kullanılır.
görüntülenen "Bütçeler" başlığıyla birlikte BudgetsComponent'i gezgine gönderin.
gezinme çubuğu. Bu, kök bileşende renderScene yönteminin çağrılmasına neden olur
Sağlanan BudgetsComponent'i oluşturacak dizin dosyasındaki BudgetWatch_ReactNative
ve onu görüntüleyin. Aynı işlevsellik İşlemler için de eklendi.

3.2.3 Bütçe bileşeni

Sonuç olarak Bütçeler için ortam yaratıldı. Geçerli ayın tarihleri, JavaScript Date nesnesi kullanılarak
uygulandı ancak
Android uygulamasındakiyle aynı formattadır. Tarihin altında BudgetsComponent oluşturur ve
BudgetList adlı özel bir bileşen döndürür. BudgetList temel bileşeni uygular
Kaydırılabilir bir dinamik veri listesi oluşturmak için ListView. Veriler dizi tarafından kullanılır

ListView.DataSource ve her satır, dizideki her öğeden alınan verilerle işlenir. Herhangi bir veritabanı uygulanmadığından listeyi doldurmak için sabit kodlanmış veriler kullanıldı. Listenin her satırı, geliştiricinin listedeki her öğenin düzenini yalnızca bir kez belirlemesine olanak tanıyan renderRow yöntemiyle oluşturulur. Bütçe listesindeki her öğenin, React bileşeni ProgressBar'ın içe aktarılmasıyla uygulanan bir ilerleme çubuğu vardır.

Kullanıcının başlangıç sahnesine geri dönmesine veya yeni bir bütçe eklemesine izin vermek için kullanıcı, gezinme çubuğundaki yeni sahneye yönlendiren bir simgeye basabilir.
Bu işlevsellik, üç yöntem içeren NavigasyonBarRouteMapper'da uygulanır: LeftButton, RightButton ve Title; bunların hepsinde argüman olarak rota, navigatör, indeks ve navState bulunur. Bir geri düğmesi sağlamak için Sol Düğme, indeks 0'dan büyük olduğu sürece bir simge döndürür. Bu, simgenin başlangıç sahnesinde oluşturulmamasına neden olur ve basıldığında gezgin açılır ve önceki sahne gösterilir. Aynı prensip RightButton için de geçerlidir ancak gezinme çubuğunun sağ tarafındaki simge sahneler arasında farklılık gösterdiğinden, işlev rotayı karşılaştırır ve mevcut rotaya bağlı olarak bir görünüm döndürür. Görünüm, basıldığında gezgine yeni sahneyi oluşturan yeni bir bileşeni gönderen belirli bir simge içerir.

3.2.4 Bütçe Ekle Bütçe

eklemek için bir sahnenin uygulanması basitti çünkü bu sahne, TextInput bileşeni kullanılarak oluşturulan biri ad, diğeri değer için olmak üzere iki giriş alanından oluşuyordu. Alanların değerleri durum olarak saklanır ve alandaki değer her düzenlendiğinde güncellenir. Bütçeyi iptal etme veya kaydetme düğmelerini taklit etmek için James Ide tarafından geliştirilen harici bir bileşen2 kullanıldı. İptal düğmesi gezgini açar ve Kaydet düğmesi verileri kaydetmelidir. Ayrıca Kaydet düğmesi, girişlerin boş olmadığını ve girişin geçerli olmaması durumunda kullanıcıya geri bildirim sağlaması gerektiğini doğrular. Orijinal uygulamada bu, ekranın altında bir mesaj görüntüleyen Android nesnesi Snackbar3 tarafından yapılıyordu. Ancak React Native için Snackbar bileşeni mevcut olmadığından bu, geleneksel Toast nesnesi olarak değiştirildi. Kullanıcı boş değerlerle bir bütçe kaydetmeye çalıştığında ve hangi alanın hatalı olduğu konusunda geri bildirimde bulunulduğunda mesaj oluşturulur ve görüntülenir.

3.2.5 Listeler ve veritabanı

Uygulama için yerel bir depolama hizmeti uygulamak amacıyla Realm (bölüm 2.5'te açıklanmıştır) kullanıldı. Şekil 3.1'de görülebileceği gibi şemalar ve indeks dosyasındaki işlemler ve bütçeler için başlangıç verileri oluşturularak veriler, aksesuarlar kullanılarak bileşenlere aktarılabilir.

2https://github.com/ide/react-native-button
3https://developer.android.com/reference/android/support/design/widget/Snackbar.html

```
const BudgetSchema ={ name:
    'Bütçe', PrimaryKey: 'id',
    özellikler: { id: 'int', name:
    'string', maxValue:
        'int',


    }
};

const TransactionSchema ={ name:
    'Transaction', PrimaryKey: 'id',
    Properties: { id: 'int',
    transactionType: 'int',
        name: 'string',
        budget: 'string', account: {type: 'string'
        ', isteğe bağlı: true},
        değer: {type: 'float',
        varsayılan: 0}, not: {type: 'string', isteğe bağlı: true}, tarih: 'string',
        datems: 'int', makbuz: { şunu yazın: 'dize', isteğe bağlı: true},




    }
};
```

Şekil 3.1: Bölge Şemaları

Budget nesnesinden gelen veriler ListView.DataSource'a eklenmiştir ve AddBudget için Kaydet düğmesi, Realm'deki Bütçeler için yeni bir giriş yazan ve bütçe listesine geri dönmek için gezgini açan bir işlevi çağırır. Ancak mevcut kod, veritabanına yeni bir öğe eklendiğinde ve listenin bulunduğu bütçe sahnesine geri çekildiğinde liste güncellenmeyeceğinden, amaçlandığı gibi çalışmayacaktır. Bunun nedeni ListView'in Android sürümündeki bir hatadır; çünkü liste bileşeni için durum olarak yeni veriler kullanılsa bile yeniden oluşturulmaz. Bu durum, şekil 3.2'de görülen kodla çözüldü ve ilk olarak Navigator'a, kök bileşende, argüman olarak bir rota alan aynı isimdeki bir yöntemi çağıran onDidFocus özelliği eklendi. Yöntem, bir geçişten veya ilk montajdan sonra her sahnenin yeni rotasıyla çağrılır. Yeni rotanın Bütçeler olup olmadığını kontrol ederek bileşenin yeniden oluşturulması zorlanabilir.

```
onDidFocus(route){ if(route.name
    === "Bütçeler"){ var data = realm.objects('Budget').sorted('name');
        return <Budgets navigator={navigator} realm={realm} data={data} />


    } else if(route.name === "İşlemler"){
        var veri = realm.objects('İşlem'); var budgetName = Route.passProps.budgetName; return
        <İşlemler

            navigator={navigator} realm={realm}
            data={data}

            budgetName={budgetName}/>
    }
},

render()
    { this.createInitialItemsForDatabase(); return ( <Navigator ref={(nav) =>
    { navigator =
        nav }} style={{flex:1}}
            başlangıçRoute={{ name: 'Application1', bileşen: Main}}
            renderScene={this.renderScene}
            onDidFocus ={this.onDidFocus} bölge = {bölge} navigasyonBar={




                <Navigator.NavigationBar style=
                    {styles.navigationBar}
                    RouteMapper={NavigationBarRouteMapper(realm)} />


        } />
    )
}
```

Şekil 3.2: Liste içeren sahnelerin yeniden oluşturulmasını zorlamak

Ancak daha önce bahsedilen hata nedeniyle bu yeterli olmadı ve Bud-getList bileşeninin, ComponentWillUpdate yöntemini uygulamasını gerektirdi. Yeni donanımlar alındığında veya yeni bir durum ayarlandığında, bu yöntem oluşturma işleminden önce çağrılır. DataSource'un yeni durumunun yeni alınan verilere ayarlanmasıyla listenin güncellenmesi zorlanır. Ancak yeni bir durum ayarlanmadan önce eski veri dizisinin yeni diziden farklı olup olmadığının kontrol edilmesi gerekir. Aksi halde, durum ayarlanırken bileşen sonsuz bir döngüde sıkışıp kalacak, daha sonra yeni bir durum oluşturacak olan ComponentWillUpdate'i çağıracak yeni verilerle işlenecek ve bu böyle devam edecek. ComponentWillUpdate yöntemi şekil 3.3'te görülebilir.

```
bileşenWillUpdate (nextProps, nextState) {
    if (this.state.dataSource._cachedRowCount !== this.props.data.length)) {
        this.setState({ data:
            this.props.data, dataSource:
            this.state.dataSource.cloneWithRows(this.props.data) })

    }
}
```

Şekil 3.3: Listelerdeki verileri güncellemeye zorlama

3.2.6 İşlem listesi

Bütçelerin temelleri uygulandığında, aynı işlevsellik İşlemler için de eklendi. Uygulamanın iki yarısı da bir ölçüde benzerdir ancak İşlemlerin bazı yardımcı işlevleri vardır. İlk olarak, TransactionsComponent, biri giderler, diğeri gelirler için olmak üzere iki bölümden oluşur; burada kullanıcı, parmağını ekranda kaydırarak veya gezinme çubuğunun altındaki sekmelere basarak iki sahne arasında gezinebilir.

ViewPagerAndroid, kullanıcının sola ve sağa geçiş yapmasını sağlayan bir konteyner bileşenidir

çocuk görüşleri arasında. Sekmeler, Albert Brand tarafından oluşturulan react-native-android-talayout4 harici bileşeni kullanılarak uygulandı. İki bileşenin aynı anda iletişim kurabilmesi ve doğru geçerli görünümü görüntüleyebilmesi için geçerli görünümün indeksini içeren bir durum oluşturuldu. Ayrıca kullanıcı sağ üst köşedeki ikona basarak işlem ekleyebilmelidir. Bu, bir işlemi eklemenin iki farklı çeşidi olması dışında Bütçelerde olduğu gibi uygulandı; gider eklemek veya gelir eklemek. Gider sayfasını görüntülerken kullanıcı bir gider ekler ve Gelir sayfasını görüntülerken bir gelir eklenebilir. Bu, AddTransactionComponent'in oluşturulması Nav-igationBarRouteMapper'da yapıldığından ve geçerli sayfanın dizini ve ne tür bir işlemin oluşturulması gerektiği TransactionsComponent içinde olduğundan, gezginin TransactionsComponent ile iletişim kurmasını zorunlu kıldı. React Native'de destek nedeniyle ebeveyn-çocuk iletişimi basittir ancak diğer bileşenler arasında iletişim kurmanın uygun bir yolu yoktur. Ancak Realm'e AppData adlı yeni bir şema eklenerek, geçerli işlem türünün (Gider veya Gelir) dizinini içeren ve TransactionsCom-ponent'teki mevcut görünümün durumuyla eş zamanlı olarak güncellenen, cur-rentTrans adlı bir özellik eklendi. . Bu özelliğe, bileşeni doğru verilerle oluşturabilen Gezgin tarafından erişilebilir. Son olarak, bir çalışma listesi ve işlem veri tabanı nesnesi ile her bütçe için işlem toplamının hesaplanması uygulandı. BudgetList için her satırın oluşturulmasında Realm'deki tüm işlemler seçilerek ve tüm giderlerin toplamı, gelirlerin toplamından çıkarılarak hesaplanarak, sonuç ilerleme çubuğuna ve metne sunulur.

3.2.7 İşlem Ekle

İşlem ekleme uygulaması, temelini bütçe eklemekten aldı ancak daha fazla alan ve ek bilgi içeriyordu. Öncelikle kullanıcının işlemin hangi bütçeye bağlı olduğunu seçmesi gerekiyor. Bu, bütçeleri Realm'den alan ve bunları JavaScript'teki Ar-ray.prototype.map yöntemini kullanarak bir listeye eşleyen Picker bileşeni kullanılarak uygulandı. Ayrıca kullanıcının işlemin gerçekleştiği tarihi seçebilmesi için bir takvim bileşeninin uygulanması gerekiyordu.

Formun alanı, durumdan geçerli tarihi okuyan basit bir Metin bileşenidir.
Metne basıldığında, standart bir Android tarih seçici iletişim kutusunu açmak için DatePickerAndroid kullanılır. Tarih seçicinin varsayılan tarihi, eyaletteki tarihtir ve yeni bir tarih seçildiğinde eyalet güncellenir.

3.2.8 Öğeleri düzenleme veya kaldırma

Uygulama hem bütçeleri hem de işlemleri görüntüleyebildiğinde ve ekleyebildiğinde, öğeleri kaldırma veya düzenleme olanağı da uygulandı. Bu, listelerdeki her öğeyi saran TouchableHighlight için LongPress'e yeni bir özellik eklenerek yapıldı. Varsayılan olarak yanlış olan bir duruma bağlı olarak görünürlük sağlayan bir model oluşturuldu ve Bütçeler sahnesine eklendi. Kullanıcı bir öğeye uzun bastığında görünürlük durumu doğru olarak ayarlanır ve mod görüntülenir. Modal, Android'in desteklediği ContextMenu'ya benzeyecek şekilde tasarlandı. Modal bir "Düzenle" metni içerir ve basıldığında AddBudget'ın bir sürümünü açar. Yeni bir bütçe eklemekten ve mevcut bütçeyi düzenlemekten farklı olan şey, giriş alanları için düzenlenebilecek önceden ayarlanmış değerlerin bulunmasıdır. onLongPress, seçilen bütçeye ilişkin verileri içeren rowData'yı modal'a iletir. Kullanıcı düzenlemeye bastığında, bu veriler bir kez daha yeni oluşturulan AddBudget bileşenine destek olarak iletilir ve gezgine eklenir, indeks dosyasında oluşturulur ve görüntülenir. AddBudget'taki durum özelliklerine verilerdeki değerler verilir ve aktarılır. Eğer props'larda veri yoksa, bunlar boş değerlere ayarlanır çünkü bu, yeni bir bütçenin ekleneceği ve düzenlenmeyeceği anlamına gelir.

4https://github.com/AlbertBrand/react-native-android-talayout

Ayrıca, RightButton'da NavigasyonBarRouteMapper için rotanın adı "Bütçeyi Düzenle" olduğunda silme simgesinin görüntülendiği yeni bir durum eklendi. Simgenin işlevi, düzenlenmekte olan mevcut bütçeyi silmektir. Ancak, daha önce de belirtildiği gibi, sahneler ile navigatör arasındaki iletişim zordur ve Realm'deki AppData nesnesine, düzenlenen bütçenin kimliğini içeren yeni bir currentEditBudget özelliği eklenmiştir. Kimlik, düzenleme sahnesi oluşturulduğunda moddan ayarlanır ve kimliğin kullanılmasıyla gezgin, bütçeyi Realm'den kolayca kaldırabilir.
Aynı işlevsellik, tek farkın, basılan işlemin işlem türüne bağlı olarak sahne adının "Gideri Düzenle" veya "Geliri Düzenle" olması gerektiği işlemlere de uygulandı. Bununla birlikte ListView'in yeniden oluşturulmasında bir kez daha sorunlar ortaya çıktı. Listeyi güncellemek için daha önce uygulanan doğrulama yalnızca orijinal veriler ile yeni verilerin uzunluğunu karşılaştırdığından ek bir açıklama uygulandı. ListView'deki hata nedeniyle listeye doğru veriler sağlandı ve DataSource güncellenmiş değerleri içeriyordu ancak listeyi yeniden oluşturmadı. Bu nedenle iki farklı veri setini karşılaştıran başka bir ifade mümkün olmadı ve Realm'deki AppData'ya başka bir özellik atanması gerekti. Bu özellik, ShouldUpdate adında bir boole değeridir ve bir öğe düzenlenirken true değerine ayarlanır. Listenin güncellenmesine yönelik doğrulamaya, verilerin uzunluklarını karşılaştıran ancak aynı zamanda ShouldUpdate değerini inceleyen ve eğer doğruysa güncellemeyi zorunlu kılan bir açıklama verildi.

