T.C.

DÜZCE ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

ANDROID NATIVE SDK, FLUTTER VE REACT NATIVE PLATFORMLARININ KARŞILAŞTIRILMASI ÜZERİNE BİR TEZ ÇALIŞMASI

LİSANS MEZUNİYET TEZİ

ÖĞRENCİNİN ADI: EMRE YAVUZALP

DANIŞMAN: DOÇ. DR. TALHA KABAKUŞ

HAZİRAN 2021
DÜZCE

Önsöz

Günümüzde android ve iOS, iki popüler mobil işletim sistemleridir. Önceden, bir uygulamayı bu işletim sistemlerinde çalıştırmak istiyorsanız, uygulamayı yeniden öbür platformda yazmak zorundaydınız. Fakat, cross platform frameworklerin ortaya çıkışıyla, bu durum ciddi manada önlenmiştir. Her iki platforma birden, çok daha az maliyetle, daha az emekle uygulama geliştirmek çok kolaylaşmıştır.

Fakat bunun dezavantajları da yok değildir. Birtakım performans özelliklerinden ve bazı desteklerden mahrum kalınabilir. Önemli olan avantajların bu dezavantajları egale edip edemediğini bulmaktır. Her bir program yazmak isteyen kişi, durumu inceleyip ona göre davranmalıdır.

Bu çalışma boyunca gösterdiği her türlü destek ve yardımdan dolayı çok değerli hocam Sayın Doç. Dr. Talha Kabakuş'a en içten dileklerimle teşekkür ediyorum.

Ayrıca tüm eğitim hayatım boyunca bana yol gösteren, destek olan, en önemlisi bana eğitimin ne kadar önemli olduğu bilincini ve çalışma disiplinimi kazandıran , her zaman yanımda olan aileme en içten dileklerimle teşekkürlerimi sunuyorum.

Ayrıca eğitim hayatım boyunca bana yol göstermiş diğer hocalarıma da teşekkürlerimi sunarım.

Emre Yavuzalp 141001017

İçindekiler

ŞEKİL TABLOSU

TABLO LİSTESİ

ÖZET

Android Native Sdk, Flutter ve React Native denilen toplam 3 platform ile ilgili birtakım projelerin yapılmasını ve bunlardan yola çıkarak ve kod satır miktarı, karmaşıklığı, dosya boyutu,cpu ve memory kullanımı, FPS, estetik ve performans gibi açılardan karşılaştırılmasını içerir. İlgili platformlarda, sadece android tarafı incelenmiş olup, ios konu dışı bırakılmıştır.

Yapılan projeler şöyle özetlenebilir. 3 platformu da bilgisayarda kurma ve daha sonra; yerelden değişkenleri alıp gösteren bir tablo projesi, izin mekanizmasını gösteren ve mevcut lokasyon iznini sorup, lokasyonu alıp şehir bilgisine çevirme(reverse geocoding) yapan proje, ve son olarak da firebase'den verileri çekip bunları yine tablo üzerinde gösteren bir proje olmak üzere 3 proje yapılmıştır. Daha sonra bunları test etmek amacıyla, bir otomasyon test framework'ü kullanılmıştır.

Ekstra bilgi olması amacıyla, dışarıdan bununla ilgili başka makaleler de bu teze eklenmiş olup, bu kaynaklara atıf yapılmıştır.

SUMMARY

Making some projects related to 3 platforms called Android Native SDK, Flutter and React Native and starting from these; The amount of lines of code, complexity, file size, CPU and memory usage, FPS, aesthetics and performance are compared. On the relevant platforms, only the Android side has been examined and iOS has been excluded.

Projects can be summarized as follows. Installing all 4 platforms on the computer and then the following projects were carried out:
1) A table project that retrieves and displays local variables
2) A project that shows the permission mechanism and asks for the current location permission, takes the location and converts it to city information (reverse geocoding).
3) 3 projects have been made, including a project that pulls data from Firebase and displays them on the Tablo.
4) To test all of these projects, we are going to use an automated test framework

For the sake of extra information, other related articles have been added to this thesis, and these sources have been cited.

1. GİRİŞ

Mobil uygulama geliştirmeyle ilgili büyük bir sorun, mobil pazarın birkaç platform arasında bölünmüş olmasıdır. Bu nedenle geliştirme süresi uzar, daha fazla geliştirme becerisine ihtiyaç duyulur ve uygulamanın bakımı zorlaşır. Buna bir çözüm, aynı anda birkaç platform için bir uygulama geliştirmenize olanak tanıyan cross-platform geliştirmedir. Eylül 2015'ten bu yana, Facebook tarafından oluşturulan cross-platform çerçevesi React Native, herkesin kullanımına sunuldu. Ve bir başka cross-platform framework olan Flutter, Mayıs 2017 de Google tarafından herkesin kullanımına sunuldu. Bu tez, hem Android için React Native ve Flutter'ı performans, platform kod paylaşımı ve görünüm ve his açısından değerlendirir. Her üç platform için dört uygulama geliştirildi,. Farklı sürümler, performansı değerlendirmek için otomatik test senaryoları, platform kod paylaşımı için manuel kod incelemesi ve görünüm ve hissi değerlendirmek için bir kullanıcı çalışması ile karşılaştırıldı. Kullanıcı araştırması, uygulamanın React Native sürümlerinin, performansı önemli ölçüde etkilemeden yerel muadilleriyle benzer kullanıcı deneyimlerine sahip olduğunu gösterdiğinden, sonuçlar umut vericidir. Sonuçlar ayrıca, belirtilen uygulama için React Native kodunun yaklaşık %75'inin her iki platform için kullanılabileceğini ve platforma özel kod eklemenin kolay olduğunu gösteriyor.

1.1 CROSS PLATFORM FRAMEWORK'LERİN ÖNEMİ

Mobil uygulama geliştirme, milyarlarca kullanıcıdan oluşan potansiyel bir hedef kitleye sahip, yazılım mühendisliğinde nispeten daha yeni bir alandır. Statista, 2016 yılında dünya genelinde yaklaşık 1,5 milyar Android veya iOS akıllı telefonun satıldığını tahmin ediyor. Böyle büyük bir olası pazar, mobil uygulamaların geliştirilmesini yazılım geliştiriciler için arzu edilen bir odak noktası haline getiriyor. Yalnızca 2017'de toplam 197 milyar uygulamanın indirildiği tahmin ediliyor. Ne yazık ki, mobil uygulamalar için dünya çapındaki pazar son derece parçalıdır.

Son on yılda pazar, birkaç rakip mobil işletim sisteminden (BlackBerry, Windows Mobile, Symbian, Android ve iOS) ağırlıklı olarak iki platforma, Android ve iOS'a geçti. Bir geliştirici yalnızca Android ve iOS'a odaklanmayı seçse bile, parçalanma hala önemli bir sorundur. Joorabchi, Mesbah ve Kruchten, mobil geliştirme alanında iki farklı türde parçalanma tanımlamaktadır: platformlar arası parçalanma ve aynı platform içinde parçalanma. Android ve iOS'un her birinin kendi geliştirme ortamları, yazılım geliştirme kitleri (SDK'lar), programlama dilleri ve kullanıcı deneyimi standartları vardır. Tek bir platformda bile

telefonlar, Tablotler, saatler, TV cihazları ve daha fazlasını içeren çok çeşitli olası cihaz türleri bulunabilir. Her cihazın kendi donanım yapılandırması, ekran boyutu ve işletim sistemi desteği vardır. Bu parçalanma düzeyi, hedef mobil pazarın tamamına ulaşmaksa, mobil yazılım geliştirmeyi özellikle zorlaştırır.

On yıldan uzun bir süredir araştırmacılar, mobil uygulama geliştiricilerinin en az miktarda geliştirme çabasıyla mümkün olan en geniş kitleye nasıl ulaşabileceklerini belirlemeye çalışıyorlar. 2017'de El-Kassas, Abdullah, Yousef ve Wahba platformlar arası geliştirme yaklaşımlarının altı geniş kategorisini tanımladı ve yaklaşık 20 farklı platformlar arası araç ve çerçeve listeledi. Bu araçlardan bazıları artık geliştiriciler için mevcuttur (MoSync, Apache Cordova, Xamarian, vb.) ve bazıları araştırma veya deneysel prototiplerdir (MobDSL, MD2 ve ICPMD). Bu kadar çok sayıda seçenekle, bir geliştiricinin birden çok mobil işletim sistemini desteklemek için bir yaklaşım seçerken nereden başlayacağını bilmesi zor olabilir. Bu sorun mobil uygulama geliştirme alanına özel olmasa da, mobil geliştirmenin benzersiz doğası ve pazarın ve geliştirme çerçevelerinin sürekli evrimi nedeniyle açık ve önemli bir araştırma alanı olmaya devam etmektedir.

1.2 KULLANIM ALANLARI

Android Native, zaten kullanılan bir platform olup, React Native ve Flutter, cross platform olması dolayısıyla Native'den daha hızlı olduğu iddia edilen 2 platformdur. İkisi de genelde oyunlar gibi çok fazla kütüphane gerektirebilen ve donanıma erişim gerektiren uygulamalar hariç, hemen her türlü uygulamada kullanılabilir. Özellikle alışveriş sitesi, yemek sipariş gibi uygulamalarda oldukça kullanılmaktadır.

Flutter kullanan bazı uygulamalar şunlardır: Tencent, Cryptomaniac Pro, The New York Times, Flydirekt, Birch Finance, NuBank, SpaceX Go, eBay Motors.

React Native kullanan bazı uygulamalar ise şunlardır: Bloomberg,Facebook, Uber Eats,Airbnb, Discord, Instagram, Skype, Pinterest, Salesforce, Baidu, Walmart, Wix

BÖLÜM 2: İLGİLİ ÇALIŞMALAR

2.1 MOBİL CİHAZLARDA GÜNLÜK YAPILI DOSYA SİSTEMİ İÇİN KULLANICI DENEYİMİ GELİŞTİRMELİ MODEL KILAVUZLU DOSYA SIKIŞTIRMA

Mobil uygulamalar, çoğunlukla çoğunlukla yazılan dosyalara ve salt okunur dosyalara rastgele erişim içeren benzersiz dosya erişim modelleri sergiler. Mobil uygulamaların yüksek yazma stresi, flash tabanlı mobil depolamanın kullanım ömrünü önemli ölçüde etkiler. Yazma stresini azaltmak ve kullanıcı tarafından algılanan gecikmeden ödün vermeden yerden tasarruf etmek için, bu çalışma FPC, dosya erişim modeli kılavuzlu sıkıştırmayı tanıtmaktadır. FPC, mobil uygulamaların rastgele yazma ve parçalı okuma işlemleri için optimize edilmiştir. Çift modlu sıkıştırma özelliğine sahiptir: Ön plan sıkıştırması, yazma stresini azaltmak için çoğunlukla yazma dosyalarını işlerken, arka plan sıkıştırması, artırılmış okuma performansı için rastgele okuma dosya bloklarını paketler. FPC, önerilen çift modlu sıkıştırmanın en iyi etkisi için mobil cihazlar için günlük yapılı bir dosya sistemi olan F2FS'deki yerinde olmayan güncelleme tasarımından yararlanır. Deneysel sonuçlar, FPC'nin toplam yazma trafiği hacmini ve yürütülebilir dosya boyutunu sırasıyla ortalama %26,1 ve %23,7 oranında azalttığını ve uygulama başlatma süresini %14,8'e kadar iyileştirdiğini gösterdi.

2.2 ANDROID KÖTÜ AMAÇLI YAZILIM SAVUNMALARI İÇİN DERİN ÖĞRENME: SİSTEMATİK BİR LITERATÜR İNCELEMESİ

Kötü amaçlı uygulamalar (özellikle Android platformunda) geliştiriciler ve son kullanıcılar için ciddi bir tehdittir. Bu nedenle, Android kötü amaçlı yazılımlarını savunmak için etkili yaklaşımlar geliştirmeye yönelik birçok araştırma çalışması yapılmıştır. Bununla birlikte, Android kötü amaçlı yazılımlarının olağanüstü büyümesi ve şaşırtma ve yansıtma gibi kötü amaçlı yazılımdan kaçınma teknolojilerinin sürekli ilerlemesiyle, manuel kurallara veya geleneksel makine öğrenimine dayalı android kötü amaçlı yazılım savunmaları, sınırlı ön bilgi nedeniyle etkili olmayabilir. Son yıllarda, güçlü özellik soyutlama yeteneğine sahip baskın bir araştırma alanı olan derin öğrenme (DL), Nature Language işleme ve görüntü işleme gibi çeşitli alanlarda çekici ve umut verici bir performans göstermiştir. Bu amaçla, Android kötü amaçlı yazılım saldırılarını engellemek için derin öğrenme tekniklerini kullanmak, son zamanlarda araştırmaların büyük ilgisini çekti. Yine de, Android Kötü Amaçlı Yazılım savunmaları için derin öğrenme yaklaşımlarına odaklanan sistematik bir literatür taraması bulunmamaktadır. Bu yazıda, Android ortamında kötü amaçlı yazılım savunması bağlamında derin öğrenme yaklaşımlarının nasıl uygulandığını araştırmak ve analiz etmek için sistematik

bir literatür taraması yaptık. Sonuç olarak, 2014-2020 döneminde toplam 104 çalışma tespit edildi. Araştırmamızın sonuçları, bu çalışmaların çoğunun hala Android kötü amaçlı yazılım algılamasına dayalı DL tabanlı olduğunu düşünmesine rağmen, 35 birincil çalışmanın (%33,7) savunma yaklaşımlarını diğer senaryolara dayalı olarak tasarladığını gösteriyor. Bu inceleme ayrıca, DL tabanlı Android kötü amaçlı yazılım savunmalarında araştırma eğilimlerini, araştırma odaklarını, zorlukları ve gelecekteki araştırma yönergelerini açıklar.

2.3 REACT NATİVE KULLANILARAK PLATFORMLAR ARASI UYGULAMA GELİŞTİRME İÇİN PERFORMANS VE KULLANILABİLİRLİK ÜZERİNDEKİ ETKİLER

React native in iç yapısını ve mimarisini, native sistemlerle etkileşimini irdeleyen bir çalışma olup, ayriyeten React Native ve Android performansını çeşitli açılardan karşılaştırmıştır.Performans değerlendirmesi beş farklı performans faktörüne bölündü: CPU kullanımı, bellek kullanımı, saniyedeki kare sayısı, yanıt süresi ve uygulama boyutu. Bu faktörler kapsamlı bir sonuç vermek için seçilmiştir. Bazı ölçümler ilgili çalışmalarda kullanılmış ve platformlar arası çerçeveler için problemler olduğu görülmüştür. Bunların her biri, ndroid ve iOS'ta işletim sistemine özel uygulama araçları kullanılarak değerlendirildi. Android için kullanılan cihaz, Android Lollipop 6.0.1 işletim sistemi sürümüne sahip bir LG Nexus 5X idi. iOS cihazı, iOS 9.3.1 çalıştıran bir iPhone 5 idi.

3 farklı senaryo üzerinde yoğunlaşılmıştır:
1. Ana istatistik ekranındaki her iki grafiği de sırayla genişletin. 5 saniye bekleyin.
2. Radyatör istatistikleri ekranındaki her iki grafiği de sırayla genişletin. 5 saniye bekleyin.
3. Bir cihaz seçin ve anahtarı çevirin. Geri düğmesine basın ve cihazın anahtarını tekrar çevirin.

CPU kullanımı, Android'de adb shell komutu top kullanılarak ölçülmüştür. Bellek tüketimi verileri, Android için adb shell komutu dumpsys ile toplandı. Veriler ortalama, medyan, maksimum ve minimum değerler açısından analiz edildi. Veriler ayrıca yüzde kullanımını sonuçlandırmak için mobil cihazın toplam RAM miktarıyla karşılaştırıldı. Uygulamaların gerçek performansı izlendi ve birbirleriyle ve bu kıyaslama ile karşılaştırıldı. Android uygulamasının FPS'sini ölçmek için Systrace aracı kullanıldı. Bir süre boyunca uygulama kullanımını kaydeder ve analiz edilebilecek uygulama olaylarının izini döndürür. Bu iz, her yeni kare oluşturulduğunda net bir şekilde işaretler ve aynı zamanda kare düşmesi olarak da

bilinen, işlenmesi çok uzun süren bir kareyi işaretler. Ortaya çıkan iz analiz edildi ve ortalama düşen kare sayısı ve her senaryo için kare düşme yüzdesi hesaplandı. Android için yanıt süresini ölçmek için, daha önce bahsedilen Systrace aracı kullanılarak uygulamanın bir izi kaydedildi. Bu araç, uygulamanın yaydığı tüm eylem ve olayların bir izini oluşturur ve bundan, izleme yanıt süreleri sonuçlandırılabilir. Yüklü bir uygulama için ne kadar sabit disk alanı gerektiğini kontrol etmek için yerleşik işletim sistemi özellikleri kullanıldı. Hem Android'de hem de iOS'ta yüklü uygulama boyutu ayarlar menüsünde bulunabilir. Her dosya, her iki platform için ne kadarının kullanılabileceğini ölçmek için manuel olarak incelendi. Kod, Android'e özgü, iOS'a özgü veya paylaşılan olarak sınıflandırıldı. Daha sonra paylaşılan kodun yüzdesi hesaplandı. Android ve iOS'a özel kod arasındaki boyut farkı da ölçülmüştür, böylece bir işletim sistemi diğerinden daha fazla işletim sistemine özel koda ihtiyaç duyarsa, bu dikkate alınır. Bir kullanıcı görevleri yerine getirmeden önce bir Android veya iOS cihazını kullanırken mi daha rahat olup olmadığı soruldu. Kullanıcıya daha sonra bir uygulama ile kendini en rahat hissettiği bir OS sunuldu. Kullanıcı, test edilen uygulamanın bir React Native uygulaması mı yoksa yerel bir uygulama mı olduğu konusunda bilgilendirilmedi. Kullanıcıya diğer her seferinde React Native uygulaması ve diğer her seferinde yerel uygulama sunuldu. Kullanıcı, aşağıda açıklanan kullanıcı görevleri listesine göre her seferinde bir görev gerçekleştirdi. Tüm görevleri tamamladıktan sonra, kullanıcıdan hemen bir anket doldurması istendi. Anket, 14 iOS kullanıcısı ve 18 Android kullanıcısı tarafından dolduruldu; bunların yarısı yerel sürümü, yarısı ise React Native sürümünü kullandı. Seçilen anket, Ek E'de gösterilen daha önce bahsedilen UEQ anketidir. Bu anket, nispeten az sayıda kullanıcı için güvenilir sonuçlar verdiği için seçilmiştir. Aynı ürünün iki versiyonunun karşılaştırılması için de önerilen bir yaklaşımdır. Kullanıcı görevleri ise şunlardır:

1. Yatak odasındaki tavan lambasını açın.
2. Evin ortalama sıcaklığını kontrol edin.
3. Oturma odasındaki radyatörde efekti 4'e ayarlayın.
4. Oturma odasının istatistiklerini kontrol edin.
5. Bir lambanın ışığını %78 oranında kısın.
6. Fırın lambasının güç tüketimini 6 W olarak ayarlayın.

2.4 YEREL BENZERİ ÇAPRAZ PLATFORM MOBİL GELİŞTİRME ÇOKLU İŞLETİM SİSTEMİ MOTORU VE KOTLİN NATİVE VS FLUTTER

Bu tezin amacı, platformlar arası mobil geliştirmenin nispeten yeni bir yolunu ve araçlarını incelemek ve göstermek ve Multi-OS Engine, Kotlin/Native ve Flutter gibi teknolojileri

araştırmaktı. Kotlin/Native ve Flutter çalışması. Tez, diğer platformlar arası geliştirme seçeneklerinin yanı sıra platformlar arası gelişime karşı yerel gelişim teorisini tartıştı. Bu teknolojiler nispeten yeni olduğu için konuyla ilgili daha önce yapılmış çok fazla çalışma bulunmamaktadır. Tez, araçların nasıl kullanılabileceğini, özelliklerini ve bir çalışma örneğini göstermek için açıklayıcı örnekler sağlamayı amaçladı. Tez, farklı araçlar kullanılarak üç örnek uygulamanın yapıldığı bir vaka çalışmasıyla sona ermektedir. Vakanın ana amacı, bu teknolojilerin gerçek dünyada nasıl kullanılabileceğini göstermekti. Vaka çalışması, bu üç aracın olanaklarının bir gösterimi olarak hizmet eder. Sonunda her iki aracın avantaj ve dezavantajları ve bunların gerçek kullanım durumları bulundu.

2.5 REACT NATİVE VE FLUTTER KARŞILAŞTIRMASI, 2 MOBİL CROSS PLATFORM FRAMEWORK'Ü

React Native, Mart 2015'te Facebook tarafından tanıtıldı. Birkaç yıl önce Facebook tarafından yayınlanan React çerçevesine dayanıyor. [2] React çerçevesi, basitliği ve kolay olması ve aynı zamanda etkili geliştirme süreci nedeniyle geliştiriciler tarafından yaygın olarak kullanılmaktadır. [3] Öte yandan Google, 2016'nın sonunda Flutter adında başka bir mobil SDK yayınlıyor. React Native'den ilham alan Flutter uygulaması her iki platformda da çalışabilir, böylece iOS ve Android'de uygulama üretiminin maliyetini ve karmaşıklığını azaltır. Flutter tamamen sıfırdan yapılmıştır ve bu çalışmanın yazıldığı sırada (Ağustos 2017), yalnızca Google ticari proje için kullanır. React Native ve Flutter'a benzeyen platformlar arası çerçeveler, daha önce farklı şirketler tarafından tartışılmış ve uygulanmıştı. Ancak bunların hiçbiri endüstriyel gelişme gereksinimini karşılamaya yeterli değildir. Ancak React Native ve Flutter, Facebook ve Google'ın destekleriyle büyük ilgi görüyor. Bu tezin amacı, React Native ve Flutter üzerine kapsamlı bir çalışma yürütmektir. Her iki platform için temel kavramlar ve özellikler açıklanacak ve gösterilecektir. Tezde, React Native ve Flutter arasındaki performans ve geliştirme süreci açısından karşılaştırmalar ele alınacaktır. Ayrıca, tamamen çalışan bir React Native ve Flutter uygulaması olan React Native ve Flutter uygulaması arasındaki farkları ortaya çıkarmak için, ek olarak Flutter kullanılarak tamamen çalışan bir React Native uygulaması yeniden yazılacaktır.

2.6 APACHE CORDOVA VE FLUTTER MOBİL GELİŞTİRME ÇERÇEVELERİNİN VE BUNLARIN GELİŞTİRME SÜRECİ VE UYGULAMA ÖZELLİKLERİ ÜZERİNDEKİ ETKİSİNİN DEĞERLENDİRİLMESİ

Mobil uygulama geliştirme, kendine özgü geliştirme yaklaşımları ve zorlukları olan bir yazılım mühendisliği alanıdır. Mobil uygulamalar pazarı, yalnızca farklı mobil işletim sistemi platformları arasında değil, aynı platformdaki cihazlar ve işletim sistemi sürümleri arasında da oldukça parçalıdır. Herhangi bir uygulama geliştiricisinin amacı, geliştirme süresini, maliyetini ve çabasını en aza indirirken mümkün olan en geniş kitleye ulaşmaktır. Mevcut pazar, Android ve iOS olmak üzere iki büyük platformdan oluşmaktadır. Platformlar arası araçların kullanımı olmadan, her iki pazara da ulaşmak için aynı uygulamayı iki kez üretmek gerekli olacaktır.

Geliştiriciler için bir dizi platformlar arası geliştirme çerçevesi mevcuttur ve her birinin kendi avantajları ve dezavantajları vardır. Bu makale, iki platformlar arası geliştirme çerçevesi, Google'ın Flutter ve Apache Cordova ile Ionic çerçeve arasındaki karşılaştırmayı detaylandırıyor. İki çerçeveyi birbiriyle ve her platform için yerel uygulamalar geliştirme deneyimiyle karşılaştırmak için dört farklı çerçeve kullanılarak bir uygulama prototipi geliştirildi. Yerel bir Android uygulaması, yerel bir iOS uygulaması, bir Flutter çapraz platform uygulaması ve Ionic kullanıcı arayüzü çerçevesi kullanılarak bir Apache Cordova çapraz platform uygulaması olarak bir görev yönetimi uygulaması geliştirildi. Platformlar arası geliştirme çerçevesi seçmenin hem geliştirme deneyimi hem de dağıtılan uygulamaların performansı ve algılanan kalitesi üzerindeki etkisini belirlemek için bir dizi kaynak kodu ve uygulama performansı profili değerlendirmesi yapıldı. Flutter ve Cordova'nın geliştirme deneyimleri arasındaki niteliksel farklılıkların çeşitli alanları da ayrıntılı olarak açıklanmıştır.

2.6 REACT YEREL VS FLUTTER, ÇAPRAZ PLATFORM MOBİL UYGULAMA ÇERÇEVELERİ

Android ve iOS için ayrı ayrı mobil uygulamalar geliştirmek külfet haline geldi. Farklı platformlar için geliştirme, bakım, test etme ve dağıtmayı dikkate alan genel bir çözüm önemli bir konudur. Bu tür bir çözüm, mobil uygulama geliştirme sürecini birleştirebilmelidir. Facebook tarafından geliştirilen React Native, platformlar arası SDK'da bir kilometre taşı olarak kabul edildi. Güçlü ve aktif bir topluluk, React Native'i en popüler platformlar arası çerçeve haline getirdi. Ancak, React Native'in artılarını ve eksilerini düşündükten sonra Google, gelişmeleri birleştirmek için Flutter'ı piyasaya sürdü. Flutter, mobil ortama uyum

sağlamak için son derece optimize edilmiştir ve geliştiricilere nihai çözümü sağlamayı amaçlamaktadır. Bu tez, React Native ve Flutter'ın en önemli özelliklerinden bazılarını incelemektedir. Bu araştırma, bu özellikler arasındaki farklılıkları denemekte ve araştırmakta ve bunların arkasındaki nedeni anlamaya çalışmaktadır. Umarım, React Native ve Flutter, platformlar arası çerçeveyi yeni bir seviyeye taşıyacaktır.

2.7 TEK ANAHTARLA HER ŞEYİ ELE GEÇİRME: ANDROID'İN DOSYA TABANLI ŞİFRELEMESİNİ KIRMAK İÇİN MASTER ANAHTARI RAM'DEN ALMAK

On yıldan beri bilindiği gibi, bir saldırgan Android akıllı telefonlar da dahil olmak üzere açık bir cihaza fiziksel erişime sahip olduğunda, soğuk başlatma saldırıları yazılım tabanlı disk şifrelemesini bozabilir. Ham bellek görüntüleri, bir cihazı sıfırlayarak ve kötü niyetli bir önyükleyici ile yeniden başlatarak, güvenli önyükleme veya kısıtlayıcı BIOS ayarları nedeniyle bunun mümkün olmadığı veya RAM modüllerinin fiziksel olarak kontrolü altındaki bir sisteme nakledilmesiyle elde edilebilir. saldırgan. Bir aygıtın bellek görüntülerine dayalı olarak, geçmişte Tam Disk Şifrelemesini (FDE) kırmak için BitLocker, dm-crypt ve ayrıca Android'in FDE'si dahil olmak üzere farklı anahtar kurtarma algoritmaları önerildi. Google'ın yeni Android cihazlar için standart şifreleme yöntemi olarak FDE'den Dosya Tabanlı Şifrelemeye (FBE) geçmesiyle birlikte, mevcut araçlar etkisiz hale getirildi. Bu açığı kapatmak ve ham bellek görüntüsü verilen şifreli Android disklerin adli analizini yeniden etkinleştirmek için FBE için özel olarak hazırlanmış yeni bir anahtar kurtarma yöntemi sunuyoruz. Ayrıca, FBE etkin EXT4 görüntüleri üzerinde çalışırken dosya adlarının ve dosya içeriklerinin şifresini otomatik olarak çözmek için Sleuth Kit'i (TSK) ve ayrıca şifreli EXT4 bölümlerinden olayları çıkarmak için Plaso çerçevesini genişletiyoruz. Son olarak, ana anahtarların FBE bölümlerinden kurtarılmasının, Google'ın anahtar türetme yöntemindeki bir kusur nedeniyle özellikle kolay olduğunu savunuyoruz.

BÖLÜM 3: DENEK ORTAMLARININ KURULUMU

3.1 ANDROID NATIVE SDK

Bunun için, android studio yu indirmeliyiz. Bunu yaptıktan sonra, sdkları ve android emülatör dosyalarını indireceğiz. Bunun için önce settings den, sdk manager kısmına tıklıyoruz. Buradan, android 9 u kuruyorsak eğer, android 9 un sdk sını seçip apply a tıklıyoruz. SDK kurulmuş oluyor.

Android simülatör için ise, yine aynı menüden AVD manager'a geliyoruz. Yine bir telefon modeli seçip, android 9 modelini, yani aynı seçilen modelimiz için AVD dosyasını seçip indiriyoruz. Ayrıca, eğer BIOS'umuzdan, virtualization'ı açmadıysak, restart açıp BIOS'a girip bunu açıyoruz. Aksi halde emülatör açılmayacaktır.

Artık Android Native SDK kurulmuştur. Native platform olduğundan, diğer Flutter ve React Native kadar ekstra işleme ihtiyaç yoktur. Paketler tek tek import edilerek kurulabilmektedir.

3.2 FLUTTER

Öncelikle buradan flutter'ı indiriyoruz: https://flutter.dev/docs/get-started/install
Ben linux için kurulum adımlarını takip ediyorum. Daha sonra yapılacak adım, environment table'a flutter'ı eklemektir. Linux'ta ise bu işleme path'e eklemek denir. Biz de path'e ekleyeceğiz. Bunun için, şu linkte görülen adımları yapmalıyız:
https://flutter.dev/docs/get-started/install/linux#update-your-path

```
export PATH="$PATH:[PATH_OF_FLUTTER_GIT_DIRECTORY]/bin"
```

Bunu, home klasöründeki .bashrc'nin sonuna ekliyoruz. Daha sonra

```
source $HOME/.bashrc
```

yazıp, pathe eklemiş oluyoruz. Eklendiğinden emin olmak için which flutter yazıp görebiliriz. Yüklediğimiz yeri gösteriyor olmalıdır.
Ayrıca linux için bu dosyaları da kurmak gerekmektedir. Apt ile kurabiliriz:

```
sudo apt-get install clang cmake ninja-build pkg-config
libgtk-3-dev
```

Ayrıca desktop support için bunu da kurmak gerekebilir. Opsiyoneldir. Şahsen ben kurmadım:

```
flutter config –enable-linux- desktop
```

3.2.1 Flutter'da Proje Oluşturma ve Çalıştırma

Öncelikle vscode kullanabiliriz, android studio da kullanabiliriz. Bu tezde, vscode kullanacağız. Bunun için, istediğimiz bir klasörde terminal açıp,

```
flutter create projeİsmi
```

şeklinde, projeyi oluşturuyoruz. Daha sonra, android emülatörümüzü açıyoruz ve:

```
cd projeİsmi && flutter run
```

komutunu verince proje emülatörümüzde çalışacaktır.

Daha fazla bilgi için terminal flutter –help komutuyla diğer komutları görebilmekteyiz.

3.2.2 Flutter'da Paket İndirme

Mesela html paketi üzerinden bir örnek vermek gerekirse:

https://pub.dev/packages/flutter_html#installing

```
flutter_html:^2.0.0
```

Bunu, oluşturulan flutter dosyasının içindeki, pubspec.yaml dosyasının içinde belirtilen yere yazmalıyız. Ardından, terminalde flutter pub get yazmalıyız. Yeni flutter versiyonlarında,

```
Add the following to your pubspec.yaml file:

dependencies:
  flutter_html: ^2.0.0
```

dosyayı save yaparsak da otomatik olarak bu komutu yazmaktadır.

3.3 REACT NATIVE

React Native denek ortamını indirmek için şu adımları izlemeliyiz:

https://reactnative.dev/docs/environment-setup

Öncelikle nodejs yi indirmeliyiz.

```
sudo apt install nodejs
```

```
```

Daha sonra, npm ve yarn ı kurmak bizim için faydalıdır.

```
sudo apt install npm

npm install -g yarn
```

npm ve yarn'ı aynı proje içinde birlikte kullanmak risklidir. Ya npm ya da sadece yarn'ı kullanmalıyız.

Daha sonra java jdk yi kurmalıyız. Openjdk 14 veya 15 kurmalıyız. Yeni sürümler sorunlu olabilmektedir.

```
sudo apt install openjdk-15-jdk
```

Burada, android studio indirilmiş olmalıdır. Eğer kuruluysa, path'e tekrar birtakım şeyler eklememiz gerekmektedir. .bashrc nin sonuna bunları ekleyiniz:

```
export ANDROID_HOME=$HOME/Android/Sdk
export PATH=$PATH:$ANDROID_HOME/emulator
export PATH=$PATH:$ANDROID_HOME/tools
export PATH=$PATH:$ANDROID_HOME/tools/bin
export PATH=$PATH:$ANDROID_HOME/platform-tools
```

Daha sonra,

```
source $HOME/.bashrc
```

ile path'e eklemiş oluyoruz.

3.3.1 React Native ile Proje Oluşturma ve Çalıştırma

```
npx react-native init projeİsmi
```

Bunu terminalde yazıp, daha sonra dosyanın içine girip:

```
npx react-native start
```

yazınca, npx server çalışmaktadır. Emülatörümüz de açık olmalıdır. Bundan sonra, başka bir terminal daha açıp:

```
npx react-native run-android
```

yazınca uygulamamız emülatöre yüklenecektir ve açılacaktır.

3.3.2 React Native Paket İndirme

Mesela Tablo paketini ele alalım: https://www.npmjs.com/package/react-native-Tablo-component

Aynı klasörde terminal açıp:

```
npm install react-native-Tablo-component
```

yazınca, ilgili paket yüklenmiş olur. Genelde paket kurulumları bu şekildedir. Her paket için bir arama motorunda arama yapıp, ilgili install komutlarını bulmamız gerekmektedir.

BÖLÜM 4: DENEYSEL SONUÇLAR VE TARTIŞMA

4.1 BASİT BİR HELLO WORLD, BAŞLANGIÇ UYGULAMASI

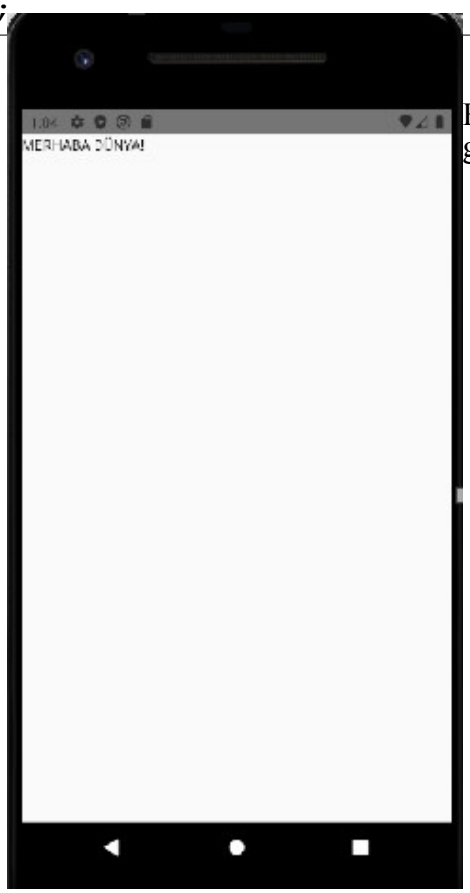
Konu: Her üç platformda da sadece hello world çıktısı veren, basit bir program yazmak.

4.1.1 React Native

Öncelikle oluşturulan şablon projede, App.js yi açıyoruz. Burayı şu şekilde değiştiriyoruz:

```
import React from 'react';
import type {Node} from 'react';
import {
SafeAreaView,
ScrollView,
StatusBar,
StyleSheet,
Text,
useColorScheme,
View,
} from 'react-native';
import {
Colors,
DebugInstructions,
Header,
LearnMoreLinks,
ReloadInstructions,
} from 'react-native/Libraries/NewAppScreen';
const App: () => Node = () => {
return (
<View>
<Text>MERHABA DÜNYA!</Text>
</View>
);
};
const styles = StyleSheet.create({
sectionContainer: {
marginTop: 32,
```

```
paddingHorizontal: 24,
},
sectionTitle: {
fontSize: 24,
fontWeight: '600',
},
sectionDescription: {
marginTop: 8,
fontSize: 18,
fontWeight: '400',
},
highlight: {
fontWeight: '700',
}
,
yazi: {
marginTop: 32,
paddingHorizontal: 24,
}
});
export default App;
```



Ekran çıktısı bu Şekil 1 deki gibi olacaktır:

Şekil 1: Proje 1, React Native ekran çıktısı

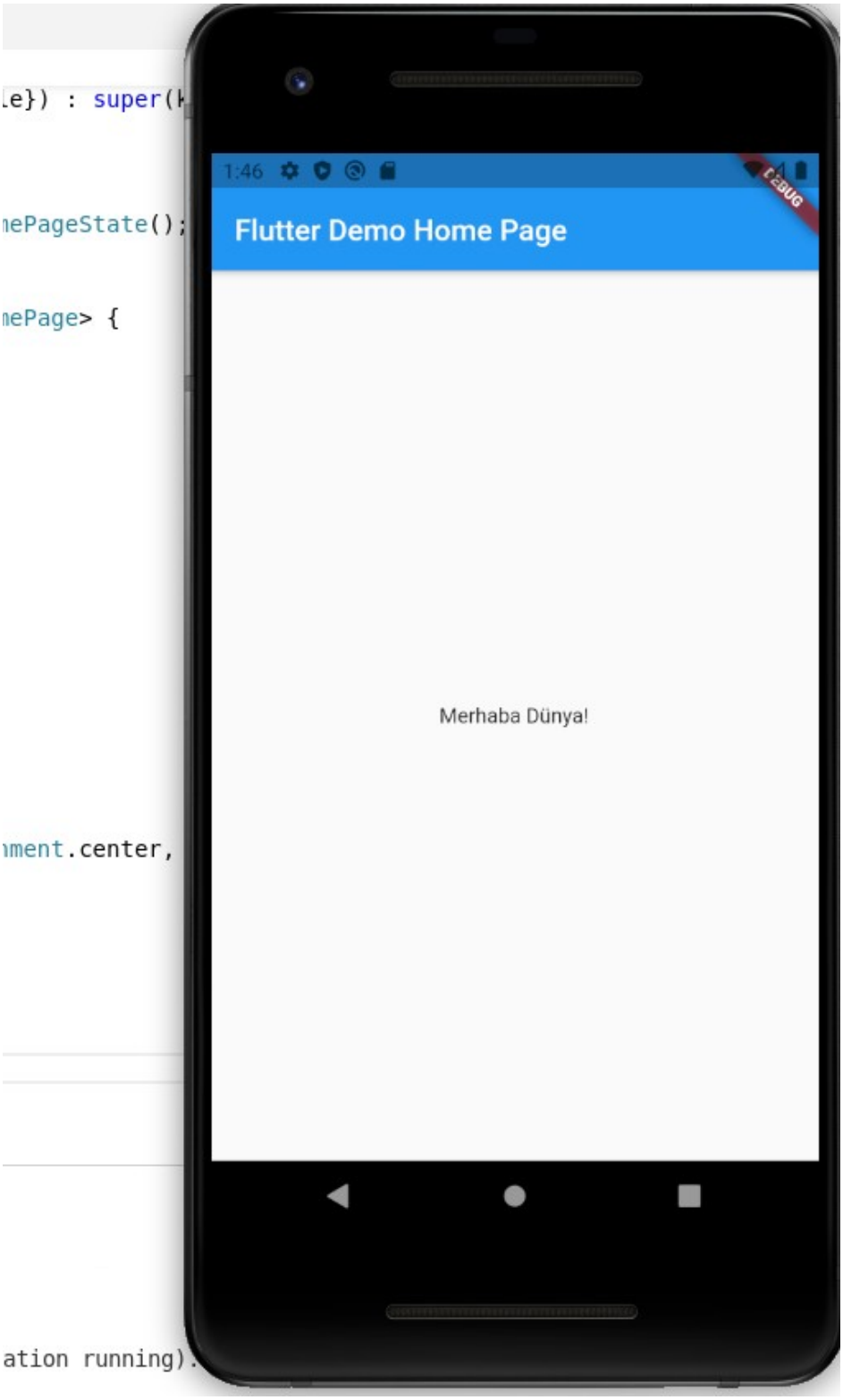
4.1.2 Flutter

Flutterda oluşturduğumuz bir projenin yine içindeki main.dart dosyasını vscode ile açıyoruz. Bu şablon projeyi şu şekilde değiştiriyoruz:

```
import 'package:flutter/material.dart';
void main() {
runApp(MyApp());
}
class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Flutter Demo',
theme: ThemeData(
primarySwatch: Colors.blue,
),
home: MyHomePage(title: 'Flutter Demo Home Page'),
);
}
}
class MyHomePage extends StatefulWidget {
MyHomePage({Key? key, required this.title}) : super(key: key);
final String title;
@override
_MyHomePageState createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
int _counter = 0;
void _incrementCounter() {
setState(() {
_counter++;
});
}
@override
```

```
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
title: Text(widget.title),
),
body: Center(
child: Column(
mainAxisAlignment: MainAxisAlignment.center,
children: <Widget>[
Text(
'Merhaba Dünya!',
),],
),
),
);
}}
```

Ekran çıktısı şekil 2’deki gibi olacaktır:



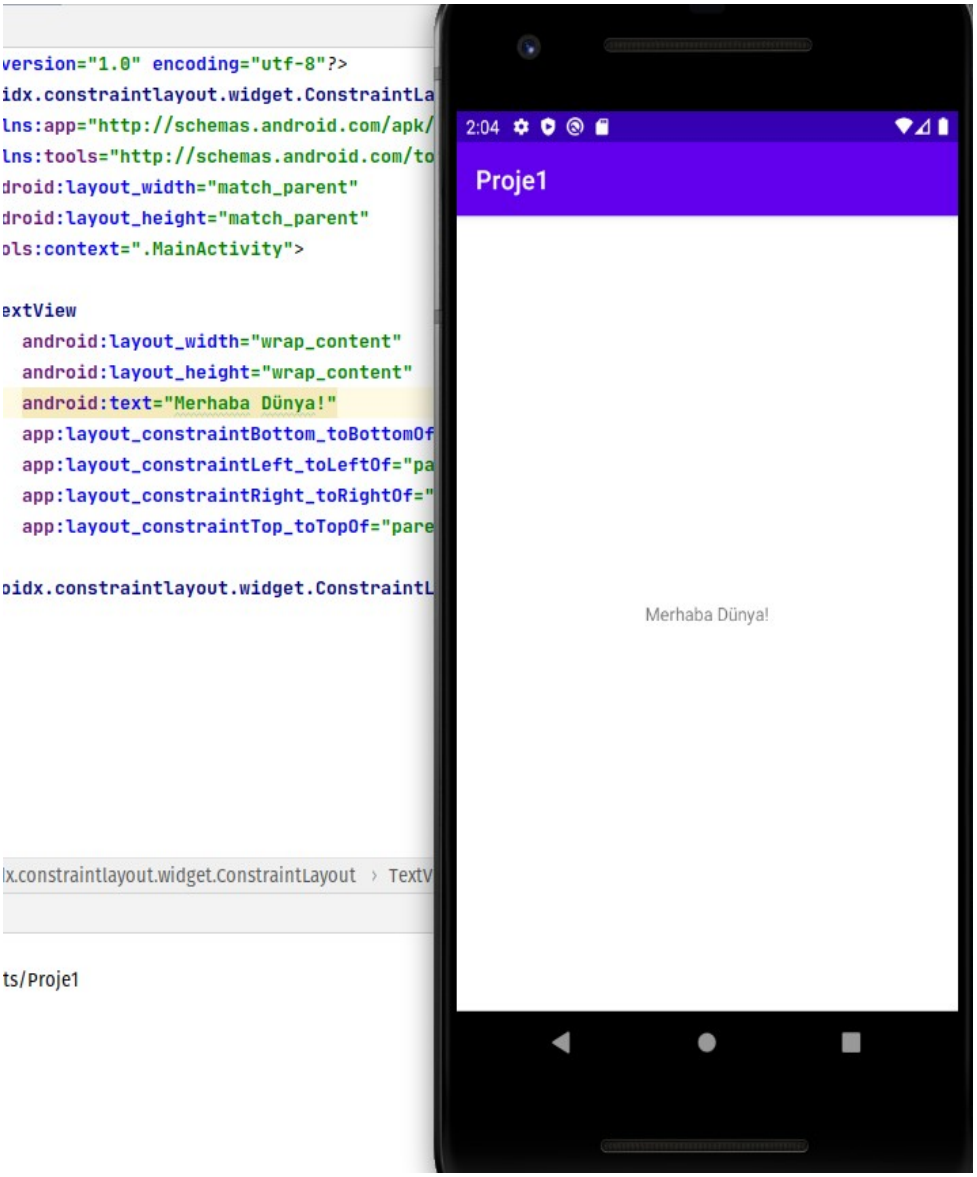
Şekil 2: Proje 1, Flutter ekran çıktısı

4.1.3 Android Native SDK

Android studio SDK ve AVD leri zaten yüklediğimizi varsayarsak, buradan sonra standart bir şablon uygulama, yani basic activity şablonunu seçiyoruz. Daha sonra, burada activity_main.xml dosyasındaki şu satırı, Merhaba Dünya olarak değiştiriyoruz:

```
android:text="Merhaba Dünya!"
```

Ekran çıktısı şekil 3'teki gibi olacaktır:



Şekil 3: Proje 1, Android Native SDK ekran çıktısı

4.2 BİR TABLO UYGULAMASI

Konu: Her üç platformda da, yerel birtakım değişkenleri gösteren basit bir tablo uygulaması yapmak.