3.2.9 Donanım - geri düğmesi ve kamera

Bir Android uygulamasında, donanım geri düğmesi5, uygulamayı kapattığı ilk ekran dışında, varsayılan olarak önceki sahneye geri adım atmak için kullanılan bir navigasyondur. React Native'de geri düğmesi, üzerine yazılmadığı sürece her zaman uygulamanın tamamını kapatacaktır. Bu, API kullanılarak ve BackAndroid'e, donanım geri düğmesine basıldığını algılayan ve bunu kendi işlevselliğimizle programlı olarak çağırmamıza izin veren bir eventListener eklenerek yapıldı.

```
BackAndroid.addEventListener('hardwareBackPress', () => {
    if (navigator && navigator.getCurrentRoutes().length > 1) { navigator.pop(); doğruyu döndür;


    } false değerini
döndür; });
```

Şekil 3.4: Fiziksel geri düğmesi için gezinmenin uygulanması

Geriye kalan tek özellik, yapılan işleme ilişkin makbuzun fotoğrafını çekebilmekti. React Native o zamanlar kamera için bir bileşeni desteklemiyordu ancak React Native için birçok bileşenin yaratıcısı olan Lochlan Wansbrough, bir kamera dönemi bileşeni6 geliştirdi . Bir işlem eklenirken veya düzenlenirken Yakala düğmesine veya Güncelle düğmesine basıldığında kamera görüntülenmeli ve tüm ekranı, hatta gezinme çubuğunu bile kaplamalıdır. React Native'de belirli bir sahne için gezinme çubuğunu gizlemek mümkün olmadığından tüm ekranı kaplayan ve kamera bileşenini saran bir model oluşturuldu. Daha önce bahsedilen düğmelerden birine basıldığında mod görünür olarak ayarlandı. Kameranın belgeleri yalnızca iOS'u kapsıyor ve fotoğraf çekebilecek görsel bir düğme uygulamak için, kamera ekranının üstüne mutlak konumlandırmayla harici bir görünümün uygulanması ve düzenlenmesi gerekiyordu. Kameranın uygulanması ve oluşturulması şekil 3.5'te görülebilir. Bir görüntü yakalandığında, görüntünün yolu işlemin durumuna eklendi ve ardından Realm'e kaydedildi. Kullanıcı ne zaman

5https://developer.android.com/design/patterns/navigation.html
6https://github.com/lwansbrough/react-native-camera

makbuzu görüntülemek için gezinir; görüntünün URI'si olarak kaynak içeren bir görüntüyü içeren bir bileşen oluşturulur ve görüntülenir.

```
<Modal
    AnimationType={'none'} Transparent={false}
    görünür={this.state.modalVisible}
    onRequestClose={() => {this._setModalVisible(false)}}> <View
    style={cameraStyles.container}> <Kamera ref={(kamera) => { this.kamera = kamera; }} style={cameraStyles.preview}
    feature={Camera.constants.Aspect.fill}> </Camera>




        // Kamera bileşeninin dışında olması gerekiyor <TouchableHighlight
        style={cameraStyles.actionButton}
            onPress={this.takePicture.bind(this)}
            underlayColor="#d6d6d6"> <View style={cameraStyles.buttonContainer}
            >

                <Resim
                    style={cameraStyles.cameraButton} source={require('./
                    images/camera-icon.png')} /> </View> </TouchableHighlight> </View> </Modal>



// Fotoğraf çekme işlevi takePicture() { var navigator =
this.props.navigator; var
    thisBileşen = bu; this.camera.capture()


        .then(işlev(veri){
            thisComponent.setState({inputReceipt: data.path});
            thisComponent._setModalVisible(false); }) .catch(hata =>

        console.error(hata));
}
```

Şekil 3.5: Kameranın uygulanması

Android uygulamasının, bir Android uygulamasının yönergelerine uyması için biraz değiştirilmesi gerekiyordu. Değişiklikler sayfa düzenindeki küçük değişikliklerdi ancak mevcut bütçeleri ve işlemleri kaydetme işlevi çok zor olacağından ve bu tezin kapsamına sığmayacağından kaldırıldı. Nihai Android uygulamasının kodu Github deposundan görüntülenebilir ve indirilebilir7 ve React Native'deki son uygulama başka bir depoda8 bulunur .

3.3 Değerlendirme

Bölüm 1.2'de listelenen iki araştırma sorusunu yanıtlamak için değerlendirme iki bölüm halinde gerçekleştirildi: çoğaltmanın yapılabilirliği ve oluşturulan uygulamanın performansı. React Native'de Android uygulaması geliştirme ve Budget Watch uygulamasının kopyalanma imkanının değerlendirilmesi iki farklı süreçte gerçekleştirildi.
Geliştirme değerlendirmesi geliştiricinin kendisi tarafından gözlemlendi ve geliştirme aşamasındaki genel ifadeye ve hedeflenen uygulamanın React Native çerçevesi kullanılarak çoğaltılıp çoğaltılamayacağına dayanıyordu. Dolayısıyla bu değerlendirme herhangi bir görsel sonuç değil, nihai bir tartışma üretecektir. Tüm işlevlerin kopyalanması son tarihten önce tamamlanamazsa, aynı işlevlere sahip iki uygulamayı elde etmek için android uygulamasının yedek işlevselliği kaldırılacaktır.

7https://github.com/willedanielsson/BudgetWatch_android
8https://github.com/willedanielsson/BudgetWatch_ReactNative

3.3.1 Çoğaltma

Ayrıca geliştirmenin son ürünü Valtech çalışanları tarafından test edildi. Bu ürün yelpazesi, bir Android uygulamasının nasıl çalışması gerektiği konusunda bilgi sahibi olan ve React Native uygulamasındaki hataları ayırt edebilecek niteliklere sahip kullanıcılara sağlar.
Daha gerçekliğe dayalı bir sonuç sağlayacak uygulamayı farklı bilgi birikimine sahip kişilerin değerlendirmesine olanak tanınabilir, ancak alanın uzmanlarını seçerek daha profesyonel bir sonuç elde edilebilir. Kullanıcılara orijinal Budget Watch uygulamasının ve replikasının yüklü olduğu bir mobil cihaz verildi ve "Hangisi React Native'de oluşturuldu?" sorusu soruldu. Kullanıcılar, kullanıcıya ne yapması gerektiğini söylemeden tarafsız bir sonuç elde etmek için her iki uygulamayı da özgürce kullanabilme ve diledikleri eylemleri gerçekleştirebilme olanağına sahip oldu. Uygulamalara Uygulama1 ve Uygulama2 adı verildi ve kullanıcıya cevabını kaydetmesi için basit bir form verildi. Form, kullanıcının hangi uygulamanın yerel kökenli olduğunu bilip bilmediğini veya kullanıcının yanıt verip veremeyeceğini sordu. Ayrıca, kullanıcı uygulamalardan birine cevap vermişse, cevabın kesinliğini soran bir takip sorusu verilmiştir. Son soru, kullanıcının bir React Native uygulamasını kullanmayı sorun edip etmeyeceğini veya deneyimin gelecekteki kullanımı caydıracak kadar kötü olup olmadığını sordu. Bu kullanıcı testi, uygulamalardaki farklılıkları fark edebilen kaç kullanıcının sayısını ve dolayısıyla React Native'de oluşturulan bir uygulamanın kullanıcı deneyiminin yerel bir uygulamayla karşılaştırıldığında ne kadar iyi olduğunu gösteren bir sonuç üretti.

3.3.2 Performans

Aşağıda bölüm 3.4'te açıklanan profil oluşturma aracı Trepn Profiler kullanılarak, iki uygulamanın performansı farklı yönleriyle test edilebilir. İki uygulamanın sonuçlarının cihaza bağlı olarak farklılık göstermediğini tespit etmek amacıyla testler aynı mobil cihaz kullanılarak yapılmıştır. Ayrıca uygulama herhangi bir internet bağlantısı gerektirmediği için mobil cihaz uçuş moduna alınarak o anki trafikten etkilenecek sonuçlar ortadan kaldırılmıştır. Çevresel etkileri daha da en aza indirgemek için testler iki kez gerçekleştirildi. Öncelikle Android uygulaması kullanılarak bir test yapıldı ve ardından React Native uygulaması test edildi. Son olarak her testten önce uygulamanın başlangıç dışı verileri kaldırılarak sonlandırılmıştır.

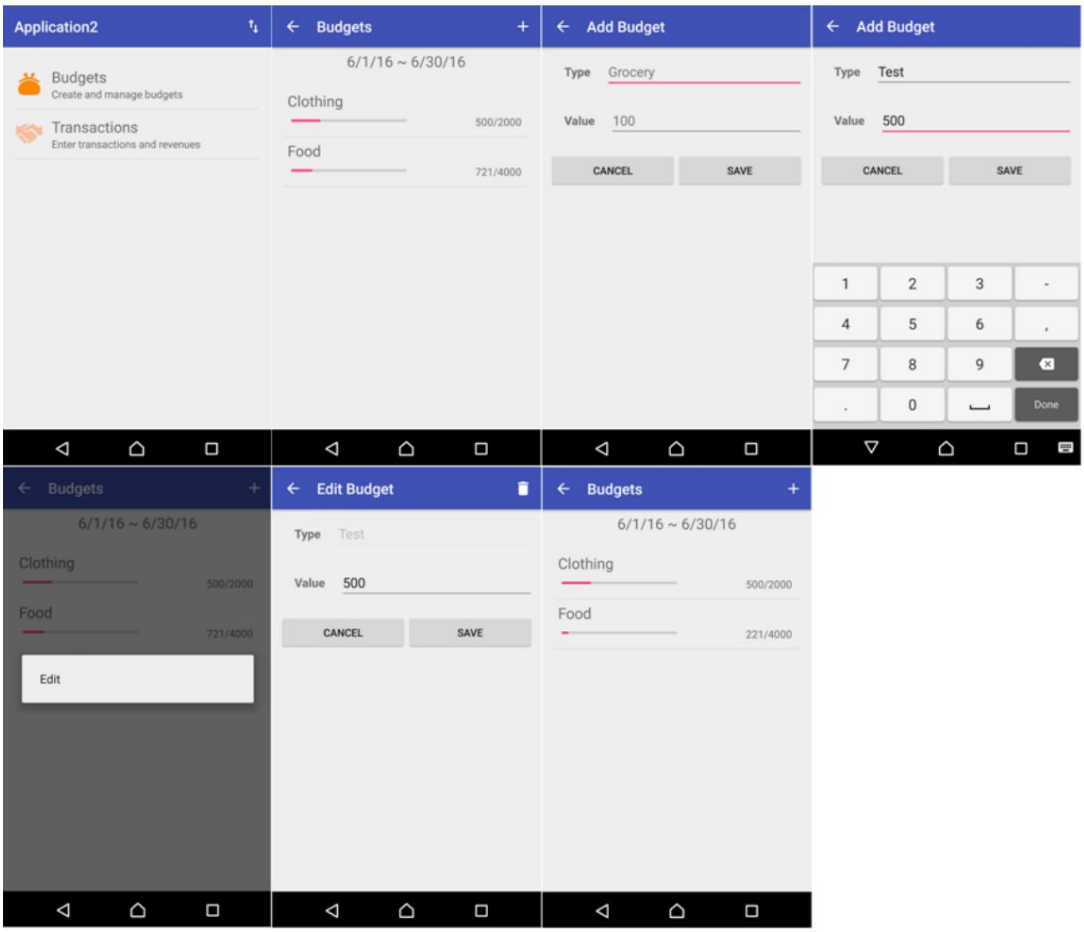
Performans ölçülürken bir uygulamanın analiz edilebilecek birçok parçası vardır.
ancak bu makale aşağıdaki veri noktalarına odaklandı:

- Megahertz (MHz) cinsinden bir değer görüntüleyen ve GPU'nun çalışma hızının bir ölçüsü olan GPU frekansı (GPU, grafiksel hesaplamaları gerçekleştiren ve telefondan ekrana gelen sinyali işleyen ve bunun tersini yapan birimdir).
- CPU'nun (birim olan) ne kadarının yüzdelik değerini sağlayan CPU yükü programları çalıştıran) uygulama tarafından kullanılır.
- Uygulamanın kaç Megabayt (MB) gerektirdiğini döndüren bellek kullanımı tion.
- Ne kadar gücün tüketildiğine ilişkin miliwatt (mW) cinsinden bir değer sağlayan Pil Gücü profilli uygulama tarafından boşaltılır.

Ayrıca uygulamanın performans açısından test edilen üç durumu bulunmaktadır. İlk test uygulama başlatıldığında ve 30 saniye boşta tutularak gerçekleştirildi. Daha sonra, uygulamanın zaten başlatıldığı ve kullanıcının uygulama içinde gezindiği performansı ölçerken iki kullanıcı vakası yapıldı.

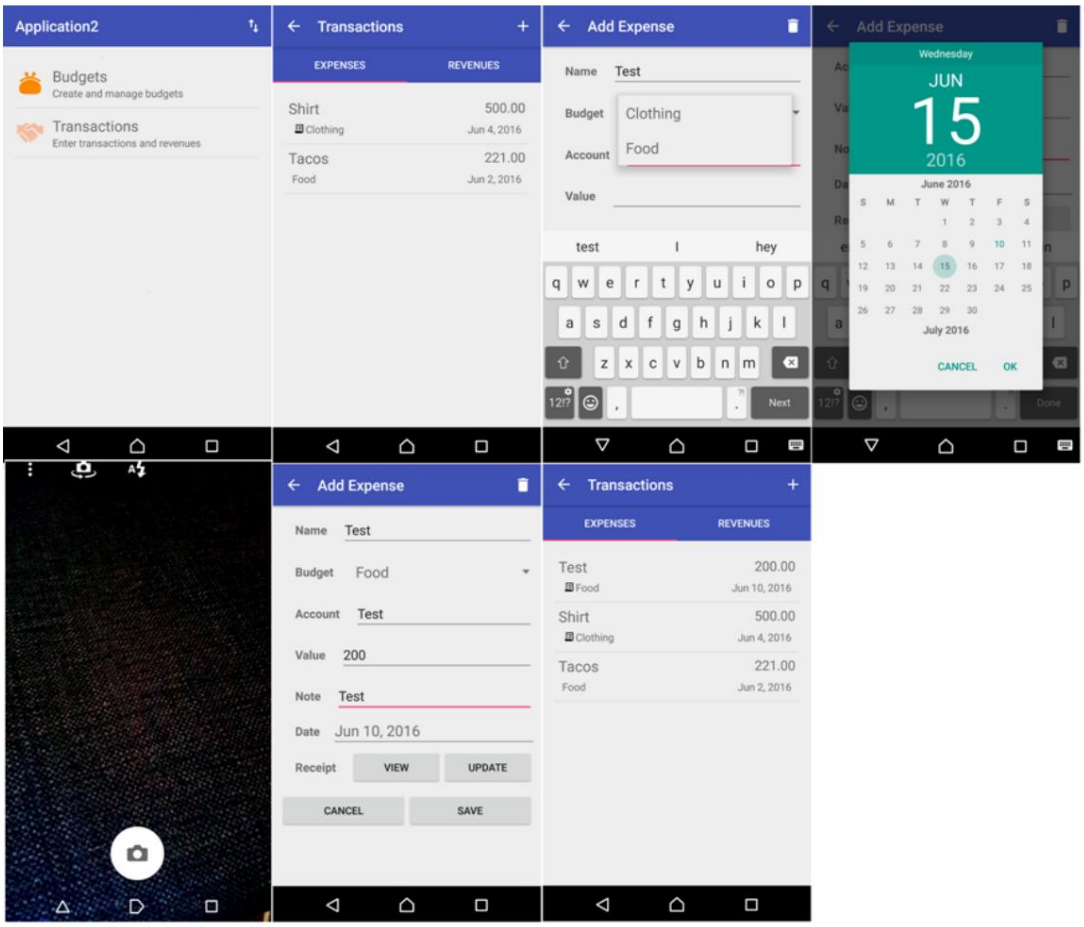
İlk durum, kullanıcının Bütçeler'e gidip yeni bir kategori oluşturmasıydı. Kategori "Test" tür değerini ve 500 değerini aldı. Kategori oluşturulduğunda kullanıcı bütçe listelerine yönlendirildi. Daha sonra kullanıcı bütçeyi düzenledi ve

kaldırdı. Bütçe kaldırılıp bütçe listesi güncellendiğinde test yapıldı ve profil oluşturma durduruldu. İlk kullanıcı senaryosunun kullanıcı yol haritasındaki değerlerin yer aldığı ekran görüntüleri şekil 3.6'da görülebilir.



Şekil 3.6: İlk kullanıcı senaryosunun yol haritası

İkinci kullanıcı durumu, kullanıcının İşlemler'e gidip yeni bir senaryo oluşturmasıydı. Bundan sonra kullanıcı yeni harcamanın ayrıntılarını yazdı ve makbuz olarak iliştirilen bir fotoğraf çekti. Kullanıcı sonuç olarak işlemi kaydetti ve profil oluşturma durdurulduğunda test yapıldı. Bu kullanıcı yol haritasındaki değerlerin yer aldığı ekran görüntüleri şekil 3.7'de görülebilir.



Şekil 3.7: İkinci kullanıcı senaryosunun yol haritası

3.4 Araçlar

3.4.1 Trepn Profiler Trepn

Profiler9 , Android cihazlara yönelik bir güç ve performans profili oluşturma uygulamasıdır ve pro-filer Qualcomm'un bir ürünü olduğundan Qualcomm Snapdragon işlemcili cihazlar için ek özelliklere sahiptir. Trepn, verileri gerçek zamanlı olarak görüntüleyebilen ve pil gücünü, CPU ve GPU frekansını ve kullanımını, ağ kullanımını ve GPS veya ekran olarak cihazın farklı durumlarını ölçebilen özelleştirilebilir katmanlar sunar. Toplanan veriler, bilgilerin analiz edilmesine yardımcı olan .csv- veya .db-formatında kaydedilebilir. Bölüm 1.5'te konuyla ilgili daha önce yazılmış makaleler sunulmuş olup Arnesson[2] tarafından yazılan bir makalede Trepn kullanılmıştır. Arnesson, performansı ölçerek farklı platformlar arası araçları karşılaştırırken veri toplamak için Trepn'i bir araç olarak kullandı ve deneyimine göre, araç güvenilir ve kullanışlıydı. Ayrıca Bakker, Android için farklı profil oluşturucular üzerinde bir değerlendirme gerçekleştirdi ve Trepn'in bileşenlerin gerçek kullanımını ölçmek için en iyi yöntem olduğu sonucuna vardı.[4]

3.4.2 Sony Experia Z1

Tüm proje boyunca kullanılan mobil cihaz, Android 5.1.1 yüklü, 2015 model bir Sony Experia Z1'di. Cihazda Qualcomm MSM8974 Snapdragon bulunuyor

9https://developer.qualcomm.com/software/trepn-power-profiler

800 işlemci (Trepn Profiler ile ek özellikler sağlayan), Dört çekirdekli 2,2 GHz CPU ve GPU bir Adreno 330'dur. Pil, çıkarılamayan bir Li-Ion 3000 mAh pildir ve ekran 5 inç çapında 1080 x 1920 piksel, bu da 441 ppi piksel yoğunluğuyla sonuçlanır. Cihaz Valtech tarafından sağlandı ve geliştirme, performans analizi ve kullanıcı testleri sırasında kullanıldı.