4.2.1 React Native

Önce, şu paketi indirmemiz gerekmektedir: https://www.npmjs.com/package/react-native-table-component

```
npm install react-native-Tablo-component
```

İşbu linkte, 1. örneği alıp, kendi durumumuza göre değiştirdik. Daha sonra, klasörümüzün içinde, assets diye bir klasör oluşturup, erkek logomuzu bu klasörün içine atıyoruz. Buradan da, içeride çağırıyoruz. Sonuç olarak, kodumuz şöyle olmaktadır:

```
import React, { Component } from 'react';
import { StyleSheet, Text, View } from 'react-native';
import { Table, Row, Rows } from 'react-native-table-
component';
import { getTimeSinceStartup } from 'react-native-startup-
time';
export default class ExampleOne extends Component {
constructor(props) {
super(props);
<View style={styles.container}>
<StatusBar style="auto" />
</View>
this.state = {
tableHead: ['Ad', 'Soyad', 'Doğum Tarihi', 'Mezuniyet
Durumu','Fotoğraf'],
tableData: [
['Ali', 'Veli', '12.03.1995', 'Mezun',<Image
style={styles.stretch}
source
={require("./assets/erkek1.png")}/>],
['Ahmet', 'Mehmet', '22.09.1999', 'Mezun değil',<Image
```

```
style={styles.stret
ch} source={require("./assets/erkek2.png")}/>],
['Ayşe', 'Fatma', '02.01.1998', 'Mezun değil',<Image
style={styles.stretch
} source={require("./assets/erkek1.png")}/>],
]
}
}
render() {
const state = this.state;
return (
<View style={styles.container}>
<Table borderStyle={{borderWidth: 1, borderColor: '#c8e1ff'}}>
<Row data={state.tableHead} style={styles.head}
textStyle={styles.text}/>
<Rows data={state.tableData} textStyle={styles.text}/>
</Table></View>
)
}
}
const styles = StyleSheet.create({
container: { flex: 1, padding: 5, paddingTop: 50,
backgroundColor:
'orange' }
,
head: { height: 30, backgroundColor: '#f1f8ff' },
text: { margin: 1 },
stretch: {
width: 50,
height: 50,resizeMode: 'stretch',
},
});
```

Ekran çıktısı bu şekil 4'teki gibi olmaktadır:

Şekil 4: Proje 2, React Native ekran çıktısı

4.2.2 Flutter

main.dart'ı şu şekilde değiştiriyoruz:

```
import 'package:flutter/material.dart';
void main() {
runApp(MyApp());
}
class MyApp extends StatefulWidget {
@override
_MyAppState createState() => _MyAppState();
}
```

```
class _MyAppState extends State<MyApp> {
double iconSize = 40;
@override
Widget build(BuildContext context) {
return MaterialApp(
home: Scaffold(
appBar: AppBar(
title: Text('Flutter Table - tutorialkart.com'),
),
body: Center(
child: Column(children: <Widget>[
Container(
margin: EdgeInsets.all(10),
child: Table(
border: TableBorder.all(),
children: [
TableRow(children: [
Column(children: [Text('Ad')]),
Column(children: [Text('Soyad')]),
Column(children: [Text('Doğum Tarihi')]),
Column(children: [Text("Mezuniyet Durumu")]),
Column(children: [Text("Fotoğraf")]),
]),
TableRow(children: [
Text("Ali"),
Text("Veli"),
Text("12.03.1995"),
Text("Mezun"),
Image(image: AssetImage('assets/erkek1.png')),
]),
TableRow(children: [Text("Ahmet"),
Text("Mehmet"),
Text("22.09.1999"),
Text("Mezun değil"),
```

```
Image(image: AssetImage('assets/erkek2.png')),
]),
TableRow(children: [
Text("Ayşe"),
Text("Fatma"),
Text("02.01.1998"),
Text("Mezun değil"),
Image(image: AssetImage('assets/kadin.png')),
]),
],
),
),
])),
);
}
}
```

Ekran çıktısı şekil 5’teki gibi olacaktır:

Şekil 5: Proje 2, Flutter ekran çıktısı

4.2.3 Android Native SDK

Öncelikle, şu kaynaktaki mesajdaki tablo örneğini kullandım ve değiştirdim: https://stackoverflow.com/questions/33995938/how-to-create-a-Tablo-by-using-Tablolayout-in-android-studio

Logo resimlerini res/drawable klasörünün içine atıyoruz ve oradan çekiyoruz.
activity_main.xml'i bu şekilde değiştiriyoruz:

```
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:shrinkColumns="*"
android:layout_width="fill_parent"
android:layout_height="match_parent"
android:stretchColumns="0,1,2,3,4"
android:gravity="center">
<TableRow
android:background="#FFFFFF"
android:layout_width="fill_parent"
android:layout_height="0dp"
android:layout_margin="1dp"
android:layout_weight="1"
>
<TableRow
android:background="#000000"
android:layout_width="fill_parent"
android:layout_height="0dp"
android:layout_margin="1dp"
android:layout_weight="1"
>
</TableRow>
</TableRow>
<TableRow
android:background="#000000"
```

```
android:layout_width="fill_parent"
android:layout_height="0dp"
android:layout_margin="1dp"
android:layout_weight="1"
>
<TextView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text=" Ad"
android:layout_margin="1dp"android:layout_column="0"
android:background="#FFFFFF"
android:gravity="center"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text="Soyad "
android:layout_margin="1dp"
android:layout_column="1"
android:background="#FFFFFF"
android:gravity="center"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text="Doğum Tarihi"
android:layout_margin="1dp"
android:background="#FFFFFF"
android:gravity="center"
android:layout_column="2"
/>
```

```
<TextView
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text="Mezuniyet Durumu"
android:layout_margin="1dp"
android:background="#FFFFFF"
android:gravity="center"
android:layout_column="3"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text="Fotoğraf"
android:layout_margin="1dp"
android:background="#FFFFFF"
android:gravity="center"
android:layout_column="4"
/></TableRow>
<TableRow
android:background="#000000"
android:layout_width="fill_parent"
android:layout_height="0dp"
android:layout_margin="1dp"
android:layout_weight="1"
>
<TextView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text=" Ali"
android:layout_margin="1dp"
android:layout_column="0"
```

```
android:background="#FFFFFF"
android:gravity="center"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text=" Veli"
android:layout_margin="1dp"
android:layout_column="1"
android:background="#FFFFFF"
android:gravity="center"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text="12.03.1995"
android:layout_margin="1dp"
android:background="#FFFFFF"
android:gravity="center"
android:layout_column="2" />
<TextView
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text=" Mezun"
android:layout_margin="1dp"
android:layout_column="3"
android:background="#FFFFFF"
android:gravity="center"
/>
<ImageViewandroid:layout_width="fill_parent"
android:layout_height="wrap_content"
```

```
android:layout_column="4"
android:src="@drawable/erkek.jpg"
/>
</TableRow>
<TableRow
android:background="#000000"
android:layout_width="fill_parent"
android:layout_height="0dp"
android:layout_margin="1dp"
android:layout_weight="1"
>
<TextView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text=" Ahmet"
android:layout_margin="1dp"
android:layout_column="0"
android:background="#FFFFFF"
android:gravity="center"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text=" Mehmet"
android:layout_margin="1dp"
android:layout_column="1"
android:background="#FFFFFF"
android:gravity="center"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="match_parent"
```

```
android:textAppearance="?android:attr/textAppearanceLarge"
android:text=" 22.09.1999"
android:layout_margin="1dp"
android:background="#FFFFFF"
android:gravity="center"
android:layout_column="2" />
<TextView
android:layout_width="match_parent"
android:layout_height="match_parent"android:textAppearance="?
android:attr/textAppearanceLarge"
android:text=" Mezun değil"
android:layout_margin="1dp"
android:layout_column="3"
android:background="#FFFFFF"
android:gravity="center"
/>
<ImageView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/erkek.jpg"
android:layout_column="4"
/>
</TableRow>
<TableRow
android:background="#000000"
android:layout_width="fill_parent"
android:layout_height="0dp"
android:layout_margin="1dp"
android:layout_weight="1"
>
<TextView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
```

```
android:text="Ayşe"
android:layout_margin="1dp"
android:layout_column="0"
android:background="#FFFFFF"
android:gravity="center"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text="Fatma"
android:layout_margin="1dp"
android:layout_column="1"
android:background="#FFFFFF"
android:gravity="center"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text="02.01.1998"
android:layout_margin="1dp"
android:background="#FFFFFF"
android:gravity="center"android:layout_column="2" />
<TextView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text="Mezun değil"
android:layout_margin="1dp"
android:layout_column="0"
android:background="#FFFFFF"
android:gravity="center"
/>
```

```
<ImageView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_column="4"
android:src="@drawable/erkek.jpg"
/>
</TableRow>
</TableLayout>
```

Ekran çıktısı şekil 6'daki gibi olacaktır:

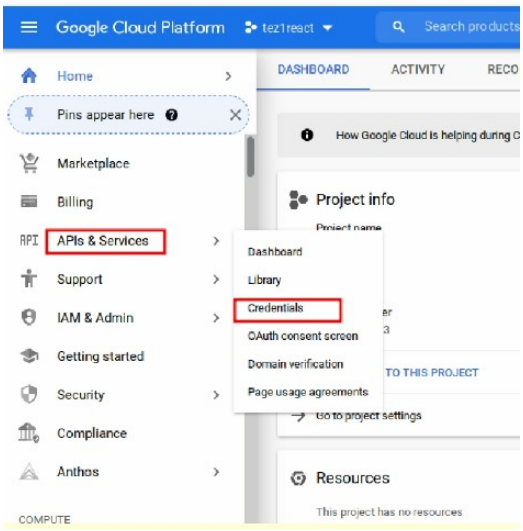
Şekil 6: Proje 2, Android Native SDK ekran çıktısı

4.3 KAMERA VE LOKASYON İZİN UYGULAMASI

Konu: 2 buton ve her bir butonda sırayla kamera ve lokasyon izni soran, lokasyon izni aldıktan sonra da mevcut lokasyonu gösteren bir uygulama geliştirmek amaçlanmaktadır.

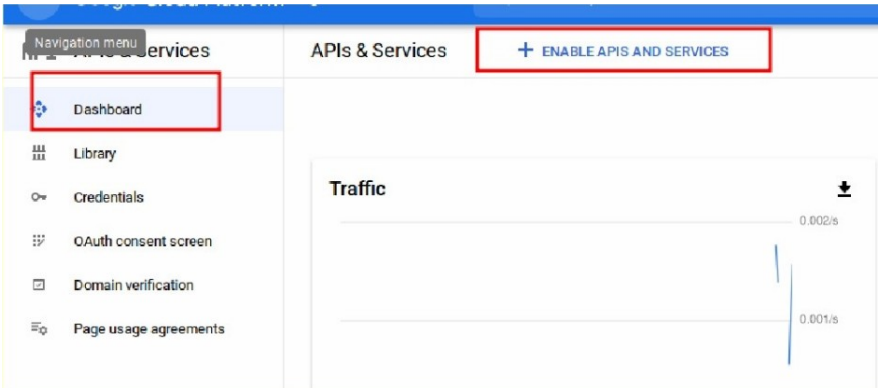
4.3.1 React Native

Öncelikle, bir google api key almak şarttır. Onu da buradaki şekil 7'de gösterildiği gibi alabiliriz: https://console.cloud.google.com/



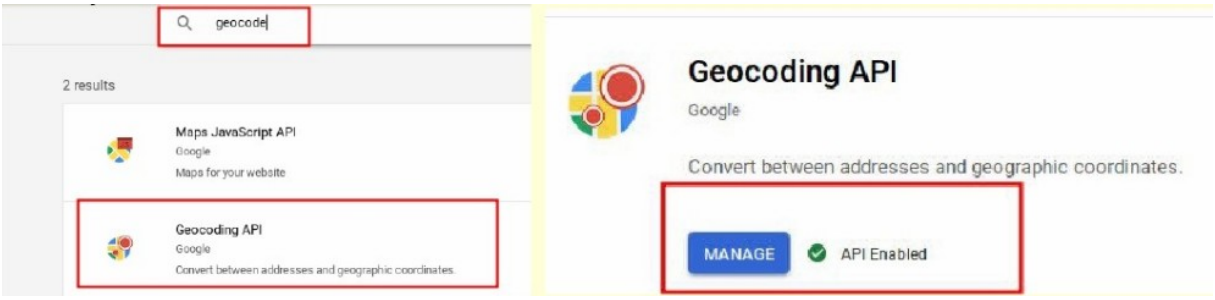
Şekil 7: Google API key alınması

Üstteki linkten girip, kayıt olup alabiliriz.Burada bir mail adresiyle oturum açıp, APIs & Services > Credentials kısmından, bir api key oluşturuyoruz. Daha sonra, aynı menünün dashboard kısmında bulunan, Enable API's and Services kısmına tıklıyoruz:



Şekil 8: API key'i aktif hale getirme

Burada, Geocoding API kısmına girip, enable a tıklıyoruz. Ben zaten enable ettiğim için burada API enabled yazmaktadır. Fakat normalde enable diye mavi bir buton mevcuttur. Ona basmalıyız.



Şekil 9: Geocoding API'yi aktif hale getirme

Bu, matematiksel koordinatlardan şehir verisine çevirmektedir. Kullanımı ücretsiz olsa da, oldukça limitlidir. Ücretli kullanımda ise limiti artırılmaktadır.

Bu api key, aşağıdaki kodun &key= den sonrasına yazılmalıdır:

```
https://maps.googleapis.com/maps/api/geocode/json?latlng=$
{locationPublish}&sensor=true&result_type=street_address&key=k
endiAPIKodunuz
uGiriniz
```

Bu URL'yi de, daha sonra aşağıda verilen fonksiyonun içinde kullanacağız. API kodları asla kimseyle paylaşılmamalıdır.

Öncelikle kullanılan fonksiyonları inceleyeceğiz:

Alttaki locationToAddress fonksiyonu, verilen adresi google maps geocode api si ile, şehir verisine çevirmektedir.

```
const locationToAdress = async (location) => {
const locationPublish = `${location.latitude},$
{location.longitude}`;
//PUBLISH
const                          url                          =
`https://maps.googleapis.com/maps/api/geocode/json?
latlng=$
{locationPublish}&sensor=true&result_type=street_address&key
=AIza
SyBL9MI07Uz
ZndYkViX7D9kdMd-LNLaMuwg`;
fetch(url)
.then((result) => {console.log(result);
})
```

```
.catch((e) => { console.log(e);
});
}
```

Bir diğer fonksiyon ise requestLokasyonPermission. Basitçe lokasyon iznini sormaktadır. locationToAddress fonksiyonu da daha sonra bu fonksiyonla birleştirilmiştir. O fonksiyonu da ayrıca çağırmaktadır:

```
const requestLokasyonPermission = async () => {
if (Platform.OS === "android") {
try {
const granted = await PermissionsAndroid.request(
PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION,
{
title: "Location Permission",
message:
"This App needs access to your location " +
"so we can know where you are.",
}
);
if (granted === PermissionsAndroid.RESULTS.GRANTED) {
console.log("You can use locations ");
} else {
console.log("Location permission denied");
}
} catch (err) {
console.warn(err);
}}
Geolocation.getCurrentPosition(
(location) => {
console.log(location);
locationToAdress(location.coords)
},
(error) => {
console.log(error);
```

```
},
{ enableHighAccuracy: false, timeout: 200000, maximumAge: 1000
}
); };
```

İçindeki Geolocation.getCurrentPosition fonksiyonu ise, mevcut konumu almaktadır. Daha sonra bu konumu da, locationToAddress fonksiyonuna parametre olarak veriyoruz. Geriye kamera fonksiyonları kalmaktadır. Bu fonksiyonlar zaten proje yönetiminde de sorunsuz çalışmaktaydı, bu yüzden ekstradan incelemeye gerek duyulmamıştır. Fakat yine de iyileştirilmiştir.

React-native in kendi permissionsAndroid özelliği kullanılmıştır. Önceden react-native-permissions kütüphanesi kullanılmaktaydı. Bu bağlamda, programın stabilitesinin arttığı söylenebilir.

Ayrıca react-native-geolocation-service kütüphanesi kullanılmıştır. Bunu indirmek için:

yarn için:

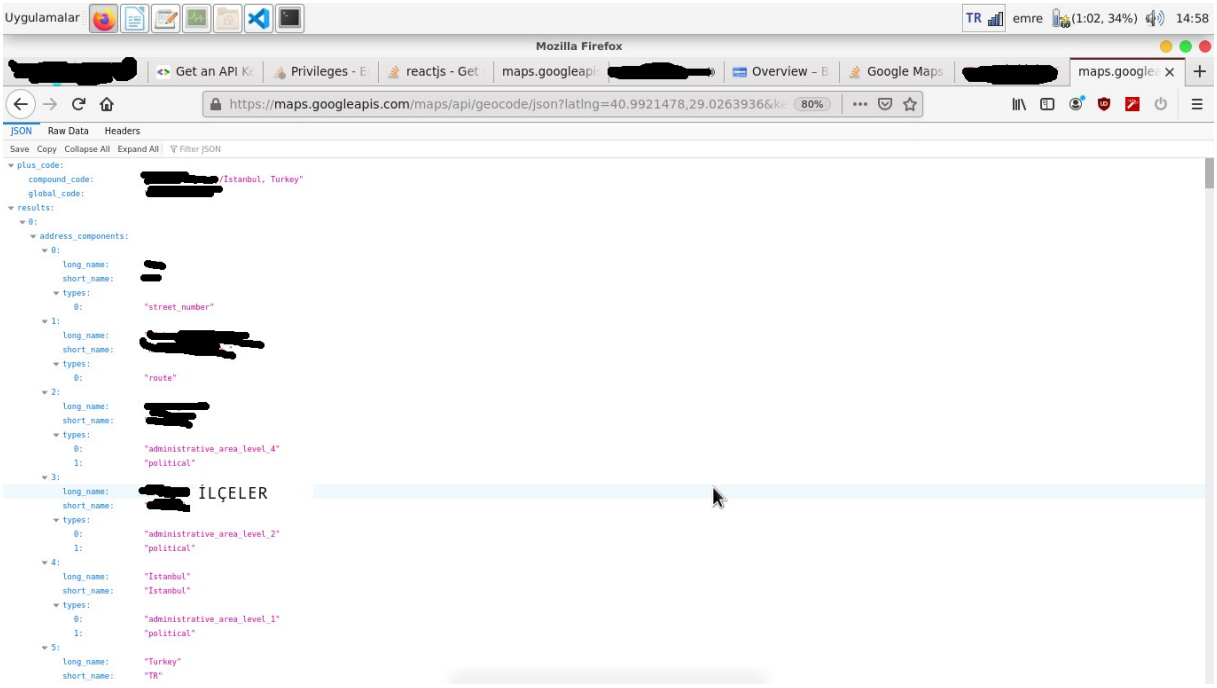
```
yarn add react-native-geolocation-service
```

npm için:

```
npm install react-native-geolocation-service
```

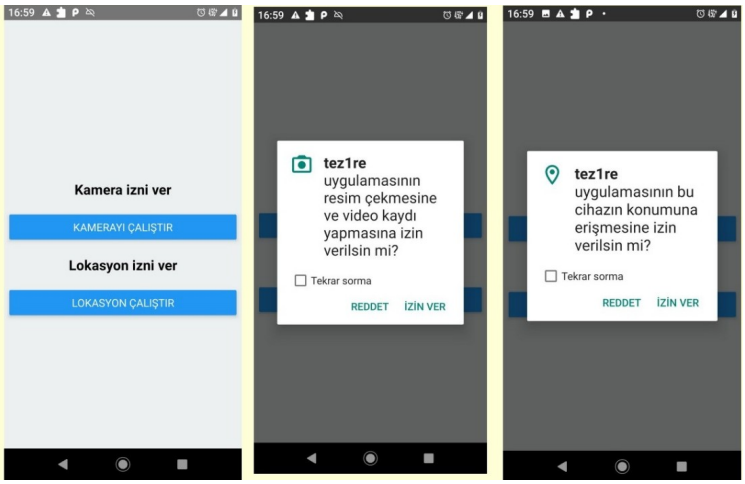
Github adresleri için: https://github.com/Agontuk/react-native-geolocation-service

Not: Bu kod, google'a lokasyon bilgilerini göndermektedir, fakat bu bilgi bir türlü uygulamada gösterilememiştir. Fakat url, tarayıcıdan girip bakıldığında adres verileri bizzat görüntülenmektedir.
Bunun kanıtı da aşağıdaki resimdir. Görüldüğü gibi, mevcut adres bilgisi burada gözükmektedir:



Şekil 10: Adres bilgisi geri gönderiliyor, fakat gösterilmemekte

Ekran çıktısı şekil 11'de görüldüğü gibidir:



Şekil 11: Proje 3, React Native ekran çıktısı

Stackoverflow'a başvurulmuştur, linkleri aşağıdadır:

https://stackoverflow.com/questions/66436762/get-user-city-from-location-coords-data-in-react-native

https://stackoverflow.com/questions/66457207/how-to-show-users-city-data-from-reverse-geocode-api-in-react-native

Tüm kod ise bu şekildedir. Sadece App.js vardır.

App.js

```
import React from "react";
import { Button, PermissionsAndroid, SafeAreaView, StatusBar,
StyleSheet,Text, View, Platform } from "react-native";
import Geolocation from 'react-native-geolocation-service'
const locationToAdress = async (location) => {
const locationPublish = `${location.latitude},$
{location.longitude}`;
//PUBLISH
const url =
`https://maps.googleapis.com/maps/api/geocode/json?
latlng=$
{locationPublish}&sensor=true&result_type=street_address&key=s
izinG
oogleAPIK
eyiniz`;
fetch(url)
.then((result) => {
console.log(result);
})
.catch((e) => {
console.log(e);
});
}
const requestLokasyonPermission = async () => {
if (Platform.OS === "android") {
try {
const granted = await PermissionsAndroid.request(
PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION,
{
title: "Location Permission",
message:
```

```
"This App needs access to your location " +
"so we can know where you are.",
}
);
if (granted === PermissionsAndroid.RESULTS.GRANTED) {
console.log("You can use locations ");
} else {
console.log("Location permission denied");
}
} catch (err) {
console.warn(err);
}}
Geolocation.getCurrentPosition(
(location) => {
console.log(location);
locationToAdress(location.coords)
},
(error) => {
console.log(error);
},
{ enableHighAccuracy: false, timeout: 200000, maximumAge: 1000
}
); };
const requestCameraPermission = async () => {
try {const granted = await PermissionsAndroid.request(
PermissionsAndroid.PERMISSIONS.CAMERA,
{
title: "Cool Photo App Camera Permission",
message:
"Cool Photo App needs access to your camera " +
"so you can take awesome pictures.",
buttonNeutral: "Ask Me Later",
buttonNegative: "Cancel",
buttonPositive: "OK"
```

```
}
);
if (granted === PermissionsAndroid.RESULTS.GRANTED) {
console.log("You can use the camera");
} else {
console.log("Camera permission denied");
}
} catch (err) {
console.warn(err);
}
};
const App = () => (
<View style={styles.container}>
<Text style={styles.item}>Try permissions</Text>
<Button title="request permissions" onPress={
() => { requestCameraPermission(); }
} />
<Text style={styles.item}>Try permissions</Text>
<Button title="request permissions" onPress={
() => { requestLokasyonPermission(); }
} />
</View>
);
const styles = StyleSheet.create({
container: {
flex: 1,
justifyContent: "center",
paddingTop: StatusBar.currentHeight,
backgroundColor: "#ecf0f1",
padding: 8
},
item: {
margin: 24,
fontSize: 18,
```

```
fontWeight: "bold", textAlign: "center"
}
});
export default App;
```

4.3.2 Flutter

Bu konuda ise şu videodan yararlanılmıştır: https://www.youtube.com/watch?v=3hiT9tlWipE

Burada, lokasyon matematiksel olarak gösterilip, şehir olarak gösterilememiştir. İlgili dokümentasyonlardan faydalanıldığı halde, birebir aynı olduğu halde, şehir bilgisi gösterilmemektedir. Nedeni çözülememiştir.

Dosyalar kısaca şu şekildedir:

main.dart

```
import
import
import
import
'package:flutter/material.dart';
'package:firebase_storage/firebase_storage.dart';
'package:cloud_firestore/cloud_firestore.dart';
'package:firebase_core/firebase_core.dart';
Future<void> main() async {
WidgetsFlutterBinding.ensureInitialized();
await Firebase.initializeApp();
runApp(MyApp());
}
class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Flutter Basic Demo',
theme: ThemeData(
primarySwatch: Colors.blue,
```

```
visualDensity: VisualDensity.adaptivePlatformDensity,
),
home: MyHomePage(title: 'new Flutter App'),
);
}
}
class MyHomePage extends StatefulWidget {
MyHomePage({Key key, this.title}) : super(key: key);
final String title;
@override
_MyHomePageState createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
var url;
Widget _buildlistItem(BuildContext context, DocumentSnapshot
document) {
void showImage() async {final ref =
FirebaseStorage.instance.ref().child('erkekyuz.png');
// no need of the file extension, the name will do fine.
url = await ref.getDownloadURL();
}
showImage();
String mezunD;
if (document.data()['mezunDurumu'] == false) {
mezunD = "Mezun Değil";
}
if (document.data()['mezunDurumu'] == true) {
mezunD = "Mezun";
}
String yasString = (document.data()['yas']).toString();
return Scaffold(
appBar: AppBar(
title: Text(widget.title),
),
```

```
body: Table(
border: TableBorder.all(),
children: [
TableRow(children: [
Text("Ad Soyad: "),
Text(
document.data()['adSoyad'],
textAlign: TextAlign.center,
),
]),
TableRow(children: [
Text("Yaş: "),
Text(
yasString,
textAlign: TextAlign.center,
),
]),
TableRow(children: [
Text("Doğum Tarihi: "),
Text(
document.data()['dogumTarihi'],
textAlign: TextAlign.center,
),
]),
TableRow(children: [
Text("Mezun Durumu: "),
Text(
mezunD,
textAlign: TextAlign.center,
),
]),
TableRow(children: [
Text(
"Fotoğraf: ",
```

```
),
Image.network(url,
width: 50,
height: 50,
),
]),
],
),
);
}
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
title: Text("Öğrenci Durumu"),
),
body: StreamBuilder(
stream:
FirebaseFirestore.instance.collection('tablolar').snapshots(),
builder: (context, snapshot) {
if (!snapshot.hasData) return const Text('Loading...');
return ListView.builder(
itemExtent: 100.0,
itemCount: snapshot.data.docs.length,
itemBuilder: (context, index) =>
_buildlistItem(context, snapshot.data.docs[index]),
);
}),
);
}
}
```

requestAssistant.dart

```
import 'dart:convert';
import 'package:http/http.dart' as http;
```

```
class RequestAssistant
{
static Future<dynamic> getRequest(String url) async
{
http.Response response=await http.get(url);
try {
if(response.statusCode == 200)
{
String jSonData = response.body;var decodeData =
jsonDecode(jSonData);
return decodeData;
}
else
{
return "failed";
}
}
catch(exp)
{
return "failed";
}
}}
```

assistantMethods.dart

```
import 'package:geolocator/geolocator.dart';
import 'package:rider_app/Assistants/requestAssistant.dart';
class AssistantMethods
{
static Future<String> searchCoordinateAddress(Position
position)
async
{
String placeAddress="";
String url=
```

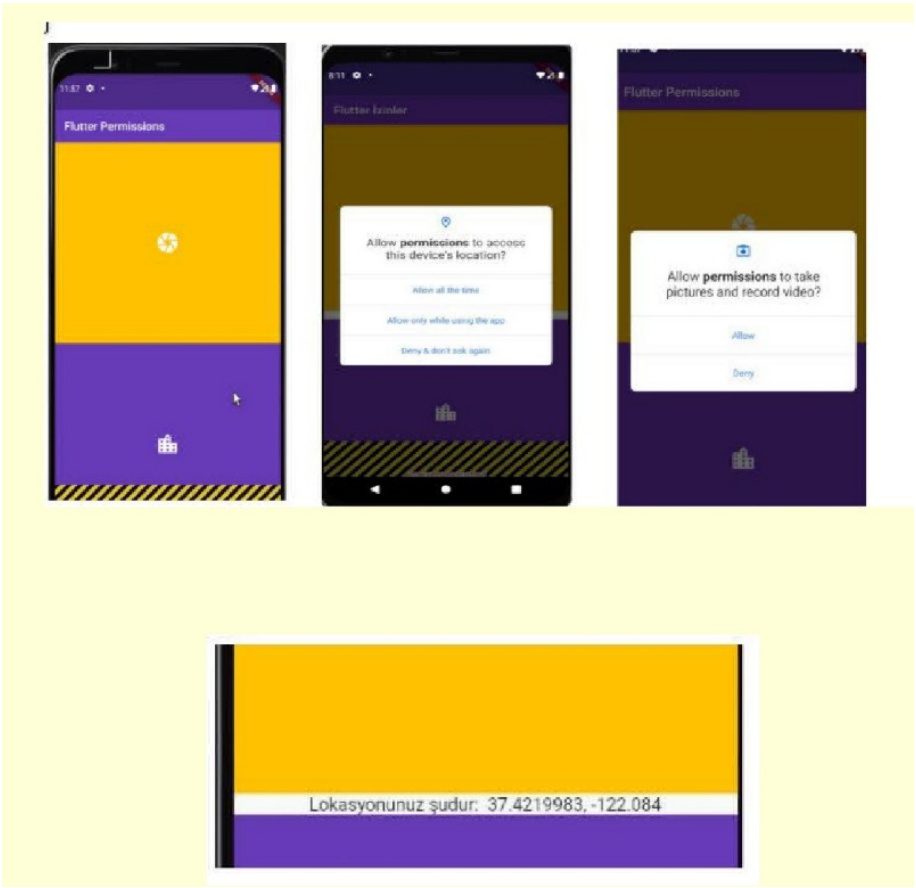
```
"https://maps.googleapis.com/maps/api/geocode/json?
latlng=${position.latitude},
$position.longitude},73,961452&key=kendiAPIkeyiniziGiriniz";
var response=await RequestAssistant.getRequest(url);
if(response !="failed")
{
placeAddress = response["results"][0]["formatted_address"];
}
return placeAddress;
}}
```

pubspec.yaml

```
name: tez1fl
description: A new Flutter project.
version: 1.0.0+1environment:
sdk: ">=2.7.0 <3.0.0"
dependencies:
flutter:
sdk: flutter
permission_handler: ^5.0.1+1
camera: ^0.5.8+8
http:^0.13
call_log: ^2.1.0
flutter_phone_direct_caller: ^1.0.1
path_provider:
path:
oktoast: ^2.3.2
geolocator: ^5.1.5
cupertino_icons: ^1.0.2
dev_dependencies:
flutter_test:
sdk: flutter
flutter:
```

```
uses-material-design:true
```

Ekran çıktısı şekil 12'deki gibidir:



Şekil 12: Proje 2, Flutter ekran çıktısı

4.3.3 Android Native SDK

Bunun için kaynak kodlar şöyledir:

MainActivity.java

```
package com.example.camloc;
import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;
import android.Manifest;
import android.content.pm.PackageManager;
import android.location.Address;
import android.location.Geocoder;
```

```
import android.location.Location;
import android.os.Build;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import
com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.tasks.OnSuccessListener;import
java.io.IOException;
import java.io.StringBufferInputStream;
import java.util.List;
import java.util.Locale;
public class MainActivity extends AppCompatActivity {
private Button btn;
private TextView textView;
private FusedLocationProviderClient
fusedLocationProviderClient;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
fusedLocationProviderClient=
LocationServices.getFusedLocationProviderClient(this);
btn= findViewById(R.id.location_btn);
btn.setOnClickListener(new View.OnClickListener() {
@RequiresApi(api = Build.VERSION_CODES.M)
@Override
public void onClick(View v) {
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
if
(getApplicationContext().checkSelfPermission(Manifest.permissi
```

```
on.AC
CESS_FINE_LOCATION)
==
PackageManager.PERMISSION_GRANTED) {
fusedLocationProviderClient.getLastLocation()
.addOnSuccessListener(new OnSuccessListener<Location>() {
public void onSuccess(Location location) {
if (location != null) {
Double lat = location.getLatitude();
Double longt = location.getLongitude();
double latitude = location.getLatitude();
double longitude = location.getLongitude();
Locale locale = new Locale("en"); //OR Locale.getDefault()
Geocoder geocoder = new Geocoder(MainActivity.this, locale);
List<Address> addresses = null;
try {
addresses = geocoder.getFromLocation(latitude, longitude,
1);
} catch (IOException e) {
e.printStackTrace();
}
Object updatedCity = addresses.get(0).getLocality();
if (updatedCity == null) {
updatedCity = addresses.get(0).getAdminArea();
}
String updatedCountry = addresses.get(0).getCountryName();
TextView location_text =
(TextView)findViewById(R.id.location_text);
location_text.setText(updatedCountry+" ,
"+updatedCity);Toast.makeText(MainActivity.this, "Success",
Toast.LENGTH_SHORT).show();
}
}
});
```

```
} elserequestPermissions(new String[]
{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
}
}
});
}}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">
<TextView
android:id="@+id/location_text"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Location"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"app:layout_constraint
Horizontal_bias="0.523"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/location_btn"
app:layout_constraintVertical_bias="0.606" />
<Button
android:id="@+id/location_btn"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

```
android:text="Get Location"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintHorizontal_bias="0.498"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.393" />
<Button
android:id="@+id/cameraAc"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Kamerayi calistir"
app:layout_constraintBottom_toTopOf="@+id/location_btn"
app:layout_constraintHorizontal_bias="0.497"
app:layout_constraintLeft_toLeftOf="parent"app:layout_constrai
ntRig
ht_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.235" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

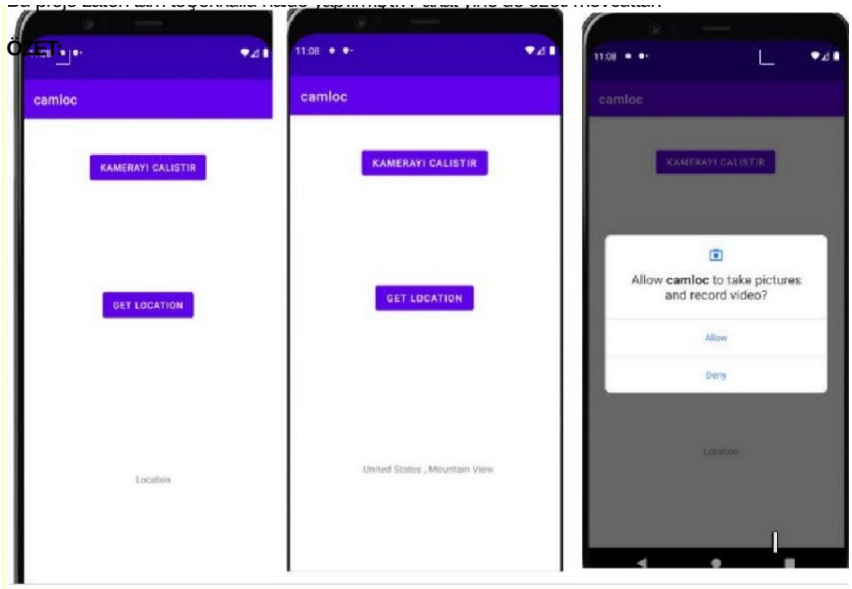
Androidmanifest.xml kısmına:

```
<uses-permission android:name="android.permission.CAMERA"
/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
```

eklenmiş ve build.gradle da dependencies kısmına:

```
    implementation 'com.google.android.gms:play-services-
    location:17.1.0'
```

Ekran çıktısı şekil 13'teki gibidir:



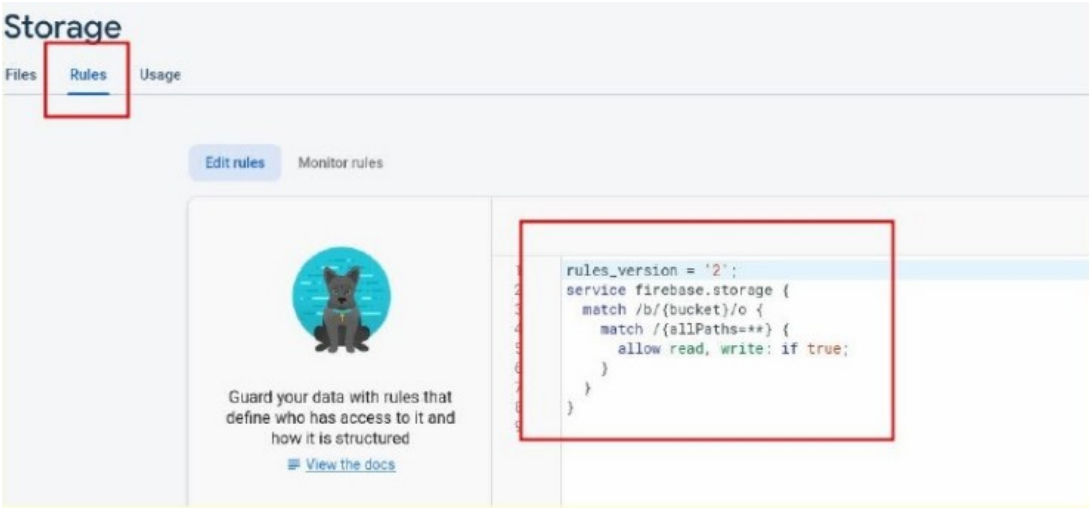
Şekil 13: Proje 3, Android Native SDK ekran çıktısı

4.4 CLOUD FIREBASE UYGULAMASI

Konu: Firebase'den belli string, boolean, fotoğraf ve number verilerini çekip, bunları bir tabloda göstermek amaçlanmaktadır.