3.4.3 GitHub

GitHub, web tabanlı olan ve 15 milyondan fazla kullanıcısı ile dünyanın en büyük kaynak kodu barındırıcısı olan Git depolarını barındırmaya yönelik bir hizmettir. Git'in kendisi, Linux'un yaratıcısı Linus Torvalds tarafından 2005 yılında geliştirilen sürüm kontrolüne yönelik bir komut satırı aracıdır.
Git ile geliştirici, projeyi sıklıkla taahhüt ederek kodda yapılan değişiklikleri takip edebilir. Bir taahhüt tamamlandığında Git, kodun anlık görüntüsünü alır ve yapılan farklılıkları saklar. Bunun sonucunda, geliştiricinin de geri alabileceği proje sürümleri akışı elde edilir ve geliştiricinin kodu kırma konusunda endişelenmesine gerek kalmayacağı için güvenlik sağlanır. Ayrıca, projenin farklı özelliklerinin ayrı ayrı geliştirilmesine ve tamamlandığında birleştirilmesine olanak tanıyan şubeler oluşturulup birleştirilebilir. [13][6]

3.4.4 Yüce Metin 3

Uygulamanın geliştirilmesi sırasında kullanılan editör, kod ve işaretleme için platformlar arası bir metin editörü olan Sublime Text10'du. Sürüm 3, 2013 yılında piyasaya sürüldü ve varsayılan olarak Sublime, çerçeve için birçok dili ve parçacığı destekler ancak çok sayıda eklentiyle genişletilebilir. Paketler, Sublime Text için üçüncü taraf bir paket yöneticisi olan Package-Control tarafından yüklenir. Yöneticinin kullanımıyla, React JSX sözdizimi uzantılarıyla ES6+ JavaScript için dil tanımları desteği ekleyen Babel11 paketi kuruldu.

3.5 Analiz

Bölüm 3.3.1'de açıklanan kullanıcı testinden alınan cevaplar özetlenmiş ve grafikler halinde çizilmiştir ve testlerden elde edilebilecek üç farklı sonuç bulunmaktadır.
İlki, kaç kullanıcının gerçek Android uygulamasını ayırt edebildiğini gösteriyordu. Bu, kaç kişinin yerel uygulamayı doğru yanıtladığını, kaç kişinin yanlış tahmin ettiğini ve kaç kullanıcının bilmediğini gösteriyor. İkinci sonuç, kullanıcıların yerel uygulamayı ayırt edebildiğinin kesinliğini gösterdi. Üçüncüsü, kullanıcının React Native uygulamasını beğenip beğenmediğini ve React Native uygulamasını kullanmayı sorun edip etmeyeceğini yanıtlarken kullanıcı deneyimini özetledi. Seçimlerine ilişkin yorumlarla birlikte, replikasyonun değerlendirilmesinden elde edilen sağlam sonuçlar oluşturulabilir ve bu sonuçlar bölüm 4.1'de sunulmaktadır.

Performans testleri birbirinden ayrı olarak yapıldı ve daha adil ve tam bir sonuç için ortalamayı elde etmek amacıyla her test üç kez yapıldı. Öncelikle android uygulaması üzerinde tüm testler yapılmış ve bunun sonucunda React Native uygulaması test edilmiştir.
Örneklenen veriler, virgülle ayrılmış bir değer dosyası olan ve verilerin birleştirilebildiği, kategorize edilebildiği, ortalama bir değere göre hesaplanabildiği ve aynı zamanda orijinal verilerden grafikler oluşturulabildiği Excel'e kolayca aktarılabilen bir csv dosyası olarak kaydedildi. Ortaya çıkan veriler bölüm 4.2'de gösterilmiş ve açıklanmıştır.

10https://www.sublimetext.com/3
11https://packagecontrol.io/packages/Babel%20Snippets

4 Sonuçlar

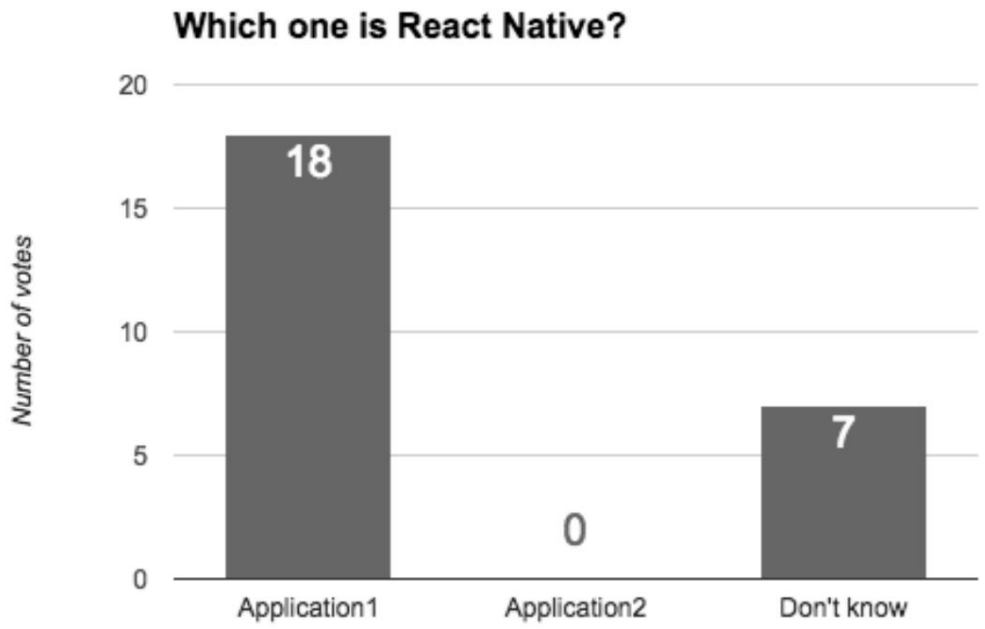
Bu bölümde gerçekleştirilen iki testin (çoğaltma testi ve performans testi) sonuçları görüntülenir. Bölüm 3.3.1'de replikasyon testi daha ayrıntılı olarak açıklanmaktadır ve performans testi için kullanılan yöntem bölüm 3.3.2'de açıklanmaktadır.

4.1 Çoğaltma

İlk araştırma sorusu, React Native'de yerel bir Android uygulamasından ayırt edilemeyen bir uygulama oluşturmanın mümkün olup olmadığını incelemeye yönelikti. Valtech danışmanlarına oluşturulan iki uygulamayı test etme fırsatı verildi ve ne yapmalarına izin verildiği konusunda hiçbir kısıtlama yoktu. Uygulamalar her test sonrasında sıfırlanarak sonlandırılarak tüm kullanıcılara aynı deneyimin yaşatılması sağlandı. Kullanıcı bir karara vardığında kullanıcı, deneyimlerine ilişkin soruların ve elbette hangi uygulamanın React Native olduğuna dair bir sorunun yer aldığı basit bir form doldurdu.

Öncelikle kullanıcılara React Native'de oluşturulan uygulamayı ayırt edip edemedikleri sorgulandı. Uygulamalar Application1 ve Application2 olarak adlandırıldı ve ilki React Native uygulamasıydı. Şekil 4.1'de görüldüğü gibi kimse yanlış tahminde bulunup React Native uygulaması olarak Application2'yi seçmedi. 25 danışmandan 18 kullanıcı React Native'de oluşturulan doğru uygulamayı seçti ve 7 kullanıcı bilmiyordu ve bu nedenle yanıt vermedi. Kullanıcıların ayrılmasına izin verildiği yorumlarından, doğru uygulamayı ayırt edebilmenin en yaygın nedeni sahneler arası geçişler oldu. Uygulama1'i seçenler geçişlerin native animasyona benzemediğini söylerken, doğru cevabı bilmeyenler ise animasyonlarda farklılık olduğunu ancak hangi animasyonun native olduğunu söyleyemediklerini söyledi.

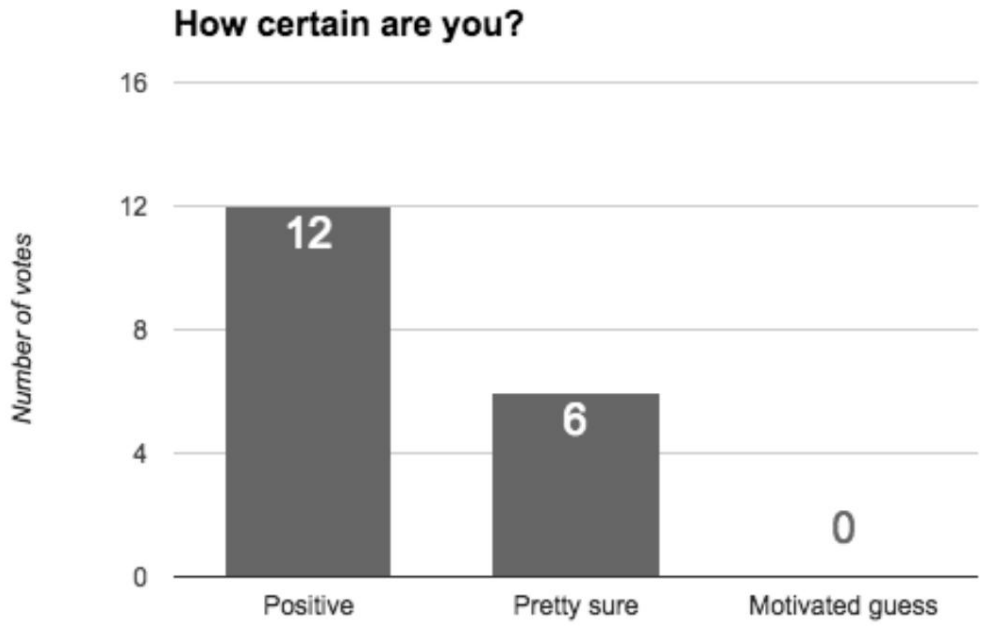
Ayrıca, birkaç kullanıcı da uygulamaların başlama süresinde bir gecikme fark etti ve bu nedenle daha yavaş olan uygulamanın React Native uygulaması olduğunu tahmin etti.



Şekil 4.1: Uygulama1'in React Native uygulaması olduğu kullanıcı yanıtları

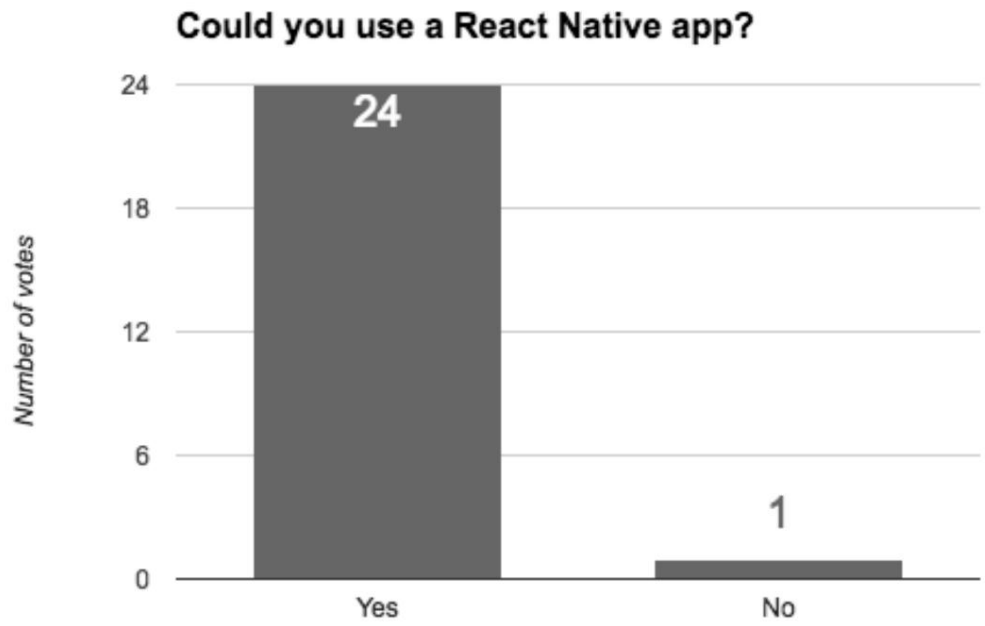
Kullanıcı ilk soruya bir cevap seçtiğinde, uygulamalardan birini cevaplayanlara ek bir soru daha sunuldu. Soru onların seçimlerinden ne kadar emin olduklarını sordu. Bu soru, önceki cevabı analiz edebilmek ve farklılıkların ne kadar net olduğunu görebilmek için kullanıcının cevabının kesinliğini belirlemek amacıyla gerekliydi. Doğru cevabı seçen 18 kişinin sonuçları aşağıda şekil 4.2'de gösterilmektedir. React Native uygulamasını ayırt edebilen kullanıcıların cevaplarından emin oldukları açık çünkü hiç kimse bu kesinliği "Motive edilmiş bir tahmin" olarak tanımlamadı.

Ayrıca 12 kullanıcı Uygulama1'in React Native uygulaması olduğunu biliyorken geri kalan 6 kullanıcı bunun Android uygulaması olmadığından oldukça emindi. Olumlu yanıt verenler ise Android uygulamaları geliştirme ve kullanma deneyimleri olduğundan animasyonların nasıl görünmesi gerektiğini bildiklerini ve Uygulama1 geçişlerini tanımadıklarını söylediler. "Oldukça eminim" diyen kullanıcılar da benzer motivasyonlara sahipti ve bazıları Application1'in açılış süresinin daha yavaş olması nedeniyle muhtemelen React Native olduğunu düşündüler ancak tam olarak emin olamadılar.



Şekil 4.2: Bir önceki soruda Uygulama1'i seçen kişilerin kesinliği

Son olarak formun son sorusunda kullanıcıya, React Native uygulamasının hatalarının, uygulamayı hiç kullanmamalarını gerektirecek kadar ciddi olup olmadığı soruldu. İki uygulama arasında bazı kusurlar ve farklılıklar olsa da, React Native uygulamasının kullanıcının onu kullanması için yeterince iyi olup olmadığını bilmek önemlidir. Sorunun kendisi basit bir evet veya hayır sorusuydu ve sonuçlar şekil 4.3'te görülebilir. Görüldüğü gibi React Native'de oluşturulan uygulamayı kullanan bir kullanıcı dışında hiçbir kullanıcı herhangi bir sorun yaşamadı. React Native'de hangi uygulamanın oluşturulduğunu bilmeyen 7 danışman, bunun native olabileceğini düşündü ancak hangi uygulamanın React Native olduğu söylendi ve buna rağmen React Native uygulamasında herhangi bir sorun yaşamadılar. React Native uygulamasını kullanmak istemeyen bir kullanıcı, Android uygulamasına kıyasla animasyonların yavaş geldiğini ve bu durumun kendisini huzursuz ettiğini söyledi. Ancak kullanıcı aynı zamanda mükemmeliyetçi olduğunu ve geçiş sorunu çözülecekse React Native uygulamasını düşünmeyeceğini de belirtti.



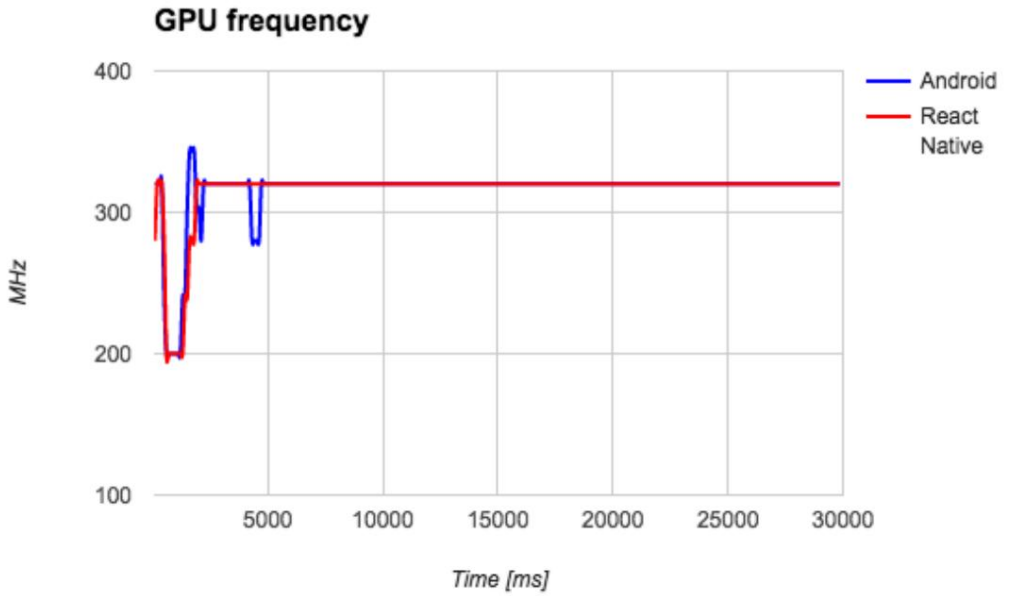
Şekil 4.3: Hala React Native uygulamasını kullanan kaç kullanıcının sonuçları

4.2 Performans

İkinci araştırma sorusunun amacı, React Native'de oluşturulan bir uygulamanın native Android uygulamasıyla aynı performansa sahip olup olmadığını, değilse ne kadar farklı olduğunu araştırmaktı. Bölüm 3.3.2'de açıklandığı gibi, üç test gerçekleştirildi; ilki iki uygulamanın boştayken performansını inceledi, ikinci test bütçe oluşturulduğunda ve kaldırıldığında performansı ölçtü ve üçüncü test bütçeyi inceledi. Uygulamaların işlem olarak verimliliği yaratıldı. Son iki testin yol haritası şekil 3.6 ve 3.7'de görülebilir. Ayrıca performansın ölçülen dört yönü vardı: GPU frekansı, CPU yükü, bellek kullanımı ve güç tüketimi.

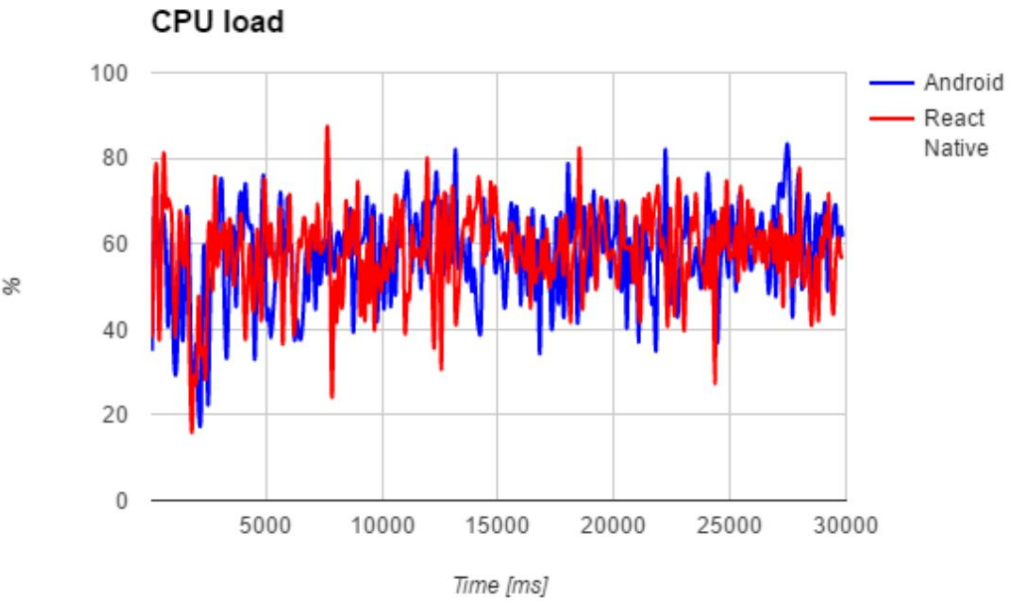
4.2.1 İlk test

Uygulamalar boştayken ilk testler 30 saniye süreyle gerçekleştirildi ve her uygulama için üç çalıştırma kaydedildi. Üç veri kümesinin ortalaması hesaplanmış ve aşağıda grafikler halinde sunulmuştur. Öncelikle şekil 4.4'te görülebileceği gibi uygulama başlatılıp kullanıcıya görüntülendiğinde GPU frekansı önemli ölçüde düştü. Uygulamayı kapatıp arka planda çalıştırdığınızda frekans arttı ve Android uygulamasının frekansı yaklaşık 340 MHz'e yükseldi, React Native uygulamasıyla birlikte 320 MHz'e düştü. Bu, Android uygulamasında yaklaşık 4,5 saniye sonra yaşanan küçük düşüş hariç, testin geri kalanı boyunca ölçülen frekanstır. Android uygulamasının ortalama GPU frekansı 315,78 MHz, React Native uygulamasının ise 315,58 MHz oldu.



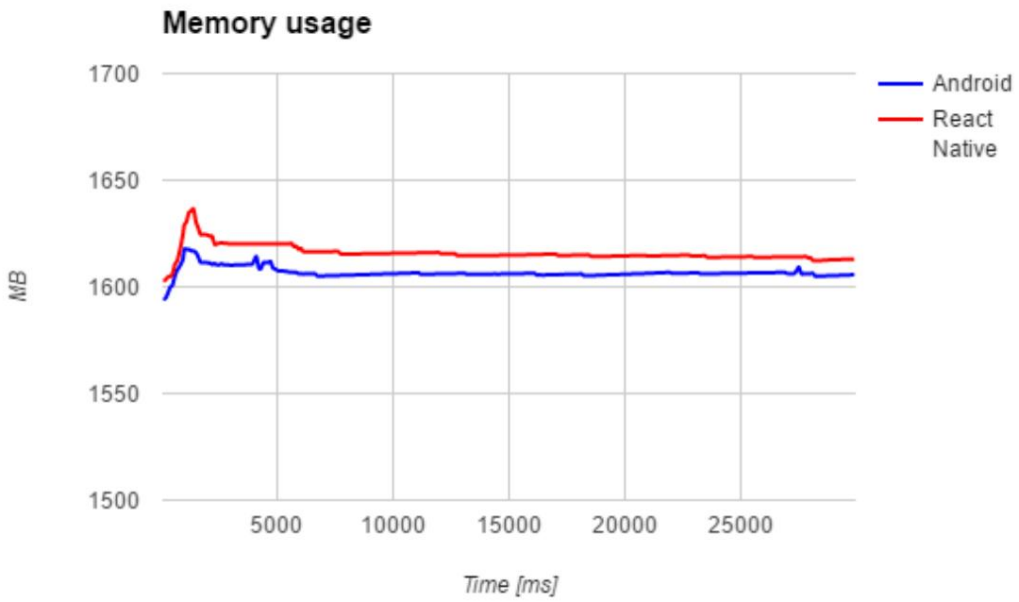
Şekil 4.4: Boş uygulamaların GPU frekansı

Şekil 4.5'teki CPU yükünün grafiğini, pürüzlü sonuçlar nedeniyle okumak biraz daha zordur, ancak her iki uygulamanın da yüksek derecede aynı CPU yüküne sahip olduğu ve React için 7,6 ve 7,8 saniyeden sonra iki farklı ani artışın beklendiği belirtilebilir. Yerel uygulama. Ortalama değer Android için %57,60, React Native uygulaması için ise %57,89 oldu.



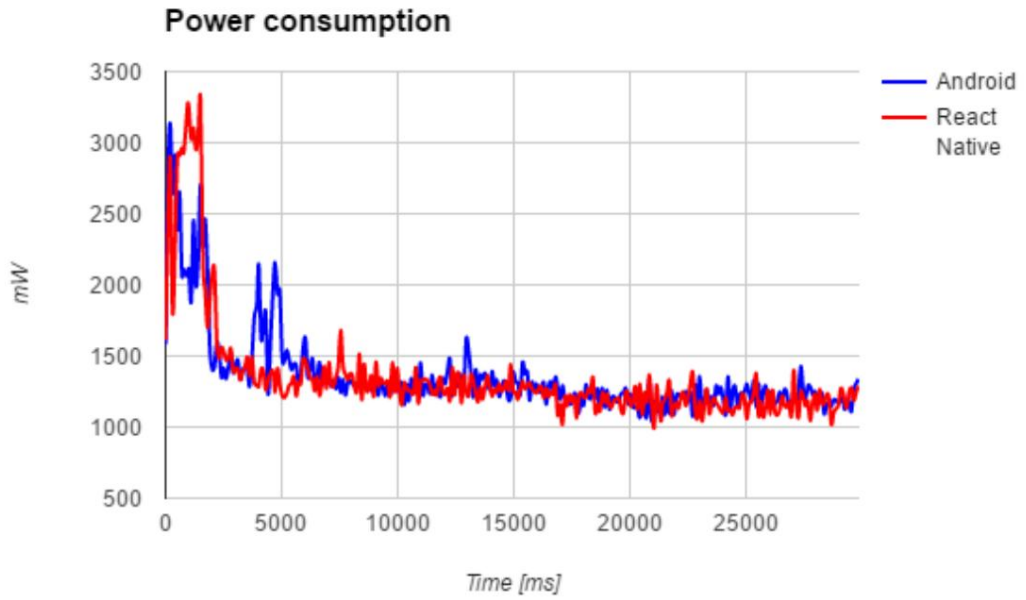
Şekil 4.5: Boş uygulamaların CPU yükü

Ayrıca, uygulamaların bellek kullanımı, uygulamaların başlatılması sırasındaki kullanım artışıyla aynı kalıba sahipti ancak daha sonra istikrarlı bir düşüş gösterdi. Aradaki en belirgin fark, 1607 Megabayt bellek gerektiren Android uygulamasının biraz daha düşük kullanımı, React Native uygulamasının ise ortalama 1616 MB bellek kullanımıydı.



Şekil 4.6: Boş uygulamaların bellek kullanımı

Son olarak, güç tüketimi de uygulamalarla aynı modele sahipti. Uygulamalara başlandığı dönemde tüketim çok yüksekti ancak zamanla azaldı. React Native uygulamasının 1 saniye, Android uygulamasının ise 4,5 saniye sonraki birikimleri dikkat çekicidir. Ortalama güç tüketimi Android uygulaması ve React Native uygulaması için 1362 miliwatt ve 1350 miliwatt olarak gerçekleşti.



Şekil 4.7: Boşta kalan uygulamaların güç tüketimi

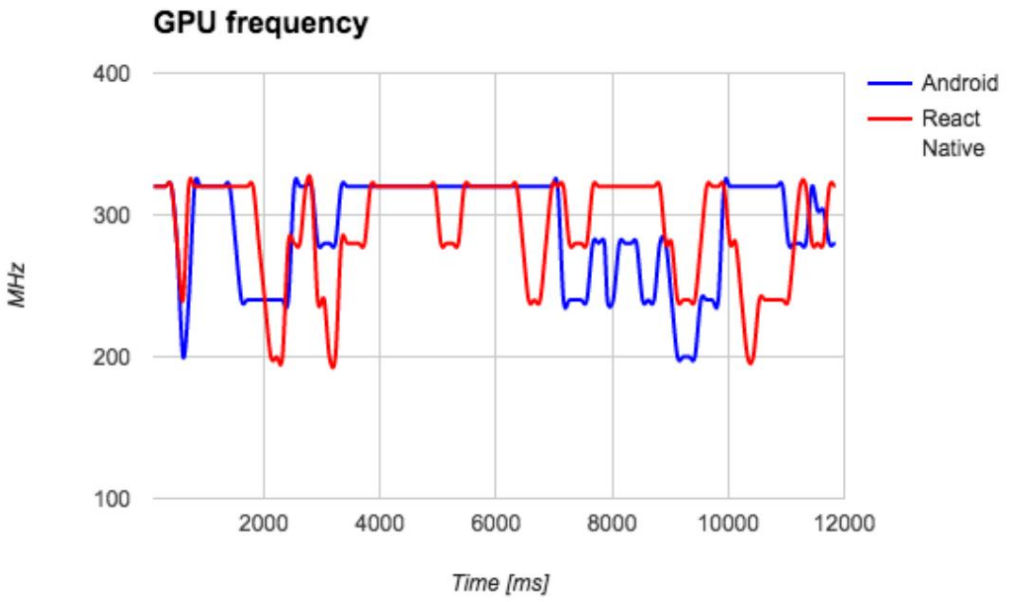
4.2.2 İkinci test

İkinci test bölüm 3.3.2'de açıklanmaktadır ve kullanıcının yeni bir bütçe eklemesi ve ardından bunu silmesinden oluşur. Bu eylemleri gerçekleştirmek için, testlerin her iki uygulamanın yürütülmesi de kabaca 12 saniyeden kısa sürdü. Test sırasında gerçekleştirilen ana eylemler aşağıda şekil 4.8'de gösterilmektedir.

Action	Time [s]
Navigate to Budgets	1.5
Pressing "Add budget" icon	3
Saving budget	7
Pressing "Edit budget" button	9
Deleting budget	11

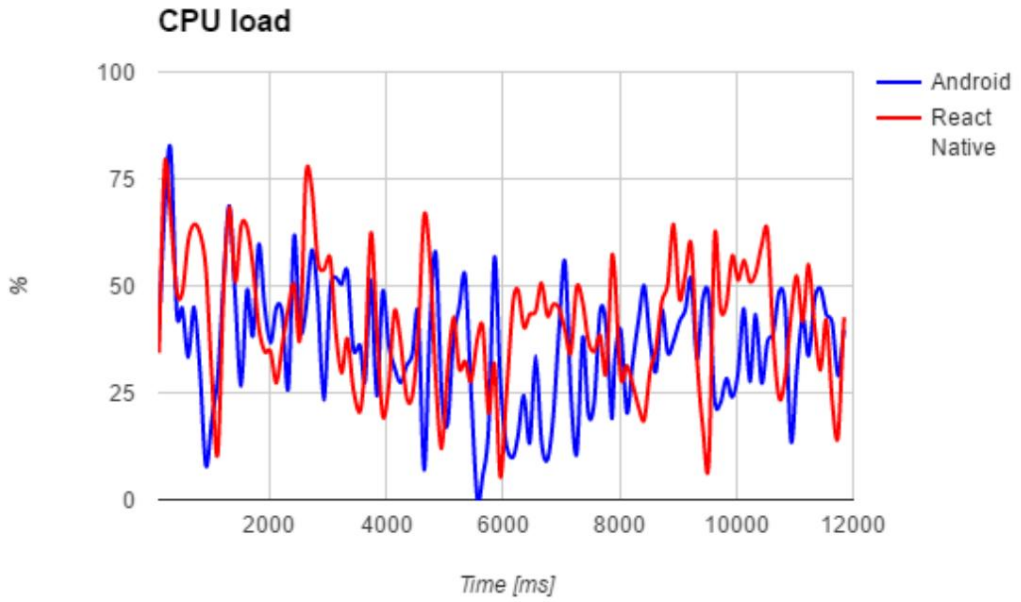
Şekil 4.8: İkinci performans testi için yaklaşık zaman damgalarında gerçekleştirilen eylemler

Bütçeleri işlerken uygulamaların GPU frekansı aşağıda şekil 4.9'da gösterilmektedir. Kullanıcı bütçe listesine gittiğinde frekans Android uygulaması için 200 MHz'e, React Native uygulaması için ise 240 MHz'e düştü. Yaklaşık 3 saniye sonra bütçe ekleme sahnesi oluşturulduğunda frekans bir kez daha düştü.
Daha sonra Android uygulaması 320 MHz'de sabit bir frekansa ulaşırken, React Native uygulamasının frekansında bazı düşüşler yaşandı. Daha sonra, kullanıcı bir bütçe kaydettiğinde ve bütçe listesine yönlendirildiğinde, bütçe düzenleme ve bütçe kaldırma işlemleri yapılırken frekans bir kez daha düştü ve testin geri kalanı boyunca artmaya ve azalmaya devam etti, bu da önceki sahnelerin yeniden oluşmasına neden oldu. -birkaç küçük değişiklikle render alıyorum.
Hem Android hem de React Native uygulaması için ortalama frekans 291 MHz idi.



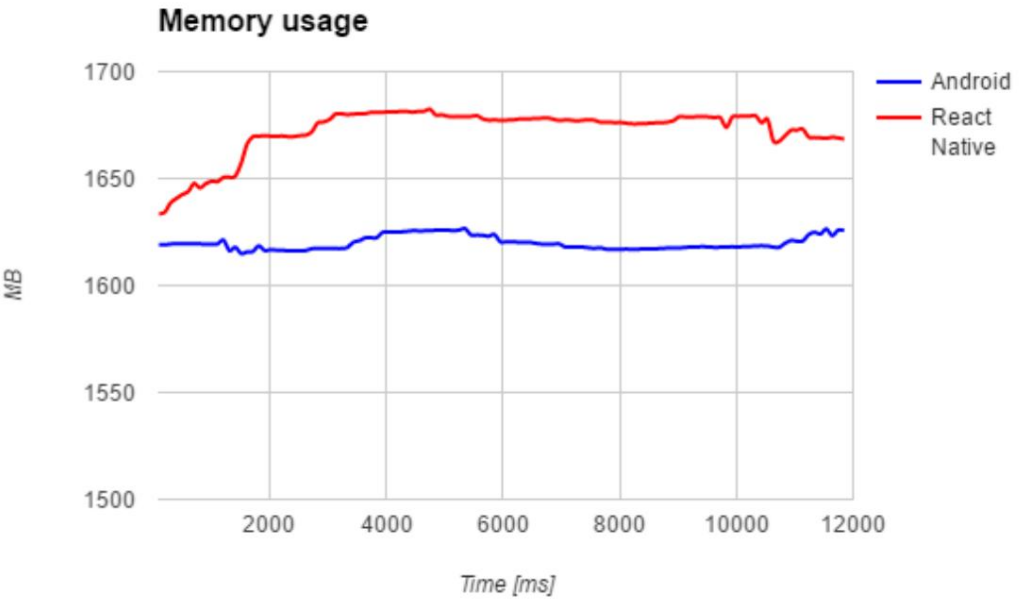
Şekil 4.9: Bütçeleri yönetirken GPU frekansı

Android ve React Native uygulaması için CPU yükü şekil 4.10'da gösterilmektedir. Önceki testte olduğu gibi, grafik düzensizdir ancak testin başında yüksek bir CPU yükü gösterir; bütçe listesi görüntülendikten ve yeni bütçe ekleme düğmesine henüz basılmadıktan sonra çok büyük bir düşüş görülür. Daha sonra, kullanıcı alanlara ilişkin değerleri düzenlerken yük azaldı, ancak kullanıcı bu bütçeyi kaydettiğinde, düzenlediğinde ve sildiğinde arttı. Ayrıca grafik, Android uygulamasının CPU yükünün React Native'den biraz daha az olduğunu gösteriyor; bu, Android uygulamasının React Native'de oluşturulan uygulama için %36,14 ve %41,64 ortalama yüke sahip olmasıyla da doğrulanıyor.