4.4.1 React Native

Öncelikle yine firebase kısmına gireceğiz. Burada, firebase storage'a fotoğrafımızı yüklüyoruz.



Şekil 14: Firebase storage kuralları değiştirme

Daha sonra, rules kısmına girip, oradan kuralları değiştirmeliyiz. Test amaçlı olduğu için, ben tüm izinleri vermiş bulunmaktayım. Fakat gerçek bir program yaparken bu haliyle yapılmamalıdır.

Bu kuralları yazmanız yeterlidir:

```
rules_version =
'2';
service firebase.storage
{
match /b/{bucket}/o {
match /{allPaths=**}
{
allow read: if
true
allow write: if
false
}
}
}
```

Daha sonra ise, stackoverflow'daki şu konu uyarınca, seçilmiş cevabı uyguluyoruz:

https://stackoverflow.com/questions/66553270/how-to-show-the-image-from-the-cloud-firestore-in-react-native

app.js

```
import React, {Component} from 'react';import { DataTable }
from 'react-native-paper';
import {
Platform,
SafeAreaView,
StyleSheet,
ScrollView,
View,
Text,
Image,
Button,
StatusBar,
```

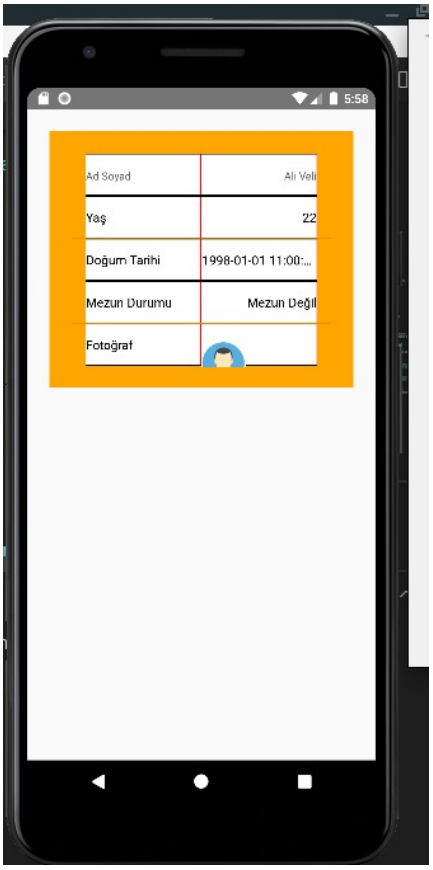
```
} from 'react-native';
import firebase from '@react-native-firebase/app'
import firestore from '@react-native-firebase/firestore'
import { format } from "date-fns";
import storage from '@react-native-firebase/storage';
import { Table } from 'react-native-table-component';
class App extends Component {
state = {
tablo: {
adSoyad: "",
yas: "",
dogumTarihi: "",
mezunDurumu: "",
imageUrl: null,
}
}
constructor(props) {
super(props);
this.getUser();
this.subscriber=firestore().collection("tablo").doc
('J6mAav1kjkcjrMupeOqX').onSnapshot(doc => {
this.setState({
tablo: {
adSoyad: doc.data().adSoyad,
yas: doc.data().yas,
dogumTarihi: doc.data().dogumTarihi,
mezunDurumu:doc.data().mezunDurumu,
}
}
)
});
}
async componentDidMount() {
var imageRef = firebase.storage().ref('erkek.jpeg');
```

```
var imageUrl = await imageRef.getDownloadURL();
this.setState({ imageUrl });
}
getParsedDate(date){
date = String(date).split(' ');
var days = String(date[0]).split('-');
var hours = String(date[1]).split(':');return
[parseInt(days[2]), parseInt(days[1])-1, parseInt(days[0]),
parseInt(hours[0]), parseInt(hours[1]), parseInt(hours[2])];
}
getUser= async() => {
const userDocument= await firestore().collection("tablo").doc
("J6mAav1kjkcjrMupeOqX").get()
console.log(userDocument)
}
render() {
var mezund;
var date = new String(this.state.tablo.dogumTarihi);
if(this.state.tablo.mezunDurumu==false){mezund="Mezun Değil"}
else if(this.state.tablo.mezunDurumu=true){mezund="mezun"}
return (
<View style={styles.body}>
<DataTable>
<DataTable.Header>
<DataTable.Title style={styles.row}>Ad Soyad</DataTable.Title>
<DataTable.Title numeric
style={styles.row}>{this.state.tablo.adSoyad}</
DataTable.Title>
</DataTable.Header>
<DataTable.Row >
<DataTable.Cell style={styles.row}>Yaş</DataTable.Cell>
<DataTable.Cell numeric
style={styles.row}>{this.state.tablo.yas}</DataTable.Cell>
</DataTable.Row>
```

```
<DataTable.Row>
<DataTable.Cell style={styles.row}>Doğum
Tarihi</DataTable.Cell>
<DataTable.Cell numeric
style={styles.row}>{date}</DataTable.Cell>
</DataTable.Row>
<DataTable.Row>
<DataTable.Cell style={styles.row}>Mezun
Durumu</DataTable.Cell>
<DataTable.Cell numeric
style={styles.row}>{mezund}</DataTable.Cell>
</DataTable.Row>
<DataTable.Row>
<DataTable.Cell style={styles.row}>Fotoğraf</DataTable.Cell>
<DataTable.Cell style={styles.row}><Image
style={styles.stretch}
source={{ uri: this.state.imageUrl }}/></DataTable.Cell>
</DataTable.Row>
</DataTable>
</View>
);
}
}
const styles=StyleSheet.create({
body:{
padding: 25,
margin:25,//flex:10,
backgroundColor:'orange',
borderColor:'black',
},
stretch:
{
width:50,height:50,
padding: 250,
```

```
margin:250,
justifyContent: 'center',
alignItems: 'center',
marginBottom:50,
paddingBottom:50,
borderTopWidth:1,
backgroundColor:'#fff',
resizeMode:'contain',
},
row:{
borderRightColor:'red',
borderRightWidth:1,
borderBottomWidth:1,
borderTopWidth:1,
backgroundColor:'#fff',
textAlign:'center',
}
})
export default App
```

Ekran çıktısı şekil 15'teki gibidir:



Şekil 15: Proje 4, React Native
ekran çıktısı

4.4.2 Flutter

Bu tam halde yapılabilmiştir. Yeni kodlar ise şu şekildedir:

pubspec.yaml dependencies kısmına:

```
dependencies:
...
firebase_storage: ^8.0.5
cloud_firestore: ^1.0.7
firebase_core: ^1.1.0
path_provider: ^2.0.1
...
```

main.dart

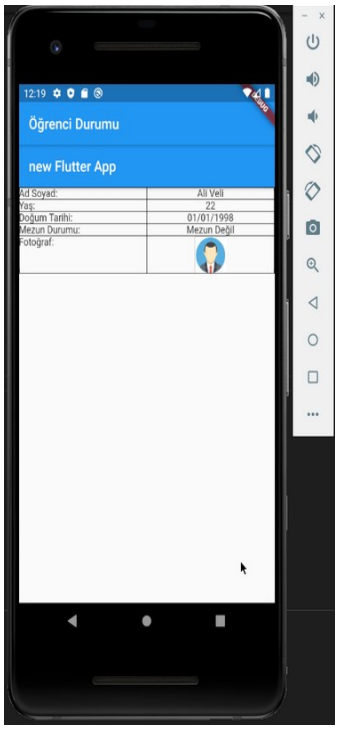
```
import 'package:flutter/material.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_core/firebase_core.dart';
Future<void> main() async {
WidgetsFlutterBinding.ensureInitialized();
await Firebase.initializeApp();
runApp(MyApp());
}
class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Flutter Basic Demo',
theme: ThemeData(
primarySwatch: Colors.blue,
visualDensity: VisualDensity.adaptivePlatformDensity,
),
home: MyHomePage(title: 'new Flutter App'),
);
}
}
class MyHomePage extends StatefulWidget {
MyHomePage({Key key, this.title}) : super(key: key);
final String title;
@override
_MyHomePageState createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
var url;
Widget _buildlistItem(BuildContext context, DocumentSnapshot
document) {
void showImage() async {final ref =
FirebaseStorage.instance.ref().child('erkekyuz.png');
```

```
// no need of the file extension, the name will do fine.
url = await ref.getDownloadURL();
}
showImage();
String mezunD;
if (document.data()['mezunDurumu'] == false) {
mezunD = "Mezun Değil";
}
if (document.data()['mezunDurumu'] == true) {
mezunD = "Mezun";
}
String yasString = (document.data()['yas']).toString();
return Scaffold(
appBar: AppBar(
title: Text(widget.title),
),
body: Table(
border: TableBorder.all(),
children: [
TableRow(children: [
Text("Ad Soyad: "),
Text(
document.data()['adSoyad'],
textAlign: TextAlign.center,
),
]),
TableRow(children: [
Text("Yaş: "),
Text(
yasString,
textAlign: TextAlign.center,
),
]),
TableRow(children: [
```

```
Text("Doğum Tarihi: "),
Text(
document.data()['dogumTarihi'],
textAlign: TextAlign.center,
),
]),
TableRow(children: [
Text("Mezun Durumu: "),
Text(
mezunD,
textAlign: TextAlign.center,
),
]),
TableRow(children: [
Text(
"Fotoğraf: ",
),
Image.network(url,
width: 50,
height: 50,
),
]),
],
),
);
}
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
title: Text("Öğrenci Durumu"),
),
body: StreamBuilder(
stream:
FirebaseFirestore.instance.collection('tablolar').snapshots()
```

```
,
builder: (context, snapshot) {
if (!snapshot.hasData) return const Text('Loading...');
return ListView.builder(
itemExtent: 100.0,
itemCount: snapshot.data.docs.length,
itemBuilder: (context, index) =>
_buildlistItem(context, snapshot.data.docs[index]),
);
}),
);
}
}
```

Ekran çıktısı şekil 16'daki gibi olacaktır:



Şekil 16: Proje 4, Flutter ekran çıktısı

4.4.3 Android Native SDK

Öncelikle bu şekilde, firebase'i başlatmalıyız:

```
FirebaseFirestore db = FirebaseFirestore.getInstance();
```

koduyla, database e bağlanmamız şarttır. Bunu yazmazsak database e bağlanılamaz. Daha sonra ise, aşağıda bahsedilen db.collection ile başlayan fonksiyon ile, oluşturulan koleksiyonu yani tabloya girecek verilerimizi seçeriz.

Tüm kodlar ise şu şekildedir:

MainActivity.java

```
package com.example.mezuniyet2nat;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import android.nfc.Tag;
import android.os.Bundle; import android.util.Log;
import android.widget.ArrayAdapter;
import android.widget.ListView; import
android.widget.TextView;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.FirebaseApp;
import com.google.firebase.firestore.FirebaseFirestore;
import com.google.firebase.firestore.QueryDocumentSnapshot;
import com.google.firebase.firestore.QuerySnapshot; import
com.google.firebase.storage.FirebaseStorage; import
com.google.firebase.storage.StorageReference;
import java.util.ArrayList;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);final String TAG =
"bisey";
```

```
TextView adSoyadAlan= (TextView)
findViewById(R.id.adSoyadDeger);
TextView yasAlan= (TextView) findViewById(R.id.yasDeger);
TextView dogumTarihiAlan= (TextView)
findViewById(R.id.dogumTarihiDeger); TextView mezunDurumuAlan=
(TextView) findViewById(R.id.mezunDurumuDeger);
FirebaseFirestore db = FirebaseFirestore.getInstance();
FirebaseStorage storageRef=FirebaseStorage.getInstance();
db.collection("tablo")
.get()
.addOnCompleteListener(new OnCompleteListener<QuerySnapshot>()
{
@Override
public void onComplete(@NonNull Task<QuerySnapshot> task) {
if (task.isSuccessful()) {
for (QueryDocumentSnapshot document : task.getResult()) {
Log.d(TAG, document.getId() + " => " + document.getData());
//document.getData().values().toArray(dataz);
String adSoyad = document.get("adSoyad").toString();
String yas=document.get("yas").toString();
String dogumTarihi=document.get("dogumTarihi").toString();
String mezunDurumu=document.get("adSoyad").toString();
String mezunDurumu2;
if(mezunDurumu=="true")
{
mezunDurumu2="mezun";
}
else{
mezunDurumu2="mezun değil";
}
adSoyadAlan.setText(adSoyad);
yasAlan.setText(yas);
dogumTarihiAlan.setText(dogumTarihi);
mezunDurumuAlan.setText(mezunDurumu2);
```

```
//String isCalender =
documentChange.getDocument().getData().get("Calender").toStrin
g();//String isEnablelocation =
documentChange.getDocument().getData().get("Enable
Location").toString();
}
} else {
Log.w(TAG, "Error getting documents.", task.getException());
}
}
});
}
}
```

activity_main.xml

```
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:background="#FFA500"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:gravity="center"
android:stretchColumns="0,1,2">
<TableRow
android:layout_width="100dp"
android:layout_height="121dp"
android:layout_margin="1dp"
android:layout_weight="1"
android:background="#FFA500"
>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_column="0"
```

```
android:layout_margin="1dp"
android:background="#FFA500"
android:gravity="center"
android:text="Anahtar"
android:textAppearance="?android:attr/textAppearanceLarge"
android:textStyle="bold" />
<TextView
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:layout_column="1"
android:layout_margin="1dp"android:background="#FFA500"
android:gravity="center"
android:text="Değer"
android:textAppearance="?android:attr/textAppearanceLarge"
android:textStyle="bold" />
</TableRow>
<TableRow
android:layout_width="100dp"
android:layout_height="100dp"
android:layout_margin="1dp"
android:layout_weight="1"
android:background="#FFA500"
><TextView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_column="0"
android:layout_margin="1dp"
android:background="#FFA500"
android:gravity="center"
android:text="adSoyad"
android:textAppearance="?android:attr/textAppearanceLarge" />
<TextView
android:id="@+id/adSoyadDeger"
android:layout_width="wrap_content"
```

```
android:layout_height="match_parent"
android:layout_column="1"
android:layout_margin="1dp"
android:background="#FFA500"
android:gravity="center"
android:textAppearance="?android:attr/textAppearanceLarge" />
</TableRow>
<TableRow
android:layout_width="fill_parent"
android:layout_height="0dp"
android:layout_margin="1dp"
android:layout_weight="1"
android:background="#FFA500">
<TextView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_column="0"
android:layout_margin="1dp"
android:background="#FFA500"
android:gravity="center"
android:text="yas"
android:textAppearance="?android:attr/textAppearanceLarge" />
<TextView
android:id="@+id/yasDeger"
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:layout_column="1"
android:layout_margin="1dp"
android:background="#FFA500"android:gravity="center"
android:textAppearance="?android:attr/textAppearanceLarge" />
</TableRow>
<TableRow
android:layout_width="fill_parent"
android:layout_height="0dp"
```

```
android:background="#FFA500"
android:layout_margin="1dp"
android:layout_weight="1"
>
<TextView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_column="0"
android:layout_margin="1dp"
android:background="#FFA500"
android:gravity="center"android:text="dogumTarihi"
android:textAppearance="?android:attr/textAppearanceLarge" />
<TextView
android:id="@+id/dogumTarihiDeger"
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:layout_column="1"
android:layout_margin="1dp"
android:background="#FFA500"
android:gravity="center"
android:textAppearance="?android:attr/textAppearanceLarge" />
</TableRow>
<TableRow
android:layout_width="fill_parent"
android:layout_height="0dp"
android:layout_margin="1dp"
android:background="#FFA500"
android:layout_weight="1"
>
<TextView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_column="0"
android:layout_margin="1dp"
```

```
android:background="#FFA500"
android:gravity="center"
android:text="mezunDurumu"
android:textAppearance="?android:attr/textAppearanceLarge" />
<TextView
android:id="@+id/mezunDurumuDeger"
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:layout_column="1"
android:layout_margin="1dp"
android:background="#FFA500"
android:gravity="center"android:textAppearance="?
android:attr/textAppearanceLarge" />
</TableRow>
<TableRow
android:layout_width="fill_parent"
android:layout_height="0dp"
android:layout_margin="1dp"
android:layout_weight="1"
android:background="#FFA500">
<TextView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_column="0"
android:layout_margin="1dp"
android:background="#FFA500"
android:gravity="center"
android:text="fotoğraf"
android:textAppearance="?android:attr/textAppearanceLarge" />
<TextView
android:id="@+id/fotografDeger"
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:layout_column="1"
```

```
android:layout_margin="1dp"
android:background="#FFA500"
android:gravity="center"
android:textAppearance="?android:attr/textAppearanceLarge"
/></TableRow>
</TableLayout>
```

Ekran çıktısı şekil 17'deki gibi olacaktır:

Not: Bu çıktı, aslında borderless bir tablodur.

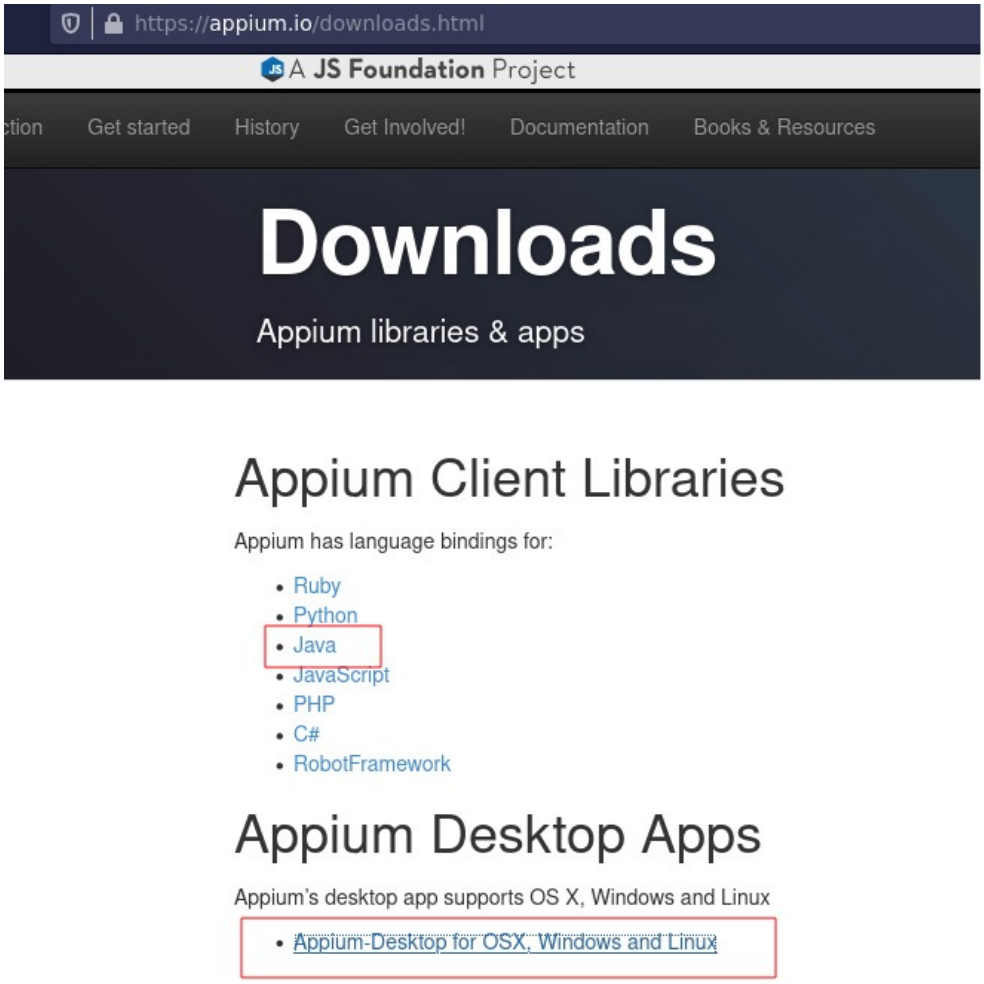
Şekil 17: Proje 4, Android Native SDK ekran çıktısı

4.5 OTOMASYON TEST FRAMEWORK

Not:Şu anda sadece, calculator üzerinde örnek bir uygulama üzerinden gösterilebilmiştir. Buradan benzeşim yoluyla, hepsi için bir test sistemi rahatlıkla yapılabilir.

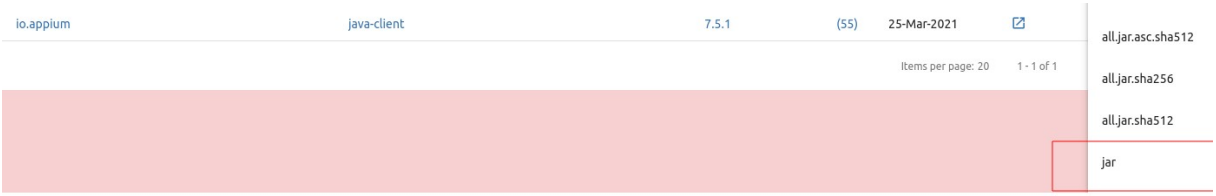
Bunun için, çeşitli platformlar bulunmaktadır. Ben appium'u kullandım. Bunun için, öncelikle appium adlı uygulamayı, intellij idea programını ve android studio programını yüklemeliyiz.

Şekil 18'de görüldüğü gibi appium'u java için indiriyoruz:



Şekil 18: Appium'u indirme (java için)

Desktop app ini, ve java ya tıklayıp, buradan download linkine tıklayıp, jar olanı indirmeliyiz. Bu daha sonra kullanılacaktır.



Şekil 19: Jar dosyası olarak indirme

Daha sonra, appium u açıp, android studio dan emülatörü de açıyoruz. Appium daki standart ekrandan, start server a basıyoruz.

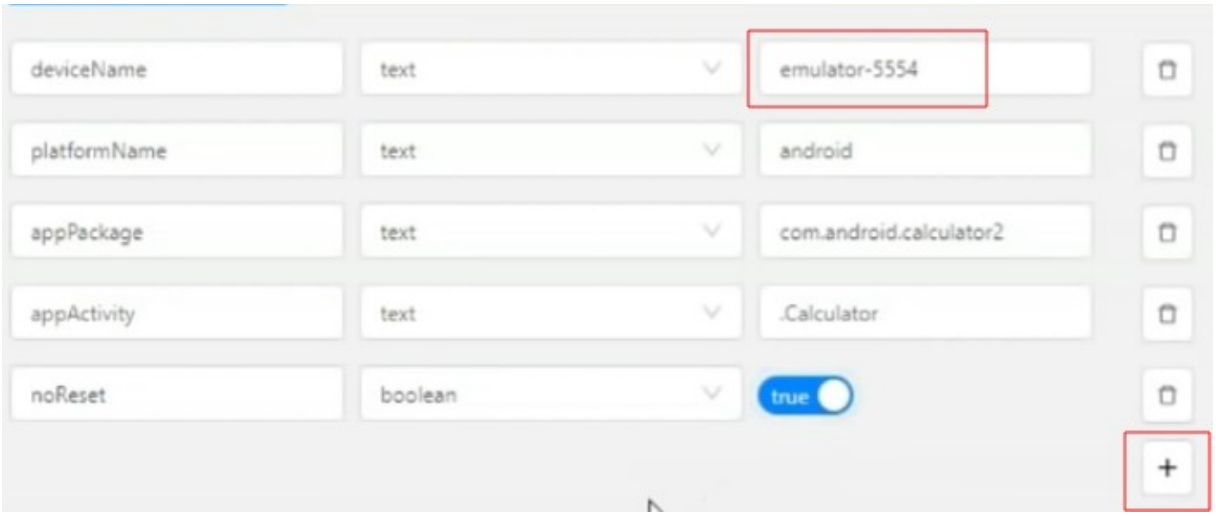
Şekil 20: Appium serveri başlatma

Emulator açıldıktan ve android ana ekranı geldikten sonra, terminal veya cmd den şunu yazıyoruz:

```
adb devices
```

Şekil 21: adb'nin cihazı tanıdığını gösteren bilgi

Oluşan çıktıyı biraz sonra kullanacağız. Appium da start server a bastıktan sonra bu şekilde bir ekran gelmektedir. Burada, mercek ikonuna tıklamalıyız:

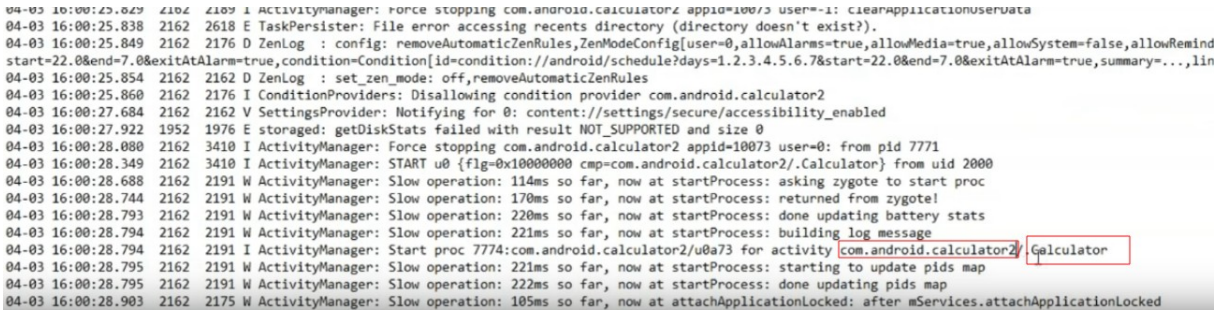


Şekil 22: Appium'a emülatör bilgilerini girme

Daha sonra, yukarıda görüldüğü üzere, deviceName diye bir text field oluşturup, bu çıktıyı oraya yerleştiriyoruz. PlatformName android olmalı. AppPackage I ise şu şekilde elde edebiliriz:

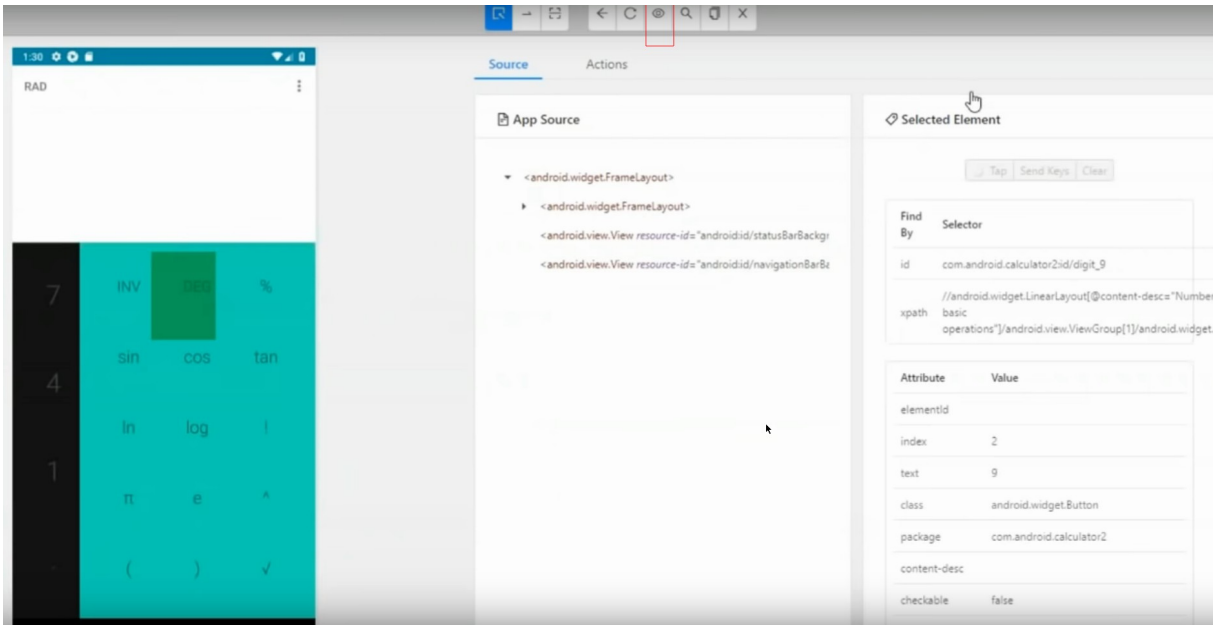
```
Adb logcat >orneklogfile.txt
```

diye örnek bir log dosyası oluşturuyoruz, bu anda, calculator uygulamasını çalıştırıyoruz. Adb, bu sırada akan veriyi dinleyip, bu txt dosyasına kaydetmektedir. Biz bu log dosyasından, ilgili verileri çekip, buraya yerleştirebiliriz. İlgili örnek txt dosyasında, calculator yazan kısmı arıyoruz:



Şekil 23: Appium log dosyası

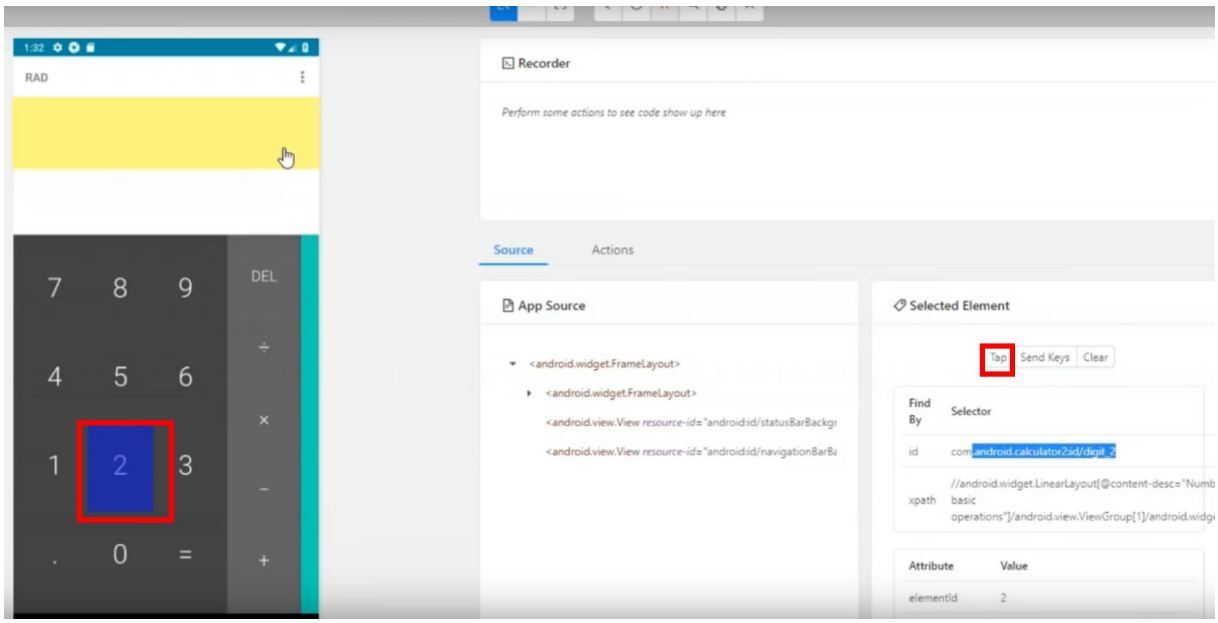
Sonra, start server a basıyoruz(sağ alttadır):



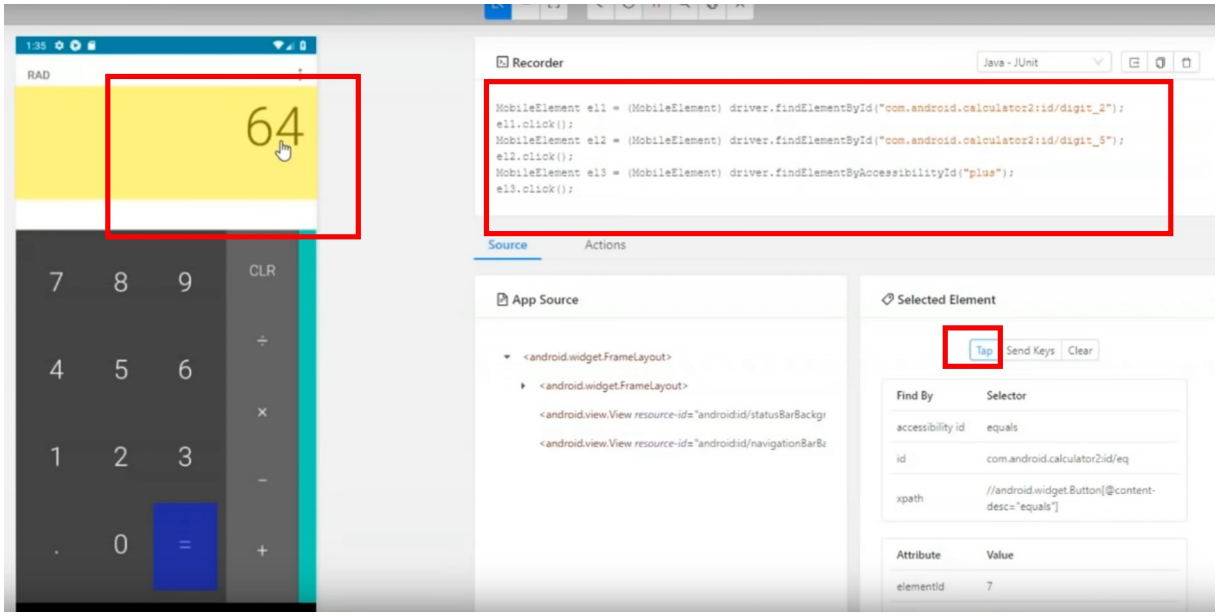
Şekil 24: Appium calculator otomasyon yapma

Üstte bulunan göz simgesine, yani start recording e basıyoruz. Buna bastıktan sonra, eğer calculatorde herhangi bir rakama basıp, sağdaki "tap" butonuna basınca, bunu kaydetmiş olur, ve bunu uygun bir java koduna dönüştürmüş olur. Ayrıca, tapa basınca bu basmayı gerçekten emulate edip, emulatorde de basmaktadır. Eğer bunu emulatörü açarsak görebiliriz. Ki bu

isteğe göre python koduna da dönüştürülebilir. Fakat biz java kısmıyla ilgilenmekteyiz. Bunu daha sonra kullanacağız. Misal 25+39=64 işlemini kayıt edeceğiz.



Şekil 25: Tuşlara basmayı emüle etme



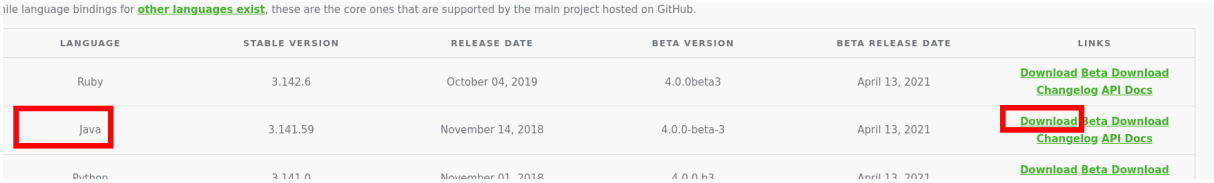
Şekil 26: Calculator cevap

Sağ üstteki kodları kopyalayıp alıyoruz. Daha sonra, intellij idea da yeni bir proje oluşturuyoruz. Burada, public static void main fonksiyonu içine, bu kodları kopyalıyoruz.

Daha sonra birtakım indirmeler yapacağız. Ama önce, idea projectinin olduğu yere gidip, src içinde lib adlı bir klasör oluşturmalıyız. İndirilen kütüphaneleri buraya ekleyeceğiz.

Buradaki linkten, download linkine basıp, java uzantılı olanı indirmeliyiz:
https://search.maven.org/search?q=g:io.appium%20AND%20a:java-client

Selenium server(grid) latest sTablo versiyonunu indirmeliyiz:
https://www.selenium.dev/downloads/
Ayrıca, alttaki java dosyasını da indirmeliyiz(aynı sayfadadır):



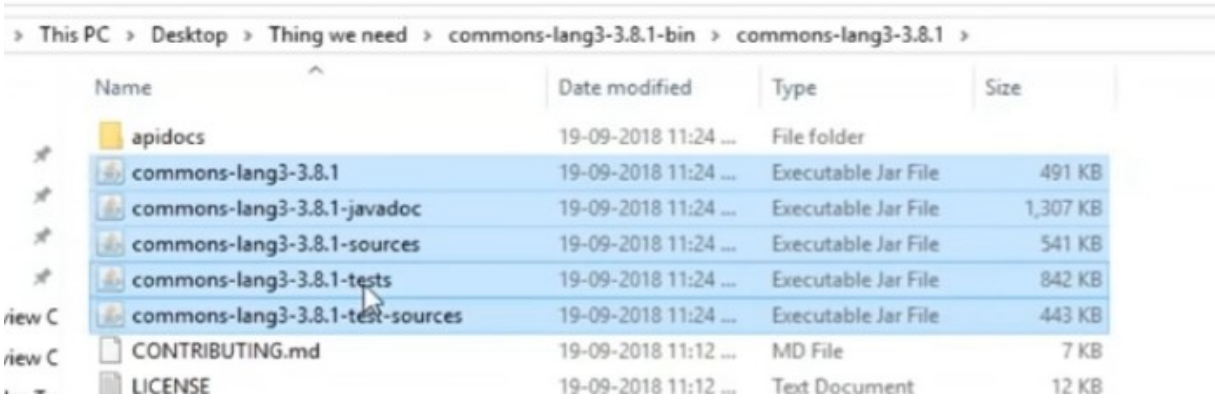
Şekil 27: Selenium server dosyalarını indirme

Bu linkteki lang3 kütüphanesini de indirmeliyiz:

https://commons.apache.org/proper/commons-lang/download_lang.cgi

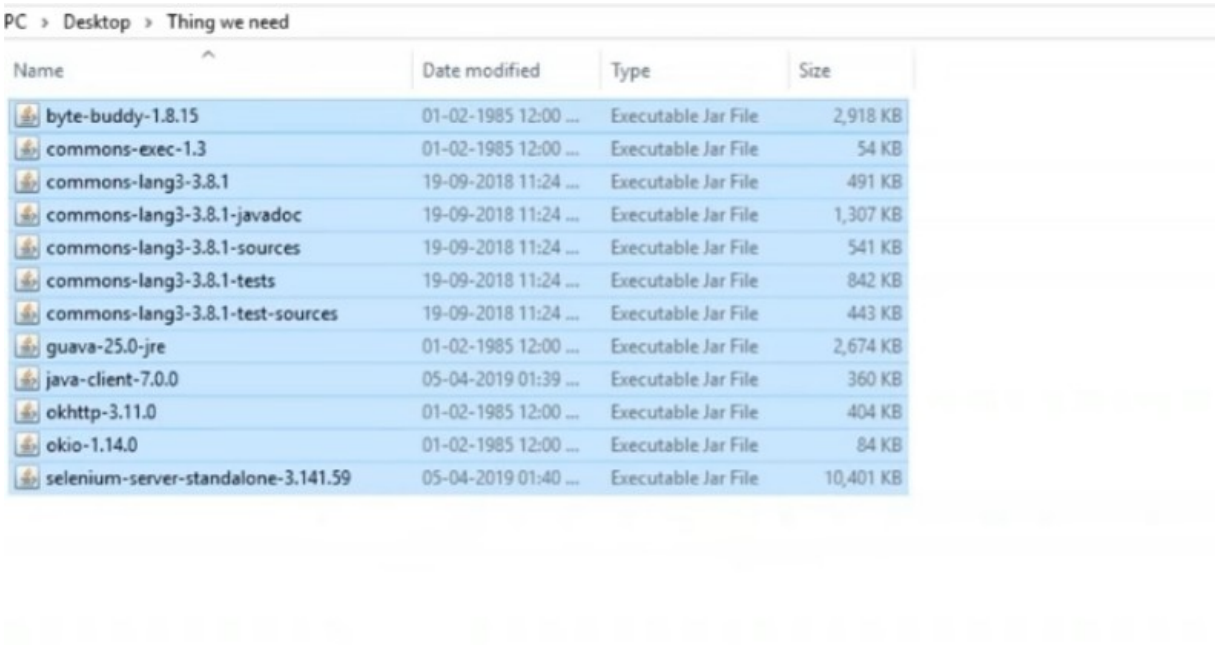
Şekil 28: commons-lang3 kütüphanesini indirme

Daha sonra, hepsinin içini açıp, mesela selenium daki libs deki tüm dosyaları çıkartıp, mevcut idea projesinde açtığımız lib klasörüne atıyoruz.

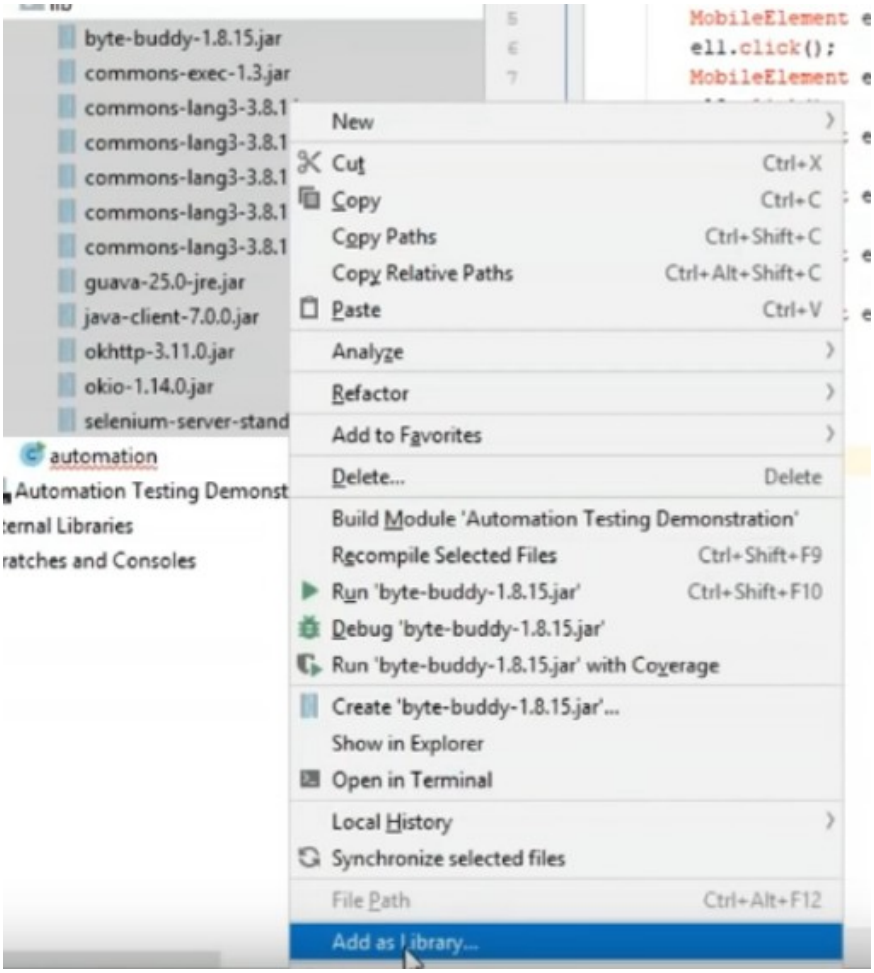


Şekil 29: commons lang dosyalarını ilgili yere yerleştirme

Lib dosyasında, toplam 12 tane dosya mevcuttur.



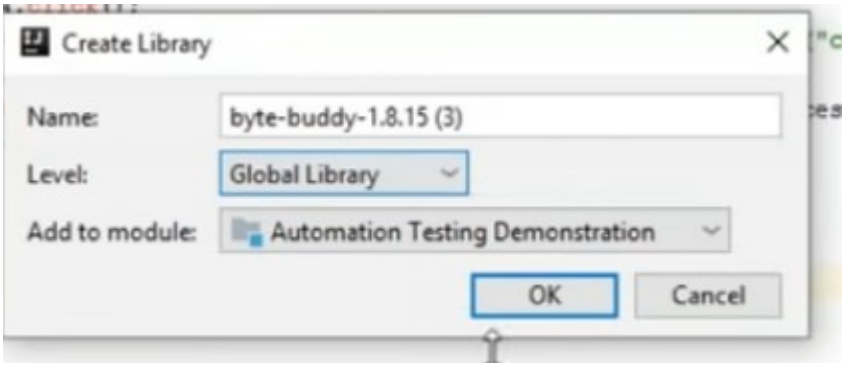
Şekil 30: Dosyaları intellij idea'ya yerleştireceğiz



Şekil 31: Intellij Idea'da kütüphane olarak tanıtma

Daha sonra, bu dosyaları, intellij idea içinden seçip, sağ tıklayıp; add as a library seçeneğine tıklıyoruz.

NOT: Global library olarak ekleyiniz.



Şekil 32: Create Library

Daha sonra, şu kodları ekliyoruz:

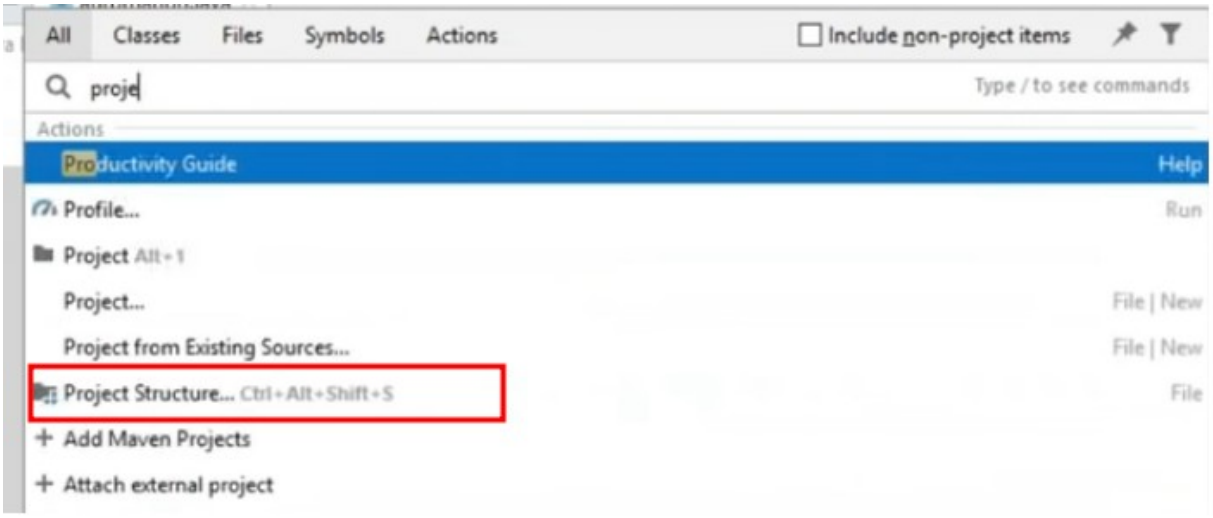
```
public static void main(String args[]) throws MalformedURLException {

    DesiredCapabilities dc = new DesiredCapabilities();

    dc.setCapability(MobileCapabilityType.DEVICE_NAME, value: "emulator-5554");
    dc.setCapability( capabilityName: "platformName", value: "android");
    dc.setCapability( capabilityName: "appPackage", value: "com.android.calculator2");
    dc.setCapability( capabilityName: "appActivity", value: ".Calculator");

    AndroidDriver<AndroidElement> ad = new AndroidDriver<AndroidElement>(new URL( spec: "http://127.0.0.1:4723/wd/hub"),dc)

    MobileElement el1 = (MobileElement) ad.findElementById("com.android.calculator2:id/digit_2");
    el1.click();
    MobileElement el2 = (MobileElement) ad.findElementById("com.android.calculator2:id/digit_5");
    el2.click();
    MobileElement el3 = (MobileElement) ad.findElementByAccessibilityId( using: "plus");
    el3.click();
    MobileElement el4 = (MobileElement) ad.findElementById("com.android.calculator2:id/digit_3");
    el4.click();
    MobileElement el5 = (MobileElement) ad.findElementById("com.android.calculator2:id/digit_9");
    el5.click();
    MobileElement el6 = (MobileElement) ad.findElementByAccessibilityId( using: "equals");
    el6.click();

    Assert.assertEquals(ad.findElementById("com.android.calculator2:id/result").getText(),64)
}
```

Şekil 33: Kodlama

Not: Bazı kodlar tanımlanmayabilir. Bu kütüphaneler eklenince, üzerine gelip import class diyebilmekteyiz.

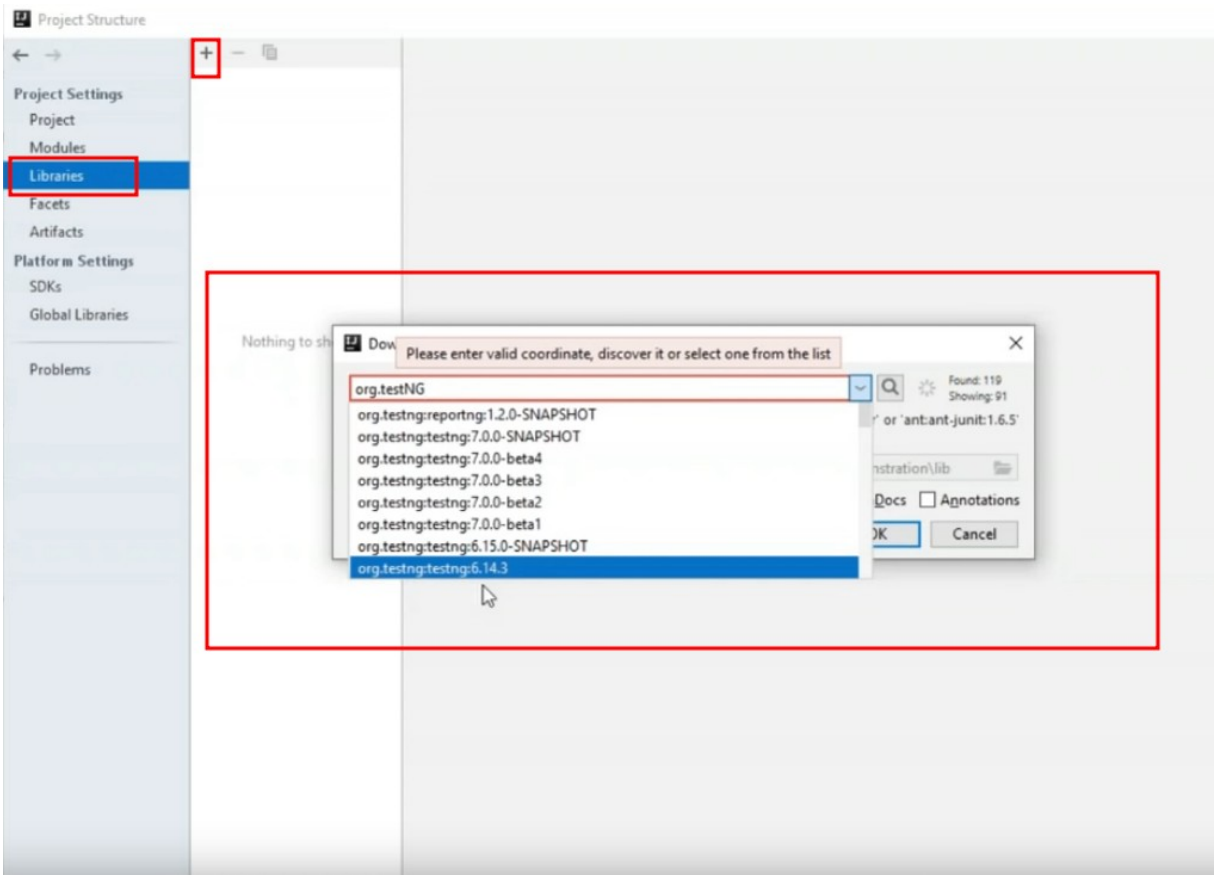
Assert fonksiyonu ise, 64 ün sonuca yazılıp yazılmadığını kontrol etmektedir. Bunu intellij loglarında görebiliriz. Buraya test etmek için bilerek yanlış rakam girilebilir. Bu daha sonra, programlanarak, test sonuçları başarısız vs gibi birtakım mesaj kutuları şeklinde verilebilir.

Ayrıca aşağıdaki Assert fonksiyonu için de başka bir kütüphane daha eklememiz gerekmektedir. Bunun için de, 2 defa shifte basıp, açılan menüde project structure a girelim. Bu, ayrıca ayarlarda da mevcuttur.



Şekil 34: Project Structure düzenleme

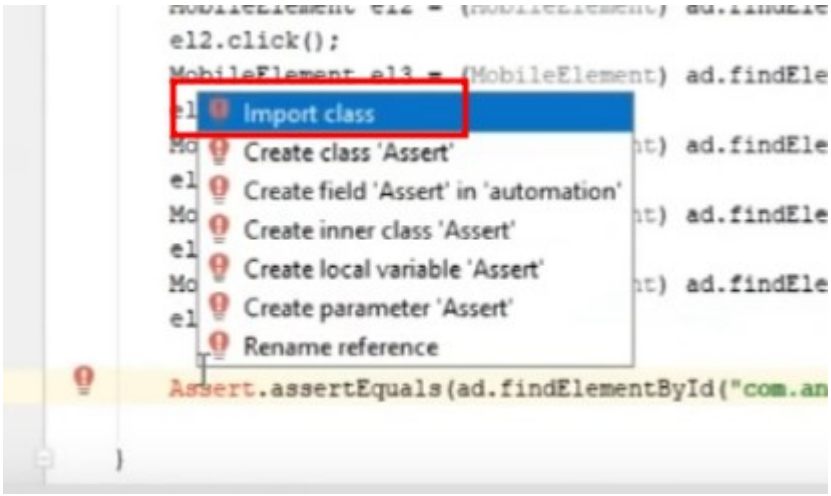
Soldan, libraries kısmına gelip, + ya basıp, arama kısmına şunu yazıyoruz:
org.testng:testng



Şekil 35: Testing için gerekli kütüphaneyi indirme

En yeni versiyondan bir önceki versiyonu seçmeniz genel mühendislik disiplini uyarınca, stabil çalışması açısından tavsiye edilmektedir. Buglardan daha az etkilenmek için bu gereklidir.

Bunu yaptıktan sonra, Assert fonksiyonunu içeren kütüphaneyi import edebiliriz:



Şekil 36: Assert fonksiyonunu ekleme

```
import org.testng.Assert;
```

adlı import yukarıya eklenmektedir.

Lütfen assert için sadece org.testng ye dahil olan Assert'i ekleyiniz. Bundan sonra, programı çalıştırdığımızda, kendi başına calculator ü çalıştırıp, tuşlara kendi başına basacaktır. Mesela proje 3(lokasyon ve kamera izni) için yapmak isteseydik, butonun yerini işbu uygulamada belirleyip, tıklayıp, çıkan kodları intellij idea ya yazıp, daha sonra çıkacak outputları assert ile kontrol edebilirdik. Sonuçta çalışıp çalışmadığını görüyor olurduk.

Kısaca, kodlar şu şekilde olmalıdır:

```
package com.company;
import io.appium.java_client.MobileElement;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;
```

```
import io.appium.java_client.remote.MobileCapabilityType;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.testng.Assert;
import java.net.MalformedURLException;
import java.net.URL;
public class Main {
public static void main(String[] args) throws
MalformedURLException {
// write your code here
DesiredCapabilities dc= new DesiredCapabilities();
dc.setCapability(MobileCapabilityType.DEVICE_NAME,"emulator-
5554");
dc.setCapability("platformName","android");
dc.setCapability("appPackage","com.android.calculator2");
dc.setCapability("appActivity","Calculator");
AndroidDriver<AndroidElement> ad = new
AndroidDriver<AndroidElement>(new
URL("http://127.0.0.1:4723/wd/hub"),dc);
MobileElement el1 = (MobileElement)
ad.findElementById("com.android.calculator2:id/digit_8");
el1.click();
MobileElement el2 = (MobileElement)
ad.findElementByAccessibilityId("multiply");
el2.click();
MobileElement el3 = (MobileElement)
ad.findElementById("com.android.calculator2:id/digit_8");
el3.click();
MobileElement el4 = (MobileElement)
ad.findElementByAccessibilityId("equals");
el4.click();
Assert.assertEquals(ad.findElementById("com.android.calculator
2:id
/result").getText(),64);
```

Bu kodu çalıştırdığımızda, eğer 8*8 işlemini calculator uygulayınca, eğer sonuç 64 yazılmazsa uyarı verecektir. Buradan, benzer şekilde tüm bu projeler için, verilen değişkenleri ekranda çıktı olarak görmedikçe, bu hata verilecektir. Böylece ekranda verilen değişkenlerin çıkıp çıkmadığı konusunu, buna benzer bir programla yapabilirdik. Fakat zaman yetişmediği için yapılamamıştır.

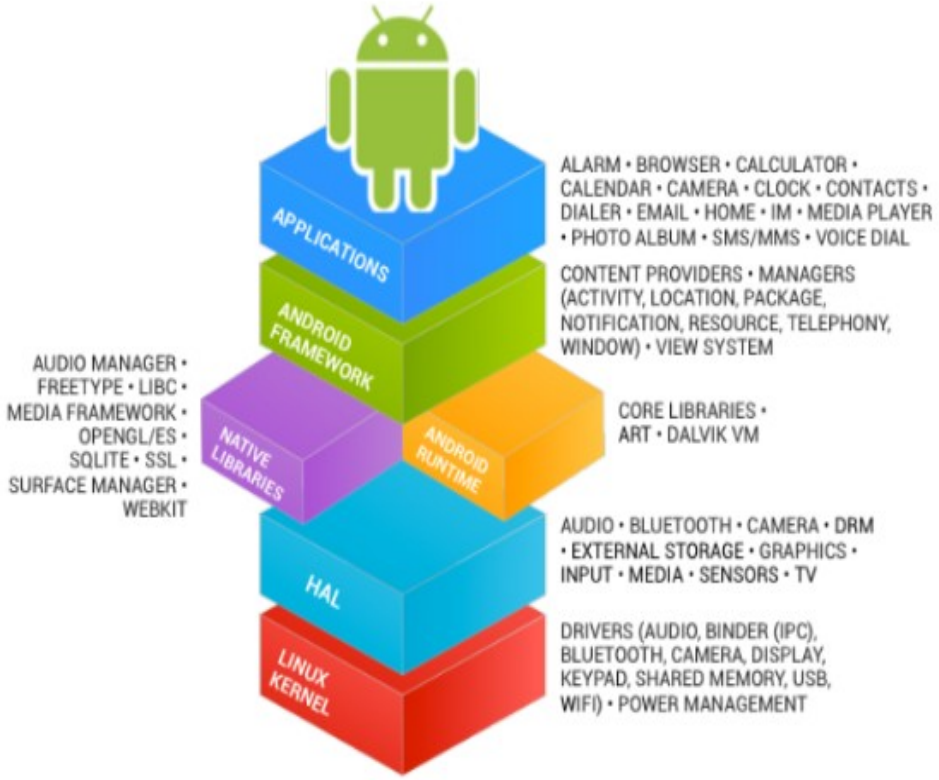
BÖLÜM 5: TARTIŞMALAR

5.1 ANDROID

5.1.1 İşletim Sistemi Yapısı

Android platformu birçok katmandan oluşur ve Linux çekirdeğini temel alır. Android mimarisi, Şekil a'da görselleştirildiği gibi birlikte Android platform yığınını oluşturan bir dizi katman olarak görülüyor. Android uygulamaları, bir API'den çekirdek bileşenlere erişilebilen uygulama çerçevesine dayanır. Çoğu durumda, uygulama geliştiricilerinin, altta yatan Android işlevselliğine erişmek için bu üst düzey çerçeveden daha derine inmeleri gerekmeyecektir. Erişim, sistem çağrılarının istemciden örneğin pencere yöneticisine veya activity yöneticisine yapılmasına izin veren Bağlayıcı İşlemler Arası İletişim (Bağlayıcı IPC) mekanizması tarafından etkinleştirilir. Pencere yöneticisi ayrıca, düşük seviyeli yüzey oluşturucu ile iletişim kurmak için bir bağlayıcı örneği kullanır ve dolaylı olarak müşterinin Android yığınındaki düşük seviyeli bileşenlerle konuşmasına izin verir.

Bahsedilen pencere yöneticisi ve activity yöneticisi gibi bileşenlerin, Android yığınındaki Android çerçevesinin altında, sistem hizmetleri katmanında olduğu kabul edilir. Sistem hizmetleri katmanı, yerel kitaplıkları ve Android çalışma zamanı bileşenlerini içerir. Bu orta bileşenler, daha düşük seviyeli sürücülere bir arayüz sağlayan Donanım Soyutlama Katmanı (HAL) ile iletişim kurar. Bu şekilde, HAL standardı ile uygulandığında bağımsız olarak çalışacakları için herhangi bir üst seviye sistemle uyumlu olacak daha düşük seviyeli sistemler tanıtılabilir. Bir donanım aygıtı dağıtıcısı, donanım bileşeni sürücüleri ile HAL arasındaki iletişimi en uygun yol hangisi olursa olsun özelleştirmekte özgürdür. Ancak, HAL ile sistem hizmet katmanı arasındaki iletişim, bir meta dosyada açıklanan uygulama spesifikasyonlarına sahip bir standardı izlemelidir. Bir mobil aygıttaki her donanım bileşeninin, en yaygın şekilde genel bir modül yapısını genişleten kendi HAL örneği vardır. Android yığınının altında, Android platformunun çekirdeği olarak hizmet veren Linux çekirdeği bulunur. Android, Binder IPC sürücüsü ve mobil ortamda yardımcı olan diğer sürücülerle genişletilmiş bir Linux çekirdeği sürümünü uygular[1,2,3].



Şekil 37: Android sistem yapısı

5.1.2 Uygulama Yapısı

Android uygulamaları, farklı uygulama bileşenleri kullanılarak oluşturulmuştur. Dört farklı bileşen türü vardır: Activityler, Hizmetler, İçerik Sağlayıcılar ve Yayın Alıcıları. Bu bileşenlerin her birinin farklı bir amacı vardır ve kendi varlığı olarak var olur. Birlikte uygulamayı oluştururlar ve davranışını tanımlarlar. Her uygulamanın bileşenlerini uygulama bildirim dosyasında bildirmesi gerekir. Bu bildirim dosyası ayrıca uygulamanın hangi izinlere ihtiyacı olduğunu, uygulamayı çalıştırmak için hangi işletim sistemi sürümünün gerekli olduğunu, uygulama tarafından hangi donanım bileşenlerinin kullanıldığını ve uygulamada herhangi bir Android kitaplığının kullanılıp kullanılmadığını da tanımlar.

5.1.3. Uygulama Bileşenleri

• Activity, kullanıcı arayüzüne sahip tek bir ekrandır. Her Activity kendi başına çalışmalı ve başka bir uygulamadan başlatılabilmelidir. Tüm Aktiviteler, tüm kullanıcı arayüzlerini ve bunlar arasındaki geçişleri temsil ettikleri için birlikte uygulamanın kullanıcı deneyimini oluşturur.

• Hizmetler, uygulamanın arka planında kullanıcı arabirimleri olmadan çalışan bileşenlerdir. Dosya indirme veya arka planda ses çalma gibi arka plan görevlerini yürütürler. Hizmetler, faaliyetler gibi diğer bileşenler tarafından başlatılır ve etkileşimde bulunur.

• İçerik Sağlayıcılar, bir uygulamada veri işlemeye yönelik bileşenlerdir. Uygulamalar ile ilgili verileri kaydetmek ve değiştirmek için kullanılırlar. Her bir İçerik Sağlayıcı, farklı uygulamaların kendi başına işlenen verileri değiştirmesine izin verecek şekilde ayarlanabilir. Örneğin, uygulamanızın bildirimi açıkça bu sağlayıcıya erişim isterse, Android adres defterini işleyen sağlayıcıya erişilebilir. İçerik Sağlayıcılar, yalnızca bir uygulamaya özel verileri işlemek için de kullanılabilir.

• Yayın Alıcıları, tüm uygulamalar tarafından erişilebilen sistem genelindeki yayınları yönetir. Sistem, örneğin ekran kapandığında veya pil zayıfladığında mesaj yayınlar. Bu mesaja abone olan uygulamalar daha sonra bu olayla ilgili davranışlarını değiştirebilir. Genellikle bu bileşenler çok fazla iş yapmaz, bunun yerine belirli bir olay alındığında bir hizmet başlatır. Ancak, kullanıcının uygulamaya özel bir şey hakkında bilgilendirilmesi gerekiyorsa durum çubuğu bildirimleri oluşturma olasılıkları vardır.

Tüm componentler bağımsız olarak işlevsel olduğundan, her uygulama başka bir uygulamanın bir componentini başlatabilir. Örneğin, bir uygulamanın kamerayı kullanması gerekiyorsa, Android sistem kamera etkinliğini başlatabilir. Her uygulama kendi süreci olarak çalıştırıldığından, bu etkinliği kendi başına başlatmak için gerekli izinlere sahip değildir. Bunun yerine, sisteme etkinliği ve amacını sağlamak için Intent sistemini kullanır. Bundan sonra, sistem ihtiyaç duyulan bileşeni başka bir işlemde başlatır ve başlatılan bileşen, intenti talep eden sürece ait değildir. Bileşen çalışmasını bitirdiğinde kapanır ve istenen verileri intenti talep eden uygulamaya geri gönderir. Android sisteminin güçlü bir özelliği, implicit intentlerin kullanılabilmesidir. Implicit intentler, bir activity tarafından gerçekleştirilebilecek eylemlerin bildirim dosyasında bildirildiği anlamına gelir. Daha sonra başka bir uygulama sisteme bu eylemi gerçekleştirmek istediğini bildirirse, kullanıcı bu tür eylem için bir activity sunan tüm uygulamalardan seçim yapabilecektir.

Herhangi bir bileşen herhangi bir uygulamadan başlatılabildiğinden, Android uygulamalarının diğer birçok program gibi ana işlevi yoktur. Bunun yerine, her bileşenin farklı olayları işleyen kendi yaşam döngüsü vardır. Activity yaşam döngüsü, başka bir activity açıldığında veya

kapatıldığında activitylerin nasıl davranması gerektiğini ele alır. Başka bir activity başlatıldığında, mevcut activity bir dizi açık activity üzerine itilir. Sistem, yığından geri çıkıncaya kadar activity durumunu bellekte tutacaktır [4].

5.1.4 Activity yaşam döngüsü

Bir activitynin üç farklı durumu vardır: devam ettirildi, duraklatıldı ve durduruldu. Sürdürülen bir activity uygulamanın ön planında çalışıyor ve mevcut activitydir. Duraklatılmış bir activity arka planda çalışıyor ve koşu activitynin altında ekranda kısmen görünüyor. Durdurulmuş bir activity ekranda görünmez, ancak hafızada hala canlıdır. Hafıza yetersizliği durumunda hem durdurulan hem de duraklatılan faaliyetler sistem tarafından kapatılabilir. Bir activity kapatıldığında ve yeniden başlatılması gerektiğinde, herhangi bir kaydedilmiş durumu olmayacak, bunun yerine baştan yaratılacaktır.

Her activity, activity yaşam döngüsünün farklı noktalarında çağrılan 6 farklı geri aramayı uygular. Activity oluşturulduğunda ilk çağrı onCreate'e yapılır ve activity çıkması gerektiğinde onDestroy'a son çağrı yapılır ve kullanılan tüm kaynakları serbest bırakır. Activity onStart geri aramasını aldığında, kullanıcı artık onu görebilir ve etkileşimde bulunabilir ve activity olayları işlemelidir. Benzer şekilde, activity artık kullanıcı tarafından görülemediğinde onStop geri araması alınır. Bu geri aramalar, kullanıcı activity gösterdiğinde ve gizlediğinde bir activity yaşam döngüsü sırasında birkaç kez çağrılabilir. OnResume geri araması alındığında, activity diğer tüm activitylerin önündedir ve kullanıcı girişi odağına sahiptir. Activity gizlendiğinde, örneğin ekran kilitliyse, Duraklatma geri araması alınacaktır. Activity kullanıcı odağına geri döndüğünde onResume çağrılacaktır. Bu iki durum arasındaki döngü, bir activity yaşam döngüsü sırasında sıklıkla yeniden ziyaret edilir [5].

5.1.5 Android Uygulamaları

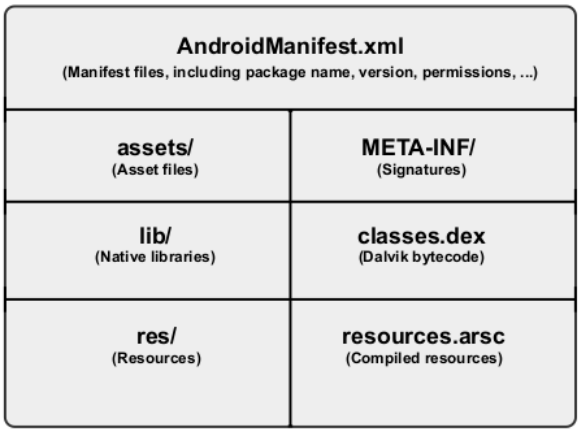
Android uygulamaları, .apk sonekiyle bir arşiv dosyasında paketlenir ve dağıtılır. Bir APK dosyasının ana içeriği Dalvik bayt kodunu (.dex dosyaları), kaynakları, varlıkları, sertifikaları ve bir manifest dosyasını içerir. Dalvik kodu, Android 5.0 a kadar kullanılmış, daha sonra yerine başka bir sistem getirilmiştir.

Bir APK'nin yapısı Şekil 1'de gösterilmektedir ve her bir bileşenin ayrıntılı sorumlulukları aşağıda sunulmuştur:

• AndroidManifest.xml: Paket adı, uygulama kimliği, istenen izinler, activityler, hizmetler, yayın alıcıları, içerik sağlayıcılar gibi uygulama bileşenleri, donanım ve yazılım özellikleri

gibi paket bilgileri dahil olmak üzere Android uygulamalarının temel bilgilerini depolayan bir Manifest dosyası .

• assets/: Dizin bir uygulamanın varlıklarını içerir.



Şekil 38: Android dosya yapısı

• META-INF /: APK'nin imzalarını ve sertifikalarını içerir.

• lib /: derlenmiş yerel kitaplıkları .so sonekiyle depolar.

• classes.dex: .dex dosyaları, Dalvik sanal makinesi tarafından anlaşılabilen, tüm sınıflardan oluşan derlenmiş bayt kodunu içerir.

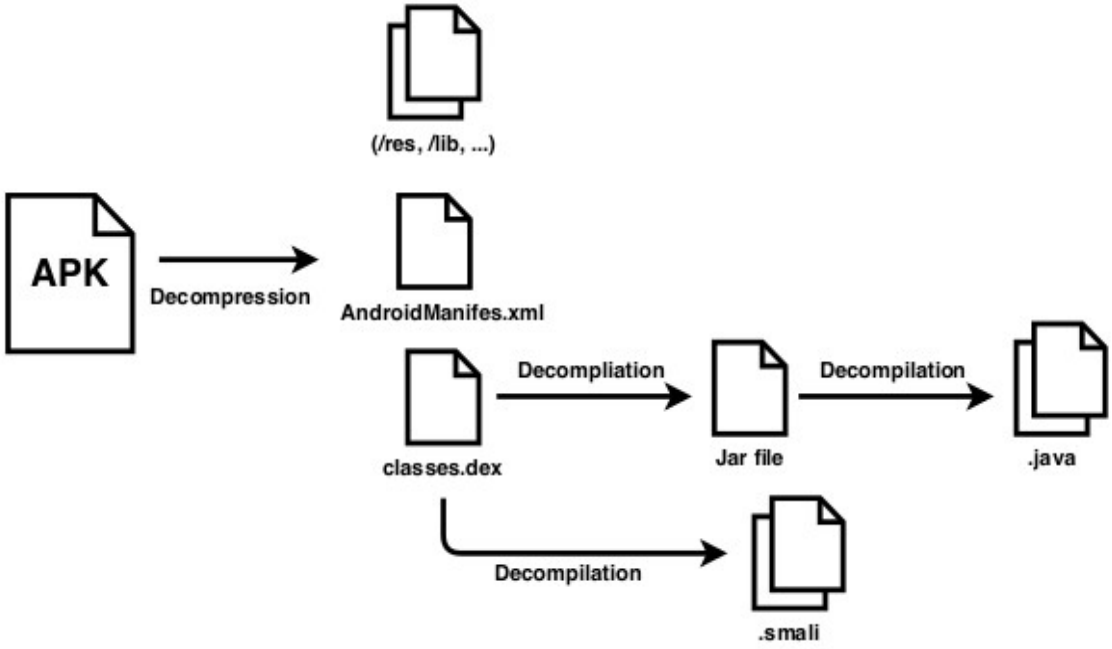
• res /: string kaynakları, düzenler vb. Gibi kaynaklar.arsc'de derlenmemiş kaynak belgelerini içerir.

• resource.arsc**:** ikili biçimde derlenmiş kaynak dosyası.

Android uygulamaları normalde Java ve Kotlin gibi üst düzey programlama dilleriyle geliştirilir ve ardından Android platformunda Dalvik sanal makinesi tarafından yorumlanabilen Dalvik çalıştırılabilir koda dönüştürülür. Bununla birlikte, Dalvik bayt kodu insan tarafından okunamaz, bu nedenle classes.dex'i yorumlanabilir bir biçime dönüştürebiliriz ve prosedürler Şekil 2'de açıklanmıştır. İlk yol, classes.dex dosyası jar dosyalarını dex2jar araçlarıyla ve ardından jar dosyalarını Java kaynak kodlarına dönüştürün. Bunun dışında, classes.dex, APKTool gibi bazı tersine mühendislik araçlarıyla kolayca ayrıştırılabilen sözde koda, smali diline de derlenebilir.

Android tabanlı mobil cihazlar, masaüstü sistemlere kıyasla çok farklı dosya kullanım modelleri sergiler: Birincisi, mobil uygulamalar işlemsel veri yönetimi için büyük ölçüde gömülü bir veritabanı katmanı olan SQLite'a dayanır; İkinci olarak, Android, yürütülebilir

ikili dosyalar ve derlenmiş kaynaklar gibi çeşitli çalışma zamanı kaynaklarını büyük yürütülebilir dosyalar halinde paketler. Veritabanı dosyaları yazma trafiğinin büyük bir kısmına katkıda bulunurken, yürütülebilir dosyaların boyutu büyüktür. Sezgisel olarak, [6-9]



Şekil 39: APK dosyası compile adımları

'te bildirilenler gibi mevcut dosya sıkıştırma teknikleri, yazma stresini azaltmak ve yerden tasarruf etmek için benimsenebilir. Bununla birlikte, mevcut tasarımlar etkili olmayabilir veya mobil cihazlarda kullanıcı deneyiminin düşmesi riskini alabilir.

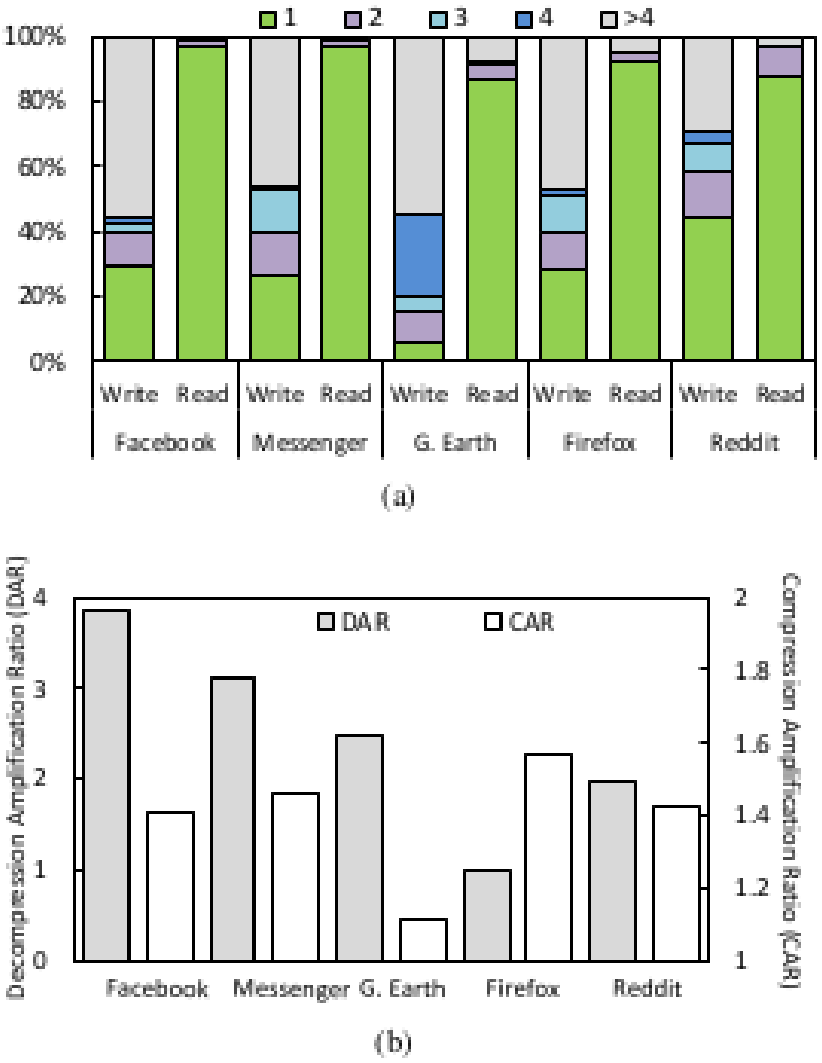
SQLite veritabanlarının işlenmesi birçok küçük dosya üzerine yazma ve ekleme işlemi oluşturur [10,11]. Bu küçük işlemler, depolama alanında büyük ölçüde parçalanmıştır ve nedeni dosya parçalanma sorunuyla ilgili olarak belirlenmiştir [12]. Parçalanmış güncellemelerle, Ext4 [13] gibi

geleneksel bir yerinde güncelleme dosya sisteminin üstünde dosya sıkıştırması, bir güncellemeden sonra sıkıştırılmış bir dosya bloğu orijinal alanına sığmayabileceğinden depolama alanında birçok delik oluşturabilir. Küçük ekleme işlemleriyle, dosya sıkıştırma, daha iyi bir sıkıştırma sonucu için yeni veriler üzerinde büyük bir sıkıştırma penceresi kullanamaz. Android yürütülebilir dosyalarıyla ilgili olarak, yükleme sırasında sırayla yazılsalar da, uygulama başlatılırken küçük, rastgele okuma işlemlerine tabidirler. Rastgele dosya ofsetlerinden dosya bloklarının açılması, blok G / Ç'nin daha büyük birim boyutu nedeniyle G / Ç okuma ek yükünü önemli ölçüde artırır [9, 14]. Ancak, Android mobil

cihazlardaki veritabanı dosyala rının ve yürütülebilir dosyaların bu benzersiz dosya erişim modelleri, önceki dosya sıkıştırma çalışmasında iyi çalışılmamıştır.

5.1.6 I / O Sistemi ve Mobil Cihazların Depolanması

Android, günümüzde mobil cihazlar için baskın işletim sistemidir. Android I / O sistemi, ana sistem yazılımı ve bir flash depolama cihazından oluşur. Ana bilgisayar yazılımı, işlemsel veri yönetimi için hafif bir veritabanı katmanı olan SQLite içerir. SQLite, iki ana seçeneğin sağlandığı dosya sistemi katmanının üstünde çalışır: Yerinde güncelleme dosya sistemi olan Ext4 [13] ve günlük yapılı bir dosya sistemi olan F2FS [15]. Rastgele dosya yazma işlemleri, flash depolamanın çöp toplama maliyetini beklenmedik şekilde artırabileceğinden, F2FS, rastgele güncellemeleri sıralı, yerinde olmayan güncellemelere dönüştürerek flash depolamadan yararlanır. Yerinde olmayan güncellemeler depolama alanında güncelliğini yitirmiş veriler oluşturduğundan, F2FS'nin gelecekteki sıralı yazmalar için bitişik boş alan sağlamak üzere geçerli verileri zamanında sıkıştırması gerekir.



Şekil 40: (a) Mobil uygulamalar için dosya yazma ve okuma boyutlarının karakterizasyonu (birim: sayfa). (b) Geleneksel sıralı sıkıştırma yaklaşımlarının düşük sıkıştırma etkinliği. Sıkıştırma Amplifikasyon Oranı (CAR) ve Dekompresyon Amplifikasyon Oranı (DAR) sırasıyla sunulmuştur.

5.1.7 Yüksek Yazma Stresi

Uygulamalar, veri bütünlüğü garantisi için büyük ölçüde SQLite günlük tutma mekanizmasına güvenerek muazzam eşzamanlı, rastgele blok yazmaları üretir [10]. Yazma trafiği, G / Ç sisteminin diğer bileşenleri tarafından daha da güçlendirilir. Spesifik olarak, F2FS için boş alan birleştirme birçok ekstra veri geçişini içerir [15] ve mobil depolama içinde flash çöp toplama, bellek silmeden önce veri hareketleri gerektirir [16]. Dosya sistemi doluluk seviyesi yüksek olduğunda, çoklu yazma büyütme seviyeleri daha da kötüleşir. Güçlendirilmiş

yazma trafiği, kullanıcı tarafından algılanan gecikmeyi önemli ölçüde azaltır [17]. Ek olarak, modern flash teknolojisi, düşük dayanıklılık pahasına yüksek bit hücresi yoğunluğuna doğru gelişirken, örneğin, bir TLC flaş bloğu yalnızca yaklaşık 1.000 P / E (program silme) döngüsüne [18] dayanabilir, aşırı yazma trafiği ayrıca depolama ömrü ile ilgili endişeler ortaya çıkarmaktadır.