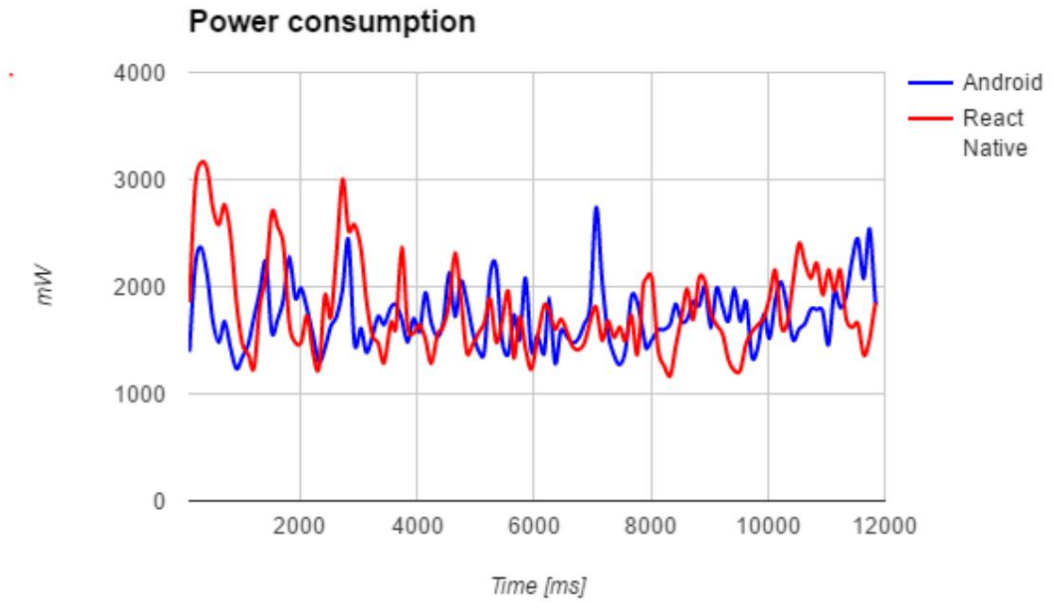
Şekil 4.10: Bütçeleri yönetirken CPU yükü

Aşağıda şekil 4.11'de görülebilecek ikinci testteki bellek kullanımı, tüm değerlendirme boyunca her iki uygulamanın da benzer bir kullanıma sahip olması nedeniyle ilk teste benzerdi. En belirgin farklar, testin başlangıcında React Native uygulamasının kullanımının artması ve Android uygulaması için ortalama 1620 Megabayt, React Native uygulaması için ise 1672 Megabayt ortalama bellek kullanımıyla genel olarak daha fazla kullanımdır.



Şekil 4.11: Boş uygulamaların bellek kullanımı

Güç tüketimi, ilk drenajla ilgili olarak ilk testten farklıydı çünkü ilk testte başlatılması gerekiyordu, ikinci testte ise ölçümler başladığında uygulama zaten çalışıyordu. Şekil 4.12'deki grafik, önemli olaylarda ve gerçekleştirilen eylemlerde güç tüketimindeki artışları göstermektedir. Başlangıçta bütçe listesi oluşturulduğunda ve 3 saniye sonra bütçe eklendiğinde drenaj arttı. Daha sonra tüketimde düşüşler ve eğilimler kaydedildi ve 7 saniye sonra bütçeler eklendiğinde, yeni bütçe kaydedilip bütçe listesi görüntülendiğinde Android uygulamasının drenajı çok yüksekti. Son olarak, bütçe düzenleme sahnesi görüntülenirken React Native uygulamasının 10,5 saniyede bir artış gösterdiği ve Android uygulamasının bütçe silindiğinden 11 saniye sonra daha yüksek bir tüketime sahip olduğu testin sonunda daha yüksek bir drenaj görülebilir. . Özetlemek gerekirse Android uygulamasının ortalama güç tüketimi 1749 miliwatt, React Native uygulamasının ise 1803 miliwatt düzeyindeydi.



Şekil 4.12: Boşta kalan uygulamaların güç tüketimi

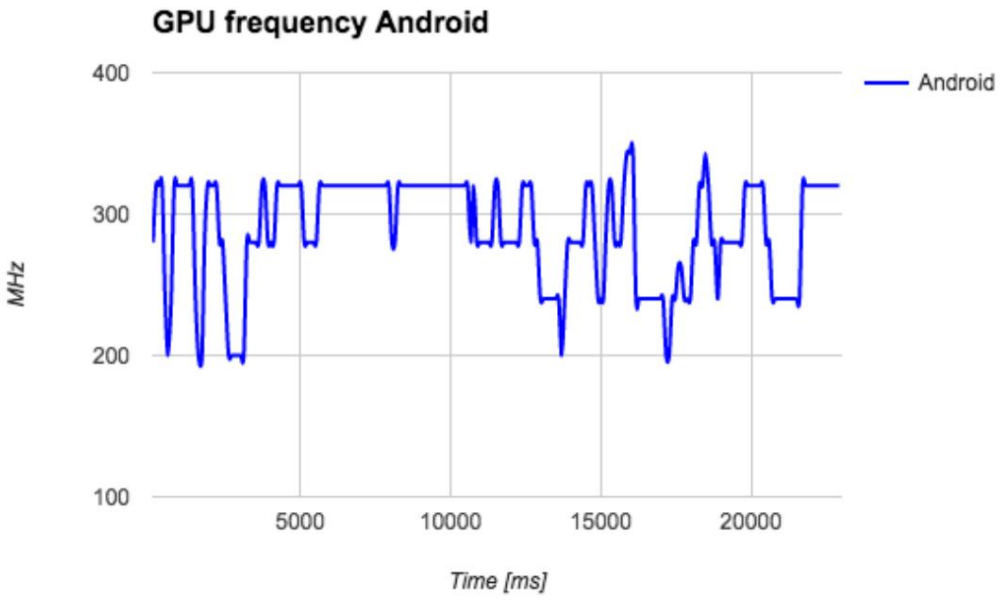
4.2.3 Üçüncü test

Üçüncü ve son test durumu da bölüm 3.3.2'de açıklanmaktadır ve kullanıcının belirli değerlerle yeni bir işlem eklemesini, tarihi değiştirmesini, bir makbuz yakalamasını ve son olarak işlemi kaydetmesini içermektedir. Ancak bu test, iki uygulamanın yürütme süresi açısından önemli bir farka sahipti. Android uygulaması tüm eylemleri 23 saniyenin altında gerçekleştirebilirken, React Native uygulaması daha yavaştı ve yaklaşık 30 saniyede tamamlandı.
Uygulama süresindeki fark, takvim açıldıktan sonra ortaya çıkar ve bu nedenle, farklılıklar nedeniyle testlerin sonuçları aynı grafiklerde görüntülenememektedir ve şekil 4.13'te görülebileceği gibi belirli eylemlerin zaman damgaları iki uygulama için farklıdır. .

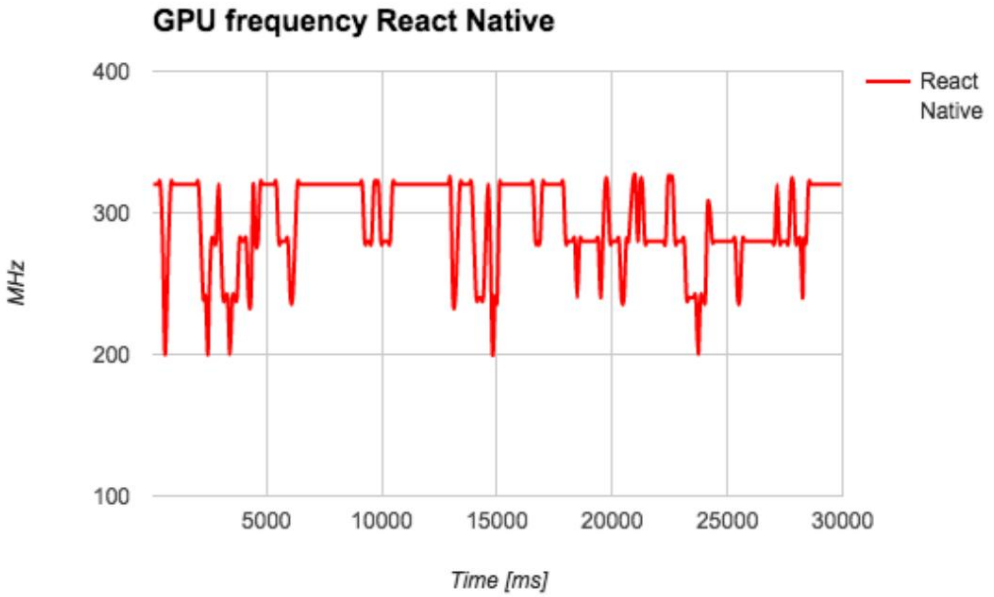
Action	Time Android[s]	Time React Native [s]
Navigate to transactions	1.5	1.5
Pressing "Add transaction" icon	2	2
Open calendar	13	13
Open camera	16	17.5
Take picture	19	23
Save transaction	21	27.5

Şekil 4.13: Üçüncü performans testi için yaklaşık zaman damgalarında gerçekleştirilen eylemler

Şekil 4.14 ve 4.15 üçüncü test sırasında uygulamaların GPU frekansını göstermektedir.
Her iki uygulama da başlangıçta işlemler için sahnelerin oluşturulması, yeni bir işlem eklenmesi ve 13 saniye sonra takvimin açılması nedeniyle frekans düşüşleriyle aynı başlangıç ölçümlerine sahipti. Android uygulamasını gösteren grafik, 16 saniye sonra kamera açıldığında ve görüntü yakalandığında frekansın nasıl 200 MHz'e düştüğünü gösteriyor. Ayrıca, yeni işlem kaydedildiğinde ve işlemler listesine yönlendirildiğinde GPU frekansı azaldı. React Native uygulaması aynı yapıya sahiptir ancak tablo 4.13'te görülebilen ilgili zaman damgalarıyla. Android uygulamasının ortalama GPU frekansı 289 MHz iken React Native uygulamasının frekansı 294 MHz oldu.

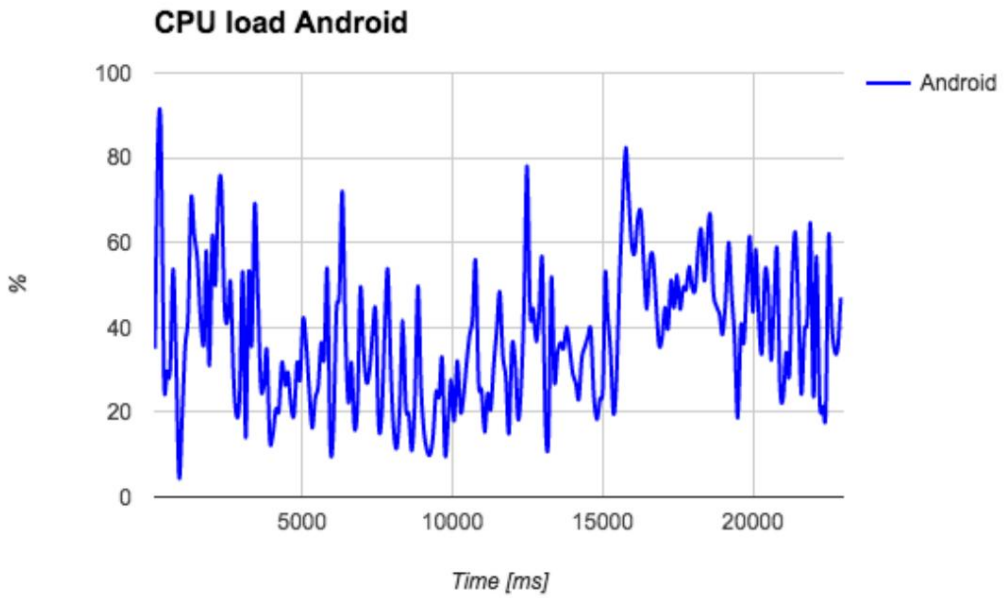


Şekil 4.14: İşlem gerçekleştirilirken Android uygulamasının GPU frekansı

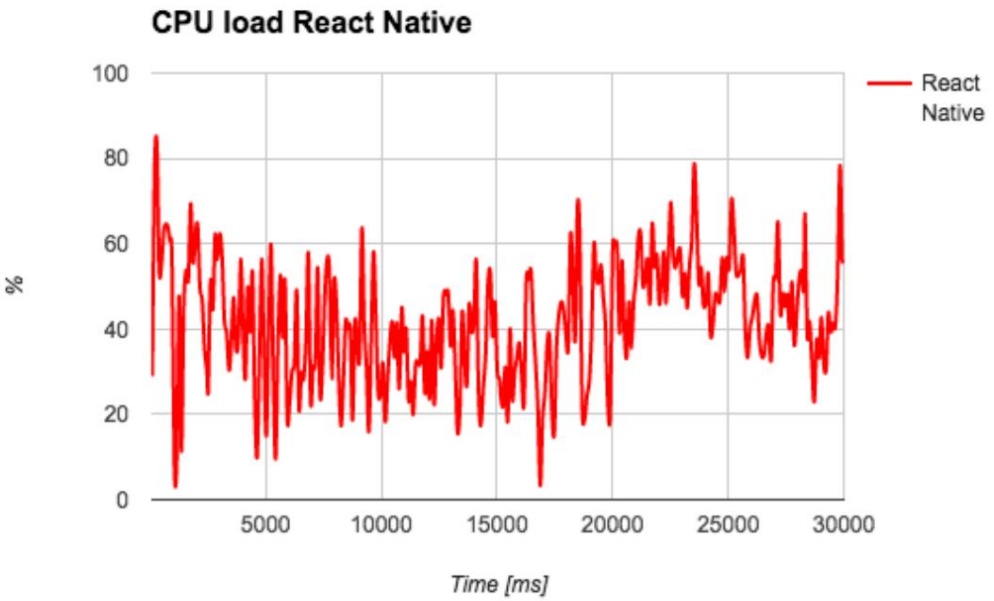


Şekil 4.15: İşlemi gerçekleştirirken React Native uygulamasının GPU frekansı

Android uygulamalarının CPU yükü şekil 4.16'da görülebilir ve şekil 4.17, React Native uygulamasının yükünü gösterir. İki grafiğin karşılaştırılması zordur ancak öne çıkan iki değer vardır. İşlemlere geçiş gerçekleşmeden önce CPU yükünde büyük bir düşüş yaşandı. Ayrıca Android uygulamasında 16 saniye sonra ve React Native uygulamasında 20 saniye sonra CPU yükünde önemli bir genel artış oldu. Sonuç olarak Android uygulamasının ortalama CPU yükü %37,88 iken React Native uygulamasının ortalama %41,76 olduğu görüldü.



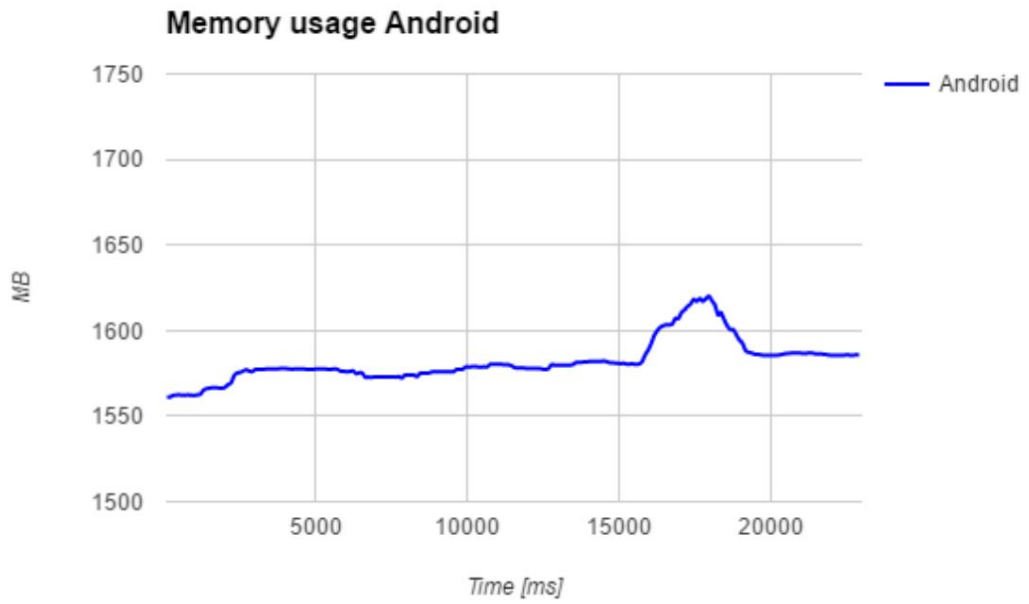
Şekil 4.16: İşlemi gerçekleştirirken Android uygulamasının CPU yükü



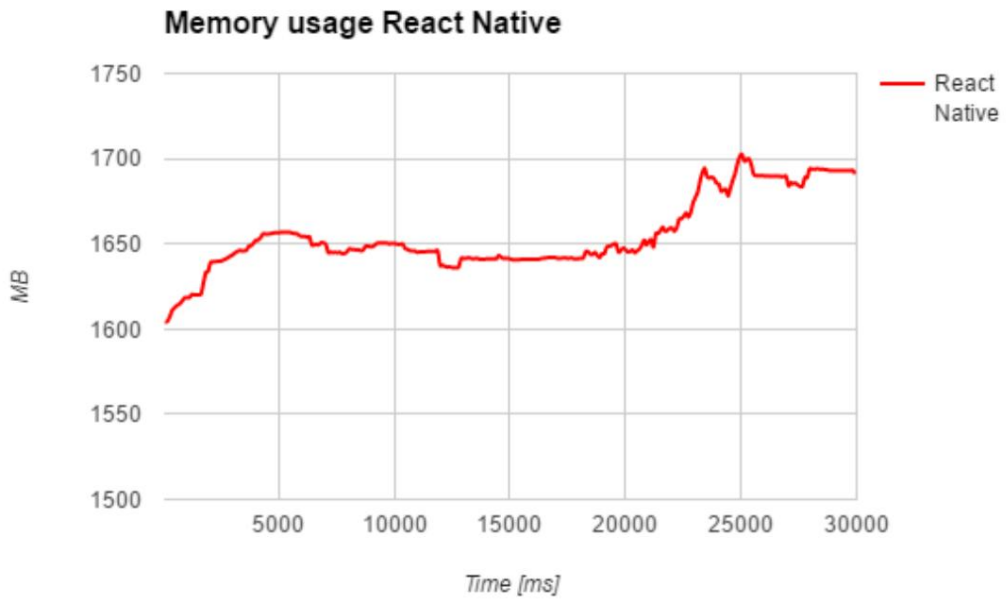
Şekil 4.17: İşlemi işlerken React Native uygulamasının CPU yükü

Şekil 4.18 ve 4.19 iki uygulamanın bellek kullanımını göstermektedir. Android uygulamasının bellek kullanım yaşı, 1560 MB civarından başlayarak zamanla yavaş yavaş arttı ve
1586 MB değerine ulaşır. Ancak testin başlangıcında biraz daha büyük bir artış oldu ve yaklaşık 16 saniye sonra büyük bir geçici artış oldu.
bir kez daha düşmeden önce iki saniye süren kullanım. Bellek kullanımı

React Native uygulaması da zamanla arttı ancak aynı seviyede veya kararlılıkta değil. Kullanım 1600 MB'tan başladı ve test sonunda yaklaşık 1700 MB'a ulaştı. Ayrıca bellek kullanımı iki saniye sonra ve bir kez daha 23 saniye sonra büyük ölçüde arttı.
Android uygulamasının ortalama bellek kullanımı 1582 MB, React Native'de oluşturulan uygulamanın ise 1655 MB oldu.