5.1.8 Dosya Sıkıştırmanın Dezavantajları

Dosya sıkıştırmanın depolama alanından tasarruf etmesi ve G / Ç miktarını azaltması beklenmektedir. Ancak, mobil uygulamaların dosya okuma ve yazma işlemlerinin oldukça rastgele doğası nedeniyle mobil depolama için durum böyle olmayabilir. Bu bölümde, seçilen popüler uygulamalar altında yüksek rasgele okuma ve yazma olduğunu gösteriyoruz ve mevcut dosya sıkıştırma tasarımlarının alan kullanımı ve okuma performansı için zararlı olabileceğini gösteriyoruz. Android cihazlarda, okuma trafiğine ve yazma trafiğine esas olarak sırasıyla yürütülebilir dosyalar ve SQLite dosyaları katkıda bulunur [10, 11]. Şekil 1 (a), * .apk yürütülebilir dosyalarındaki dosya okuma işlemlerinin ve SQLite dosyalarındaki dosya yazma işlemlerinin boyut dağılımını bildirmektedir. Sonuçlar, SQLite dosyalarına yazma işlemlerinin kabaca yarısından fazlasının 4 sayfadan (16KB) büyük olmadığını göstermektedir. Bu küçük yazma işlemlerinin çoğu rasgele dosya uzaklıklarına bağlıydı. Okuma işlemleri için, tüm uygulamalarda tüm okuma işlemlerinin yaklaşık% 90'ı bir sayfadan büyük değildi. Bu okumaların dosya ofsetleri de oldukça parçalanmıştı. Mevcut birçok sıkıştırma dosya sistemi, örneğin Btrfs [9], JFFS2 [19] ve EROFS [20], aynı depolama bloğunda yalnızca ardışık ofsetlerin sıkıştırılmış dosya bloklarının depolanmasına izin verir. Son sıkıştırılmış dosya bloğunun dosya ofsetine devam etmezse, sıkıştırılmış bir dosya bloğu için yeni bir depolama bloğu tahsis edilir. Sıralı sıkıştırma yöntemi olarak adlandırılan bu tasarım, mobil depolamada alan kullanımını ciddi şekilde azaltabilir.

Problemlerin ciddiyetini değerlendirmek için, Şekil 1 (b) 'de sıralı sıkıştırma yöntemiyle bir dizi sıkıştırılmış mantıksal bloğu depolamak için gereken fiziksel blokların 1 toplam sayısının oranı olan Sıkıştırma Yükseltme Oranını (CAR) bildiriyoruz. tüm sıkıştırılmış mantıksal blokları depolamak için gereken minimum fiziksel blok sayısı. CAR, sıralı sıkıştırma yönteminin, dahili parçalama yoluyla rastgele yazmada alan verimliliğini nasıl düşürdüğünü yansıtır. CAR değerleri, sıralı sıkıştırma yönteminin 5 uygulamadan 4'ü için% 40'tan fazla ekstra alan gereksinimi doğurduğunu göstermiştir. Ardından, sıralı sıkıştırma yöntemiyle uygulama başlatmanın güçlendirilmiş bir okuma ek yükünden nasıl zarar gördüğünü

gösteriyoruz. Bir uygulamayı başlatmak için okunan fiziksel bloklarda depolanan toplam sıkıştırılmış mantıksal blok sayısının, uygulamayı başlatmak için gerçekten gerekli olan sıkıştırılmış mantıksal blokların toplam sayısına oranını göstermek için Dekompresyon Yükseltme Oranını (DAR) kullandık. Örneğin, bir fiziksel blok dört sıkıştırılmış mantıksal blok depoluyorsa ve bunlardan yalnızca biri gerçekten kullanılıyorsa DAR 4'tür. Şekil 1 (b), DAR değerlerinin CAR değerlerinden bile daha yüksek olduğunu göstermektedir çünkü tek sayfalık okumalar genel okuma trafiğine hakim olmuştur. CAR ve DAR sonuçları, mevcut sıralı sıkıştırma yönteminin beklenmedik bir şekilde alan kullanımını azalttığını ve mobil cihazlarda uygulama başlatmak için okuma ek yükünü artırdığını gösterdi. Bu, mobil uygulamaların benzersiz rastgele I / O modeliyle başa çıkmak için yeni bir alan yönetimi stratejisine duyulan ihtiyacın altını çiziyor.

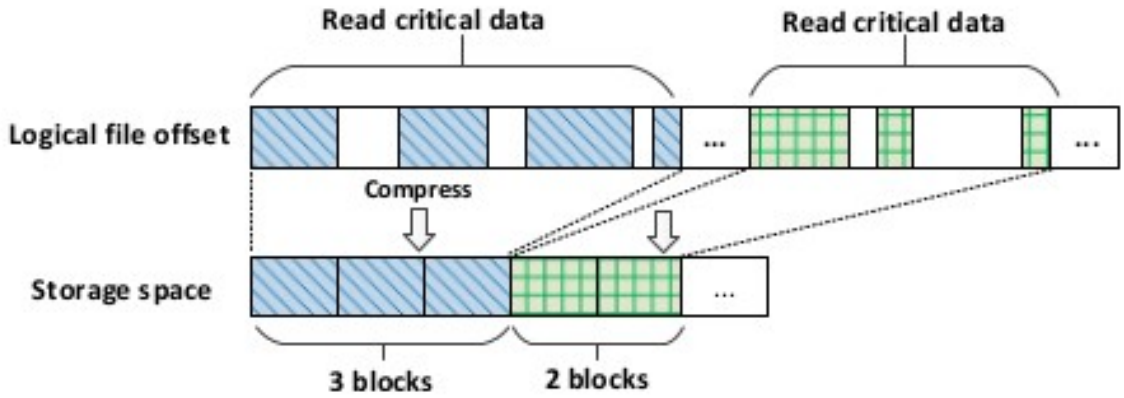
5.1.9 Yürütülebilir Dosyaların Son Derece Rastgele Okumaları

.Apk, .dex, .odex ve .oat dosyaları dahil olmak üzere yürütülebilir dosyalar, büyük bir depolama alanı [14] oranına katkıda bulunur ve yüksek oranda sıkıştırılabilirler. Uygulamanın başlatılması sırasında yürütülebilir okuma gecikmeleri, kullanıcı deneyimi için çok önemlidir, çünkü kullanıcıların gerekli tüm yürütülebilir veriler açılıp belleğe yüklenene kadar beklemesi gerekir. Çalıştırılabilir dosyaların tipik mobil uygulamalarla nasıl okunduğunu inceledik. Gerekli dosya verilerini doğru bir şekilde tanımlamak için, ilgili sistem çağrılarını ayıklamak için Sanal Dosya Sistemine (VFS) rutinler ekledik, ör. Bellek eşlemeli yürütülebilir dosyalar üzerindeki okumaları kaydetmek için do_mmap işlevini kullanmak ve olmayanlarda okumaları ayıklamak için do_generic_file_read işlevini kullanmak. -bellek eşlemeli dosyalar. Başlatma süresi, uygulamanın çalıştırılabilir dosyası bir saniye boyunca herhangi bir okuma almadığında sona erdi. Şekil 4, Facebook, Chrome ve Messenger için ana yürütülebilir dosya olan base.apk dosyası içindeki okuma adreslerinin dağılımlarını göstermektedir. Sonuçlar, incelenen uygulamaların okuma taleplerinin küçük ve rastgele olduğunu göstermektedir. Yürütülebilir dosyalar büyük olsa da, çalıştırılabilir dosya verilerinin yalnızca küçük bir kısmı gerçekten başlatılmak üzere getirildi. Örneğin, Facebook'un çalıştırılabilir dosyası 80,6 MB'tı, ancak Facebook'u başlatmak için dosyanın yalnızca 632 KB'ı okundu. Dosya sayfaları, çalışma zamanı talep sayfalama yoluyla getirildi, ancak adres dağılımı, bu gerekli sayfaların korelasyonlarına göre iyi organize edilmediğini gösteriyor. Apk dosyaları aslında bir kaynak dosyaları paketidir. Bu dosyalar, uygulamanın başlatılması sırasında okuma trafiğinin önemli bir kısmına katkıda bulunur; ör. Facebook, Chrome, Messenger için toplam okuma isteklerinin sırasıyla% 49,% 27 ve% 21'i. Apk dosyalarının rastgele okunmasının, 5. Bölümde

gösterilecek olan uygulama başlatma gecikmeleri üzerinde büyük etkileri oldu. Facebook'un base.apk dosyasını derledik ve başlatma için hangi kaynak dosyalarının gerçekten okunduğunu analiz ettik. Erişilen kaynak dosyaları, apk içindeki kaynak dosyalarının okuma adresleri ve dosya ofsetleri eşleştirilerek tanımlandı. Base.apk, 17.439 kaynak dosyası içeriyordu. Tablo 1, kaynak dosyalarının yalnızca küçük bir alt kümesinin (17.439'dan 110'u) okunduğunu göstermektedir. Bu gerekli kaynak dosyaları (* .xml, * .png, vb.) Apk içinde rastgele konumlara dağıtıldı ve hepsine tek blok okuma istekleriyle erişildi. Bu küçük, rastgele okumalar, blok G / Ç sayısını artırdı ve kullanıcının algılayabileceği uygulama başlatma gecikmesini düşürdü.

Tablo 1: Facebook'un base.apk dosyasından okunan kaynak dosyaları

Tip	.xml	.arsc	.png	.dex	diğerleri	apk metadata
Dosya sayısı	51	1	41	9	8	1
Okunan bloklar	1	32	1	1	1	17

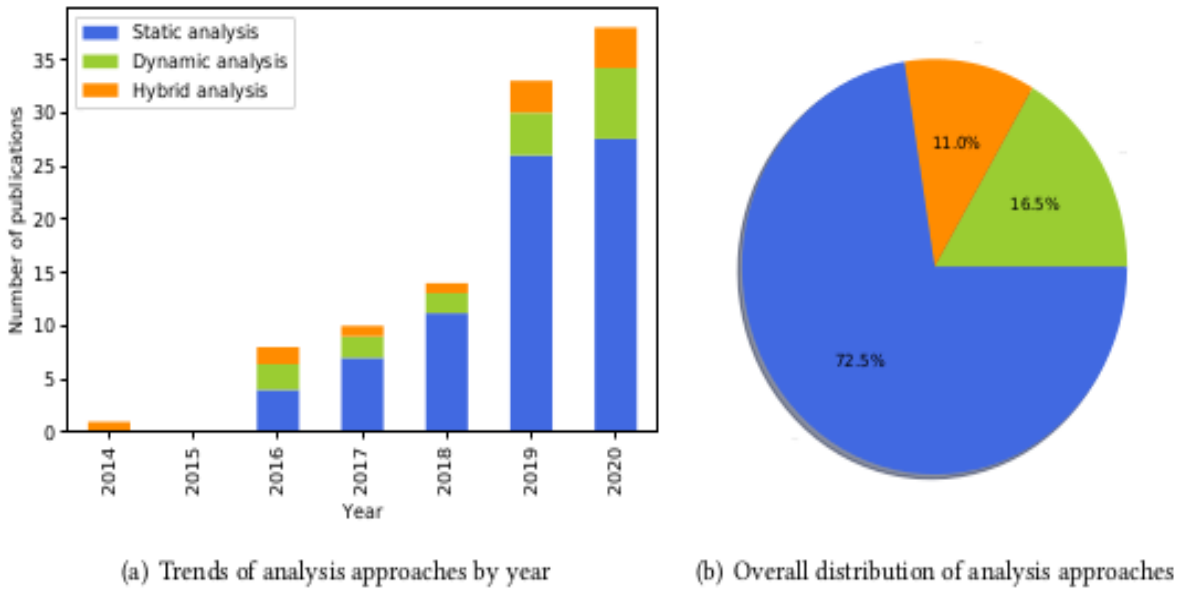


Şekil 41: Kritik okuma verilerinin yürütülebilir dosyalarda sıkıştırılması.

5.1.10 APK Karakterizasyonu

Android APK, ikili yürütülebilir bir dosyadır, bu nedenle, DL modellerine beslemeden önce, Android APK dosyasını veya toplanan verileri DL modelleriyle uyumlu, genellikle vektörize biçimde biçimlendirilmiş bir temsile dönüştürmek gerekir. Araştırma çalışmaları genellikle ilk başta APK'leri işler ve analiz eder, ardından uygulamalardan çıkarılan bilgiler üzerinde daha fazla işlem yapar. Analizin programlanmasına giden yolda, analiz statik analiz, dinamik analiz

ve hibrit analiz olarak kategorize edilebilir. Analizden sonra, işlenen dosya veya toplanan veri bilgileri, DL modellerine beslenmeden önce özellik kodlama yaklaşımları ile biçimlendirilmiş formlara kodlanır. Şekil 5, toplanan çalışmalardaki APK karakterizasyon yaklaşımlarını özetlemektedir ve bu bölüm, incelenen çalışmada kullanılan analiz yöntemlerine ve özellik kodlama yaklaşımlarına genel bir bakış sunmaktadır.



(a) Trends of analysis approaches by year (b) Overall distribution of analysis approaches

Şekil 42: Analiz bakış açılarının özeti

5.2 ANDROID GÜVENLİĞİ

5.2.1 Statik Analiz

Statik analizin hedef dosyası, ikili dosyalar veya demonte dosyalar olabilen sabit değildir. Şekil 2'ye göre, APK, Manifest dosyasına, classes.dex'e, vb. Açılabilir. Açıldıktan sonra, bu dosyalar hala insanlar tarafından okunamayan ve okunabilir bir formatta demonte edilmesi gereken bir ikili formattadır. Daha spesifik olmak gerekirse, ikili Manifest dosyası düz metin XML'e dönüştürülebilirken, classes.dex Java dili veya smali dili gibi okunabilir biçimlere daha da dönüştürülebilir. AndroidManifest.xml meta veriler için tasarlanmıştır, bu nedenle gerekli izinler, donanım bileşenleri, uygulama bileşenleri ve filtrelenmiş amaçlar statik olarak toplanabilir. Class.dex'in ayrıştırılmasıyla, API çağrıları, kullanılan izinler veya ağ adresleri gibi dizgi özellikleri, demonte edilmiş java kodlarından veya smali kodlarından çıkarılabilir. Öte yandan, APK, classes.dex, kaynaklar dosyaları ve smali dahil olmak üzere kaynak

dosyalar kodlar , metin özellikleri veya bayt kodu özellikleri olarak doğrudan girilebilir. Kaynak dosyalardan gelen özellikler dışında, uygulama pazarlarındaki uygulama bilgileri, Android uygulamasına ilişkin meta veriler de özellik olarak kullanılabilir. İncelenen çalışmalarda en yaygın olan statik analiz türüdür. Statik analizden yararlanan 79 ilgili birincil çalışma bulundu. Özellik çıkarma yaklaşımları iki kategoriye ayrılabilir. Birincisi, Android uygulamalarının orijinal bilgilerinin tutulmasını en üst düzeye çıkarmak için doğrudan özellik girişi olarak APK dosyalarının ham kodlarını kullanmaktır. Derin öğrenme teknikleri, ham kodlardan özellik gösterimini öğrenir, ancak bu yöntemle ilgili önemli bir sorun, APK'lerin genellikle büyük bir boyuta sahip olması ve önemli miktarda gürültü içermesi nedeniyle büyük miktarda hesaplama maliyeti gerektirmesidir. Bu nedenle, tersine mühendislik, geleneksel ML tabanlı ve DL- tabanlı Android kötü amaçlı yazılım savunma modelleri. Tersine mühendislik araçları, APK'lardan bilgi çıkarır ve bu çıkarılan ham özellikler, eğitimi modellemek için daha fazla kullanılır. Geleneksel makine öğrenimi tekniklerine dayalı Android kötü amaçlı yazılım analizinde en yaygın yaklaşım, izinler, API çağrıları, Uygulama bileşenleri, donanım özellikleri vb. Gibi manifest veya classes.dex'ten anlamsal özellikler çıkarmaktır. Bu anlamsal özellikler elde edildikten sonra, kötü amaçlı yazılım örneklerini analiz etmek için geleneksel makine öğrenimi algoritmaları uygulanır.

Derin öğrenmeyle ilgili toplanan çalışmalarımızda, ilk olarak APK'lardan ham özellikleri çıkarmak için aynı yolu kullanan çalışmaların yarısından fazlası var. Bununla birlikte, geleneksel ML tabanlı yaklaşımlardan farklı olarak, özellik seçimi, DL tabanlı yaklaşımlarda gereksiz bir adımdır çünkü derin öğrenme algoritmaları, özellik temsilini doğrudan ham özelliklerden öğrenebilir. Özelliklerin formatına ve kategorilerine bağlı olarak statik özellikleri, toplanan çalışmalarda en yaygın olan 7 kategoriye ayırıyoruz. Çıkarılan semantik özelliğin türü çeşitli olduğundan ve çoğu anlambilimsel özellik genellikle toplu çalışmalarda tek başına kullanılmadığından, API çağrıları veya izinleriyle birlikte kullanıldığından, aralarında yalnızca en yaygın iki semantik özelliği dikkate alırız, Aşağıdaki tartışmada API çağrısı ve izni. Her çalışmada çeşitli özellik türleri kullanılabildiğinden, belirli bir statik özellik türü kullanan çalışma sayısının statik analiz kullanılarak toplam makale sayısına oranı hesaplanarak yüzde elde edilir.

İzin: İzinler genellikle Android uygulamalarındaki güvenlikle ilgili activitylerle ilişkilendirilir ve bu nedenle genellikle kötü amaçlı yazılımları analiz etmek için kullanılır. Android Varlık Paketleme aracı (AAPT) kullanılarak Booz ve ark. Android APK dosyalarından izin bilgilerini

çıkardı ve son olarak, çekirdek-Android ve kullanıcı tanımlı izinler dahil olmak üzere 22300 izin ve izin bağlamı elde ettiler. Daha sonra her uygulama için izin bilgilerini kaydetmek için 22300 boyutlu bir ikili gösterge vektörü kullanıldı. Vinayakumar vd., izin dizilerini girdi olarak kabul etti ve ardından anlamsal anlamı izin dizilerinden öğrenmek için sözcük düzeyinde bir dil modeli kullandı. Ek olarak, izinle ilgili diğer birincil çalışmalar, bir Android kötü amaçlı yazılım savunma modeli tasarlamak için genellikle izin ve diğer anlamsal özellikleri birlikte ele alır. Wang vd. örneğin, Androguard ve Android SDK Araçlarının yardımıyla, kullanılan izinler, istenen izinler, filtrelenmiş amaçlar, kısıtlanmış API çağrıları, donanım özellikleri, dizgi özellikleri ve şüpheli API çağrıları dahil olmak üzere yedi anlamsal özellik kategorisi elde etti.

CFG / DFG: API çağrı graflarından farklı olarak, CFG düğümleri program ifadelerini ve DFG düğümleri veri mesajlarını temsil eder. DeepFlow, hassas kaynağı ve havuz yöntemlerini tanımlamak için Android uygulamalarından DFG'yi çıkarmak için FlowDroid'i ve SUSI tekniğini kullanır. Bu sayede hassas veri akışlarının 323 özelliği elde edilir. Xu vd. CFG'leri ve DFG'leri çıkardı ve daha sonra elde edilen grafları sabit boyutlu bir bitişik matris halinde kodladı.

API Call: API call başka bir yaygın anlambilimsel özellik türüdür ve Tablo 5'ten statik analiz kullanılarak toplanan çalışmalarda en büyük yüzdeye sahiptir. Hou et al. API çağrılarını uygulamaların küçük kodlarından çıkardı ve aynı yöntemlere ait çıkarılan API çağrılarını tek bir özellik olarak bir bloğa atadı. Nix vd. ve MalDozer, her uygulama için API çağrı dizilerinin ham dizilerini elde etmek için smali kodlar üzerinde bir psuedo-dinamik kod analizi gerçekleştirirken, Pektacs ve ark. ve EveDroid, her uygulama için API çağrıları bilgilerini temsil edecek API çağrı grafını oluşturmak için FlowDroid çerçevesini uygular.

Bununla birlikte, bu yaklaşımlar, her API çağrısını bir tanımlayıcıya eşlemek için önceden tanımlanmış bir API çağrısı sözlüğüne dayalı olarak belirli API çağrılarını çıkarır ve bu nedenle, bu API çağrılarının zaman içinde Android OS ve Android kötü amaçlı yazılımlarının geliştirilmesi. Yeni API çağrılarını yakalamak ve modası geçmiş özelliklerin neden olduğu model eskimesini önlemek için API çağrılarını işlemek için benzerlik temelli yaklaşımlar kullanılır. Zhang vd. , eğitilen modelin evrimleşmiş kötü amaçlı yazılım örneklerini sınıflandırmaya devam edebilmesi için API'ler arasındaki anlamsal benzerliği yakalamak için ilişki graflarına dayalı APIGRAPH'ı önerdi. Benzer şekilde EveDroid, API çağrılarından daha

yüksek düzeyde anlambilim yakalar ve API çağrılarını kümeleme yoluyla çeşitli olay gruplarına böler ve ardından her gruptaki davranışları temsil etmek için işlev kümeleri benimsenir.

Opcode: Classes.dex, Android platformu için Google tarafından Dalvik sanal makine tasarımında gerçekleştirilebilen program işlem kodlarını içerir. Dalvik işlem kodları ayrıca okunabilir sözde kod, smali formata dönüştürülebilir. En yaygın olarak ayıklanan yaklaşımlardan biri, doğrudan classes.dex'te onaltılık formatta bayt kodu kullanmaktır. Öte yandan, küçük kodlardaki işlem kodlarının anlamsal bilgileri de özellik olarak kullanılabilir. Dalvik bayt kodları bir uygulama hakkında ayrıntılı bilgi sağlasa da, analizin karmaşıklığı önemli ölçüde artmaktadır. Bu nedenle, Dalvik işlem kodu basitleştirme, işlenenleri kaldırma, çağrı grafı oluşturma gibi ileri bir işleme adımı olarak yaygın şekilde kullanılmaktadır.

Java kodu: Classes.dex, daha yüksek seviyeli bir biçime, .java dosyalarına dönüştürülebilir. Java kodları işlenecek metin bilgisi olarak görülebilir, bu nedenle NLP'deki kodlayıcılar java kodlarından özellik bilgisi çıkarmak için kullanılabilir. Bunun dışında Zhang ve ark., java kaynak kodlarında bir benzerlik analizi gerçekleştirmek için kelime torbası tabanlı bir kod klon yaklaşımı olan SourcererCC uyguladı ve ardından bir benzerlik özelliği vektörü oluşturdu.

İkili kod: Parçalanmış kodların yanı sıra, APK dosyasındaki ham veriler, ikili kod da girilen özellikler olarak kullanılabilir ve fazlalık API enjeksiyonları gibi gizleme tekniklerine umut vaat eden bir çözümdür. Araştırmacılar, tüm ikili kodu veya ham APK dosyalarının belirli baytlarını 8 bitlik işaretsiz tam sayılardan oluşan bir vektöre dönüştürür.

Uygulama meta verileri: Feng ve ark. ve Feichtner ve ark., izin kullanımını tanımlamak için özellikler olarak uygulama açıklamalarını kullandı. Bir uygulamanın simgesi de kötü amaçlı yazılım varyantlarını algılama ve kötü amaçlı yazılım algılama için özellikler olarak işlendi. Statik analizle ilgili birincil araştırma çalışmalarının yarısı, bu anlamsal özellikleri girdi olarak alır ve analiz için genellikle izinlerin ve API'lerin en yaygın olduğu çeşitli semantik özellik türleriyle birleştirilir. Bu tür özelliklerin benzer bir özelliği, insanlar tarafından profesyonel bir tanıma sahip olmaları ve Manifest dosyalarından ve classes.dex'ten çıkarıldıktan sonra, normalde insan müdahalesi özellik seçiminin gerekli olmasıdır. Aksine, öznitelikler olarak kaynak kodunu kullanmak, daha az ön bilgi ile öznitelik seçimi sürecini

atlayabilir, ancak ham kodları sayısal vektörlere dönüştürmek için uygun bir yaklaşımın nasıl tasarlanacağı yeni bir zorluktur.

5.2.2 Dinamik Analiz

Uygulamaları yürütmeden statik analizin aksine, dinamik analiz, çalışma zamanı davranışlarını ve android uygulamaların sistem ölçümlerini incelemek için örnekler çalıştırarak çalışır. Özellikle kötü amaçlı yazılım analistleri, bir Android emülatörü veya gerçek bir mobil cihaz gibi kontrollü bir ortamda bir Uygulama örneğini yürütür. Bir Uygulamayı Android Sanal Cihaz (AVD) veya Genymotion gibi bir emülatörde çalıştırmak, araştırmacıların kişisel bilgisayarlarında veya sunucularında doğrudan tamamlanabildiğinden uygulamaları dinamik olarak analiz etmek için oldukça yaygın bir yaklaşımdır. Dinamik analizle ilgili toplanan 18 çalışmada, 18 çalışmadan 11'i uygulamanın dinamik davranışını izlemek için emülatör kullanmaktadır. Bununla birlikte, dinamik analiz, kötü niyetli gizleme stratejileriyle karşı karşıya kaldığında statik analizden daha güçlü ve etkili olsa da, gelişmiş Android kötü amaçlı yazılımlarındaki kötü amaçlı activityleri gizlemeye yönelik tespitten kaçınma teknikleri akıllıdır ve anti-emülatör teknikleri dahil olmak üzere sürekli geliştirilmektedir ve çalışmalar Alzaylaee tarafından yapılmıştır. et al. kararlılık ve algılama yeteneği açısından cihaz üzerindeki dinamik analizin simülatör dinamik analizinden çok daha iyi performans gösterdiğini ortaya koymuştur.

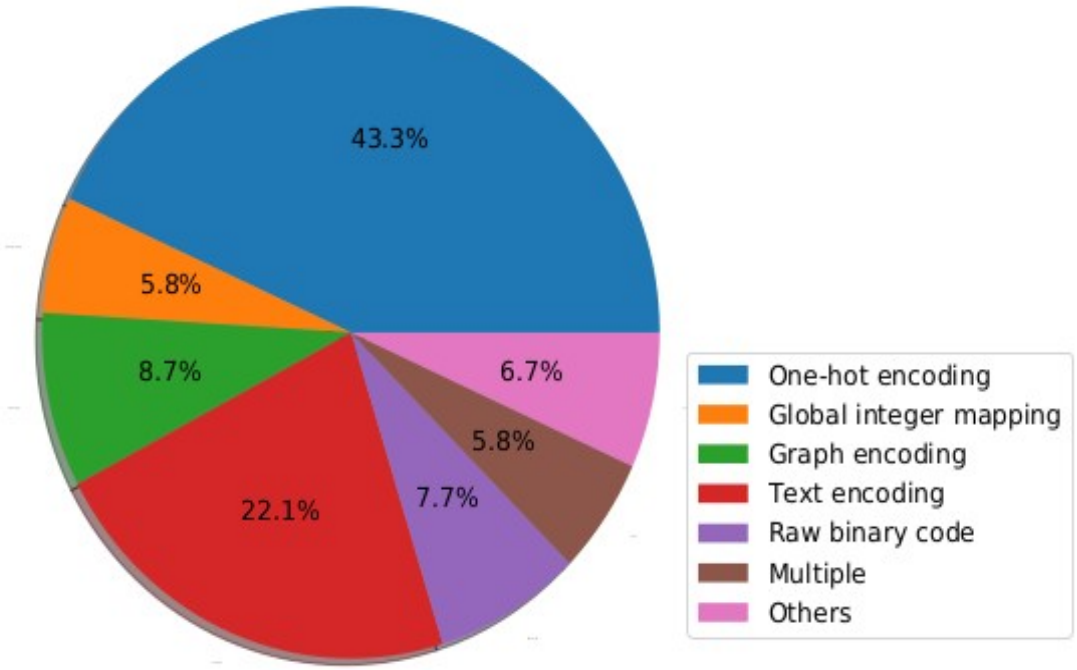
Martinelli vd., Android 6.0 (Marshmallow) ile bir Google Nexus 5X üzerinde 7100 gerçek dünya Android uygulaması için dinamik analiz gerçekleştirdi ve bundan daha fazlası, Alzaylaee ve ark., ölçüm hatasını azaltmak için aynı konfigürasyona sahip sekiz farklı cep telefonu markası üzerinde dinamik analiz gerçekleştirdi. Kontrollü bir Android ortamına yüklendikten sonra, her Uygulamanın belirli simüle edilmiş insan eylemlerine tepki vermesine izin verilir ve bir uygulamanın izleri daha fazla analiz için kaydedilecektir. Özellikle, analistler; tıklamalar, dokunmalar veya hareketler gibi bir dizi Kullanıcı Arayüzü (UI) etkileşimi ve bir dizi sistem düzeyi olay oluşturmak için girdi oluşturma araçlarını kullanır. Kötü niyetli davranışları tetiklemek için yeterli kod kapsamını sağlamak için, durum bilgisi olmayan (rastgele tabanlı), durum bilgisi olan ve durum bilgisi olmayan ile durum bilgisini birleştiren hibrit tabanlı olmak üzere üç tür test girişi oluşturma yöntemi vardır. Monkey, bir tohum tarafından kontrol edilen sözde rastgele olay akışları üreten en yaygın rastgele tabanlı araçtır. ve 8 incelenmiş çalışmada Monkey aracı kullanılmıştır. Durum bilgisi olmayan giriş oluşturma araçlarından farklı olarak, durum bilgisi olan yöntem, bir uygulamanın mevcut

durumunun değerlendirilmesine dayalı olarak yürütülecek olayı seçer ve Alzaylaee ve diğerleri., yaptıkları çalışmada girdi oluşturma araçlarının istikrarlı ve sağlam performansını kanıtladı.

Dinamik analiz için, çıkarılan dinamik özellikler de çeşitlidir. İncelenen çalışmalarda, birçok çalışmanın dinamik analiz nesneleri olarak Linux çekirdek sistemi çağrılarını seçtiğini gördük. Android, Linux tabanlı bir mobil işletim sistemidir ve sistem çağrısı, Linux çekirdeği tarafından sağlanan arayüzdür. API çağrılarının aksine, Linux çekirdek sistemi çağrıları Android işletim sisteminden sürümden bağımsızdır ve bu nedenle kötü amaçlı yazılımdan kaçınma stratejilerine karşı daha dayanıklıdır. Xiao vd. ve Martinelli ve ark., kötü amaçlı uygulamaları tespit etmek için sistem çağrı dizilerinden yararlanırken, Lorenzo ve ark., kötü amaçlı davranışları daha iyi yerelleştirmek için Android kötü amaçlı yazılımlarının sistem çağrılarını görselleştiren bir araç olan VizMal'ı öneriyor.

Bunun dışında, dinamik analiz ile çıkarılan API çağrılarına dayanan 4 ilgili çalışma bulduk ve bunun bir örneği D'Angelo ve diğerleri tarafından yapılan çalışmadır. burada yazarlar, otomatik, hepsi bir arada bir mobil uygulama olan Mobile Security Framework kullanarak API çağrılarını dinamik olarak çıkarır. Ye ve ark. Tarafından uygulanan bir başka önemli çalışma, gerçek zamanlı bir Android kötü amaçlı yazılım tespiti olan AiDroid'i önerdi. İlk olarak, uygulamaların dinamik davranışına ve API çağrıları ile Uygulama arasındaki ilişkiler ve bir Uygulama ile geliştiriciler arasındaki ilişkiler gibi zengin ilişkilere dayalı olarak heterojen graflar oluştururlar. Ve sonra, ilk olarak örnek içi düğüm yerleştirmelerini elde etmek ve ardından HG yerleştirmelerini yeniden çalıştırmadan / ayarlamadan örnek dışı düğümlerin temsillerini öğrenmek için HG-Learning yöntemini önerirler. Diğer dinamik analiz nesneleri arasında ağ trafiği, kaynak tüketimi (CPU, Bellek, vb.), Amaç bileşenli iletişim amacı ve dosya işlemi ve SMS gibi diğer dinamik davranışlar yer alır. Wang vd. , örneğin, URL dizesini HTTP trafiğinden çıkarın ve URL vektörünü çok görüntülü bir sinir ağına besleyin.

5.2.3 Hibrit Analiz



Şekil 43: JavaScript tarafı, bir JavaScript threadi ile çalışır ve ana thread üzerinde çalışan native tarafla JavaScript köprüsü üzerinden eşzamansız olarak iletişim kurar.

Statik analizi ve dinamik analizi birleştiren ilgili yönteme hibrit analiz adı verilir ve toplu çalışmada 12 ilgili çalışma bulunur. Karma analizde kullanılan özellikler, statik ve dinamik analizde bildirilenlere benzerdir, çünkü statik analizle API çağrıları ve izinleri ve dinamik analizle sistem çağrıları, örneğin, Refs 'de en yaygın olarak kullanılanlardır. . Bunun dışında, iki tür özelliğin kombinasyonu çeşitlidir. Bu, Lu et al. kaynak dosyası bilgilerini, API çağrılarını, izinleri, Meta bilgilerini ve altı tür dinamik davranışı birleştirir.

a) Feature Encoding

Program analizinden sonra, çıkarılan bilgilerin derin öğrenme modellerine beslenmeden önce özellik vektörlerine kodlanması gerekir. İncelenen çalışmalarda, Android uygulamalarından çıkarılan bilgilerin biçimi çeşitli olduğundan özellik kodlamanın birçok yolu vardır. Bu nedenle, özellik kodlama yaklaşımlarını aşağıdaki yedi türe sınıflandırıyoruz:

b) One Hot Encoding

Bir tek sıcak vektör, 1 x N'lik bir matristir (vektör) ve sütunlar, uygulamalar için her bir olası özellik değerinin varlığını belirtmek için kullanılır. İncelenen literatürde, araştırmacılar

genellikle her bir anlamsal özelliğin varlığının bilgisini depolamak için çıkarılan anlamsal özellikleri ikili değişkenler olarak görürler. Dolayısıyla, bir APK x n = {a 1, a 2, olarak temsil edilebilir. . . , a n}, burada a n, n'inci özelliğin gösterge değeridir. Çıkarılan semantik özelliklerin miktarı yeterince büyüktür ve bu nedenle, bu birincil çalışmalardaki özellik listelerini tanımlamak için genellikle insan müdahalesi tanıtılmaktadır. Örneğin, DroidDetector, hibrit analiz yoluyla önceki deneyime dayalı olarak her uygulama için toplam 192 özelliği dikkate alır ve uygulamada bir özellik ortaya çıkarsa, özellik değerinin atandığı her uygulama için 192 boyutlu bir vektör oluşturulur. 1; aksi takdirde, özellik değeri 0'dır. Her semantik özelliğin varlığını belirtmek için kategorik değişkenlerle uğraşmanın yanı sıra, tek sıcak kodlama, opcode dizileri gibi gözden geçirilmiş çalışmalarda sıralı özellikleri kodlamak için yaygın olarak kullanılır ve API çağrı dizileri. Basitliği nedeniyle tek sıcak kodlama en yaygın strateji olmasına rağmen, yüksek boyutlu oluşturma ve sözcükler arasına tek başına yerleştirme dahil olmak üzere iki büyük dezavantaj vardır.

c) Global Integer Mapping

Gözden geçirilen çalışmalarda, her bir özelliğe ayırt edici bir tamsayı atayarak global ara eşleme, çıkarılan kategorik anlamsal özellikleri işlemek için başka bir yaklaşımdır. Oak vd., örneğin, her dinamik aktivitenin ayrı bir sayı olarak temsil edildiği 174 dinamik aktiviteden oluşan bir arama tablosu oluşturur ve bu şekilde, bir vektördeki tüm dinamik aktiviteleri temsil etmek için bir tamsayı dizisi kullanılır. Bunun dışında Zegzhda ve ark. API çağrısını bir RGB piksel değerine eşleyin ve Uygulamaları bir görüntü olarak kodlayın.

d) Graph Encoding

Çeşitli araştırma çalışmaları CFG ve DFG gibi uygulamaların özellik graflarını oluşturur, çıkarılan grafları bir vektöre veya bir vektör setine dönüştürmek için graf kodlaması gerekir. Deep4MalDroid, ağırlıklı yönlendirilmiş bir graf oluşturmak için dinamik analiz araçlarıyla sistem çağrılarını elde eder ve her bir kenarın ağırlıkları ve her bir düğümün derece ve dış derecesi dahil olmak üzere graf yapı bilgisi, girişler olarak vektörlerde depolanır. Xuet al., ham kodlardan çıkarılan CFG ve DFG'yi sırasıyla bitişiklik metriklerine kodlar ve CFG ve DFG'yi gömme katmanlarında tek bir metrikte birleştirir.

e) Text Encoding

Doğal dil işleme alanından özellik temsil yaklaşımları kullanmak oldukça yaygındır, çünkü derlemenin ayrıştırılmasıyla elde edilen özellikler anlamsal bilgi içerir. Ham Android kodlarından anlamsal anlam elde etmek için, özellik temsilini öğrenmek için birçok son

teknoloji kelime düzeyinde kodlama yaklaşımı getirilmiştir. Aslında, tek sıcak kodlama, metin kodlamanın en basit yöntemidir, ancak dezavantajlardan biri, daha önce tartıştığımız yüksek boyutlu problemdir. Ek olarak, bazı araştırmacılar kelimeleri seçmek için Bag of Words (BOW), Terim Frekansı-Ters Belge Frekansı (TF-IDF) ve N-Gram gibi ayrı temsil yaklaşımları kullanır, ancak bu yöntemler hala veri seyrekliği ve yüksek boyutluluk sorunundan muzdariptir. Bu nedenle, birçok birincil çalışma, Android APK'larının çıkarılan semantik özelliklerinden bağlamsal olmayan gösterimi öğrenmek için Continuous Word2vec ve GloVe gibi önceden eğitilmiş kelime yerleştirme modellerinin etkinliğini daha da araştırmaktadır. Kelimeleri yoğun vektörler olarak temsil etmenin dışında, paragraf, cümle veya belgeye dayalı bazı gömme yaklaşımları vardır ve örneğin, üç temel çalışma Doc2vec belgeyi düzeyinde yerleştirmeyi öğrenir.

f) Raw Code Encoding

APK dosyalarından anlamsal bilgi çıkarmak için program analiz araçlarını gerektiren eski kodlama yaklaşımlarından farklı olarak, ham kod dönüştürme, ham ikili kodları veya bayt kodunu doğrudan bir vektöre veya bir vektör setine dönüştürür. Daha ayrıntılı olarak, APK, Android OS'nin paket dosyası biçimidir ve ikili kod biçiminde depolanan bir programın tüm kodlarını içerir. Birincil çalışmaların çoğu genellikle APK dosyalarını yeniden derleyerek elde edilen anlamsal bilgilere dayanan savunma modelleri önerir, ancak bu, gizleme gibi bazı ciddi sorunları beraberinde getirebilir. Bu nedenle, ikili kodlar veya sınıflar.dex bayt kodu gibi ham kodları kodlamak umut verici bir yaklaşımdır. IMCFN belirli bir Android ikilisini 8 bitlik işaretsiz tamsayıların bir vektörü olarak okur ve ardından vektörü 2 boyutlu bir diziye dönüştürür. RGB renk haritasına dayalı olarak Android ikili, renkli bir görüntü olarak görselleştirilir. Birçok araştırma çalışması, ham kodları doğrudan girdi olarak kullanmak için benzer yaklaşımlar kullanır.

g) Others

Özellik kodlama türü çeşitlidir ve yukarıda tartışılan yöntemlerden farklı özellik kodlama yaklaşımlarını kullanan bazı araştırma çalışmaları vardır. Farklı anlamsal özellikler arasındaki bağımlılık bazı çalışmalar tarafından ele alınmıştır. Xiao vd., bitişik sistem çağrıları arasında belirli korelasyonları almak için sistem çağrı dizilerinden Markov zincirlerine dayalı bir kötü amaçlı yazılım tespit modeli önermektedir. D'Angelo ve diğerleri. ayrıca iki API çağrısı arasındaki bağımlılığı da göz önünde bulundurur, ancak aralarındaki fark, API çağrı çiftlerini doğrudan iki boyutlu bir dizide eşleştirmeleridir. Öte yandan, ham kodları işlemek için diğer

kodlama yaklaşımlarını kullanan bazı çalışmalar da vardır. Yuan vd. kötü amaçlı yazılım ikili kodlarını bayt aktarım olasılığı matrislerine göre Markov görüntülerine dönüştürür ve ardından kötü amaçlı yazılım Markov görüntülerini ayırt etmek için sınıflandırıcılar kullanır.

h) Multiple

Daha zengin özellikleri işlemek için farklı özellik kodlama yaklaşımlarını birleştirmek de toplanan çalışmalarda yaygındır. Kim vd. çeşitli özellik türlerine göre özellik gösterimini elde eder. Yazarlar, izin, dize ve uygulama bileşenleri gibi kategorik özelliklerin varlığını kaydetmek için tek sıcak vektörler oluşturur. Aynı zamanda, şaşırtma tekniklerinin etkilerini hafifletmek için, özellik temsilini öğrenmek için benzerliğe dayalı bir özellik vektörü oluşturma süreci tanıtılmıştır ve bu özelliklerin örnekleri, yöntem işlem kodu, yöntem API'si ve paylaşılan kitaplık işlevidir. Bu çalışmalar aynı zamanda Android uygulamalarının simgelerini veya resimlerini de dikkate aldığından, özellik temsillerini öğrenmek için hem görüntü gömme yaklaşımları hem de metin gömme algoritmaları kullanılmaktadır.

5.2.4 Kötü Amaçlı Uygulamalar(Malware)

Kötü amaçlı yazılım (kötü amaçlı yazılım), kötü niyetli bir yazılım türüdür ve saldırganlar, dijital cihazlarınızı kullanıcıların bilgisi olmadan zararlı işlemleri tamamlamak için kullanır. Ayrıca, tüm kötü niyetli saldırıları önlemek için kesinlikle güvenli bir işletim sistemi yoktur ve cep telefonları, Kişisel Bilgisayarlar (PC'ler) veya Nesnelerin İnterneti (IoT'ler) gibi her türlü dijital cihaz kötü amaçlı yazılım bulaşmaları için bir hedeftir. PC tabanlı kötü amaçlı yazılımlar, diğer dijital cihazlara kıyasla çok daha uzun bir geçmişe sahiptir. Nesnelerin İnternetinin son yıllarda hızla gelişmesiyle birlikte, IoT'lerdeki kötü niyetli saldırılar da büyüyen bir tehdit haline geliyor. Çeşitli platformlar arasında işletim sistemleri ve uygulamalarda büyük farklılıklar olsa da, bunların sunduğu kötü niyetli davranışlar doğaları gereği benzerdir. Bu bölüm kötü amaçlı yazılım kategorilerini, özelliklerini ve diğer ilgili tanımları açıklamaktadır.

5.2.5 Kötü Amaçlı Yazılım Kategorileri

Kötü amaçlı yazılımları niyetlerine ve davranışlarına göre sınıflandırabiliriz. Popüler mobil kötü amaçlı yazılım türleri aşağıda sunulmuştur:

• Fidye yazılımı: Fidye yazılımı, kurbanın fidye ödeyene kadar özel anahtar şifrelemesini kullanarak verilere erişmesini önlemek için kullanıcı cihazını kilitleyen bir kötü amaçlı yazılım türüdür.

• Casus yazılım: Kullanıcı kişisel bilgilerini ve activitylerini gizlice izleyen ve ardından toplanan bilgileri kullanıcının haberi olmadan uzak sunucuya gönderen bir kötü amaçlı yazılım türüdür.
• Botnet: Saldırganlar bir yazılım programı oluşturur, bot oluşturur ve komuta ve kontrol kanallarını kullanarak kurbanın cihazını uzaktan kontrol eder.
• Truva Atı: İyi huylu bir uygulama gibi görünen ancak aslında zararlı eylemler gerçekleştiren bir tür kötü amaçlı yazılım.
• Adware: Adware kullanıcılara istenmeyen reklamlar sunar.
• Arka Kapı: Mağdur cihazlarda bir arka kapı sağlayarak diğer kötü amaçlı yazılımlara zemin hazırlayan bir kötü amaçlı yazılım türüdür.
• Solucan: Solucan bir kod parçasıdır ve çoğaltılabilir ve ağ aracılığıyla diğer cihazlara yayılabilir.

5.2.6 Kötü Amaçlı Yazılım Teknikleri

Mobil kötü amaçlı yazılım saldırılarının yakın bir tehdit olmasının ana nedeni, kötü amaçlı yazılım türlerinin ve kötü amaçlı yazılım istilacı tekniklerin sürekli olarak evrim geçirmesi ve ilerlemesidir. Aşağıda, saldırganlar tarafından yaygın olarak kullanılan, dikkate alınan kötü amaçlı yazılım tekniklerini açıklıyor ve özetliyoruz:
• Yeniden paketleme: Kötü amaçlı yazılım yazarları, yeni eklenen kötü amaçlı yazılım yüküyle popüler Uygulamaları parçalara ayırıp yeniden birleştirir ve ardından bu virüslü uygulamaları, kullanıcıları indirmeye ve yüklemeye ikna etmek için üçüncü taraf pazarlara yükler.
• Drive-by Download: Kurbanlar kötü amaçlı sayfalara çekilir ve kötü amaçlı yazılım indirmeleri için kandırılır.
• Dinamik Yük: Bu teknik, uygulamalar eklenen güncelleme bileşenlerinin yardımıyla kötü amaçlı yükü otomatik olarak indirdiğinden, yükü bir kerede ekleyen yeniden paketlemeden farklıdır.
• Gizleme teknikleri: Bu tür teknikler, kötü amaçlı kodu, tespit edilmekten kaçınmak için anlaşılması zor bir forma şifreler.

5.2.7 Android Kötü Amaçlı Yazılım Analizi

Özellik yapısının türüne bağlı olarak, kötü amaçlı yazılım analizi, statik ve dinamik analizi birleştiren statik analiz, dinamik analiz ve hibrit analiz olarak kategorize edilebilir. Statik analiz, uygulamayı çalıştırmadan uygulama kodunu ve Androguard ve Flowdroid gibi son teknoloji statik analiz yaklaşımlarını inceler. Statik analizden farklı olarak dinamik analiz,

simülatörler veya gerçek cihazlarda çalışırken bir uygulamanın özelliklerini toplamak için kullanılan bir tekniktir. Bununla birlikte, bu iki aracın avantajları ve dezavantajları vardır. Statik analiz tüm kodu kapsayabilirken dinamik analiz için kodun bölümleri yalnızca çalıştırılabilir. Bunun dışında, gerekli özellikleri elde etmek için sınırlı bilgi işlem kaynaklarına sahip bir platformda bir uygulama çalıştırmak, kodu statik olarak analiz etmeye kıyasla oldukça zaman alıcıdır. Ancak elbette dinamik analiz, dinamik yükleme ve kod gizleme gibi birçok statik analiz probleminin üstesinden gelir. Karma analiz, hem statik hem de dinamik analizin avantajlarından yararlanmak için tasarlanmıştır, ancak yoğun hesaplama gerektirmeye devam etmektedir.

Analiz işlemi sırasında, bu analiz yöntemleriyle bir dizi özellik çıkarılacaktır. En popüler analiz yaklaşımı olarak, statik analiz ile oluşturulan özellikler, manifesto temelli özellikler, koda dayalı özellikler ve meta verilere dayalı özellikler olarak kategorize edilebilir. Manifest dosyası, her Android uygulaması için temel bilgileri depolar ve bu nedenle her uygulamanın davranışları, bildirim tabanlı özelliklerle iyi bir şekilde tanımlanır. Tipik bildirim tabanlı özellikler aşağıda sunulmuştur:

• Permissions: Android, bir işlemin gerçekleştirebileceği işlemleri kısıtlamak için permission tabanlı bir erişim mekanizması sağlar. Uygulamalar yüklenmeden önce, kullanıcıların, saldırganların önündeki ilk engel gibi bazı güvenlikle ilgili davranışların mülkü olarak izin ayrıcalığı vermeleri gerekir. Bir uygulamanın gerekli permissionları, bildirim dosyasında bildirilir ve fiilen kullanılan permissionlar, classes.dex'ten elde edilebilir.

• App Component: Sistem veya kullanıcıların uygulamaya erişmesi için bir uygulamanın temel yapı taşları olan Android, farklı amaçlara sahip dört tür bileşenden oluşur: activityler, hizmetler, içerik sağlayıcılar ve yayın alıcıları

• Filtered Intents: Amaç, bir uygulamanın her bileşeninin mesajları veya verileri kaydetmesine ve teslim etmesine yardımcı olabilir ve farklı uygulamalar arasında bilgi paylaşımı da amaçlarla gerçekleştirilir.

•Hardware Feature: Uygulamanın gerekli donanım bileşenlerini bildirim dosyasında bildirmesi gerekir. GPS, ağ modülleri veya kameralar gibi bu donanım bileşenlerine erişim genellikle kullanıcının gizliliği üzerindeki zararlı faaliyetlerle ilgilidir.

Adından da anlaşılacağı gibi, kod tabanlı özellikler, sınıflar.dex'in demonte edilmiş dosyalarından toplanır ve ana kod tabanlı özellik türleri şunlardır:

• API Call: Uygulama Programlama Arayüzü (API) çağrıları, uygulama oluşturmak için bir bilgi işlem arayüzü olarak hizmet ettikleri için Android kötü amaçlı yazılım analizindeki en popüler statik özelliklerden biridir. Bir API normalde bir uygulamanın bir tür işlevini belirtir. API çağrıları, demonte edilen koddan çıkarılabilir.

• String Feature: Demonte dosyalardaki dize değerleri, dize özellikleri olarak çıkarılabilir. En normal dize, IP adresleri, ana bilgisayar adları ve demonte koddan çıkarılan URL'ler gibi ağ adresleridir.

• Control Flow Graph: Kontrol akış grafiği (CFG), uygulamanın kaynak kodunu tüm yürütme yollarını içeren yönlendirilmiş bir grafik olarak temsil eder.

•Data Flow Graph: Veri akış grafiği (DFG) de yönlendirilmiş bir grafiktir ancak uygulamaların işlemleri arasındaki veri bağımlılıklarını açıklar.

Bir Uygulamanın açıklamasından veya diğer bilgilerinden çıkarılan özellikler, meta veriye dayalı özellikler olarak görülür. Bilgiler genellikle kategori, açıklama, izin, derecelendirme, geliştirici bilgileri vb. Gibi doğrudan uygulama pazarlarında gösterilir. Öte yandan, dinamik analizle çıkarılan özellikler de aynı şekilde çeşitlidir. Android örneklerini çalıştırırken, kullanılan API çağrıları, kullanılan izinler, sistem çağrıları veya internet erişimi ve kaynak tüketimi gibi dinamik davranışlar, Android uygulamalarını analiz etmek için kullanılabilir.

5.2.8 Cold Boot Saldırısı ile RAM'den Master Keyi Elde Edip Dosya Bazlı Şifrelemeyi Kırma

On yıldan beri bilindiği gibi, bir saldırgan Android akıllı telefonlar da dahil olmak üzere açık bir cihaza fiziksel erişime sahip olduğunda, cold boot saldırıları yazılım tabanlı disk şifrelemesini bozabilir. Ham bellek görüntüleri, bir cihazı sıfırlayarak ve kötü niyetli bir önyükleyici ile yeniden başlatarak, güvenli önyükleme veya kısıtlayıcı BIOS ayarları nedeniyle bunun mümkün olmadığı veya RAM modüllerinin fiziksel olarak kontrolü altındaki bir sisteme nakledilmesiyle elde edilebilir. saldırgan. Bir aygıtın bellek görüntülerine dayalı olarak, geçmişte Tam Disk Şifrelemesini (FDE) kırmak için BitLocker, dm-crypt ve ayrıca Android'in FDE'si dahil olmak üzere farklı anahtar kurtarma algoritmaları önerildi. Google'ın yeni Android cihazlar için standart şifreleme yöntemi olarak FDE'den Dosya Tabanlı Şifrelemeye (FBE) geçmesiyle birlikte, mevcut araçlar etkisiz hale getirildi. Bu açığı kapatmak ve ham bellek görüntüsü verilen şifreli Android disklerin adli analizini yeniden etkinleştirmek için FBE için özel olarak hazırlanmış yeni bir anahtar kurtarma yöntemi sunuyoruz. Ayrıca, FBE etkin EXT4 görüntüleri üzerinde çalışırken dosya adlarının ve dosya içeriklerinin şifresini otomatik olarak çözmek için Sleuth Kit'i (TSK) ve ayrıca şifreli EXT4 bölümlerinden olayları çıkarmak için Plaso çerçevesini genişletiyoruz. Son olarak, ana

anahtarların FBE bölümlerinden kurtarılmasının, Google'ın anahtar türetme yöntemindeki bir kusur nedeniyle özellikle kolay olduğunu savunuyoruz.

Açık ancak kilitli bir akıllı telefon kaybolduğunda, çalındığında veya ele geçirildiğinde, ona fiziksel erişim sağlama yeteneklerinden yararlanan saldırılarla karşı karşıya kalır. 2008 yılında Halderman ve ark. (2008), soğuk önyükleme saldırısı olarak adlandırılan BitLocker, TrueCrypt ve FileVault dahil olmak üzere FDE'yi kırmak için bir yöntem sundu. Bu saldırı, bellek modüllerinin güç kesildikten sonra kısa bir süre için içeriklerini koruduğunu söyleyen DRAM'ın kalıcılık etkisine dayanmaktadır. Bu süre, RAM modülleri soğutma spreyleri ile veya cihazı bir dondurucuya koyarak soğutulursa, bir saniyeden daha kısa süreden birkaç dakikaya kadar uzatılabilir (Müller ve Spreitzenbarth, 2013). Soğutmadan sonra, iki farklı türde soğuk önyükleme saldırısı uygulanabilir: Hedef makine sıfırlanır ve RAM'den şifreleme anahtarlarını kurtarmak için bir adli önyükleyici ile başlatılır. Bu durumda, güç sadece kısa bir süreliğine kesilir ve doğru kurtarılan bitlerin oranı yüksektir. Veya hedef makinenin güvenli önyükleme veya BIOS ayarları gibi önyükleme kısıtlamaları varsa, RAM modüllerinin saldırganın kontrolünde hızlı bir şekilde bir kurtarma makinesine nakledilmesi gerekir. İkinci durumda, güç birkaç saniyeliğine kesilir ve başarıyla kurtarılan bitlerin hızı, DRAM'in diğer fiziksel özelliklerinin yanı sıra bellek modüllerinin sıcaklığına bağlıdır.

Halderman ve arkadaşlarının çalışmasının ardından, soğuk önyükleme saldırılarının yaygın masaüstü bilgisayarlara (Gruhn ve Müller, 2013) ve Android tabanlı akıllı telefonlara (Müller ve Spreitzenbarth, 2013) karşı uygulanabilirliğini kanıtlayan daha fazla akademik çalışma yayınlandı. DDR3 bellek karıştırmanın tanıtılmasıyla birlikte, bellek karıştırma, RAM'i "şifrelemek" için LFSR tabanlı bir algoritma kullandığından, soğuk önyükleme saldırılarını zorlaştırır. Bununla birlikte, hiçbir zaman şifreleme için değil, sadece verimliliği artırmak için birler ve sıfırların eşit dağılımı içindi. Sonuç olarak, her önyüklemede karıştırma farklı şekilde başlatılsa da, sonraki çalışmalar, düz RAM'e erişmek için bellek karıştırmanın nasıl kırılacağını gösterebilir (Bauer ve diğerleri, 2016).

Savunma tarafında, FDE anahtarlarını ve diğer kripto anahtarlarını korumak için, anahtarları RAM'de değil, yalnızca CPU kayıtlarında tutmak için özel anahtar depoları önerilmiştir (Müller ve diğerleri, 2011; Garmany ve Müller, 2013). Ancak bu sistemler, bildiğimiz kadarıyla, üretken ortamlarda kullanılmayan saf akademik kavramlardır. İkincisi, bu tür sistemler tipik olarak FDE anahtarı gibi yalnızca bir anahtarı korurken, diğer içerikler

RAM'de şifrelenmemiş halde kalır. Bu nedenle, soğuk başlatma saldırıları kriptografik anahtarlara değil, Android cihazlardaki görüntüler ve mesajlar gibi diğer bellek içeriğine odaklanan bir tehdit olmaya devam ediyor (Hilgers ve diğerleri, 2014). Sonuç olarak, yazılımda (Gotzfried ve diğerleri, 2016a, 2016b) veya donanımda (Würstlein ve diğerleri, 2016) RAM'in büyük bölümlerini şifreleyen daha karmaşık karşı önlemler, aynı zamanda bu sistemler üzerinde kullanılmayan akademik kavramlar olarak kalır basit bir nedenden dolayı gerçek dünya cihazları: Bellek şifreleme, genel sistem performansını bir büyüklük sırasına göre yavaşlatır.

Özetle, aradan on yıl sonra da cold boot saldırılarının açtığı güvenlik açığı henüz genel olarak kapatılamadı. Gerçekten de, donanım destekli anahtar depoları ve güvenli önyüklemenin etkinleştirildiği (TrustZone) ARM güdümlü Android cihazlarda, artıklık etkisinden yararlanmak bugün son derece zordur. Android satıcıları tarafından zorunlu kılınan kısıtlayıcı platform ayarları ve kilitli cihazlarda özel önyükleyicilerin başlatılmasına izin verilmemesi nedeniyle, RAM modüllerinin nakli genellikle bellek dökümlerini elde etmenin tek yolu olarak kalır. Ancak bu yöntem, akıllı telefonlar gibi gömülü cihazlarda kolayca mümkün değildir, çünkü RAM modülleri kart üzerine lehimlenmiştir ve fişten çıkarılamaz.

Bunlar, bireysel güvenlik araştırmacıları gibi mevcut kaynakların alt ucundaki rakipler için, güncel, tamamen yamalı Android akıllı telefonlardan ham bellek dökümleri elde etmek bugün karşılanamaz hale geliyor. Bunun aksine, devlet düzeyindeki aktörler gibi mevcut kaynakların üst ucundaki rakipler birden fazla seçeneğe sahiptir: Sıfır gün güvenlik açıkları için BootROM istismarları (Checkra1n Jailbreak: Anal, 2020; exynos-usbdl: unsigned co, 2020) satıcılardan “gizli” önyükleyici aşamaları olarak (Redini et al., 2017). Devlet düzeyindeki rakipler için maliyetler önemli değildir, satıcılar genellikle işbirliği yapmaya zorlanabilir ve yonga saldırıları, araştırmaları yürütmenin yerleşik bir yoludur (Mikhaylov, 2016).

2008 yılında Halderman ve ark. (2008), FDE anahtarlarını RAM'den kurtarmak için aeskeyfind aracını sundu. Bu araç, bellekteki AES anahtar zamanlama yapısını tanımlayan bir algoritmaya dayanmaktadır. Ancak araç tek başına ana anahtarı FBE bölümlerine kurtarmak için kullanılamaz.

2017 yılında Loftus ve ark. (2017), Android'in FBE'sine genel bir bakış verdi ve FDE'ye karşı bilinen saldırıların hala uygulanabilir olup olmadığını araştırdı. Sonuç olarak, Android 7.0 ve

sonraki sürümlerde bazı saldırıların hala mümkün göründüğünü belirtiyorlar. Bu yazıda, özellikle anahtar kurtarma yönteminin ve adli alet zincirlerinin FBE zamanlarında nasıl uyarlanması gerektiğini gösteriyoruz.

2019 yılında Grob ve ark. (2019), Android'in FBE özelliğini kullanırken akıllı telefon kullanıcılarının kullanım davranışlarıyla ilgili hassas verilerin sızıp sızamayacağını araştırdı. Meta veriler şifrelenmemiş olarak kaldı ve gerçekten de yazarlar, meta verilerin yüklü uygulamaların bir listesini çıkarmak ve WhatsApp mesajlarının gönderildiği veya alındığı zaman gibi kullanım davranışının izlerini kurtarmak için yeterli olduğu sonucuna vardı. Grob ve diğerleri tarafından gösterilen saldırı. (2019), ancak şifrelenmemiş bir FBE bölümünden çıkarılabilen temel olaylarla sınırlıydı. Bu yazıda, bir RAM görüntüsü elde edildikten sonra FBE'nin tamamen kırılabileceğini gösteriyoruz.

2016 yılında, Unterluggauer ve Mangard (2016), şifreleme anahtarını RAM yerine diferansiyel güç analizi ve diferansiyel hata analizi ile elde etmek için farklı disk şifreleme şemalarını araştırdı. EXT4'ün FBE özelliği dahil olmak üzere Android, Mac OS X ve Linux'un (dm-crypt) FDE uygulamalarını kontrol ettiler. FBE için, ana anahtarı bir dosyaya özgü veri şifreleme anahtarından ve daha sonra kullanacağımız dosyaya özgü olmayan bir olgudan kurtarmanın mümkün olduğundan bahsettiler.

Daha ilgili çalışmaları özetlemek gerekirse: Skillen ve ark. (2013), Wang ve diğerleri. (2012) ve Teufl et al. (2014), Android'in FDE'sinin güvenliğini değerlendirdi ve FDE'yi sertleştirmek için yeni yöntemler önerdi. (Müller ve Spreitzenbarth, 2013), FDE anahtarını soğuk önyüklemeli bir Android cihazından çıkaran FROST adlı bir adli önyükleyici uyguladı. FROST gibi saldırıların üstesinden gelmek için Gotzfried€ ve Müller (2014), FDE anahtarlarını CPU kayıtlarında depolayarak Android'in FDE'sini bellek forrensik saldırılarına karşı sertleştiren bir yöntem uyguladı.

Android 7.0 ile Google, FDE için olası bir ikame olarak FBE'yi tanıttı. Android 10.0 ile FBE, yeni cihazlar için zorunlu şifreleme şemasını alır. FBE, tek tek dosyaların şifrelenmemiş kalabilmesi için bir bölümün tamamı yerine kullanıcı dosyalarını şifreler. Sonuç olarak, bir akıllı telefonun temel özellikleri (örneğin, çalar saatler, çağrı alma) önce telefonun kilidini açmadan çalışabilir ve farklı kullanıcı hesaplarının dosyaları farklı şekilde şifrelenir. Tüm

bölümler şifrelenmediğinden FBE, dosya adlarını ve dosya içeriklerini şifreleyen bir EXT4 özelliği olarak uygulanır.

Yalnızca son Android telefonlarda ek destek ile meta veriler de şifrelenir. Bu nedenle, Android 9.0 ile Google, meta veri şifreleme desteği sunmuştur (Metadata Encryption, 2020). Meta veri şifrelemesi etkinleştirildiğinde, tüm bölümü şifrelemek için FBE'nin üstünde FDE kullanılır. Bu nedenle, dosya verilerinin şifresini çözmek için, önce FDE bölümünün şifresinin çözülmesi ve ardından dosya içeriğinin FBE tarafından şifrelenmesi gerekir. Tipik olarak, bu meta veri şifreleme FDE, ilk olarak cihazın kilidini açmadan temel hizmetleri önyükleme sırasında başlatabilmek için cihaza bağlı bir anahtarla gerçekleştirilir.
FDE'yi soğuk başlatma saldırılarıyla kırma konusunda birçok ilgili çalışmanın zaten yayınlandığını unutmayın. Sonuç olarak, FDE ve FBE kombinasyonu, yeni cihazlarda şifre çözme zincirinin bir parçası olarak saldırımızı hala değerli kılıyor. İlk olarak, FDE'nin, örneğin Halderman ve diğerleri tarafından olduğu gibi yaklaşımlarla kırılması gerekir. (2008) ve daha sonra FBE'nin yaklaşımımız tarafından kırılması gerekecekti.

Teknik olarak, FBE ile her klasör, dahil olan dosya adlarını ayrı bir anahtarla şifreler ve benzer şekilde, her dosya içeriği farklı bir anahtarla şifrelenir. Unterluggauer ve Mangard'a (2016) göre, veri şifreleme anahtarını bir dosya veya klasöre DEKf olarak adlandırıyoruz. DEKf anahtarları, bir ana anahtar MK'den türetilir. Tipik bir Android sisteminde, en az iki farklı ana anahtar türü vardır. Cihaza bağlı olan (MKd olarak işaretlenmiştir) ve cihaz açıldıktan sonra bir pin koduyla kilidini açmadan erişilmesi gereken dosyaları şifrelemek için kullanılır. Tipik olarak, böyle bir ana anahtarın cihaz bağlaması, anahtarın kök ve çekirdek düzeyindeki saldırganlardan korunmasını sağlayan ARM TrustZone'un yardımıyla uygulanır. Diğer ana anahtarlar, bir kullanıcının kimlik bilgilerinden (örneğin, MKc) türetilir. Bu türetme, saldırganların zayıf pin kodlarını kolayca kaba kuvvet uygulayamamasını sağlamak için TrustZone'da da yapılır. MKc anahtarı, örneğin sohbet geçmişleri gibi daha gizli olması gereken dosyaları şifrelemek için kullanılır.

EXT4'teki her şifreli dosya ve klasör, DEKf'yi türetmek için hangi ana anahtarın kullanılması gerektiğini bir anahtar tanımlayıcı ile belirtir. Bir anahtar türetme işlevi (KDF), MK girişlerini ve dosyaya özgü nonce'yi kullanarak DEKf'nin çıktısını verir.

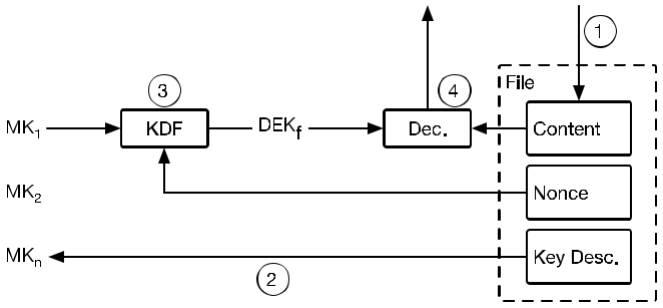
Şekil 44, bir dosyanın içeriğine erişirken farklı tuşların kullanımını göstermektedir. 1. adımında, bir dosyaya erişim istenir. Dosyanın anahtar tanımlayıcısı yardımıyla uygun ana anahtar MK seçilir. 2.adımda, dosya nonce ve ana anahtar daha sonra anahtar türetme işlevi 3. adımda yardımıyla DEKf'yi türetmek için kullanılır. Türetilmiş DEKf ile dosya içeriğinin şifresi, şifre çözme işlevi (DF) ile çözülebilir.

TSK, dosya sistemlerinin ve bölümlerin adli analizi için açık kaynaklı bir kitaplık ve araç koleksiyonudur (Carrier, 2020). İşlevselliği, bir uzantıyı olabildiğince kolay hale getirmek için her biri bir depolama katmanına (örn. görüntü, bölüm, dosya sistemi) odaklanan birden çok modüle bölünmüştür.

TSK, akıllı telefonun belleğinin bit-bit bir kopyasının analiz edildiği fiziksel bir alım gerçekleştirmeye izin verir. Bu teknik, potansiyel olarak silinen ve gizli verilerin kurtarılmasına izin verir. Fiziksel edinimin aksine, akıllı telefonun işletim sisteminin (OS) veri toplama için kullanıldığı mantıksal edinimdir (Sammons, 2012). Bu yöntemin dezavantajları, silinen hiçbir verinin kurtarılamaması ve işletim sistemi ve çalışan uygulamalar sürekli olarak arka planda dosyalara eriştiği için verilerin kazara veya kaçınılmaz olarak değiştirilebilmesini içerir.

Dosya sistemi veya bölüm analizi ile ilgili her görev için TSK, tek amaçlı bir komut satırı aracı sağlar. Örneğin, bir dosyanın meta verilerini göstermek için istat kullanılır ve bir dosyanın içerik verilerini çıkarmak için icat kullanılır. Ayrıca fls adlı bir klasörde bulunan dosyaları listelemek için bir araç da vardır. Örneğin günlük verileri, blok düzeyinde erişim veya bölüm düzeyinde verilere erişim gibi işleyebilecek birçok başka araç vardır.

Doğal olarak, EXT4'te FBE etkinleştirildiğinde, TSK'da yukarıda belirtilen araçların işlevselliği etkilenir. FDE ile şifre çözme işlevini dosya sistemi analizinden ayırmak mümkündür ve bu nedenle TSK'nın şifre çözmeye ihtiyacı olmayacaktır. Ancak FBE ile şifreleme, dosya sistemine sıkı bir şekilde bağlıdır ve bu nedenle, EXT4 FBE dosya sistemlerinden verileri verimli bir şekilde analiz edebilmek ve çıkarabilmek için TSK'yı şifre çözme işleviyle genişletmek gerekir.



Şekil 44: Bir dosyanın içeriğine erişirken dosya tabanlı şifre çözme işlemi.

a) Plaso

Plaso Projesi, log2timeline aracının halefidir. Bu projenin araçları, bir sistemin tüm zaman damgalı verilerinin tek bir zaman çizelgesinde toplandığı süper bir zaman çizelgesi oluşturmamıza olanak tanır (Plaso belgesine Hoş Geldiniz, 2020). Böyle bir süper zaman çizelgesi, adli uygulamalarda olayın yeniden yapılandırılması için iyi bir temel oluşturur.

Plaso, zaman damgalı verileri çıkarmak için bölümler, dosya sistemleri ve uygulama dosyaları gibi farklı depolama katmanları üzerinde yinelenir. Dosya sistemi düzeyinde erişim işlevi, dijital adli sanal dosya sistemi (dfVFS) projesinde özetlenmiştir (Digital Forensics Virtual, 2020). Tipik olarak, zaman damgalı veriler örneğin günlük dosyalarında, anlık mesaj göndericilerinde ve tarayıcı veritabanlarında ve ayrıca dosyaların meta verilerinde bulunabilir. Plaso, birçok dosya türü için farklı uygulama verileri için bir ayrıştırıcı ve özel çözümleyici sağlar. Örneğin, Chrome tarayıcısının geçmişi bir SQLite veritabanı olarak saklanır. Plaso, bir SQLite format ayrıştırıcı modülü ve geçmiş veritabanının farklı tablolarındaki girişleri nasıl yorumlayacağını bilen özel bir analizör sağlar.

TSK için zaten tartıştığımız gibi, Plaso'nun FBE şifreli görüntüleri işleyebilmek için genişletilmesi gerekiyor. FDE ile, bölümlerin şifresi önceden çözülebildiğinden Plaso'nun şifrelenmiş görüntüleri işlemesine gerek yoktu. Ancak FBE ile şifreleme, dosyalara sıkı sıkıya bağlıdır, öyle ki FBE şifreli dosyaları işleyebilmek için Plaso ve dfVFS'yi genişletmemiz gerekir.

b) Dosya tabanlı şifreleme saldırısı

Saldırgan modeli olarak FBE ile Android cihaza fiziksel erişimi olan bir saldırgan seçiyoruz. Disk şifrelemenin görevi, fiziksel erişimi olan saldırganların bir cihazda kalıcı olarak

depolanan verileri elde etmesini önlemek olduğundan bu bir sınırlama değildir. Ek olarak, cihazın açık olduğunu varsayıyoruz, ancak birçok akıllı telefon için tipik bir durum olan kilitli olabilir. Son olarak ve bu bir sınırlamadır, bir cihazın RAM görüntüsüne tam erişime (ve tabii ki onun şifreli diskine erişime) ihtiyacımız var. Yalnızca tam bir bellek görüntüsü verildiğinde, sonunda depolanan verilerin şifresini çözmek için kullanılabilecek ana anahtarı elde etmek için FBE'nin anahtar türetme işlevindeki bir zayıflıktan yararlanabiliriz.

c) Önkoşullar

Saldırımız için (1) bir bellek görüntüsüne ve (2) doğal olarak saldırıya uğrayan cihazdan kullanıcı verileri bölümünün bir kopyasına ihtiyacımız var. Aşağıda detaylandıracağımız bu görüntüleri elde etmek için farklı olasılıklar vardır. Ancak genel olarak, telefona Android Hata Ayıklama Köprüsü (ADB) veya önceden yüklenmiş başka geliştirici erişimi gerekmediğini unutmayın. Genel olarak saldırımız için "bir şekilde" bir RAM görüntüsü elde edilmelidir. Bu, yukarıda bahsedilen soğuk önyükleme saldırılarına bir alternatif olarak, önyükleyicinin kilidinin açılmasını veya RAM modüllerinin, kodun erken bir önyükleme aşamasında yürütülmesine izin veren açıklardan yararlanmalar ve diğer ince ayarlarla bağlantısının kesilmesini gerektirebilir.

d) Bellek görüntüleri

İlke olarak, erken bir önyükleme aşamasında kodun yürütülmesiyle açık aygıtlardan bellek içeriğinin alınmasına izin veren en az iki yol biliyoruz.

Önyükleyici Aşamaları Modern mobil cihazlar, sonunda kullanıcıya yönelik uygulamaları barındıran işletim sistemi başlayana kadar birden çok önyükleyici aşamasından geçer. İlk önyükleyici aşaması, üretim işlemi sırasında cihazda değişmez bir şekilde depolanır ve BootROM olarak adlandırılır. Sonraki tüm aşamalar değiştirilebilir ancak kriptografik imzalar kullanılarak doğrulanarak güvenilir bir önyükleme zinciri elde edilir. Bu güvenilir önyükleme zincirinin tipik bir tasarımı, önyükleyici aşamalarının görevlerine göre ayrıcalıklarını düşürmesine izin vermektir. Örneğin, BootROM en yüksek ayrıcalık seviyesinde başlar ve kontrol işletim sistemine aktarılmadan önce ayrıcalıklar düşer (Dent, 2020). Üreticiler genellikle çiplerini, cihazın sıcak sıfırlanmasından sonra girilebilecek alternatif önyükleme modlarıyla donatırlar. . Girilen önyükleme modu, daha önce açıklandığı gibi ayrıcalık seviyesini de belirler. Kurtarma modu ve hızlı başlatma modu yaygın olarak bilinse ve halihazırda RAM'e doğrudan erişim sağlayabiliyor olsa da, belgelenmemiş ve güçlü erken

aşama önyükleme modları da mevcuttur. Tüm Huawei cihazlarında kullanılan HiSilicon yongalarında, erken bir önyükleyici aşaması, imzalı kodu aktarmak ve önyüklemek için bir seri konsol sunar (hisi-idt, 2020). Tüm Nexus ve Pixel cihazlarında kullanılan Qualcomm yongalarında, Acil İndirme (EDL) modu olarak adlandırılan bir önyükleme modu, doğrudan RAM'e erişim sağlayan EDL programcılarının dağıtılmasına izin verir (Hay ve Hadad, 2018a). Bu programcıların da üretici tarafından imzalanması gerekiyor, ancak birçoğu internete sızdırılmış durumda.

Önyükleyici İstismarları Daha önce bahsedilen belgelenmemiş önyükleyici aşamaları için kriptografik olarak imzalanmış kod gerekli olsa da, erken önyükleyici aşamalarından herhangi birinde bir istismar aynı erişimi sağlar. Bir aygıtın savunmasız bir önyükleyici aşamasına sıcak bir şekilde yeniden başlatılması, başarılı bir kullanımdan sonra kalan bellek alanlarının elde edilmesini sağlar. Bu senaryo yalnızca gerçekçi olmakla kalmaz, aynı zamanda ilk önyükleyici aşaması olan BootROM etkilenirse, yakın zamanda checkm8 açığıyla Apple ürünlerinde gösterildiği gibi (Checkra1n Jailbreak: Anal, 2020) yama yapılamaz. Benzer şekilde ciddi bir kusur, Samsung'un en azından Samsung Galaxy S7 modelinin hatalı bir BootROM içerdiği amiral gemisi serisi Galaxy'yi etkiledi (exynos-usbdl: unsigned co, 2020). Bu notta, Redini ve ark. (2017), mobil cihazların saldırı yüzeyi ile ilgili olarak bootloader kodunun dikkate alınması gerektiğinin altını çizmektedir. Araştırmalarında, Huawei cihazlarında kullanılan önyükleyici aşamalarında birden fazla kusur buldular.

e) Kullanıcı verileri bölümü

Kalıcı kullanıcı verisi bölümünün görüntüsünü almak için saldırganın en az iki seçeneği vardır. İlk seçenek, yukarıda açıklandığı gibi kalıcı bellekten bölümleri boşaltmak için özellikler sunan bir önyükleyici aşaması kullanmaktır. Bu, daha önce tartışıldığı gibi birçok EDL programcısı için geçerlidir (Hay ve Hadad, 2018b).

Diğer seçenek, veri depolama yongasını yontmaktır. Örneğin Nexus 6P, eMMC flash bellek kullanır. Bu tür belleğin içeriği, Etemadieh ve diğerleri tarafından gösterilen yöntem kullanılarak boşaltılabilir. (2017). Kalıcı veri depolama için başka bir seçenek, veri çıkarma için benzer yöntemlere sahip NAND yongalarıdır (Mikhaylov, 2016).

e) Ana anahtar türetme

Saldırımız, Android EXT4 FBE'de kullanılan KDF'nin gizli olmayan verileri şifreleme anahtarı olarak alması gerçeğine dayanmaktadır. İncelediğimiz FBE uygulamasında KDF olarak AES256-ECB kullanılmıştır. Liste 1, bir MK'den bir DEKf türeten ilgili kodu gösterir. Açık kaynak olan Android çekirdeğinin bir parçasıdır. Koda Google Kaynak deposundan erişilebilir[21]. Gösterilen kod alıntıları, ASB-2018-12-05_4.14-p-release taahhüt etiketindeki fs/crypto/keyinfo.c dosyasındandır.

derive_key_aes işlevi, aralarında bazı işlevler bulunan fscrypt_get_encryption_info işlevinden dolaylı olarak çağrılır. Bu çağrıda nonce dosyası türetme_anahtar parametresi olarak alınır ve ana anahtar, kaynak_anahtar parametresinin bir parçasıdır. fscrypt_get_encryption_info işlevi, diğer şeylerin yanı sıra DEKf'yi türetmek için EXT4 dosya sistemlerinden kullanılır. Benzer şekilde, DEKf anahtarlarını türetmek için F2FS dosya sistemi için aynı işlev kullanılır.

Liste 1, türetme_anahtarının (nonce) türetme anahtarı ve kaynak_anahtarının (ana anahtar) kaynak anahtar olarak alındığını gösterir. Son parametre yalnızca türetilmiş anahtarın hedefini tanımlar. derive_key_aes fonksiyonunu araştırırsak, şifre olarak AES-ECB'nin kullanıldığını görürüz (6. satır). Ayrıca, türev_key parametresinin şifreleme anahtarı olarak kullanıldığını görüyoruz (8. satır).

Dosyaya özel veri şifreleme anahtarını DEKf ¼ AESECBnoncef ðMKÞ olarak almak için ana anahtarın AES ECB şifrelemesini nonce dosyasıyla not ederiz. AES ile şifre çözme, şifreleme ile aynı işlevdir. Ana anahtarı MK ¼ AESECBnoncef ðDEKF Þ olarak hesaplayabiliriz, bu da dosyaya özgü şifre çözme anahtarının doğal nonce ile şifresini çözdüğümüz anlamına gelir.

Android çekirdeğinin sonraki sürümlerinde, EXT4 FBE için ikinci bir KDF uygulandı. Bu sürümde KDF, dosya adı şifreleme moduna göre seçilir. EXT4_EN-CRYPTION_MODE_AES_256_HEH modu kullanılırsa, yeni KDF kullanılır. Bu karar, Liste 2'de gösterilen ext4_derive_key işlevinde verilir. Bu kod, https://android.googlesource.com/kernel/msm commit 9db9532a. deposundaki fs/ext4/crypto_key.c dosyasının bir parçasıdır. KDF'nin anahtarı olarak ana anahtar ve bizim saldırı yöntemimiz artık doğrudan geçerli değil. Şu anda, Bölüm 5'te görülebileceği gibi, yeni KDF, değerlendirilen telefonlarımızdan yalnızca biri tarafından kullanılıyor.

Bir FBE dosya sisteminin ana anahtarlarını hesaplamaya yönelik saldırımız, bellek dökümünde bulduğumuz dosyaya özel AES anahtarları DEKf'yi ve kalıcı kullanıcı verilerinin dökümünde bulduğumuz dosyaya özgü nonce'ları kullanır. DEKF anahtarlarını bellekten çıkarmak için AES anahtarlarını bulan mevcut aeskeyfind [22] aracını kullanıyoruz. Çekirdek yapılarını ayrıştırmak yerine bu yaklaşımı kullanmaya karar verdik çünkü bir aygıtın soğuk önyüklenmesi sırasında meydana gelebilecek kısmi üzerine yazmaya veya belleğin solmasına karşı daha dirençlidir.

Kullanıcı verileri bölümünden tüm dosya nonce'larını çıkarmak için, EXT4 dosya sistemini ayrıştırdığımız anlamına gelen genişletilmiş TSK'mızı kullanırız. Bulunan tüm DEKf anahtarları, set FK ¼ fDEK1 olarak not edildiğinde; DEK2; …; DEK ng ve dosya sisteminde bulunan tüm nonces N ¼ fnonce1; nonce2; …; nonceng, tüm potansiyel ana anahtarların kümesini hesaplayabiliriz M ¼ fMK1; MK2; …; AES ECB şifre çözme uygulayarak MK3g:

$AES^{ECB}{}_{n}$ðfkÞ : FK; N/M

Geçerli ana anahtarlar, M kümesinin bir alt kümesidir. AES şifresinin özellikleri, yanlış anahtar çıktısının rastgele verileriyle verilerin şifresinin çözülmesini sağlar. Çözümümüze uygulandığında, bu, aynı MK ana anahtarının çıktısını veren iki farklı FK N çifti bulursak, geçerli bir kullanılmış ana anahtar bulduğumuz anlamına gelir.

Bu çözüm, karşılık gelen MK'yi hesaplamak için aynı ana anahtardan türetilen yalnızca iki DEKf anahtarının bellek dökümünde bulunmasını gerektirir. Ana anahtar, tüm DEKf'leri türetmemize ve EXT4 dosya tabanlı şifreli dosya sisteminin tüm dosya adlarının ve dosya içeriğinin şifresini çözmemize izin verir.

Ana anahtarlar ve dosyaya özel anahtarlar 512 bit boyutunda olduğundan FBE durumunda özel dikkat gereklidir. Bu yalnızca bazı AES modları (örn. AES XTS) tarafından kullanılan ve 2 birleştirilmiş 256 bit anahtardan oluşan bir sözde anahtar boyutudur (Light, 2014). AES, maksimum 256 bit anahtar boyutlarıyla çalışabilir. FBE'de bulunan tüm 512 bitlik anahtarlar, iki birleştirilmiş 256 bitlik anahtarlardır. aeskeyfind ve bizim hesaplamamız, bu 512 bitlik anahtarların yalnızca yarısını verir, ancak tam 512 bitlik ana anahtarlar elde etmek için ana anahtar yarılarını birleştirebilir ve dosya sistemindeki verilerin şifresini çözerek birleştirmeleri doğrulayabiliriz.

f) Uygulama

TSK, FBE ile ilgili meta verilerin çıktısını alacak ve ana anahtarlarla sağlandığında dosya adlarının ve içeriğin şifresini çözebilecek şekilde genişletilmiştir. Ayrıca Plaso'yu genişletilmiş TSK'yı kullanabilmek için genişlettik, öyle ki olay rekonstrüksiyonu FBE'li Android cihazlarda yapılabilir. Şekil 2, tarafımızca genişletilen veya uygulanan tüm araçları ve bunların FBE'li bir Android telefondan olayları çıkarmak için birlikte nasıl etkileşime girdiklerini gösterir.