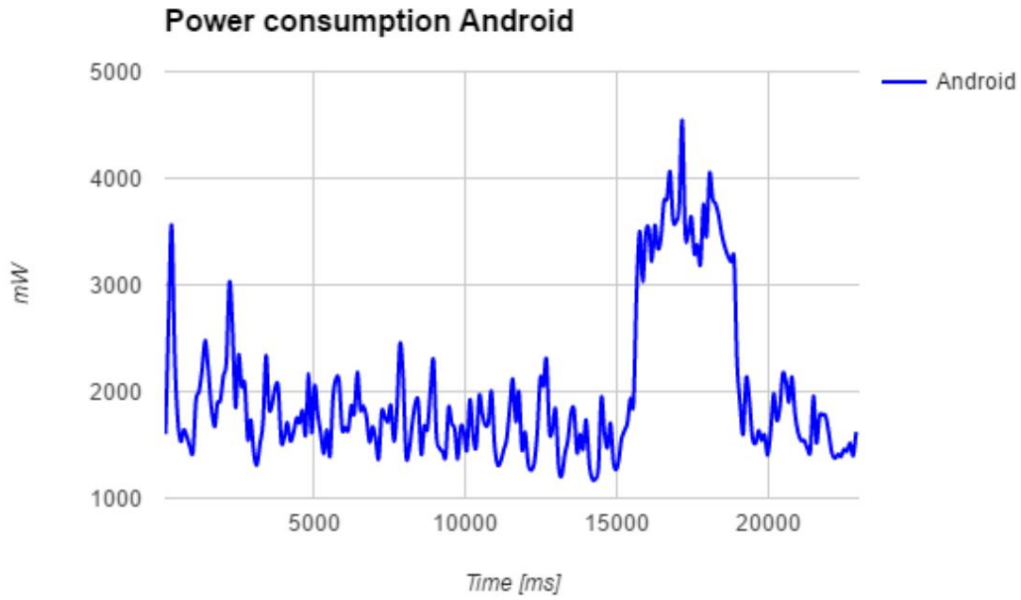


Şekil 4.18: İşlemi gerçekleştirirken Android uygulamasının bellek kullanımı

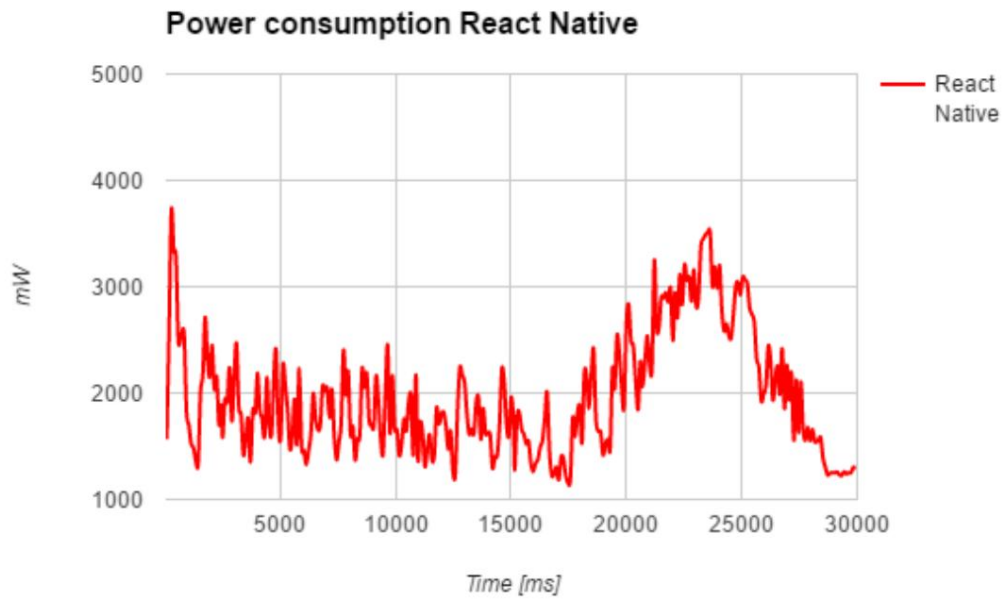


Şekil 4.19: İşlemi işlerken React Native uygulamasının bellek kullanımı

Son olarak Android ve React Native uygulamasının güç tüketimi şekil 4.20 ve 4.21'de görülebilir ve her ikisi için de eşdeğer olan modeli açıkça gösterir. Testin başlangıcında, testi gerçekleştirmek için uygulamaya geçildiğinde güç tüketimi aniden arttı. Daha sonra, kamera açılıncaya kadar drenaj aniden yükseldi ve düştü ve tüketim, Android uygulaması için yaklaşık 16 saniye ve React Native uygulaması için 20 saniye sonra büyük ölçüde arttı. Fotoğraf çekildiğinde güç tüketimi önceki seviyelere düştü. Android uygulamasının ortalama drenajı 1992 mW, React Native uygulamasının ise 1972 mW oldu.



Şekil 4.20: İşlem gerçekleştirilirken Android uygulamasının güç tüketimi



Şekil 4.21: React Native uygulamasının işlem gerçekleştirilirken güç tüketimi

5. Tartışma

Bu bölüm daha önce sunulan sonuçları özetlemekte ve tartışmakta ve toplanan verilerin nedenini açıklamaktadır. Ayrıca tezde kullanılan yöntemler incelenip açıklanmakta ve yapılan seçimlerin nedenleri açıklanmaktadır. Son olarak React Native uygulamasının gelişimini, bu sürecin nasıl hissettirdiğini ve genel olarak sağladığı hissi sonuçlandırıyor ve açıklıyor.

5.1 Sonuçlar

Önceki bölümde hem çoğaltma için kullanıcı testinden hem de performans testlerinden toplanan veriler görüntülenir. Sağlanan sonuçlar aşağıda tartışılmaktadır.

5.1.1 Çoğaltma testi

Valtech'teki çalışanların iki uygulamayı test ettiği ve React Native ile hangi uygulamanın oluşturulduğunu seçme görevinin verildiği bölüm 3.3.1'de açıklanan çoğaltma testinin sonuçları başarılı oldu. Sonuçlar, kullanıcıların çoğunun React Native uygulamasını küçük farklılıklar (özellikle sahneler arasındaki geçişler) nedeniyle ayırt edebildiği ancak kullanıcıların React Native uygulamasını kullanmaktan çekinmediği sonucuna vardı. Geçişlerle ilgili sorun, geçiş gerçekleştiğinde belirli bir animasyona sahip olan sahneler için Android uygulamasının Etkinlikler'i kullanmasının bir sonucuydu. React Native'de farklı geçişler değiştirilebilir ancak hiçbir animasyon, yeni bir Activity'nin yerel animasyonuyla aynı değildir.
React Native'de geçişler için kullanılan animasyon bu nedenle Android'deki Parçalar arasındaki geçişlere benzeyecek şekilde uygulandı. Ancak animasyon aynı değildir ve animasyonun kendisi yerel Parçalara göre biraz daha yavaştır ve bazı kullanıcılar bu tutarsızlığı yerel olmayan bir davranış olarak tanımlayabilir. React Native uygulamasının native bir Android uygulamasıyla tamamen aynı görünüme ve hisse sahip olmadığını açıkça gösteren bir sonuca rağmen, React Native uygulamasını kullanırken hala sorun yaşamayan kullanıcılar için yeterince iyiydi.

5.1.2 Performans testi

Bölüm 4.2'deki her testten görülebileceği gibi React Native uygulaması Android uygulamasıyla aynı performansa sahip değildir. Ancak fark beklenenden daha az ve üçüncü testte React Native uygulamasının ortalama güç tüketimi daha düşüktü. React Native uygulamasının native bir uygulama kadar iyi performans gösterememesinin nedeni oldukça açıktır, native bir yaklaşım hibrit ile kıyaslanmaktadır ve React Native bölüm 2.1.1'de açıklanan ART'ta oluşturulmamaktadır. React Native uygulamaları gömülü bir V8 örneğinde (iOS için JavaScriptCore) çalıştığından, uygulamalar aynı çalışma süresini kullanmaz ve bu nedenle, V8 motoru farklı sistemlerde çalışırken ART yalnızca Android için oluşturulduğundan performansta farklılıklar olacaktır. Android için optimize edilmemiş. V8 hızlı olmasına rağmen ART ile aynı sonuçları elde edememekte ve bu nedenle Android için biraz daha iyi değerlere sahip sonuçlar elde edilmektedir.[25]

İlk test

İlk test, uygulamaların başlatılması sırasındaki performansını ve arka planda çalıştırıldığında uygulamaların nasıl ele alındığını ölçtü. Dört veri noktasının tümü, Android uygulamasının daha iyi performansa sahip olmasıyla ve göze çarpan herhangi bir veri olmadan aynı sonucu görüntüler. Uygulamanın en fazla bellek ve güce uygulama başlatıldığında ihtiyaç duyduğu, GPU ve CPU'nun uygulama açıldığında en fazla kullanıldığı, uygulama kapatılıp arka planda çalıştırıldığında ise değerlerin düştüğü görülmektedir.

İkinci test

İkinci test, kullanıcının uygulamayla etkileşimine ilişkin daha ilginç bir sonuç görüntüler çünkü test, kullanıcının bir bütçe ekleyip daha sonra silmesi sırasında performansı ölçmüştür.
Şekil 4.9'daki GPU frekansı, kullanıcı ekranla etkileşim kurduğunda (örneğin kullanıcı bütçelere gittiğinde veya bütçe kaydedildiğinde) her iki uygulama için de GPU'nun nasıl düştüğünü gösterir. Bununla birlikte, frekans düşüşleri React Native uygulamasında daha sık meydana gelir, çünkü düşüşler, kullanıcı bir bütçe eklerken alanlara girdi sağladığında da gerçekleşir, Android uygulamasında düşmeler meydana gelmez. Üstelik React Native uygulamasında GPU frekansında 200 MHz değerinde daha fazla düşüşler mevcut ve en büyük fark bütçe düzenlendiğinde ortaya çıkıyor. Genel GPU sonucu, Android uygulamasının daha kararlı bir frekansa sahip olduğunu gösteriyor.

CPU yükü, React Native uygulamasının çoğu zaman tüm test boyunca biraz daha yüksek bir yüke sahip olmasıyla benzer bir sonuç gösteriyor. CPU yükünde, uygulamanın başlatıldığı ve kullanıcının bütçelere gitmesinden önceki 1 saniyeden sonra belirgin bir düşüş görülür. Kullanıcı yeni bir bütçe eklediğinde, CPU yükü artar ve ardından bütçe kaydedilene ve yeniden arttığı yere kadar azalır.

İkinci testin bellek kullanımında herhangi bir benzersiz olay yoktur. Ancak başlangıçta kullanımın oldukça benzer olduğu görülebilir ancak etkileşim başladıkça (kullanıcı 1,5 saniye sonra bütçelere gider) React Native uygulamasında bellek kullanımı ciddi şekilde artarken Android uygulaması boyunca aynı boyutta kalır. tüm test.

Güç tüketimi, iki uygulamanın performans farklılıklarını en iyi gösteren sonucu görüntüler. Testin başlangıcındaki üç ani artışın, uygulamalar açıldığında kullanıcının bütçelere gittiği ve yeni bütçe ekleme butonuna basıldığında her iki uygulama için de drenajın arttığı ancak React Native uygulamasının daha yüksek değerlere sahip olduğu görülmektedir. . Ancak yedi saniyeden sonra Android uygulaması için daha yüksek bir tüketim artışı yaşanıyor. Bu, uygulamanın bütçe tasarrufu sağladığı ve React Native için daha düşük tüketimin Realm kullanımından kaynaklandığı zamandır.

Android uygulaması için kullanılan teknik olan SQLite'tan çok daha hızlı ve verimlidir.[29]

Üçüncü test

Üçüncü ve son test, önceki teste benzer sonuçlar verdi. Bu test, bölüm 3.3.2'de belirtildiği ve daha ayrıntılı olarak açıklandığı gibi, bir kullanıcı sağlanan tüm değerlerle ve çekilen bir görüntüyle bir işlem eklediğinde performansı ölçtü. En büyük fark, veri noktalarından alınan değerler değil, eylemleri gerçekleştirmek için gereken süreydi. Android uygulaması daha hızlıydı ve yalnızca yaklaşık 23 saniye sürerken, React Native uygulaması yaklaşık 30 saniyede tamamlandı. Bunun nedeni React Native uygulamasının bazı bölümlerinin yerel uygulama kadar hızlı işlenememesi ve görüntülenememesiydi. Şekil 4.13'te görüldüğü gibi kullanıcı her iki uygulama için de takvimi açana kadar tüm işlemleri benzer zamanlarda gerçekleştirebilmiştir. Ancak takvim bileşeni uygulamanın ilk kısmıydı ve React Native uygulamasında biraz daha yavaştı çünkü Android uygulaması işlemi gerçekleştirirken uygulamanın takvimi açması, yeni bir tarih seçmesi ve diyaloğu kapatması yaklaşık 4,5 saniye sürüyordu. aynı görevleri yaklaşık 3 saniyede tamamlar. Daha sonra kullanıcı, kamerayı açan bir düğmeye bastı ve ardından kaydedilen görüntüyü kaydetti, kamera kapatıldı, kullanıcı bir önceki sahneye yönlendirildi ve ardından işlem kaydedildi. Android uygulamasında bu işlem yaklaşık 5 saniye sürerken, React Native uygulamasında aynı eylemleri gerçekleştirmek için 10 saniyeye ihtiyaç duyuldu. Aradaki fark çok fazla ve bunu yapmanın iki kat daha fazla zaman alması, React Native uygulamasında kullanılan kamera bileşeninin harici bir bileşen olmasıydı. Facebook kendi kamerasını oluştursaydı, kamerayı başlatmak ve kullanmak için gereken süre çok daha az olabilirdi.

Üçüncü test için GPU frekansı daha fazla bilgi sağlamadı çünkü sahneler oluşturulduğunda frekansın düştüğünü ve genel frekansın React Native uygulaması için daha yüksek olduğunu da doğruladı.

CPU yükü aynı sonuçları sağladı ve Şekil 4.16 ve 4.17'de, Android uygulaması için 16 saniye sonra ve React Native uygulaması için 17,5 saniye sonra kamera başlatıldığında CPU'nun en fazla yüke ihtiyaç duyduğu görülebilir. İlginç bir değer, takvim 13 saniye sonra açıldığında Android uygulamasının CPU yükünün neredeyse %80'e ulaşmasıdır, ancak bu olay React Native uygulaması için herhangi bir benzersiz değer sağlamamaktadır. Bunun nedeni Datepicker'ın React Native uygulamasında uygulanma şeklidir. Bileşen, bütçe ekleme sahnesi oluşturulduğunda ve kullanıcı tarihi değiştirmek için metne bastığında görünmez bir modelin içinde oluşturulur, model basitçe görünür olarak ayarlanır. Bu, takvim açıldığında gereken CPU yükünün, o noktada Datepicker'ı başlatan Android uygulaması için daha fazla olduğu anlamına gelir. Tarih bileşeninin daha erken başlatılması, React Native uygulaması için AddTransaction sahnesinin oluşturulmasında genel CPU yükünün daha yüksek olmasının bir nedenidir.