Uyguladığımız tüm kodlar açık kaynaklı hale getirildi ve
https://www.cs1.tf.fau.de/research/system-security-group/one-key-to-rule/
adresinde herkese açık hale getirildi.

g) Sleuth Kit uzantısı

TSK uzantımız iki kısma ayrılabilir. Birincisi, EXT4 dosya sistemleri için komut satırı aracı istat uzantısıdır. Bu araç, belirli bir meta veri adresinin (inode olarak da bilinir) tüm bilgilerini ve meta verilerini yazdırır. Uzantımızla, araç ek olarak şifrelemeyle ilgili meta verilerin çıktısını verir.

İkinci uzantı, ana anahtarlar sağlandığında TSK'nın dosya adlarının ve şifreli EXT4 bölümlerinin içeriğinin şifresini çözme olasılığını uygular.

Istat aracı, desteklenen her dosya sistemi için uygulanan dosya sistemine özel istat işlevini dahili olarak çağırır. Şifreleme meta veri çıktısı için ext2fs_istat işlevini genişlettik ve yeniden düzenledik. EXT4'te, yeni şifrelemeye özgü meta veriler, EXT'nin bir uzantı özelliği olan bir genişletilmiş öznitelik (XA) olarak depolanır. İki farklı XA türü vardır. Birincisi dahili bir XA'dır ve bir inode meta-veri yapısının içinde depolanır. XA'nın ikinci versiyonu ayrı bir blokta saklanır ve bloğa inode yapısı içinde başvurulur. Bu XA'lara harici denir. EXT4, şifreleme meta verilerinin dahili veya harici XA'lar olarak depolanmasına izin verir.

Istat işlevinde zaten mevcut olan XA'ların yeni bir ext2fs_print_xattr işlevine ayrıştırılmasını hariç tuttuk ve dahili ve harici genişletilmiş öznitelik verileri için iki kez çağırabilmek için genişlettik. XA şifrelemesini, EXT2_EA_IDX_ENCRYPTION (¼9) özel öznitelik türü (idx) ile tanımlarız. Uzantımızdaki tüm EXT4 sabitleri ve yapı tanımları Android çekirdek kaynağından alınmıştır. Şifreleme XA, ext2fs_encryption_entry yapısında tanımlanır.

Değerler sürümü, içerik şifreleme modu, ad şifreleme modu, anahtar tanımlayıcı (8 bayt) ve nonce (16 bayt) içerir. İçerik ve ad şifreleme modu, şifrelemek için hangi şifrenin kullanıldığını tanımlar. Anahtar tanımlayıcı, DEKf anahtarını türetmek için hangi ana anahtarın kullanılması gerektiğini tanımlar. Nonce, türetme işleminde kullanılır. Android çekirdeği 6 (geçersiz seçenek 7 ile) farklı şifreleme modları tanımlar: geçersiz (¼0), AES XTS (¼1), AES GCM (¼2), AES CBC (¼3), AES CTS (¼4), AES HEH (¼126) , ve özel (¼127). Nonce ve anahtar tanımlayıcı onaltılık biçimde yazdırılır; şifreleme modları, istat uzantımızdan insan tarafından okunabilen değişmez değerler olarak yazdırılır.

İçeriğin genişletilmesi ve isim şifresinin çözülmesi için TSK kodunda birkaç noktayı ayarlamak zorunda kaldık. Şekil 3, hangi işlevleri değiştirdiğimizi (kırmızıyla) veya eklediğimizi (maviyle) göstermektedir. tsk_fs_fls işlevi, dosya adlarının çıktılandığı bir çağrı için örnek niteliğindedir. tsk_fs_attr_read ve tsk_fs_attr_walk işlevleri, dosya içeriğinin çıkarılması gerektiğinde başlangıç noktasıdır.

Bağlama işleminde, şifre çözme görevlerini yerine getirebilmek için OpenSSL kitaplığı TSK yürütülebilir dosyalarına bağlanır. tsk/util/crypto_ext.c dosyasında, AES XTS ve CTS şifre çözme modlarını uyguladık. XTS için yalnızca OpenSSL işlevi için bir sarmalayıcı uyguladık. CTS, OpenSSL tarafından sağlanmaz, bu nedenle AES ECB kullanarak uyguladık.

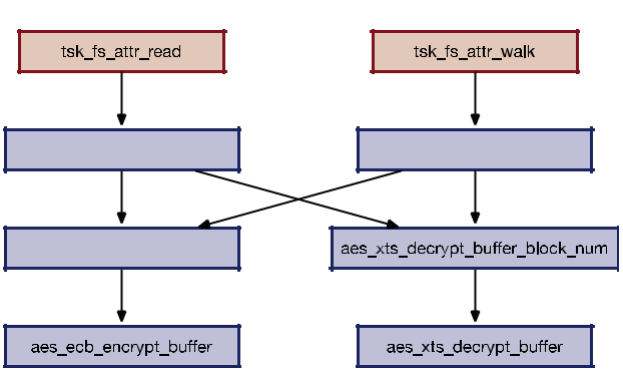
TSK'nın (EXT2FS_INFO) kök EXT4 dosya sistemi yapısına ana anahtarlar eklemeye izin veren fls, fcat, icat ve ifind komut satırı araçlarına yeni bir -K argümanı ekledik. Bu bağımsız değişkenle, kullanıcı, id:key_part1:key_part2 biçiminde satır başına bir ana anahtar içeren bir dosyaya işaret eder.

TSK, EXT4 meta veri yapısını ayrıştırır ve önemli verileri TSK_FS_META genel yapısına kopyalar. Bu genel yapıyı, muhtemelen şifreleme verilerine işaret edecek şekilde genişlettik. EXT4 dosya sistemi durumunda bu işaretçi, ext2fs_encryption_attribute türünde bir yapıya işaret eder. Gelecekteki dosya sistemleri, kendi özel şifreleme özniteliklerine başvurmak için bu genel işaretçiyi kullanabilir. Bu değişikliğe bağlı olarak, tsk/fs/fs_inode.c'de dosya sisteminden bağımsız TSK işlevlerini tsk_fs_meta_ reset ve tsk_fs_meta_close olarak ayarlamak zorunda kaldık. ext2fs_dinode_copy_encryption_attribute ve ext2fs_extract_encryption_attribute işlevleri, şifreleme özniteliklerinin başlatılmasından sorumludur ve ext2fs_dinode_copy işlevinde kullanılır.

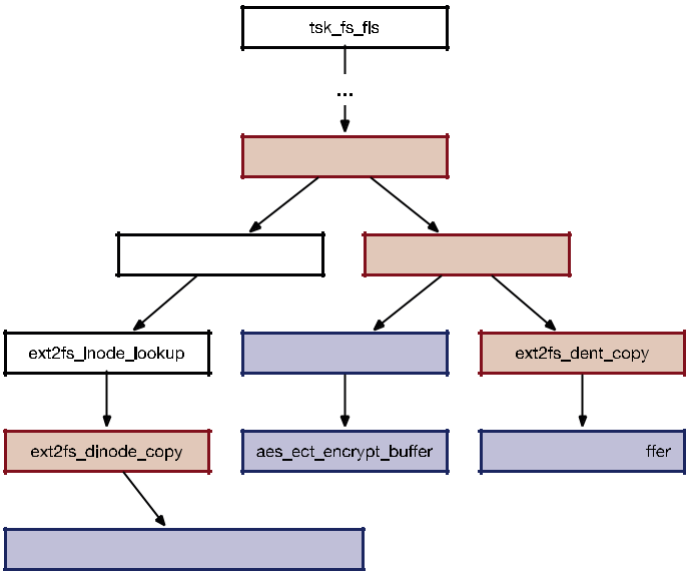
Dosya adlarının ve dosya içeriğinin şifresini çözmek için, şifrelenmiş içeriğe sahip her dosya veya klasör için DEKf'yi hesaplamamız gerekir. Anahtar türetme için dosya sisteminin ana anahtarlarına ihtiyaç vardır. Ana anahtarları, birden fazla ana anahtarı birbirine bağlayabilen EXT2FS_MASTER_KEY yapısında saklarız ve ilk anahtara bir referans, genel dosya sistemi bilgilerini depolamak için çoğu TSK işlevinde kullanılan kök veri yapısı olan EXT2FS_INFO yapısında saklanır. tsk_ext2fs_add_master_keys, ana anahtarları EXT bilgi yapısına ekler. tsk_fs_add_keys, dosya sistemine özel bir işlevi çağıran genel işlevdir.

EXT4'ün anahtar türetimi, bir dosyanın meta verilerini ve EXT2FS_INFO yapısını alan ext2fs_calculate_file_key işlevinde yapılır. Dosya adlarını EXT'de saklayan dizin girişleri, ext2fs_dent_parse_block ve ext2fs_dent_copy işlevinde ayrıştırılır. Şifrelenmiş dosya adlarının şifresini çözmek için bu işlevleri genişlettik.

Bir kullanıcı dosya içeriği verilerini TSK komut satırı araçları veya kitaplık ile okumak isterse, sonunda bu görevleri gerçekleştirmek için tsk_fs_attr_read ve tsk_fs_attr_walk işlevleri kullanılır. Yürüme işlevi, bir dosyanın her veri yığını için belirli bir geri arama işlevini çağırır. Read işlevi, belirli bir boyut ve ofsetteki bir veri yığınını çıkarmak için kullanılabilir. Bu işlevleri, meta veriler içeriğin şifreli olduğunu gösteriyorsa özel işlevlerimizi ext4_walk_encrypted_data ve ext4_read_encrypted_data olarak adlandıracak şekilde genişlettik.



Şekil 46: Değiştirilmemiş işlevler (beyaz), yeni işlevler (mavi) ve tarafımızdan değiştirilen işlevler (kırmızı) dahil olmak üzere Sleuth Kit'in (TSK) işlevlerine genel bakış.



Şekil 45: Değiştirilmemiş işlevler (beyaz), yeni işlevler (mavi) ve tarafımızdan değiştirilen işlevler (kırmızı) dahil olmak üzere Sleuth Kit'in (TSK) işlevlerine genel bakış.

h) Fbekeyrecover

Bu araç, kullanılan ana anahtarları hesaplamak için bir bellek dökümü ve dosya sisteminin bir görüntüsünü alır. Harici bir işlemde yürütülen aeskeyfind, fls ve istat araçlarını kullanır. fls ile, verilen EXT4 dosya sisteminde bulunan tüm düğümler toplanır. Her düğüm için, varsa şifreleme meta verilerini almak için istat çağrılır.

Araç için en önemli olan, bir düğümün nonce ve ana anahtar tanımlayıcısıdır. Tüm nonces ana anahtar tanımlayıcısı tarafından gruplandırılır. Bu işlemden sonra farklı nonce kutularımız var.

Araç, birden çok bellek dökümünü işleyebilir. Bu özellik, dökümleri birden çok dosyaya bölerek çok büyük bellek dökümlerini işlerken meydana gelen aeskeyfind çökmesini atlatmamızı sağlar. Her bellek dökümü aeskeyfind tarafından işlenir ve bulunan tüm 256-bit AES anahtarları tek bir listede toplanır.

Bu aracın son adımı, nonce bin başına tüm potansiyel ana anahtarları hesaplamaktır. Sekiz işçi kullanarak ana anahtarları paralel olarak hesaplar. Böylece, her ana anahtarın oluşumunu sayar. Sonunda, tuşlar azalan sayıma göre sıralanır. En çok isabete sahip nonce bin başına iki 256-bit anahtar, tanımlayıcı için ana anahtarı şekillendirir.
Yaklaşımımızla, bir ana anahtarı yeniden oluşturmak için bellek dökümünde yalnızca iki DEKF dosya anahtarının bulunması gerekir. Bu, bir soğuk önyükleme saldırısı sırasında belleğin bazı bölümlerinin zaten kaybolduğu veya silindiği durumlarda değerlidir.

j) Plaso uzantısı

Plaso'nun uzantısı üç bölümden oluşmaktadır. Öncelikle TSK kütüphanesine Python bağlamaları sağlayan pytsk3 projesini ayarlamamız gerekti. İkinci kısım, bir yol belirtimi ile farklı soyutlama seviyelerinde (örn., görüntü, dosya sistemi, dosyalar) veri erişimi sağlayan dfVFS projesinin uzantısıdır. Bu proje, EXT dosya sistemlerini açmak için TSK kitaplığını kullanır. Son kısım, Plaso projesinin EXT4 ana anahtarlarının komut satırı araçlarına bir argüman tarafından sağlanmasına izin veren uzantısıdır.

pytsk3 modülü, dosya içeriğini almak veya dizin girişlerini listelemek için kullanılabilen FS_Info nesnesini sağlar. FS_Info'yu add_keys yöntemiyle genişlettik. TSK'ya EXT4 ana anahtarlarını tanıtan TSK'nın tsk_fs_add_keys işlevini çağırır. Ek olarak, TSK'nın verilerin şifresini çözebilmesi için libcrypto kitaplığını pytsk3 modülüne bağlamamız gerekiyordu.

dfVFS projesinde TSKPathSpec sınıfına bir master_keys niteliği ekledik. Bu yol özelliği, TSK tarafından işlenen bu tür dosya sistemlerinden sorumludur (bizim durumumuzda EXT4). dfVFS'de, bir dosyadaki belirli dosyalara veya içeriğe referanslar oluşturmak için farklı yol özellikleri birbirine bağlanabilir. Öğe formatına bağlı farklı yol özellikleri dfVFS projesinde uygulanır.

Dosya sistemi görüntülerini açmaktan sorumlu olan TSKFileSystem'in _open yöntemini değiştirdik. Yol belirtiminde belirtilen ana anahtarları pytsk'den FS_info nesnesine eklemek için kod ekledik.

Her TSK yolu spesifikasyonunun ana anahtarları alt TSK yolu spesifikasyonlarına yayması gerektiğinden, aşağıdaki yöntemleri ayarlamak zorunda kaldık:

TSKDirectory sınıfından _EntriesGenerator, TSKFileEntry sınıfından Get_Linked_File_Entry ve GetParentFileEntry ve TSKFile System sınıfından GetRootFileEntry.

Son olarak, yol özelliklerinin serileştirilmesini düzeltmemiz gerekti. Bir görüntü üzerinden tarama sırasında Plaso, yol özelliklerini seri hale getirir ve seri durumdan çıkarır. Ana anahtarları saklamak ve onları alt TSK yolu özelliklerine yayabilmek için Factory (dfvfs/path/factor.py) sınıfındaki PROPERTY_NAMES'e master_keys eklemek zorunda kaldık. Bu, ana anahtarların yol özellikleriyle birlikte düzgün bir şekilde serileştirilmesini ve seri durumdan çıkarılmasını sağlar.

Plaso aracında zaten f.e. sağlamak için kullanılan bir argüman kimlik bilgisi var. BitLocker Sürücü Şifreleme parolaları. Yeni bir kimlik bilgisi türü ext4_master ekledik. Kimlik bilgisi bağımsız değişkeni ext4_master biçiminde kullanılır: <id>,<key_part1>,<key_part2>.

Plaso komut satırı araçlarının log2time-line ve psteal'in ilk çağrılarından biri StorageMedia Tool sınıfının ScanSource'udur. Bu yöntem, verilen kaynağı açar ve hangi bölümün ve dosya sistemi işleyicisinin kullanılacağını arar. Bir EXT dosya sistemi nedeniyle TSK kullanılıyorsa, ana anahtarları _addMasterKeys yöntemimizi çağırarak yol belirtimine ekliyoruz.

Değerlendirme olarak; Saldırımızın uygulanabilirliğini iki farklı şekilde değerlendirdik. İki Google cihazı için tam bir ana anahtar kurtarma işlemi gerçekleştirdik. Bu cihazlar Nexus 5X

ve Pixel XL'dir. Diğer birçok Android cihaz için, örneğin, saldırımızın uygulanabilirliği hakkında bir sonuç çıkarmak için şifreleme meta verilerini çıkararak hangi şifreleme şemasının kullanıldığını kontrol ettik. Bu, Samsung, Huawei ve Xiaomi gibi çok çeşitli satıcıların cihazlarını içerir.

Bellek dökümünde bulunan dosyaya özel veri şifreleme anahtarlarından ana anahtarları başarıyla türettiğimiz cihazlar Tablo 1'de gösterilmektedir. Nexus 5X ve Pixel XL için, LiME çekirdek modülünün yardımıyla belleği boşalttık [23]. Bu modülü yükleyebilmek ve hafızayı boşaltmak için kaynaklardan kendi Android çekirdeğimizi derledik. Yüklenebilir çekirdek modülü (LKM) desteğini etkinleştirdik ve LiME modülünü çekirdek ile birlikte derledik.

Unmkbootimg ve mkbootimg [24] araçlarıyla özel çekirdeğimizi resmi bir fabrika görüntüsünden bir boot.img dosyasına ekledik. Daha sonra LKM desteği almak için cihazı modifiye edilmiş boot.img ile flashladık. Özel çekirdeğe ek olarak, cihazları Magisk [25] ile köklendirdik.

LiME'yi yükleyerek belleği bir görüntüye aktardık. dd ve netcat araçlarıyla, root kullanıcısı olarak userdata bölümünü bir görüntüye döktük. Daha sonra, hem bellek hem de kullanıcı verileri dökümü ile ana anahtar kurtarma aracımızı çağırdık ve olası ana anahtarları değerlendirdik. Her potansiyel ana anahtar için ne sıklıkta hesaplandığını saydık. Nexus 5X için sonuç Şekil 4'te görülebilir. X ekseninde en çok sayılan dört anahtar aday görüntülenir. Bölüm 3'te açıklandığı gibi, yalnızca gerçek ana anahtarları birden çok kez saymayı bekliyorduk. Bu sonuçla, bu varsayımı doğrulayabiliriz.

Nexus 5X üzerinde ise üç farklı master key kullanıldığını gözlemleyebiliriz. Her ana anahtar tanımlayıcı için iki adet 256 bitlik anahtarı birden çok kez kurtardık. Bir ana anahtar 512 bit boyutunda olduğundan (256 bit boyutunda iki anahtar), saldırımız etkilidir. Miktarı düşünürsek, doğru şekilde 512 bitlik anahtarın tamamını kurtarabiliriz.

Ayrıca ana anahtarları bir Android Sanal Aygıtından (AVD) kurtarabildik. Android 10'dan beri AVD, FDE yerine FBE'yi de kullanır. Belleği qemu monitörle boşalttık ve kullanıcı verileri görüntüsü zaten qcow2 biçiminde mevcuttu.
Üç cihazı değerlendirmemiz sırasında, geri yüklenen ana anahtarları ve değiştirilmiş TSK sürümümüzü kullanarak dosya adlarının ve içeriğin şifresini başarıyla çözdük. Kullanıcı

verileri görüntüsünden Chrome tarama olaylarını yeniden yapılandırarak Plaso uzantısını değerlendirdik.

İkinci bölümde, erişimimiz olan akıllı telefonlar tarafından hangi disk şifreleme mekanizmasının (örn. FDE, FBE, FBE þ metadata şifrelemesi) kullanıldığını değerlendiriyoruz. Bir cihaz FBE veya meta veri şifrelemesi kullanıyorsa, ana anahtarları kurtarabileceğimiz KDF kullanımını gösteren ad şifreleme modunu kullanıp kullanmadığını da değerlendirdik. Bu, AES HEH dışındaki tüm ad şifreleme modları için geçerlidir.

Değerlendirmenin bu kısmı, mount komutunun çıktılarını ve paketlenmiş bir yerel ikili dosyayı değerlendiren, uyguladığımız bir Android uygulamasını temel alır. Bu ikili, ioctl aracılığıyla bir EXT4_IOC_GET_ENCRYPTION_POLICY isteğinin yardımıyla kendi veri klasörünün şifreleme meta verilerini okur.

Tablo 2'de incelediğimiz tüm cihazlar listelenmiştir. Bunlardan üçü, yani Nexus 6P, Mi 8 ve P20 lite, meta veri şifrelemesi olmadan hala düz FDE kullanıyor. Ad şifreleme modu, ana anahtarları kurtarabileceğimiz eski KDF'yi kullandıklarını gösterir. Pixel 2, yeni bir KDF kullanımını gösteren HEH isim şifreleme modunu kullanan tek cihazdı.
Ancak, FDE'de kullanılan DEK'i kurtarabilirseniz, yöntemimiz yine de meta veri şifreli bir dosya sisteminin şifresini tamamen çözmede bir yapı taşı olabilir. Tipik olarak, bu FDE anahtarı, bir kullanıcının parolası veya pini tarafından türetilmez. Bunun yerine aygıt donanımına bağlıdır (genellikle TrustZone'da). Bu DEK'e erişilebiliyorsa, yöntemimiz metadata şifreli cihazların FBE katmanının şifresini çözmesi için hala değerlidir. Değerlendirme, Pixel 3 ve Pixel 4 cihazlarının FBE ile birlikte meta veri şifrelemesini ve eski KDF kullanımını gösteren bir isim şifreleme modunu kullandığını gösteriyor. Galaxy S10 ve P40 Pro cihazları için, işletim sistemi erişimi reddettiği için şifreleme meta verilerine erişimimiz yoktu ve bu nedenle kullanımdaki modları daha fazla değerlendiremedik. Bu, Tablo 2'de gösterilir.

Tablo 2: Popüler akıllı telefonlar ve kullanılan şifreleme şemaları.

Sürüm	Cihaz	OS versiyonu	İçerik Şifreleme	Şifreleme ismi	eski KDF	Metadata Enc.
2015	Samsung Galaxy S6	7.0	Tam-Disk Şifreleme		e	e
2015	Google Nexus 6P	8.1.0	AES XTS	AES CBC CTS	✓	7
2016	Huawei P9 lite	7.0	Tam-Disk Şifreleme		e	e
2017	Google Pixel 2	10	Özel	AES HEH	7	7
2017	BQ Aquaris X	8.1.0	Tam-Disk Şifreleme		e	e
2018	Google Pixel 3	9	Özel	AES CBC CTS	✓	✓
2018	Xiaomi Mi 8	8.1.0	Özel	AES CBC CTS	✓	7
2018	Huawei P20 lite	8.0.0	AES XTS	AES CBC CTS	✓	7
2019	Google Pixel 4	10	Özel	AES CBC CTS	✓	✓
2019	Samsung Galaxy S10	10	✳	✳	✳	7
2020	Huawei P40 Pro	10.1.0	✳	✳	✳	(✓)

5.3 REACT NATIVE

React Native, ilk olarak 2015 React.js konferansında tanıtılan, platformlar arası mobil uygulamalar oluşturmak için kullanılan yerel bir komut dosyası oluşturma çerçevesidir [26]. 2015'in başlarında React Native yalnızca iOS uygulamalarının geliştirilmesini destekledi, ancak çerçeve, Eylül 2015'te Android desteğini içerecek şekilde genişletildi [27]. React Native, bazı platforma özgü kodlar gerekli olsa bile, uygulama kodunun büyük bölümlerinin platformlar arasında paylaşılabilmesi anlamında platformlar arası geliştirme vaat ediyor. Ayrıca, programlama topluluğunun gelişimine katkıda bulunmasına izin veren açık kaynaklı bir framework'tür.

React Native, 2013'te açık kaynaklı bir Javascript çerçevesi olan ReactJS'nın [28] ilkeleri ve kavramları üzerine kurulmuştur, ancak Facebook bunu 2011'den [29] beri dahili olarak kullanmaktadır. ReactJS'den bazen sadece React olarak bahsedilir, ancak bu raporda okuyucuyu React Native ile karıştırmamak için ReactJS olarak adlandırılacaktır. React Native kısa süre önce piyasaya sürülmüş olsa da ReactJS aracılığıyla yerleşik bir çekirdeğe sahip. React Native'in arkasındaki temel kavram "bir kez öğren, her yerde yaz". Bu, bir geliştiricinin ReactJS aracılığıyla bir web uygulaması oluşturabilmesi durumunda, önceden yerel geliştirme deneyimi olmadan tüm platformlar için React Native uygulamaları oluşturabilmesi gerektiği anlamına gelir. React Native ve ReactJS, kod yapısı açısından çok benzer. Her iki çerçeveyi de oluşturan Facebook, aralarındaki farkın ReactJS'nin Belge Nesne Modeli (DOM) üzerinde bir web tarayıcısında, React Native'in ise mobil uygulama görünümünde çalışması olduğunu açıklıyor. Bu aynı zamanda mobil uygulamalar için yazılan kodun büyük ölçüde yeniden kullanılabileceği ve ReactJS web projelerinde paylaşılabileceği anlamına gelir [26,30]

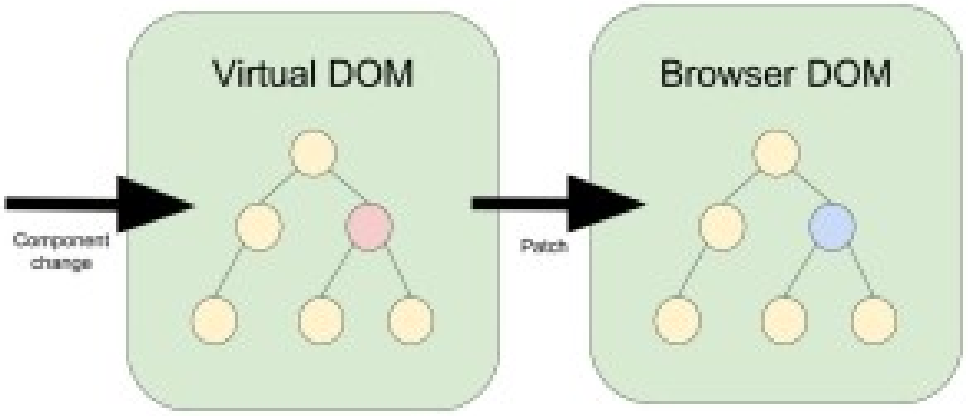
5.3.1 İç Yapı

Hem ReactJS hem de React Native'in en önemli özelliklerinden biri, değişiklikler meydana geldiğinde çerçevelerin uygulama görünümü hiyerarşisinde nasıl çalıştığıdır. Web geliştirmedeki bu değişiklikleri ele almak için üç farklı geleneksel yaklaşım vardır. Bunun bir yolu, tüm sayfayı yeniden oluşturmak için sunucuya yeni bir HTML isteği göndermektir.

İkincisi, sayfanın bölümlerini yeniden işleyecek istemci tarafı HTML şablonunu kullanmaktır. Üçüncü ve en etkili yol, zorunlu HTML DOM değişiklikleri yapmaktır. Daha önceki iki yaklaşım, DOM'un büyük bölümlerinin veya hatta tüm DOM'un yeniden oluşturulması gerektiğinden gecikmelere neden olabilir, bu da pahalı ve yavaş bir işlem olabilir. Bununla birlikte, bu yaklaşımlar genellikle okunması ve bakımı kolay olan kodlu iyi kod yapısına izin verir. DOM ağacını zorunlu programlama yoluyla manuel olarak değiştirmek daha hızlı bir yaklaşımdır, ancak daha büyük uygulamaların geliştirilmesi sırasında bu teknik genellikle hataya açıktır ve bakımı zordur [31].

ReactJS [27], yukarıda bahsedilen uygulamanın avantajlarını kullanmaya çalışır. sanal bir DOM kullanarak dezavantajları giderir ve giderir. ReactJS, bileşenlerini geleneksel olarak şablonlarla aynı amaca hizmet etmek ve bir olay meydana geldiğinde kod bloklarını yapısal olarak değiştirmek için kullanır.

Şablonlar ve ReactJS bileşenleri arasındaki önemli bir fark, tarayıcı DOM'u doğrudan güncellemek ve etkilenen alanın tamamının yeniden oluşturulmasını sağlayan eski içeriği değiştirmek yerine, bileşenlerin güncellenmesinin sanal alanın yeniden hesaplanmasını tetiklemesidir. DOM ve DOM tarayıcısına gönderilen bir yama ile sonuçlanır. Sanal DOM hiçbir zaman tarayıcıda oluşturulmaz, bu da bir şablon aracılığıyla DOM'un yeniden oluşturulması genellikle zaman alıcıdır ve sanal DOM hesaplaması nispeten ucuz bir işlemdir. Bir bileşenin değişikliğiyle güncellenen sanal DOM, daha sonra tarayıcı DOM ile karşılaştırılır ve tarayıcı DOM'u sanal DOM'un bir kopyasına dönüştürmek için gereken en az miktarda değişiklik hesaplanır. Bu değişiklikler daha sonra bir yama olarak sıraya alınır ve şekil 7'de görülebileceği gibi zorunlu DOM manipülasyonu yoluyla tarayıcı DOM'a eşzamansız olarak eklenir. Zorunlu DOM manipülasyonları hızlı olduğu için sanal bir DOM etkilidir. React Native, uygulamayı güncellerken ReactJS ile aynı yaklaşımı kullanır, ancak sanal DOM, sanal bir uygulama hiyerarşisi üzerinde çalışır. React Native hesaplamaları ana iş parçacığına aktarıldığından, her render, bir değişiklik yapıldığında artık tüm uygulamayı yeniden derlemeye gerek kalmaz. Bunun yerine, kod değiştirildiğinde React Native, sanal uygulama hiyerarşisini kullanarak gerekli değişiklikleri uygulayacaktır.



Şekil 47: Bir bileşen değiştiğinde, sanal DOM, Tarayıcı DOM'a gönderilen zorunlu DOM işlemlerinin bir yaması oluşturacaktır.

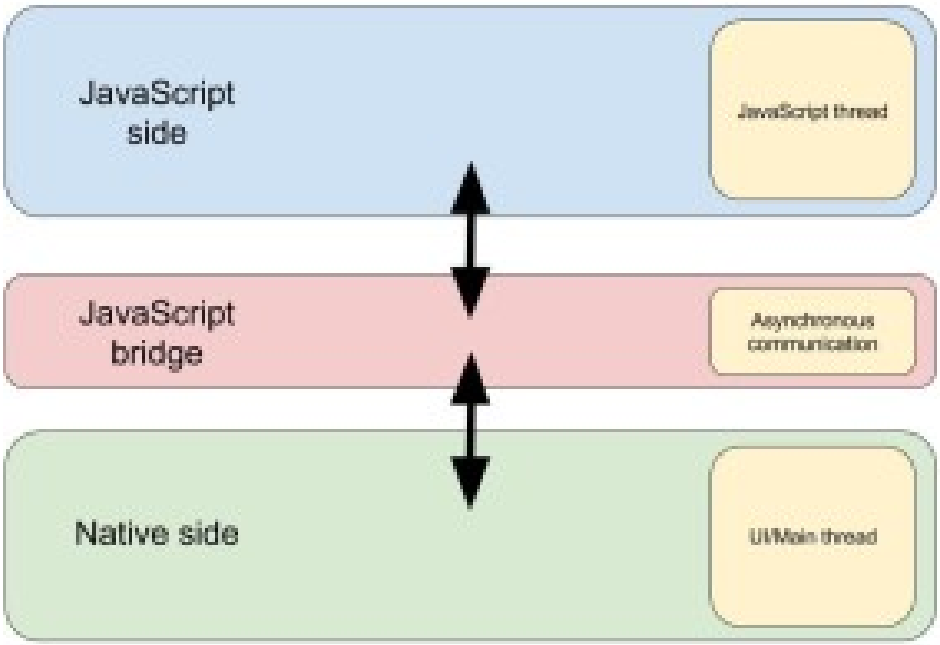
ReactJS kodunu okumayı ve yazmayı kolaylaştırmak için Facebook, JSX adı verilen XML'e benzeyen bir JavaScript sözdizimi uzantısı geliştirdi. JSX'i destekleyen bir derleyici, JSX kodunu normal JavaScript kodunda derleyebilir, ancak JSX'i kullanmak için bir gereklilik değildir, sadece bir olasılıktır. React Native, JSX'i destekleyen Babel [32] adlı bir derleyici ile birlikte gelir ve Facebook, JSX kodu yazmayı şiddetle tavsiye eder. Babel, tercüman desteğini beklemek zorunda kalmadan en yeni JavaScript sözdizimini ve özelliklerini kullanmayı mümkün kılan ES2015'i de destekler. JSX ve JS kullanılarak yazılmış bir ReactJS kodu örneği Şekil 8'de mevcuttur.

React Native, yorumlayıcı olarak JavaScriptCore [33] ile bir JavaScript köprüsü aracılığıyla yerel API'lerle iletişim kurar. JavaScript köprüsü, şekil 8'de görüldüğü gibi uygulamanın JavaScript tarafını ve yerel tarafını birbirine bağlar. JavaScript tarafı, ana iş parçacığından ayrı bir eşzamansız iş parçacığı üzerinde çalışır ve yerel kullanıcı arayüzüne müdahale etmez. Uygulamanın JavaScript tarafından bir çağrı yapıldığında, çağrı hemen gönderilemezse bir mesaj kuyruğunda saklanır. Bilgileri yerel tarafa göndermeden önce JavaScript köprüsü, JavaScript veri türlerini yerel veri türleriyle otomatik olarak eşleşecek şekilde dönüştürür. Yerel mobil ana döngü, performansı en üst düzeye çıkarmak için JavaScript köprüsünün eşleştiği saniyede 60 kare hızında çalışır. Köprünün yöntem dönüş türleri yalnızca geçersiz olabilir, bu nedenle bilgileri yerel taraftan JavaScript yan olaylarına geçirmek için veya geri aramaların kullanılması gerekir. Olay dinleyicileri çeşitli şekillerde uygulanabilir, ancak basit bir çözüm, olayı göndermek için yerel tarafta [34] bir olay yayıcı oluşturmak ve JavaScript de

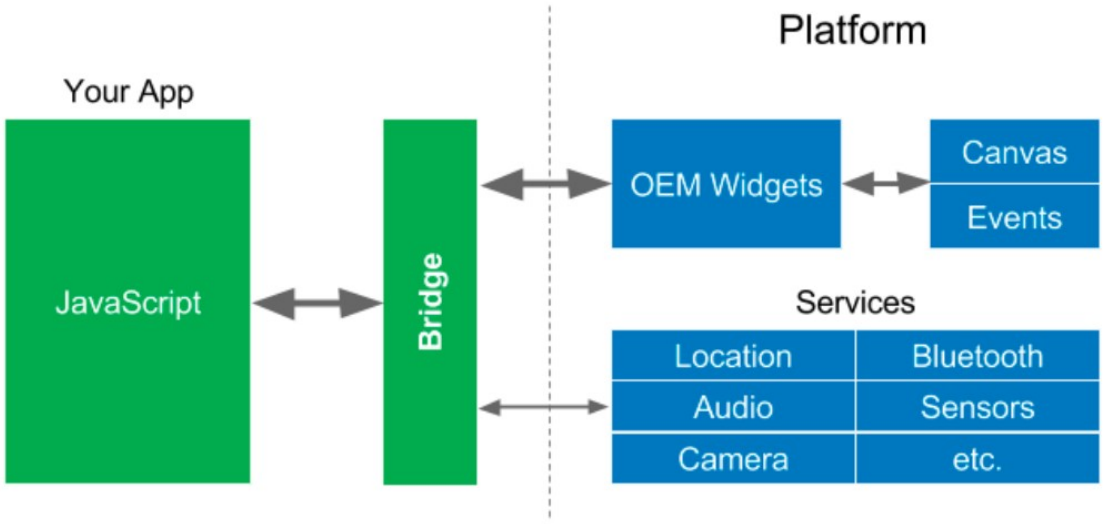
bileşen için bir dinleyici oluşturmaktır [35]. Geri aramalar, geri arama gerçekleştiğinde yerel tarafın ayarlaması gereken herhangi bir sayıda çıkış parametresi ile tanımlanan özel bir veri türüdür [36]. Bu geri aramalar JavaScript tarafında saklanır.

React Native, iki iş parçacığı üzerinde ve ek gönderimle birlikte çalışır kuyruklar. Yerel kullanıcı arayüzü, ikinci iş parçacığı sırasında ana iş parçacığı içinde çalışır. React Native uygulamasının JavaScript kodunu çalıştıran bir JavaScript threadidir [37]. Her yerel bileşen, iş parçacıklarını ve eşzamanlılığı işlemek için bir Grand Central Dispatch Queue (iOS) [38] veya özel bir MessageQueue (Android) [39] kullanır.

JSX kullanmanın enjeksiyon saldırılarını önleme avantajını getirdiğini belirtmekte fayda var. React DOM, girilen herhangi bir değeri, oluşturmadan önce normal bir dizeye işler. Bu nedenle, kullanıcı arayüzü aracılığıyla uygulamanıza hiçbir zaman herhangi bir komut dosyası veya komut enjekte edemez.



Şekil 48: JavaScript tarafı, bir JavaScript threadi ile çalışır ve ana thread üzerinde çalışan native tarafla JavaScript köprüsü üzerinden eşzamansız olarak iletişim kurar.



Şekil 49: React native in platform ile ilişkisi

5.3.2 Uygulama Yapısı

Bir React Native uygulaması, her bir bileşik bileşenin sanal uygulama hiyerarşilerinin bir alt ağacını döndüren bir işleme işlevine sahip olduğu bir bileşik bileşenler ağacı olarak temsil edilir. Her bir bileşik bileşen, dahili durumu depolar ve durum değişikliklerini dinler. Bir durum değişikliği meydana geldiğinde, bileşen değişim olayını alır ve kendisini yeniden oluşturur. Bileşik bileşen, React Native'de uygulanan yerel bileşenleri saran özel bir React Native sınıfıdır. Bu şekilde React Native uygulaması, React Native bileşenleri yerel dillerde uygulandığından ve yerel SDK'ları kullandığından, aslında yerel bileşenlerden oluşacaktır. En yaygın bileşenlerin çoğu zaten React Native'de uygulanmıştır, ancak RCTBrigeModule protokolü ve React Native işaretleme sözdizimi ile yerel programlama yoluyla yeni, özelleştirilmiş bileşenler oluşturmak da mümkündür. React Native'in bileşen ağaç yapısı, uygulamayı oldukça modüler hale getirir.

Uygulama hiyerarşisini düzenlemek için React Native, bileşenleri düzenlemek için bir düzen modu olan CSS3 Esnek kutusu veya Flexbox'ın [40] bir uygulamasını kullanır. Flexbox'ın temel konsepti, bir esnek kaptaki esnek öğelerin, komşu kapların düzenini etkilemeden kabı otomatik olarak doldurabileceği veya içinde kalmak için küçülebileceğidir. Her esnek öğenin, bu öğeye ne kadar boş alan tahsis edilmesi gerektiğine karar veren bir esnek değeri vardır. Şekil 2.4'te gösterildiği gibi, bir öğe aynı esnek kaptaki diğer öğelerle orantılı olarak

esneyecektir. Birkaç öğe aynı flex değerine sahipse, boş alanı eşit olarak paylaşacaklardır, ancak biri diğerinden daha büyük bir flex değerine sahipse, flex değerleriyle orantılı olarak daha fazla yer kaplayacaktır. React Native, bileşenlerin mizanpajı için Flexbox mizanpaj modelinin bir alt kümesini ve ortak web CSS stillerini destekledi. Flexbox React Native, farklı çözünürlükleri ve ekran boyutlarını işleyebilir, çünkü içerik sığmak için alanı doldurur veya küçülür ve bu, platformlar arası geliştirmede büyük bir sorundur [30].

5.3.3 Android ile Karşılaştırma

Dört bölüme ayrılmıştır: geliştirme, performans değerlendirme, platform kod paylaşımı ve kullanıcı çalışması görünüm ve his.

a) Geliştirme

Geliştirmenin sonuçları, uygulamanın ikisi iOS ve ikisi Android'de çalışan dört sürümüydü. Her işletim sistemi için, yerel diller kullanılarak yazılmış bir uygulama ve React Native kullanılarak yazılmış bir uygulama vardır. Ortaya çıkan uygulama ekranları, Ek B'de bulunmaktadır. Bunların, Ek A'daki UI konsept resimlerine benzemesi beklenmektedir. Geliştirme sonuçları, her iki platform için UI konseptine çok benzeyen uygulamaları göstermektedir. Hem React Native hem de native sürümler yan yana karşılaştırıldıklarında bile çok benzer görünüyor. UI konseptinin Everey platformuna özgü özelliği, ortaya çıkan uygulamaların her birinde de mevcuttur. Bu, görece kolaylıkla başarıldı ve hiçbir özellik kaldırılmadı. Geliştirmede başarılması en zor kısım, istatistikler sekmesindeki grafiklerdir. Bunlar, MPAndroidChart kitaplığı 1 üzerine kurulu bir dizi eklenti kullanılarak oluşturulmuştur. Bu kitaplık, hem yerel Android hem de iOS hem de React Native için mevcut olduğu için seçildi. Kitaplık orijinal olarak Android için oluşturulduğundan, bu platform için grafikleri özelleştirmek daha kolaydı. React Native eklentileri en az geliştirilmiş eklentilerdi, ancak her ikisi de açık kaynaklıdır ve Java veya Objective C ile oluşturulmuştur. Bu, grafikleri daha da özelleştirmek için kaynak kodda değişikliklerin yapılmasına izin verdi. React Native herhangi bir Android kaydırıcı bileşeni içermiyordu ve bu nedenle React Native topluluğu tarafından oluşturulan bir eklenti kullanıldı. Bu eklenti, animasyon özelliklerini doğru kullanmıyordu ve kaynak kodunda yapılan değişiklikler yoluyla özelleştirilmesi gerekiyordu.