Bellek kullanımı, belleğe ilişkin diğer testlerle benzer bir sonuç görüntüler. Şekil 4.19'da kameranın bir görüntü yakalayıp yüksek hızda devam etmesi durumunda kullanımın nasıl arttığı gösterilmektedir. Görüntünün çekildiği 23 saniye sonra bir ani yükseliş görülebilir ve 25 saniye sonraki ani artış ise görüntünün mobil cihaza kaydedildiği zamandır. Ancak Android uygulamasının bellek kullanımı şekil 4.18'de görüldüğü gibi aynı modeli izlemedi. Kullanım, görüntü çekildiğinde artmadı ancak kamera açıldığında ve ardından Android uygulaması, görüntü çekildikçe kullanımı hızla düşürmeyi başardı.

Ölçülen son veri noktası güç tüketimiydi ve her iki uygulamanın da uygulamalar başlatıldığında ve sahneler oluşturulduğunda nasıl daha fazla güce ihtiyaç duyduğunu gösteriyor (örneğin, bir işlem ekleme sahnesi yaklaşık 2 saniye sonra görüntülendiğinde). React Native uygulamasındaki biraz daha yüksek değerler dışında tek fark, kamera açıldığında drenajdır. Android uygulamasında muazzam bir artış yaşandı

15 saniye sonra yaklaşık beş saniye süren ve 4540 mW'lık bir tüketim değerine ulaşırken, React Native uygulamasının yaklaşık 10 saniyelik daha uzun bir zaman aralığı vardı ancak çok daha az tüketim vardı (en yüksek değer 3504 mW idi). React Native uygulamasının kamera kullanımı sırasında daha düşük drenaja sahip olmasının nedeni, React Native için kullanılan kameranın daha önce de belirtildiği gibi harici bir bileşen olması ve yalnızca temel işlevsellik desteğine sahip olmasıydı. React Native'de kullanılan kamera yalnızca görüntü yakalama desteğine sahipken, Android kamerada kamera değiştirme, flaşı etkinleştirme, yakınlaştırma ve farklı ayarlar desteği bulunuyordu. Dolayısıyla bu kameranın kullanımı daha fazla güç kullanılmasını gerektirdi ve şekil 4.20'de görülebilecek ani artışa neden oldu.

Daha önce de belirtildiği gibi, üçüncü testteki ortalama güç tüketimi React Native için daha düşüktü ve React Native için avantaj sağlayan tek test ve veri noktasıydı.
Öncelikle, Android uygulamasının yaklaşık yedi saniye daha hızlı olması ve React Native uygulamasının toplam tüketiminin çok daha fazla olması nedeniyle üçüncü test için ortalama değerlerin adil bir görünüm sağlamadığı iddia edilebilir. Ancak React Native uygulamasında ortalama tüketimin daha az olmasının nedeni, Android'de kullanılan kameraya göre daha az desteğe sahip ve daha az karmaşık olan kameranın örneklemesi ve kullanımından kaynaklanıyordu. Kullanılan harici kamera bileşeninin temel özelliklerden daha fazla özelliği desteklememesi ve Android'deki sınırlı bilgi nedeniyle orijinal uygulamanın daha az işlevselliğe sahip bir kamera kullanacak şekilde değiştirilmemesi.

5.2 Yöntem

Bölüm 3.2, React Native uygulamasının nasıl geliştirildiğini açıklar ve geliştirme aşamasında meydana gelen sorunları ve düzeltmeleri gösteren kod parçalarını içerir. Bu izlenecek yol, Github'daki her iki uygulamanın deposuyla birlikte uygulamaların ve testlerin çoğaltılmasına yardımcı olacaktır. Ancak React Native, sürekli olarak güncellenen ve yamalanan yeni bir çerçevedir. Bu, uygulamanın bazı bölümlerinin yakın gelecekte kullanımdan kaldırılmasına neden olabilir ve karşılaşılan hatalar da (listeyi güncelleme sorunu gibi) giderilebilir. Ayrıca uygulamada kullanılan bazı bileşenler haricidir ve Facebook, çerçeveyi geliştirmelerine yardımcı olmak için açık kaynak topluluğunu kullandığından, bu bileşenler gelecekte çerçeveye eklenebilir. Eğer bu yapılacaksa, bileşenlerin kendileri daha fazla işlevsellik içerebilir ancak her şeyden önce daha iyi performansa sahip olabilir. Üstelik kopyalama testinin sonuçlarının temel nedeni geçişler arasındaki animasyonlardı. React Native geliştirilme aşamasında olduğundan, yerel bir uygulamayla aynı görünüme ve hisse sahip animasyonlar sağlamak amacıyla çerçevenin bu kısmı düzeltilebilir. Yukarıda bahsedilen olasılıklardan dolayı gelecekte aynı uygulamalar test edilse bile sonuçlar farklı olabilir.

5.2.1 Mevcut bir Android uygulamasını kullanma Öncelikle iki uygulamayı

karşılaştırmak için halihazırda mevcut bir Android uygulaması elde edildi. Bunun temel nedeni geliştirme süresinden tasarruf etmek ve geliştirmeyi React Native'e odaklamaktı. Bitmiş bir Android uygulamasına sahip olunduğunda, geliştirme süresinin çoğu bir React Native uygulaması oluşturmaya odaklanabildi ve daha eksiksiz bir uygulamanın geliştirilmesine olanak tanındı. Ancak Android uygulamasının önceden belirlenmiş işlevsellik ve özelliklere sahip olması nedeniyle uygulamanın tamamını kopyalamak zordu. Örneğin, orijinal uygulama kullanıcının mevcut bütçeleri ve işlemleri bir csv dosyasına kaydetmesine ve daha sonra bunu içe aktarmasına olanak tanıyordu. Bu özelliğin daha gerçek bir kullanım deneyimi sağlayamayacağı ve geliştirilmesi zaman alacağı için özellik Android uygulamasından kaldırıldı.

Ayrıca Android uygulamasının sıfırdan geliştirilmemesi nedeniyle orijinal uygulamanın pek çok kısmı bir Android uygulamasının nasıl tasarlanması gerektiği ile ilgili değildi. Bu durumun, React Native'den zaman alan Android uygulamasında da düzeltilmesi gerekiyordu.

gelişim. Ancak mevcut bir Android uygulamasını kullanmaya karar verirken, bahsedilen gibi hatalar ortaya çıkabileceği için uygulamanın açık kaynaklı olması gerektiği açıktı.
Kaynak kodun mevcut ve düzenlenebilir olması sayesinde, orijinal uygulama için zorunlu olan hiçbir özelliği kaldırmadan, iki benzer uygulama oluşturmak amacıyla uygulamayı değiştirmek kolaydı. Android uygulaması açık kaynak olmasaydı, oluşturulan React Native uygulaması, resmi Android geliştirme belgelerinde önerilmeyen özellikleri ve tasarım seçeneklerini içermek zorunda kalacaktı.

Ayrıca Android uygulaması sıfırdan geliştirilmediğinden, Android uygulaması React Native uygulamasına uyacak şekilde tasarlanıp oluşturulmamıştır. Taraflı bir sonuç sağlayacak bir React Native uygulaması gibi görünmek ve hissetmek için Android uygulaması geliştirilebilir. Ancak mevcut bir uygulamayı seçerek tez, bir Android uygulaması gibi görünecek ve hissettirecek bir React Native uygulaması geliştirmeyi daha doğru bir şekilde deneyebilir.

5.2.2 iOS

React Native'in bu tezde değerlendirilmeyen veya incelenmeyen büyük bir kısmı, hem Android hem de iOS uygulaması oluşturma yeteneğidir. Aşağı yukarı aynı koddan iki uygulama oluşturma yeteneği React Native'in en iyi özelliklerinden biridir ancak bu tez yalnızca Android yarısına odaklanmıştır. React Native uygulamaları Android uygulamasıyla aynı performansa sahip olmasa da uygulamanın kodunun büyük bir kısmı başka bir uygulama için kullanılabildiği için bu durum adil olmayan bir sonuç doğurabilir. Ancak bu tez iki temel nedenden dolayı bu hususu kapsamamıştır. Öncelikle başka bir native iOS uygulaması oluşturmak ve ardından bu uygulama için bir React Native uygulaması oluşturmak zaman alıcı olacaktır. Üstelik hem Android hem de iOS için aynı görünen birçok popüler uygulama var ancak açık kaynak kodlu bu işletim sistemleri için iki uygulama bulmak zor, hatta imkansız. Bu, bu projenin sıfırdan bir Android ve iOS uygulaması oluşturmasını gerektirecektir ve bu da yukarıda belirtilen yöntem seçenekleriyle çelişecektir. Son olarak React Native'in yalnızca Android kısmına odaklanılarak çerçeveye yönelik daha eleştirilen bir bakış açısı kullanıldı. Çerçevenin kullanımı kolaysa ve Android için yerel bir uygulama kadar iyi bir uygulama oluşturabiliyorsa, kodu iOS'ta birkaç değişiklikle derleme yeteneği de büyük bir bonus olarak görülecektir.

5.2.3 Literatür

Son olarak, teori ve metodolojide görülebileceği gibi (özellikle bölüm 2.4'teki React Native teorisine ilişkin olarak) kullanılan bilimsel literatür eksikliği bulunmaktadır. Bunun nedeni basit; Bonnie Eisenman[8] tarafından yazılan ve çerçeveyle ilgili bilgiler içeren tek bir kitap var ve React Native hakkında genel olarak bilimsel rapor eksikliği var. React Native yeni bir çerçeve olduğundan, yazarlar henüz tam kitap yazamadılar; bu da React Native hakkında bilgi edinmek için mevcut kaynakların ya belgeleri okuyarak ya da James Long'un köprüyle ilgili yazısı[20] olarak blog yazılarını okuyarak sonuçlanmasını sağladı. Ancak React Native ile ilgili geliştirme prosedürünün ve teorinin kazanılması gerektiğinden, bu tezin yürütülebilmesi için kullanılması gereken kaynaklar bu kaynaklardı. Öncelikli olarak mevcut az sayıdaki yayınlanmış kitap ve bilimsel raporlardan yararlanılmış, bunların yeterli bilgi vermediği durumlarda ise dokümanlara ve blog yazılarına ulaşılarak kullanılmıştır.

Blog yazılarını veya benzeri çevrimiçi kaynakları kullanırken kaynağın mümkün olduğunca güvenilir olduğundan emin olmak önemlidir. Bu öncelikle yazarın ne gibi deneyimler yaşadığı araştırılarak yapıldı. React Native ile deneyimi olan veya çerçevenin geliştiricilerinden biri olan bir yazar, daha güvenilir bir gönderiyle sonuçlanır. Ancak kişinin yine de içeriği yanlış yorumlamış olabileceği veya kimlik bilgilerini taklit etmiş olabileceği için bundan emin olunamaz.
Daha sonra gönderinin kaynağı araştırıldı. Gönderinin yayınlandığı web sitesine bakarak kaynağın az çok güvenilir olup olmadığını öğrenmek mümkündü. İle

yalnızca popüler teknoloji blogları veya benzerlerinde yayınlanan gönderileri seçmek, kaynağın güvenilirliğini artırdı. Son olarak, güvenilir bir kaynak bulunduğunda, bilginin React Native'in belgeleriyle ilişkili olup olmadığının bulunması gerekiyordu. Tutarsızlıklar olması durumunda, başka yanlış bilgiler içerebileceğinden kaynak kullanılmamıştır. Ancak tüm bu önlemler alınmış olsa bile kaynaklar tam olarak doğru olmayabilir ve yazıları okurken dikkat edilmesi gereken bir husustur.

5.2.4 Çoğaltma

React Native'in bir Android uygulamasını ne kadar iyi kopyalayabildiğini test etmek için insanlara iki uygulamayı test etmelerine ve hangisinin React Native uygulaması olduğunu tahmin etmeye çalışmalarına izin verildi. Bu testin kullanıcıları, genel olarak Android geliştirme ve uygulamalar konusunda deneyime sahip danışmanlar olan Valtech çalışanlarıydı. Piyasadaki uygulamaların gerçekliğini yansıtacak bir sonuca ulaşmak için sıradan ve rastgele insanlardan faydalanılmalıdır. Ancak bu tez, daha önce de belirtildiği gibi, React Native'e yönelik eleştirilen bir bakış açısına sahiptir ve bu nedenle BT danışmanlarını kullanmayı tercih etmiştir. Bu kişiler neyi arayacaklarını bilirler ve bir uygulamanın hatalı davranan veya hatalı görünen kısımlarını, uygulamanın normal kullanıcının hiç fark etmeyeceği veya umursamayacağı kısımlarını ayırt edebilirler. Profesyonellerin uygulamaları değerlendirmesine olanak tanıyarak, elde edilen sonuçlar deneyimli kullanıcılardan elde edilen içgörülerdir ve son derece olu
React Native uygulaması replikasyon testinde şaşırtıcı derecede iyi performans gösterdiğinden, çerçeve deneyimli eleştirmenler tarafından onaylandı ve aksi takdirde fark edilemeyecek kusurlar yine de testi geçti.

Ayrıca replikasyon testi için kullanılan uygulamalar kapsamlı değildi ve fazla işlevsellik içermiyordu. Budget Watch küçük bir uygulama olduğundan kullanıcılar pek çok özelliği test edemedi. Ancak test edilmesi gereken uygulamanın özellikleri değil, uygulamanın nasıl göründüğü ve hissettirdiğiydi. Uygulama boyutuna rağmen Android uygulamalarında sıklıkla kullanılan birçok bileşeni bünyesinde barındırıyordu. Seçilen uygulamanın çok fazla özelliğe sahip olmamasının nedeni, React Native uygulaması geliştirmenin zorluğunu bilmenin zor olması ve daha gelişmiş bir uygulama oluşturmak için zaman olup olmayacağının bilinmemesiydi.

5.2.5 Performans

Tez ayrıca, CPU yükü, güç kullanımı vb. gibi farklı veri noktalarına ilişkin verileri kaydetmek için Trepn Profiler kullanarak React Native uygulamasının performansını da ölçtü. Ancak bu sonuçları farklı senaryolarda kaydedebilmek için testlerin manuel olarak yapılması gerekiyordu. .
Her iki uygulama için de aynı etkileşimi ve yürütme süresini elde etmek amacıyla çok sayıda test kaydedilmiş olsa da, kullanıcının tıklayıp değerleri yazması yerine otomatik testlerin uygulama için istenen eylemleri gerçekleştirmesi sağlanarak daha doğru bir sonuç elde edilebilirdi. . Bu, özellikle bu eylemlerin gerçekleştirilme zamanının iki uygulama arasında çok farklı olduğu üçüncü testte, sonuçlarda rol oynayabilecek testler için insan faktörünü ortadan kaldıracaktır. Ancak otomatik testlerin kullanılmamasının ana nedeni, uygulamaların sürüm modunda değil hata ayıklama modunda olmasını gerektirmesiydi. Hata ayıklama modu daha fazla geliştirme özelliği sunar ancak uygulama, sürüm moduna getirildiğindeki kadar iyi performans göstermez. Her iki uygulama da hata ayıklama aşamasında olsa da, React Native uygulaması bazen inanılmaz derecede yavaş olduğundan bunun uygulamaları ne kadar etkileyeceği belirsizdir. Ayrıca uygulamaların yayın modunda olmaması, gerçek bir pazar senaryosunu ve gerçek kullanıcıların uygulamalarla nasıl etkileşimde bulunacağını yansıtmayacaktır. İnsan hatalarından etkilenmeyen değerlerin alınabilmesi için testler, optimum zaman aşımına sahip üç hatasız durum kaydedilene kadar birçok kez gerçekleştirildi.

Ayrıca performans, GPU frekansı, CPU yükü, bellek kullanımı ve güç tüketimine odaklanılarak ölçüldü. Ancak GPU frekansı grafiği kapsasa bile-

Kullanıcı arayüzünün kullanıcı için ne kadar iyi görüntülendiğini ve yeniden oluşturulduğunu göstereceğinden, uygulamaların FPS'sini (saniyedeki kare sayısı) ölçmek de çok ilginç olacaktır. Ancak bunu ölçmek için ölçümün koda eklenmesi gerekiyordu ve uygulamaların hata ayıklama modunda olması gerekiyordu ki bu daha önce belirtildiği gibi bir seçenek değildi.
Son olarak, daha önce tartışıldığı gibi, daha fazla işlevselliğe ve özelliğe sahip daha büyük bir uygulama kullanılmış olsaydı, test senaryoları daha fazla eylemi kapsayacağından performans ölçümlerindeki farklılıklar muhtemelen daha büyük olurdu.

Mobil cihazın farklı bileşenlerinden performans elde etmek amacıyla bölüm 3.4.1'de anlatılan Trepn Profiler kullanılmıştır. Elde edilen değerlerin donanım ve diğer uygulamaların ölçümlerine dayalı tahminler olması nedeniyle, profil oluşturma için uygulamalar kullanılırken bazı endişeler vardır. En doğru sonucu alabilmek için diğer tüm uygulamalar sonlandırıldı ve herhangi bir müdahaleyi ortadan kaldırmak için mobil cihaz uçuş moduna alındı, bu da uygulamanın internet bağlantısı gerektirmemesi nedeniyle mümkün oldu. Ayrıca geçerli bir sonuç alabilmek için tüm testlerde aynı mobil cihaz kullanıldı ve iki farklı uygulama ara sıra test edildi.

5.3 Geliştirme

Bu tezin temel amacı ve ilk araştırma sorusunun bir kısmı, React Native'deki gelişimi ve çerçeve için mevcut desteği değerlendirmekti. Her şeyden önce, React Native'in dokümantasyonu olağanüstü. Yeni bir proje oluşturmak çoğu zaman zorlu bir süreç olabilir ve ortamda sıklıkla sorunlar ortaya çıktığı için zaman alıcı olabilir.
Bununla birlikte, iyi yazılmış dokümantasyon nedeniyle React Native'in başlatılması kolaydı ve dokümantasyonun hedef işletim sistemi ve geliştirme işletim sistemiyle birlikte sağlanmasıyla özel bir adım adım sayfa döndürülür ve birkaç dakika içinde geliştirme başlayabilir.
Ayrıca belgelerde farklı bileşenlerin nasıl kullanıldığı, bunlar üzerinde nasıl düzeltmeler yapılabileceği ve aynı zamanda bunların uygun şekilde nasıl kullanılacağına ilişkin kod örnekleri de yer almaktadır. Dokümantasyonda anlaşılması zor olan ve anlamak için bir blog[23] kullanılması gereken tek kısım navigasyon ve yönlendirmenin nasıl çalıştığıydı.

İyi yazılmış belgeler ve topluluk nedeniyle React Native'in anlaşılması ve kullanılması oldukça kolaydı. Çerçeve popüler hale gelen React üzerine kurulduğundan, yalnızca React Native'e odaklanmak gerekmez, bunun yerine React'a dayalı yanıtlar alınabilir.
React veya React Native'de oluşturulan hizmetler bileşenler kullanılarak oluşturulduğu ve bildirimsel programlama kullandığı için, belirli sorunların çözülmesi ve yanıt alınması kolaydır.
Örneğin, React Native'deki listelerin nasıl çalıştığıyla ilgili sorun yaşıyorsanız, o spesifik ve izole bileşeni araştırmak mümkündür. Sorun o bileşenin içinde yer aldığından çevredeki ortamın açıklanmasına gerek yoktur (veya en azından açıklanmasına gerek yoktur). Ayrılmış bileşenlere dayalı ve yalnızca destek olarak gönderilen verileri kullanan bir uygulama yapısına sahip olarak, klasik böl ve yönet paradigması elde edilir ve bu da genellikle daha kolay açıklanabilen bir sorunla sonuçlanır.

React Native uygulamasının geliştirilmesindeki bir husus, ihtiyaç duyulan kod miktarının az olmasıydı. Android uygulaması1 ve React Native uygulaması2 için depolarda görülebileceği gibi , ikincisi. Bu, React Native'de geliştirmenin kod miktarının yarısından azı kullanıldı
basitliğiyle birlikte geliştirme süresinin beklenenden çok daha kısa olmasının nedeniydi. Bu, farklı bileşenleri denemeye ve kodu yeniden düzenlemeye daha fazla zaman ayırmaya olanak sağladı.
Android uygulamasında kod miktarının daha yüksek olmasının bir nedeni, uygulamanın Java'lı yapısının, işlevselliği yönetirken aynı zamanda yapı için XML dosyalarına sahip olmasından kaynaklanıyordu. Bu, çoğunlukla React Native'de bir araya getirilir ve ayrıca bir bileşen için tüm kodun aynı dosyada olmasına katkıda bulunur, bu da çok daha az dosya ve daha fazla kontrol sağlar.

1https://github.com/willedanielsson/BudgetWatch_android
2https://github.com/willedanielsson/BudgetWatch_ReactNative

Ancak React Native'de bazı hatalar var ve en ciddi endişe çerçevedeki hatalardır. React Native, daha önce de belirtildiği gibi yeni bir çerçevedir ve mükemmel çalışmasını bekleyemeyiz. Çerçevenin bileşenlerindeki hatalar çoğunlukla bileşenin çalışması için hayati öneme sahip değildir ve genellikle kolayca düzeltilir. Ayrıca, çerçevede desteklenmedikleri için dışarıdan alınması gereken çok sayıda bileşen olduğundan, bunların hata içerme olasılığı daha yüksektir ve çoğu, daha önce piyasaya sürüldüğünden beri yalnızca iOS için belgelere ve daha fazla özelliğe sahiptir. Ancak en büyük ve en ciddi hata dışarıdan değil, çerçevedeki Liste bileşeninden kaynaklanıyordu. Bu hata bölüm 3.2.5'te açıklanmıştır ve sorun, listeye yeni verilerin destek olarak aktarılmasına rağmen listenin güncellenmemesidir. Bu hata sinir bozucuydu ve çerçevede büyük bir hataydı çünkü listeler uygulamalarda çok sık kullanılıyordu ve React ve React Native'deki ana özelliklerden biri olan, durumda yeni veriler ayarlandığında güncellenmeyen bir bileşene sahipti. gülünç.

Son olarak, React Native çerçevesinin bu tezde tam olarak ele alınmayan bir kısmı da kendi yerel modüllerini oluşturma yeteneğidir. Bazen uygulamanın platform API'sine erişmesi gerekebilir ancak React Native'in henüz buna karşılık gelen bir modülü yoktur. Bu durumda çerçeve, geliştiricilerin yerel kod yazmasına ve platformun tüm gücüne erişmesine olanak tanıyacak şekilde oluşturulmuştur. Bütçe İzleme uygulamasında harici modüllerin bulunmasının ve kullanılmasının nedeni budur çünkü çerçevede desteklenmeyen modüller topluluk tarafından oluşturulmakta ve bir projeye kolayca eklenebilmektedir. Facebook, topluluğun çerçeveyle birlikte kullanılmak üzere kendi modüllerini eklemesine izin vererek çerçevenin yalnızca çalışanları tarafından değil, aynı zamanda React Native kullanan geliştiriciler tarafından da genişletilmesine ve geliştirilmesine olanak tanıdı.

6. Sonuç

React Native, geliştiricilerin diledikleri teknik olabileceği ve uygulama oluşturma şeklimizi değiştirebileceği için övdüğü yeni ve ilginç bir çerçevedir. Bu tez, Android derlemesine odaklanarak çerçeveyi değerlendirmiş ve React Native'de geliştirme yaparken kullanıcı deneyimini ve iki uygulamanın performansını ve küçük bir ölçüde geliştirme sürecini karşılaştırmıştır.

Bir React Native uygulaması geliştirmek şaşırtıcı derecede kolaydı ve süreç hızlıydı; bu da uygulamanın beklenenden daha erken tamamlanmasına neden oldu. Dokümantasyonun kendisi, belirli işlevlerin nasıl ekleneceğine dair birçok örnek sağlar ve kullanılacak çok sayıda kılavuz ve bileşen üreten bir toplulukla birlikte, çerçeve yalnızca bir yıllık olmasına rağmen geliştirme kolaylaştırılmıştır. React Native uygulaması oluşturmanın basitliği, az sayıda kullanıcının React Native uygulamasını Android uygulamasından ayırt edememesi nedeniyle uygulamanın kullanıcı deneyimi üzerinde herhangi bir olumsuz etki yaratmadı.
Ayrıca diğerlerinin fark ettiği fark, yakın gelecekte düzeltilip destekleneceğini umduğumuz geçişler arasındaki animasyondu. Son olarak teste katılanların neredeyse tamamı React Native uygulamasını kullanırken hiçbir sorun yaşamadıklarını söyledi ki bu çok iyi bir karar ve native uygulama ile aynı görünüm ve hislere sahip olma hedefine ulaşıldı.

Son olarak, iki uygulama GPU frekansı, CPU yükü, bellek kullanımı ve güç tüketimi açısından performans testine tabi tutuldu. Üç farklı durum test edildi ve tüm ölçümler aynı sonuçla sonuçlandı; React Native uygulaması, native Android uygulaması kadar iyi bir performansa sahip değildir. Ancak testlerdeki farklar küçüktü ve React Native uygulaması, Android uygulamasına kayda değer bir şekilde meydan okumayı başardı.

Bu tez, React Native'de oluşturulan bir uygulamanın yerel bir Android uygulamasıyla karşılaştırıldığında ne kadar iyi olduğunu değerlendirdi ve gelecekte araştırılması ilginç olabilecek bazı çalışmalar var. Her şeyden önce, bu tez iOS'u bile dikkate almadığından, React Native'i iOS ile karşılaştıran benzer bir tez ilginç olurdu. Çerçevenin, yerel bir Android uygulaması gibi performans gösteren ve hissettiren bir uygulama oluşturabildiğini gördük, ancak diğer işletim sistemine göre ne kadar iyi performans gösterdiği henüz bilinmiyor. Dahası, bir iOS ile React Native'de oluşturulan bir Android uygulaması arasında kodun ne kadarının paylaşılabileceğini araştırmak ilginç olacaktır. Çerçeve mümkün olduğundan

Çok sayıda paylaşılan kod içeren iki uygulama oluşturduğunuzda, yerel bileşenler nedeniyle kodun ne kadarının platforma özgü olması gerektiğini bilmek ilgi çekicidir.

Kaynakça

[1] Android kaynak web sitesi | ART ve Dalvik. https : // kaynak . android. com / devices/tech/dalvik/ index.html. Erişim tarihi: 2016-04-19.

[2] Andreas Arnesson. “Kod adı bir ve PhoneGap, bir performans karşılaştırması”. Yüksek lisans tezi. Blekinge Teknoloji Enstitüsü, Yazılım Mühendisliği Bölümü, Haziran 2015.

[3] Ugaitz Moreno Arocena. “WiFi ve Bluetooth Kullanılarak Android Üzerinde 3G Trafik Konsolidasyonuna Yönelik Enerji Tüketim Çalışmaları”. Yüksek lisans tezi. Linköping Üniversitesi, Bilgisayar ve Bilgi Bilimi Bölümü, Ocak 2014.

[4] Alexander Bakker. Android için Enerji Profilerlerini Karşılaştırma. Twente Üniversitesi. 2016.

[5] V. Balasubramanee ve ark. "Twitter önyüklemesi ve AngularJS: Bilim ağ geçidi gelişimini hızlandırmak için ön uç çerçeveleri". İçinde: Cluster Computing (CLUSTER), 2013 IEEE Uluslararası Konferansı. Eylül 2013, s. 1–1. DOI: 10 . 1109 / KÜME . 2013. 6702640.

[6] Scott Chacon. Profesyonel Git. 1 inci. Berkely, CA, ABD: Apress, 2009. ISBN: 9781430218333.

[7] A. Cockburn ve ark. “WebView: Web Sayfalarını Tekrar Ziyaret Etmek İçin Grafiksel Bir Yardım”. İçinde: OZCHI'99 Avustralya İnsan Bilgisayar Etkileşimi Konferansı (Kasım 1999).

[8]Bonnie Eisenman. React Native'i öğrenme. O'Reilly Media, Inc., Aralık 2015. ISBN: 9781491929049.

[9]Bonnie Eisenman. React Native ile Çapraz Platform Uygulamaları Yazma. http://www.infoq. com/articles/ react-native-introduction. Erişim tarihi: 2016-04-25. Şubat 2016.

[10] Artemij Fedosejev. React.js'nin Temelleri. Packt Publishing Ltd., 2015. ISBN: 978-1-78355-162-0.

[11] Ben Frain. HTML5 ve CSS3 ile duyarlı Web Tasarımı. Packt Publishing Ltd, 1 Ocak. 2012.

[12] Cory Gackenheimer. React'a giriş. Apress, 2015. Bölüm. React Nedir?, s. 1–20. ISBN: 978-1-4842-1245-5. DOI: 10.1007/978-1-4842-1245-5_1.

[13] Georgios Gousios ve ark. “Yalın GHTorrent: Talep Üzerine GitHub Verileri”. İçinde: Madencilik Yazılım Depolarına ilişkin 11. Çalışma Konferansı Bildirileri. MSR 2014. ACM, 2014, s. 384–387. ISBN: 978-1-4503-2863-0. DOI: 10.1145/2597073.2597126.

[14] Robert B. Grady ve Deborah L. Caswell. Yazılım Metrikleri: Şirket Genelinde Bir Şirket Kurmak Programı. Prentice-Hall, Inc., 1987. ISBN: 0-13-821844-7.

[15] Erik Johansson ve Tobias Andersson. Platformlar arası destelere daha yakından bakış ve karşılaştırma akıllı telefonlar için geliştirme ortamı. Haziran 2014.

[16] Anmol Khandeparkar, Rashmi Gupta ve B. Sindhya. "Hibrite Giriş Platform Mobil Uygulama Geliştirme". İçinde: Uluslararası Bilgisayar Uygulamaları Dergisi 118.15 (2015).

[17] Benjamin LaGrone. Html5 ve Css3 Duyarlı Web Tasarımı Yemek Kitabı. Paket Yayıncılığı Ltd, 23 Mayıs 2013.

[18] Arnaud Le Hors ve ark. Belge Nesne Modeli (DOM) Düzey 3 Temel Belirtim. World Wide Web Konsorsiyumu, Tavsiye REC-DOM-Level-3-Core-20040407. Nisan 2004.

[19] Seung Ho Lim. "Mobil Sistemler İçin Hibrit ve Yerli Uygulamaların Deneysel Karşılaştırılması". İçinde: Uluslararası Multimedya ve Her Yerde Mühendislik Dergisi 10.3 (2015), s. 1–12.

[20] James Uzun. REACT NATIVE'DA KÖPRÜ KURMAK: React Native'in özüne derinlemesine bir bakış. http://jlongster.com/First- Impressions- kullanma- React- Native. Erişim tarihi: 2016-04-26. Ekim 2015.

[21] L. Ma, L. Gu ve J. Wang. "An-droid Platformu için Mobil Uygulamanın Araştırılması ve Geliştirilmesi". İçinde: Uluslararası Multimedya ve Her Yerde Mühendislik Dergisi 9.4 (2014), s. 187–198.

[22] Angel Torres Moreira, Mónica Aguilar-Igartua ve Silvia Puglisi. "Vatandaşların konumlarını anonim olarak analiz edecek bir Android uygulamasının tasarlanması ve uygulanması Barselona'da". İçinde: CoRR abs/1507.04585 (2015).

[23] Dotan Nahum. React Native'de Yönlendirme ve Gezinme. http: // blog'um. parakod. com/2016/01/05/react-native-in-yönlendirme-ve-navigasyon/. Erişildi: 2016-04-28. Ocak 2016.

[24] Johan Nordström ve Thommie Jönsson. Android üzerinden sipariş verme. Kristianstad Üniversitesi Vercity, Sağlık ve Toplum Okulu. 2012.

[25] Tom Occhino. React Native: Modern web tekniklerini mobile taşıyor. https://kod. Facebook . com / mesajlar / 1014532261909640 / tepki - yerli - getiriyor - modern web tekniklerinden mobile/. Erişim tarihi: 2016-04-23.

[26] Tom Occhino. React.js Conf 2015 Açılış Konuşması - React Native ile Tanışın. Facebook Geliştiricileri - Youtube. 2015. URL: https://www.youtube.com/watch?v=KVZ-P-ZI6W4.

[27] React Belgeleri. http://facebook.github.io/react/docs. Erişim Tarihi: 2016-04-21.

[28] React Yerel Dokümantasyon. https://facebook.github.io/react-native/. Erişim tarihi: 2016-04-23.

[29] Realm'in resmi web sitesi. https://realm.io/. Erişim tarihi: 2016-05-06.

[30] Virpi Roto, Marianna Obrist ve Kaisa Väänänen-vainio-mattila. Kullanıcı Deneyimi Değerlendirmesi Akademik ve Endüstriyel Bağlamda Kullanım Yöntemleri.

[31] Rajinder Singh. "Android İşletim Sistemine ve Güvenlik Özelliklerine Genel Bakış". İçinde: Mühendislik Araştırma ve Uygulamaları Dergisi 4.2 (2014), s. 519–521.

[32] Benny Skogberg. Android Uygulama Geliştirme. Teknoloji Okulu, Malmö Üniversitesi çok yönlülük. 2010.

[33] Andreas Somer. Mobil iş uygulamalarının geliştirilmesine yönelik platformlar arası çerçevelerin karşılaştırılması ve değerlendirilmesi. Technische Universität München. 2012.

[34] Arnold POS Vermeeren ve ark. “Kullanıcı Deneyimi Değerlendirme Yöntemleri: Mevcut Durum ve Gelişim İhtiyaçları”. İçinde: 6. İskandinav İnsan-Bilgisayar Etkileşimi Konferansı Bildirileri: Sınırları Genişletmek. NordiCHI '10. ACM, 2010, s. 521–530.
ISBN: 978-1-60558-934-3. DOI: 10 . 1145/1868914. 1868973. URL: http : / / doi . acm.org/10.1145/1868914.1868973.

[35] Marko Vitas. “ART vs Dalvik-KitKat'ta yeni Android çalışma zamanı ile tanışın”. İçinde: URL: https://www. sonsuz. co/the-capsized-eight/articles/art-vs-dalvikintroducing-the-new-android-runtime-in-kit-kat (cons. 2-2015) (2013).

[36]Tom Williams. RTC: Android, Telefonlardan ve Tabletlerden Daha Geniş Yerleşik Uygulamalara Geçmeye Hazır. http:// www.rtcmagazine.com/articles/view/102484. Erişim tarihi: 2016-04-15.

[37] Daniel Witte ve Philipp von Weitershausen. Android için React Native: İlk çapraz platform React Native uygulamasını nasıl geliştirdik. https : / / kodu . Facebook . com / gönderiler / 1189117404435352 / react - native - android için - ilk platformlar arası-tepki-yerel-uygulamasını nasıl - geliştirdik /. Erişim tarihi: 2016-04-24.

[38] L. Zhang ve ark. "Akıllı telefonlar için doğru çevrimiçi güç tahmini ve otomatik pil davranışı tabanlı güç modeli oluşturma". İçinde: Donanım/Yazılım Ortak Tasarımı ve Sistem Sentezi (CODES+ISSS), 2010 IEEE/ACM/IFIP Uluslararası Konferansı. Ekim 2010, s. 105–114.

Bü

İzleme ekran görüntüleri