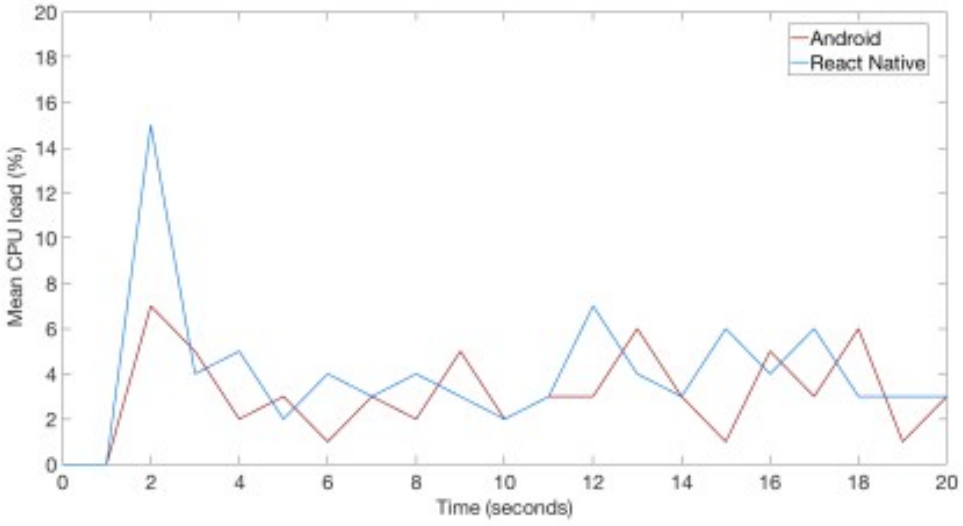
b) Performans Değerlendirmesi

Otomatik test senaryolarının ve açıklanan kullanıcı etkileşimlerinin kaydedilmesiyle aşağıdaki sonuçlar elde edildi. Ölçülerin her biri için, iki yerel uygulama React Native muadilleri ile karşılaştırıldı.

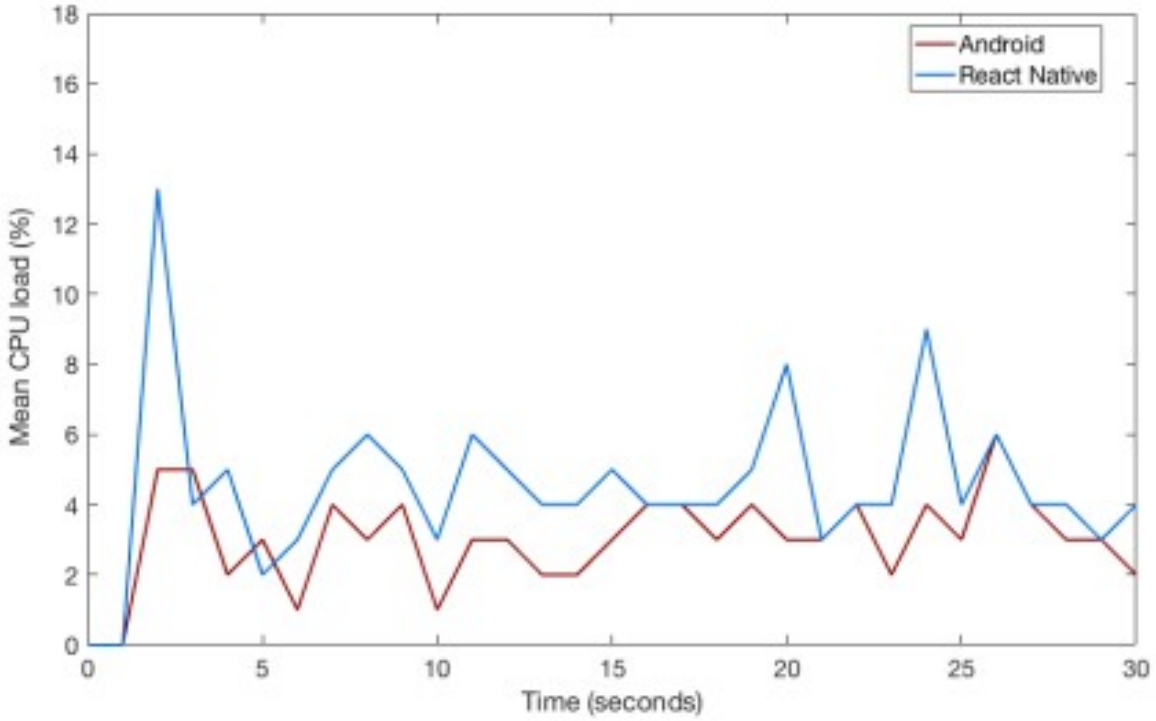
c) CPU Kullanımı

Bu bölümdeki grafikler, her bir senaryo için React Native uygulamasının CPU yükünün yüzdesini ve yerel uygulamaları gösterir. Otomatik test, React Native için çalışır ve yerel Android veya iOS senkronizasyonda başlar, ancak değişen bir süre sonra yerel sürüm geride kalır. Neyse ki, verilerdeki belirgin artışlar sayesinde hangi CPU verilerinin hangi girdi olayının sonucu olduğunu tahmin etmek kolaydır. Bu, React Native ve yerel verilerin adil bir şekilde karşılaştırılmasını sağlar.

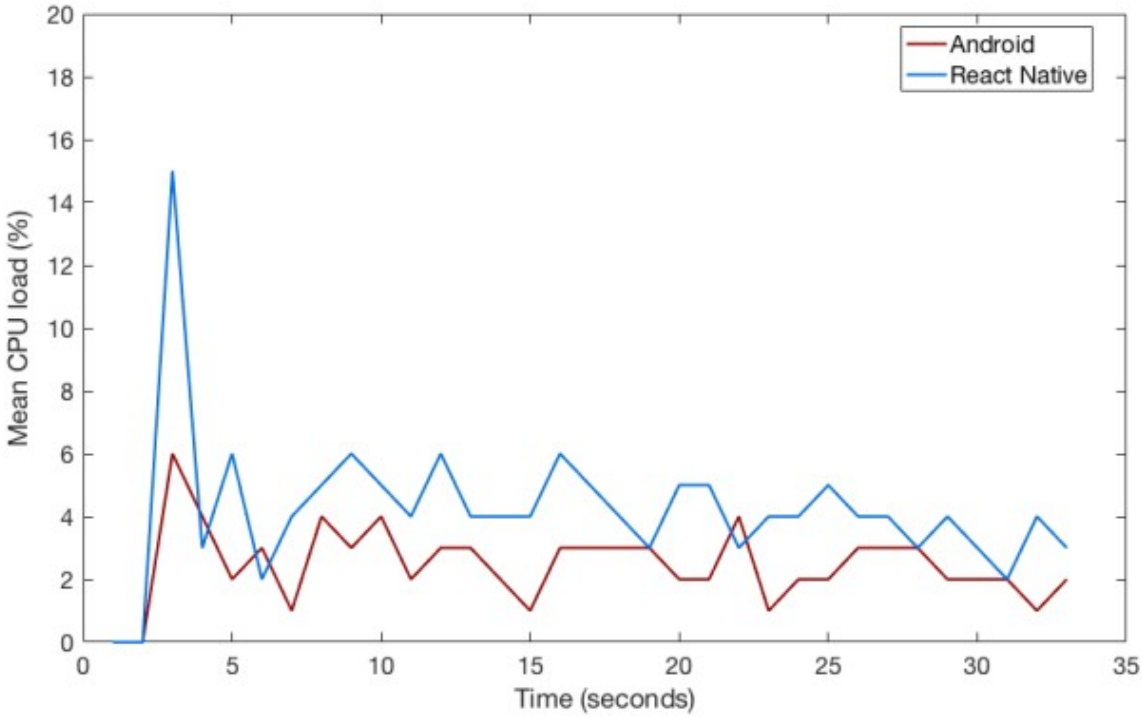
Bir Android cihazdan alınan CPU verilerini temsil eden Şekil 51, 52 ve 53'ün tümü, React Native'de yerel Android'den daha yüksek bir CPU kullanımını gösterir. Android için CPU kullanımındaki toplam ortalama fark küçüktür, yaklaşık% 1-3, ancak farkın daha büyük olduğu örnek noktalar vardır. CPU kullanımındaki ani artışların meydana geldiği bu örnek noktalar, kullanıcı girdisi olaylarından kaynaklanır.
Android'e gelince, React Native ile yerel iOS arasındaki ortalama CPU yükü farkı küçüktür, yaklaşık% 1-3. React Native, uygulama başlangıcında çok fazla CPU kullanır ve bu her şekildeki ilk artışla gösterilir. Üç senaryoda, React Native'in CPU kullanımı Android için% 13-15 ve iOS için% 82-85 değerlerine ulaşır. Üç senaryodaki CPU kullanımının ortalama değeri, Android için% 4-5 ve iOS için% 19-20 aralığındadır, bu da React Native artışlarını çok büyük yapar. Buna karşılık, başlangıçtaki yerel artışlar yalnızca Android için% 5-7'ye ve iOS için% 68-82'ye ulaşır ve ortalama değerler sırasıyla% 3-4 ve% 16-18'dir.



Şekil 50: React Native ve Android uygulaması için 1. senaryo sırasında ortalama% CPU yükü.



Şekil 51: React Native ve Android uygulaması için 2. senaryo sırasında ortalama% CPU yükü.

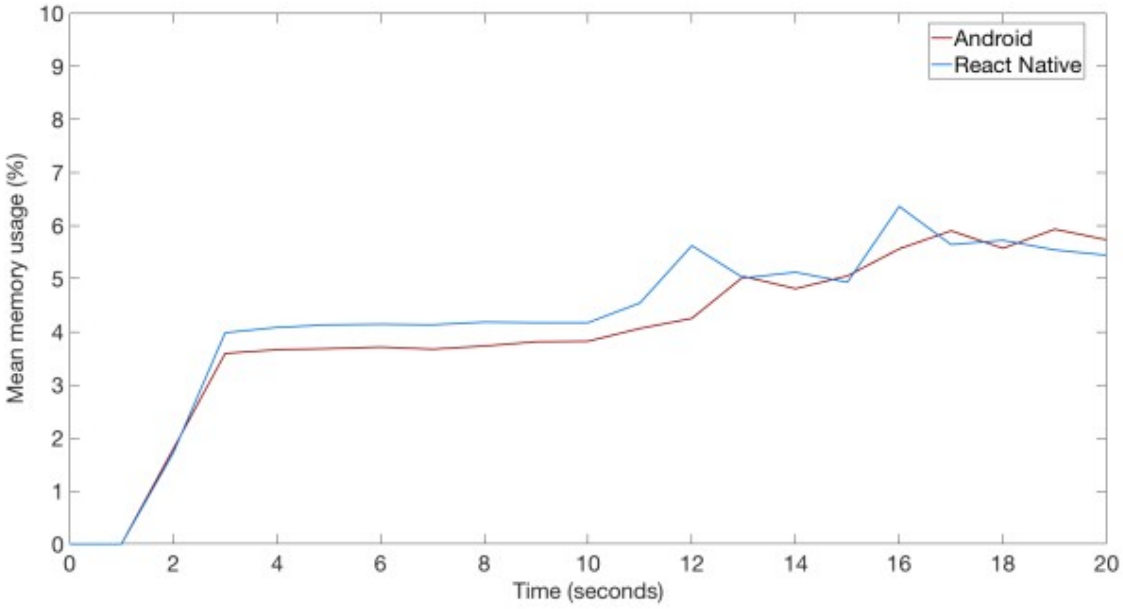


Şekil 52: React Native ve Android uygulaması için senaryo 3 sırasında ortalama% CPU yükü.

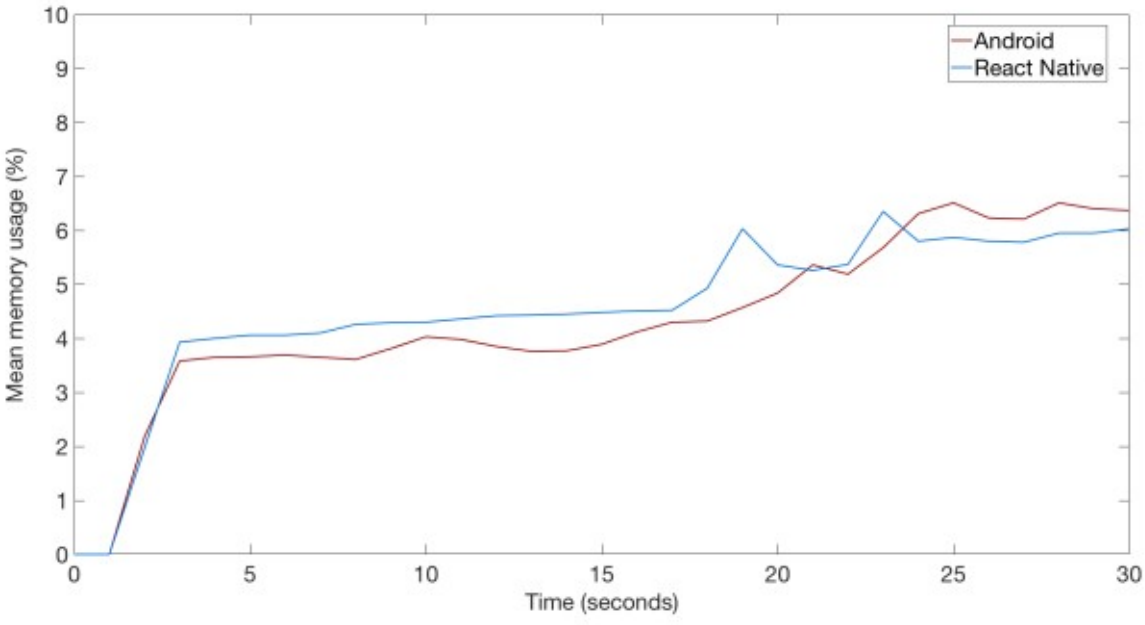
d) Hafıza Kullanımı

Bu rakamlar ayrıca, bir grafiğin her oluşturulmasının bellek kullanımında ani artışa neden olacağı benzer sonuçlar da gösterir. React Native uygulama artışları, ilgili native artışlardan daha yüksektir.

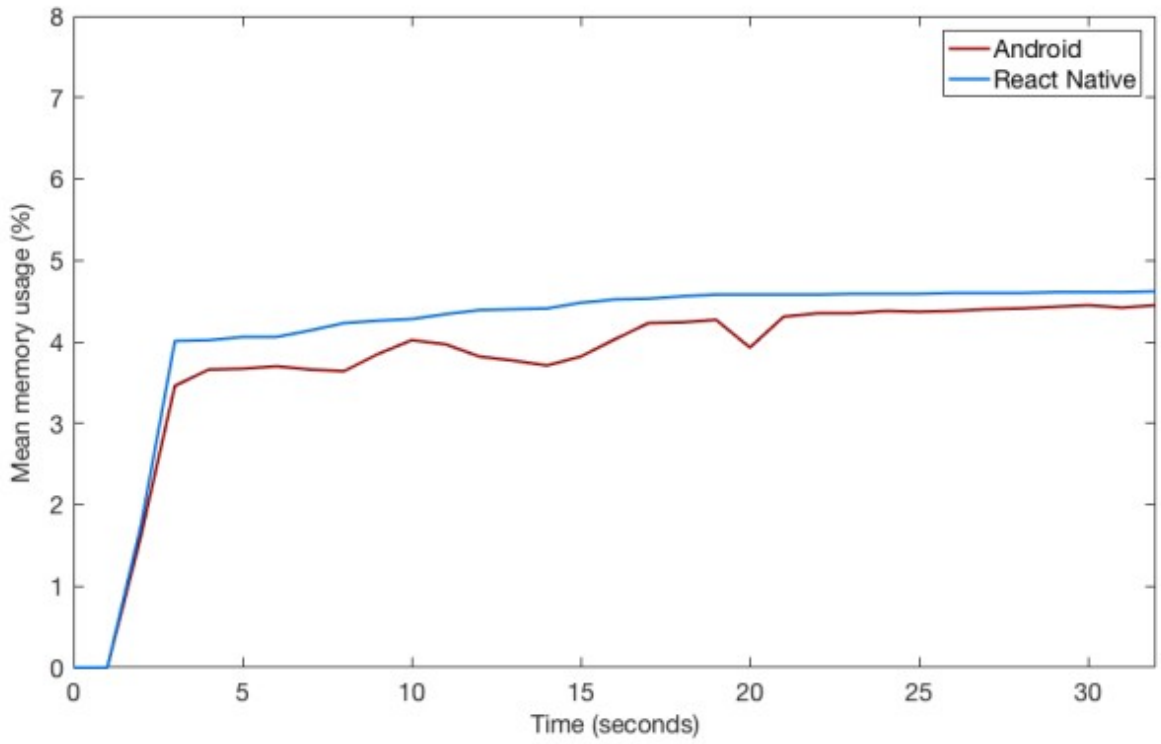
Şekil 11.3'da görülen Senaryo 3, React Native'in senaryonun başlangıcında daha fazla bellek kullandığını, ancak aynı zamanda native uygulama ne kadar çok kullanılırsa farkın azaldığını gösteriyor.



Şekil 53: React Native ve Android uygulaması için 1. senaryo sırasında ortalama% bellek kullanımı.



Şekil 54: React Native ve Android uygulaması için 2. senaryo sırasında ortalama% bellek kullanımı.



Şekil 55: React Native ve Android uygulaması için senaryo 3 sırasında ortalama% bellek kullanımı.

e) Saniyedeki Kare Sayısı(FPS)

Tablolar, her bir senaryonun 5 test çalışması sırasında bırakılan çerçevelerin ortalama sayısını göstermektedir. Ayrıca, uygulamanın 60 FPS'de çalışması gerektiği göz önüne alındığında, senaryonun süresini ve sonuçta beklenen kare sayısını görüntüler. Son olarak, tablolar, ortalama düşme oranını belirtmek için, bırakılan ortalama çerçeve sayısı ve beklenen çerçeve sayısına dayalı bir yüzde değeri görüntüler.

Tablo 3, 4 ve 5'te gösterilen sonuçların tümü, React Native'in yerel Android'e eşit veya ondan daha iyi olduğunu göstermektedir.

Tablo 3: Senaryo 1'deki FPS

Senaryo 1	Native Android	React Native
Ortalama frame düşümü	12.2	7.4
Senaryonun süresi [s]	21	21
Beklenen frame sayısı	1260	1260
Ortalama frame düşümü %	0.97%	0.59%

Tablo 4: Senaryo 2 için android FPS sonuçları

Scenario 2	Native Android	React Native
Ortalama frame düşümü	16.6	11.4
Senaryonun süresi [s]	31	31
Beklenen frame sayısı	1860	1860
Ortalama frame düşümü %	0.89%	0.61%

Tablo 5: Senaryo 3 için android FPS sonuçları

Scenario 3	Native Android	React Native
Ortalama frame düşümü	12	5.6
Senaryonun süresi [s]	33	33
Beklenen frame sayısı	1980	1980
Ortalama frame düşümü %	0.61%	0.28%

f) Tepki Süresi

Bu bölüm, Tablo 3.1'de tanımlanan kullanıcı etkileşimleri için ortalama yanıt sürelerini sunar. Tablo 6, React Native'in etkileşim 2 için biraz daha kötü, ancak diğerleri için daha iyi performans gösterdiğini göstermektedir.

Tablo 6: Kullanıcı interaksiyonları için ortalama tepki süresi(ms)

Kullanıcı interaksiyonu	Native Android [ms]	React Native [ms]
1	768.4	249.6
2	142.6	156.8
3	170.2	129.8
4	342.4	269.8
5	53.8	45.2

g) Uygulama Boyutu

Tablo 7, hem yüklü uygulama boyutu hem de APK / IPA boyutu olmak üzere ortaya çıkan uygulama boyutlarını gösterir. Hem Android hem de iOS için React Native sürümleri, hem yüklü uygulama boyutu hem de APK / IPA boyutu açısından daha büyüktür.

Tablo 7: Her iki versiyon için uygulama ve APK boyutları

Android	Native	React Native
Uygulama boyutu	13.39 MB	23.45 MB
APK boyutu	2.4 MB	8.4 MB

h) Platform Kod Paylaşımı

Tablo 6, React Native kullanılarak Android ve iOS uygulamaları arasında kaç satır kod paylaşılabileceğini göstermektedir. Tablo, tüm yazılı kodların% 62'sinin platformlar arasında paylaşıldığını ve yaklaşık% 20'sinin her platforma özgü olduğunu göstermektedir. Android'e özgü kısım, iOS'a özgü kısımdan biraz daha büyüktür.

Tablo 8: React Native için platform kodu paylaşımı oranı

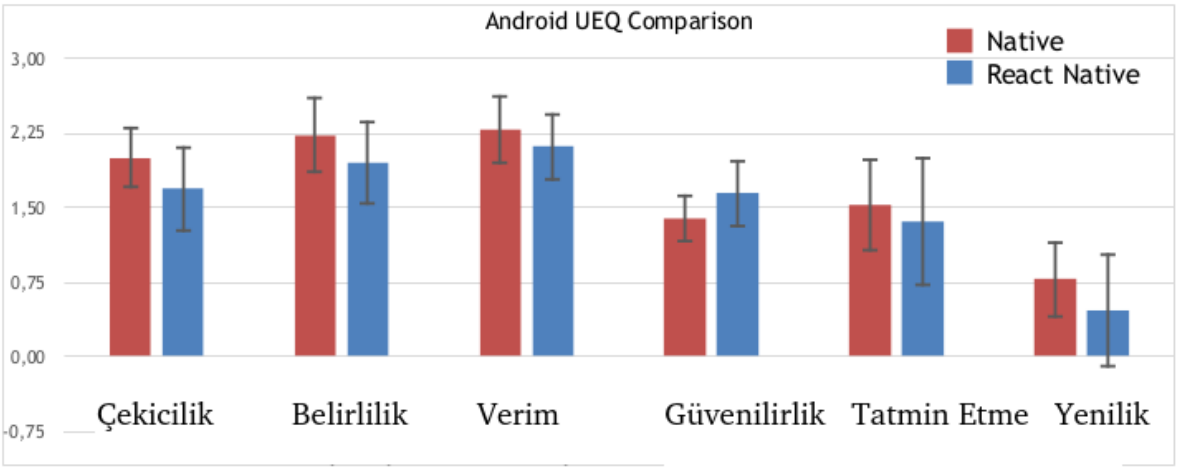
	Paylaşılan	Android'e has	IOS'a has	Toplam
Kod satır sayısı	1308	433	359	2100
Yüzde	62.29%	20.62%	17.10%	100.00%

j) Görüntü ve His, Estetik Özellikler Kullanıcı Araştırması

Bu bölümde, kullanıcı görevleri ve standartlaştırılmış UEQ ile gerçekleştirilen görünüm ve hissetme kullanıcı çalışmasının sonuçları sunulmaktadır. İlk olarak Android platformundan elde edilen sonuçlar, ardından iOS sonuçları sunulur.

Şekil 12, her bir UEQ ölçeği için yerel Android ve React Native uygulaması için ortalama değeri gösterir. Şekil 12 ayrıca, her çubuğun üstünden çıkan siyah aralık göstergesi tarafından

görüntülenen güven aralığını da gösterir. Ortalama puan değeri ve ayrıca şekil 12'teki her bir çubuğun güven değeri de Tablo 7'de sayısal olarak sunulmuştur. Bu değerleri incelerken, React Native'in daha yüksek güven değerlerine sahip olduğu açıktır. Bu, React Native test edicilerinin yanıtları arasındaki yayılmanın daha yüksek olduğunu yansıtıyor. Yerel Android uygulamasına, Güvenilirlik haricinde tüm UEQ ölçekleri için React Native sürümünden daha yüksek bir ortalama UEQ puanı verilir. Bununla birlikte, tablo 8, t-testi sonuçlarını gösterir ve bunlar, sonuçlar arasında α-değeri 0,05 kullanıldığında önemli bir fark olmadığı sonucuna varır. Tablo 4.9, iki Android uygulamasındaki her ölçek için Cronbach katsayılarını göstermektedir. Değerler, ölçek maddelerinin kullanıcı tarafından ne kadar iyi yorumlandığını gösterir. Yerel uygulamanın Güvenilirliği ve Yenilik, önerilenle karşılaştırıldığında son derece düşüktür, en az 0,6'dır.



Şekil 56: Her ölçek için ortalama değerlerle Android UEQ karşılaştırması.

Tablo 9: Android UEQ'dan her ölçek için ortalama ve güven değerleri.

Skala	Native Android		React Native	
	Ortalama	Özgüven	Ortalama	Özgüven
Çekicilik	2.00	0.31	1.69	0.43
Belirlilik	2.22	0.39	1.94	0.42
Verim	2.28	0.35	2.11	0.34
Güvenilirlik	1.39	0.25	1.64	0.34
Tatmin etme	1.53	0.47	1.36	0.65
Yenilik	0.78	0.39	0.47	0.57

Tablo 10: Android uygulamalarının her bir UEQ ölçeği için Cronbach katsayıları.

UEQ Scale	T-test değeri	Çok fark var mı?
Çekicilik	0.2599	Hayır
Belirlilik	0.3562	Hayır
Verim	0.5119	Hayır
Güvenilirlik	0.2600	Hayır
Tatmin etme	0.6899	Hayır
Yenilik	0.4016	Hayır

Tablo 11: Android uygulamalarının her UEQ ölçeği için Cronbach katsayıları.

UEQ Skalası	Native Android	React Native
Çekicilik	0.71	0.85
Belirlilik	0.77	0.87
Verim	0.61	0.55
Güvenilirlik	-0.87	0.66
Tatmin etme	0.69	0.88
Yenilik	0.08	0.74

5.3.4 React Native ve Flutter karşılaştırması

a) Navigasyon

Navigasyon ile kastımız uygulama içindeki navigasyondur. React Native önceden tamamen modern ve native bir navigasyon özelliğine sahip değildi. Fakat topluluk, bir açık kaynaklı kütüphane olan “React Native Navigation(RNN)” i seçti.

RNN'yi işlevsel hale getirmek için, tüm ekran bileşenlerinin uygulamanın giriş noktasında bir yönlendirme yolu olarak kaydedilmesi gerekir. Redux birinci sınıf durum yönetimimiz olarak seçildiğinden, uygulama durumlarının Mağazasının bir kayıt parametresi olarak geçirilmesi gerekir.

Bu, Redux'un gezinmeyi bir eylem göndermeye dönüştürmesi için faydalıdır, böylece gezinme, gerektiğinde hata ayıklama için kaydedilebilir ve yeniden oynatılabilir. Aşağıdaki şekil 58, bileşeni yönlendirme yolları olarak kaydetmek için kod parçacığını içerir.

```
/* eslint-disable import/prefer-default-export */
import { Navigation } from 'react-native-navigation';

import Drawer from './modules/_global/Drawer';
import Movies from './modules/movies/Movies';
import MoviesList from './modules/movies/MoviesList';
import Movie from './modules/movies/Movie';
import Search from './modules/movies/Search';

export function registerScreens(store, Provider) {
    Navigation.registerComponent('movieapp.Movie', () => Movie, store, Provider);
    Navigation.registerComponent('movieapp.Movies', () => Movies, store, Provider);
    Navigation.registerComponent('movieapp.MoviesList', () => MoviesList, store, Provider);
    Navigation.registerComponent('movieapp.Search', () => Search, store, Provider);
    Navigation.registerComponent('movieapp.Drawer', () => Drawer);
}
```

Şekil 57: Rota yolunun kaydı

Şekil 58'de, 4 ekranın tümü RNN'de bağımsız componentler olarak kaydedilir. “registerComponent()” fonksiyonu en az iki parametre alır:

1. Bileşenin tanımlayıcısı olarak kullanılacak bir dize etiketi,
2. Kayıtlı bileşeni döndüren bir işlev.

Mağazayı ve redux sağlayıcısını geçmek, navigasyonun uygulamanın durum yönetimine girmesi için gereklidir. Bileşen içinde yönlendirmede ise RNN'ye ait “showModal()” veya “push()” gibi yöntemler kullanılmaktadır. Her ekran bileşeni, yukarıda belirtilen kayıttan sonra “navigatör” adlı bir nesne alacaktır.

Şaşırtıcı olmayan bir şekilde, Flutter'da Navigator, bir yığın disiplini ile bir dizi alt öğeyi kontrol eden bağımsız bir pencere öğesi olarak bulunur [41]. Kullanıcının gezinebileceği tam ekran widget'lara "Rota" adı verilir ve bu "Rota" widget'ları Navigator tarafından yönetilir. Navi-gator bir yığın olarak işlev gördüğünden, rotayı Navigator'a aktarmak mümkündür ve Navigator, gerektiğinde rotayı dışarı çıkarabilir ve görüntüleyebilir.

```
onTap: (){
  String id = _movie.id;
  Navigator.push(context, new MaterialPageRoute(
      builder: (BuildContext context)=> new MoviePage(movieID: id)));
},
```

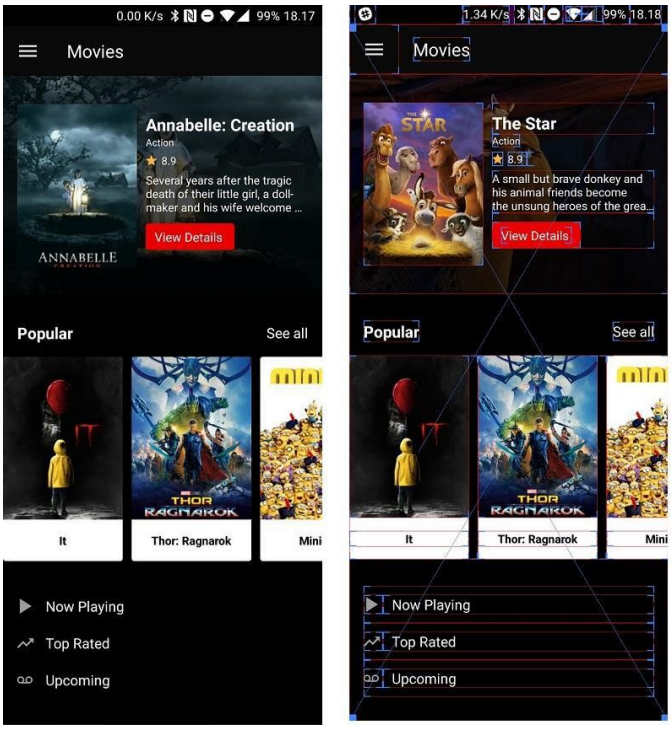
Şekil 58: Rotayı doğrudan navigatör widget'ına aktarma

Şekil 59'da, bir widget'ı gezgin yığınına itmenin temel eylemi gösterilmektedir. “push()” yöntemi iki parametre gerektirir: 1). Geçerli widget'ın bağlamı. 2). Gidilecek bir sonraki widget'ı gösteren bir rota. Rotayı açıkça Push işlevi içinde oluşturmak yaygındır.

Navigatör, bir nesne olarak, parçacığın mevcut bağlamı verilerek doğrudan oluşturulabilir. Bununla birlikte, geliştirme durumunun çoğunda, "MaterialApp" veya "WidgetApp" gibi üst düzey kapsayıcı widget'ı tarafından sağlanan gezgini kullanmak daha akıllıca bir fikir olabilir. Container navigatörünü kullanmanın en büyük kazanımlarından biri, uygulamanın giriş noktasında birden fazla rota kaydetme imkanıdır. Yukarıdaki kod parçacığı, önceden tanımlanmış rota yollarıyla bir malzeme uygulamasını nasıl başlattığımızı gösterir.

b)Görünümler

Hem React Native hem de Flutter uygulaması modülerlik tasarımına saygı duyar. Yani geliştiricilerin, uygulamaları sayfa sayfa yazmak yerine, sayfaları bileşenlere ayırması ve ardından bunları ihtiyaçlara göre birleştirmesi gerekiyor. Layout inspector kullanarak, bir sayfanın bileşenlerin sayısıyla nasıl oluşturulduğu kolayca gözlemlenebilir.



Şekil 59: Layout inspector

Şekil 60, sistem layout inspector'ünün kullanımını gösterir. Inspector'ü kullanmak, kullanıcının mevcut sayfanın nasıl oluşturulduğunu gözlemlemesini sağlar. Bu örnek ana sayfanın dört ana bölümden oluştuğu görülmektedir:

1. Drawer anahtarını içeren araç çubuğu.
2. Güncel en popüler filmin afişi.
3. Tüm popüler filmleri içeren yatay bir liste
4. Arama anahtar sözcüklerinin seçimleri.

Modülerliğin aynı zamanda yeniden kullanılabilirliği de gösterdiğini belirtmekte fayda var. Tipik bir örnek, vaka çalışmamızda kart bileşeninin kullanılmasıdır. Kart bileşeni, kendi kendini kısıtlayan düşük seviyeli ve durumsuz bir bileşen/widget'tır. Başka bir deyişle, tam olarak aynı girdiyle bileşenin her zaman aynı biçimde oluşturulmasını sağlar. Doğasını bağlamla alakasız tutmak için stillerin bileşenin içinde uygulanması ve uygulanması gerekir.

Flutter, modülerlik ilkesini başka bir düzeye taşıyor. Yalnızca görünüm componentinin kendisi bir widget değildir, görünümlere uygulanabilecek her stil ve kural da widgettir. Ayrıca, Flutter, OEM widget'larına rağmen doğrudan ekranda görüntülendiğinden, Flutter, UI araç setini işletim sisteminden ayırabilir. Bu öznitelik, kullanıcı deneyiminin ve beklentisinin tahmin edilebilir ve doğası gereği kararlı olmasını sağlar. Örneğin, yerel bir iOS 11 beta görünümüne ve hissine sahip olan Flutter uygulaması, 7 yıllık bir Android cihazında kararlı bir şekilde çalışabilir.

c) Şekillendirme

React Native ve Flutter, stilin esasen ne olduğu ve görünümlere nasıl uygulanması gerektiği konusunda farklı görüşlere sahiptir. Bu farklılıkların arkasında yatan tarihsel ve perspektif nedenler vardır. Aşağıda bu ayrımların hangi özelliklerden kaynaklandığını ve hangi sorunu çözdüklerini kısaca açıklayacağız.

Şaşırtıcı olmayan bir şekilde, React Native'in temel çerçevesi olarak React, React Native uygulamalarının nasıl düzenlendiği ve geliştirildiği üzerinde güçlü bir etkiye sahiptir. Web perspektifinden ve CSS etkisinden, stil, görünüm bileşenlerine uygulanabilen ilkel kurallar olarak ele alınır. Ancak bu, React Native uygulamasının mevcut CSS dosyalarını doğrudan kullanabileceği anlamına gelmez. Özel bir ayrıştırıcı sınıfı (StyleSheet)

View componentlerine daha fazla uygulanabilen CSS benzeri nesne oluşturmak için bir fabrika olarak oluşturulur.

Öte yandan, Flutter, widget'ın her şey için tek ve tek üst sınıf olduğu ilkesini kararlılıkla kurar ve uygular, bu nedenle stiller, bileşenleri görüntülemekle karşılaştırıldığında eşit şekilde oluşturulmalı ve ele alınmalıdır. Viewlere kurallar uygulamak yerine, Flutter'da stilleri ve görünümleri birlikte oluşturmak daha yaygındır. Tipik bir örnek olarak metin widgeti verilebilir. Yapıcı, neyin görüntüleneceğini (bir dize) ve nasıl görüntüleneceğini (bir stil nesnesi) içeren birden fazla parametre alır. Yukarıdaki kod parçası nasıl çalıştığını gösterir.

```
const myStyles = StyleSheet.create({
  red: {
    color: 'red',
  },
});

export default class StyleExample extends Component {
  render() {
    return (
      <View>
        <Text style={myStyles.red}>just red</Text>
      </View>
    );
  }
}
```

```
const myStyles = StyleSheet.create({
  red: {
    color: 'red',
  },
});

export default class StyleExample extends Component {
  render() {
    return (
      <View>
        <Text style={myStyles.red}>just red</Text>
      </View>
    );
  }
}
```

Şekil 61: Flutter Stillendirme

Şekil 61 ve şekil 62 birlikte sırasıyla React Native ve Flutter'da kırmızı bir metnin nasıl oluşturulduğunu gösterir. React Native'de stil nesnesi, gerçek görünüm bileşeninden dış olarak

bildirilir. Belirli bir stil kuralı gerektiğinde, görünüm nesnesi, ayrı stil nesnesinden kurala başvurmalıdır. Flutter'da, diğer herhangi bir adlandırılmış parametre gibi, stil gerektiğinde "stil" adlı parametre için bir stil nesnesi sağlanır.

d) Performans Karşılaştırması

Önceki bölümlerde belirtildiği gibi, Flutter ve React Native uygulamalarında çok sayıda farklılık vardır. Bu nedenle, aynı senaryonun performansı önemli ölçüde değişebilir. Teknik olarak Flutter, aşağıdan yukarıya mimarisi nedeniyle daha verimlidir ve daha az kaynak gerektirir. Bununla birlikte, React Native daha müreffeh bir topluluğa sahiptir. Örneğin, Redux gibi vaka çalışmamıza yoğun bir şekilde dahil olan olgun kütüphaneler, orijinal olarak React için geliştirilmiştir. Toplamda, duruma göre tartışmadan performansla ilgili nesnel bir sonuç çıkarmak olası değildir.

Performansı ölçmek için bazı kavramların tanıtılması gerekir. Gözlemlenmesi gereken en basit faktörlerden biri olan saniyedeki kare sayısı(FPS), bir uygulamanın akıcılığını tanımlamak için standart bir birim olarak yaygın olarak kullanılmaktadır. Bir çerçeve, geçerli pencerenin bir statik resmi anlamına gelir. Mevcut çerçevedeki herhangi bir küçük değişiklik, başka bir yeni çerçevenin üretilmesine neden olacaktır. Belirli bir cihazda her saniyede oluşturulan kare sayısını kaydederek, iki farklı uygulamanın performansı kolayca karşılaştırılabilir.

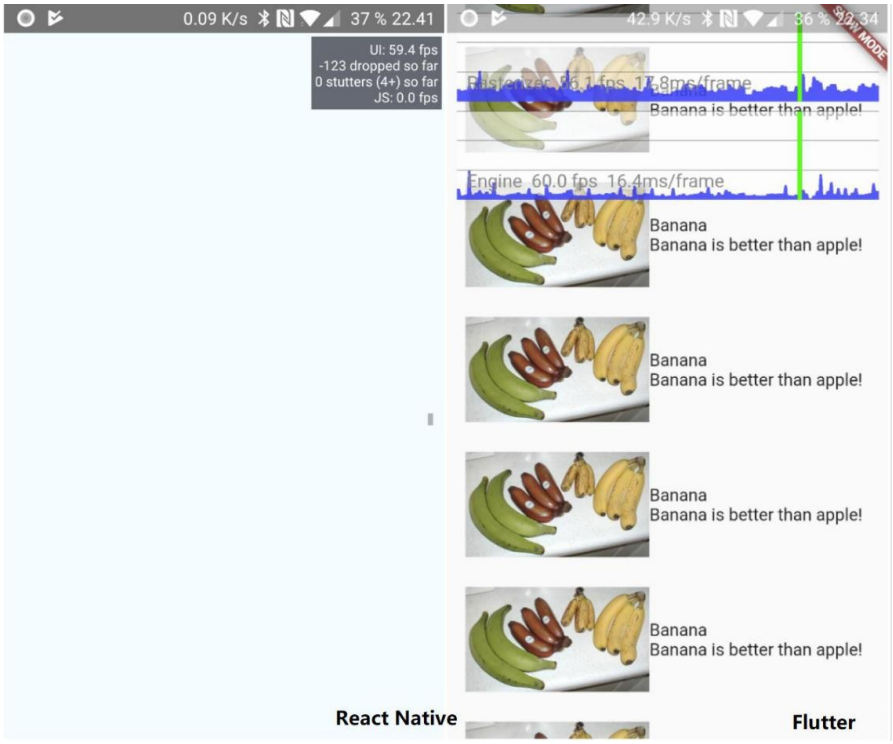
Bu bölümün amacı, Flutter ve React Native uygulaması arasındaki performansın kusurlu bir karşılaştırmasını göstermektir. En çok görülen iki mobil uygulama senaryosu, daha ikna edici bir sonuç için test senaryoları olarak seçilmiştir. Tüm testler ve karşılaştırmalar, kare hızını izlemek için tek bir Android cihazda (Oneplus A3003) yapılacaktır.

Dikey Kaydırılabilir liste, tüm mobil uygulamalarda tipik bir görünümdür. Hem yerel Android hem de iOS'un performansı optimize etmeye odaklanmak için özel bir dizi görünüm koleksiyonu ("RecyclerView" ve "UICollectionView") sağlaması çok yaygındır. Basit bir eylem olarak kaydırma, anında geri bildirim, animasyon ve ön işleme gerektirir.

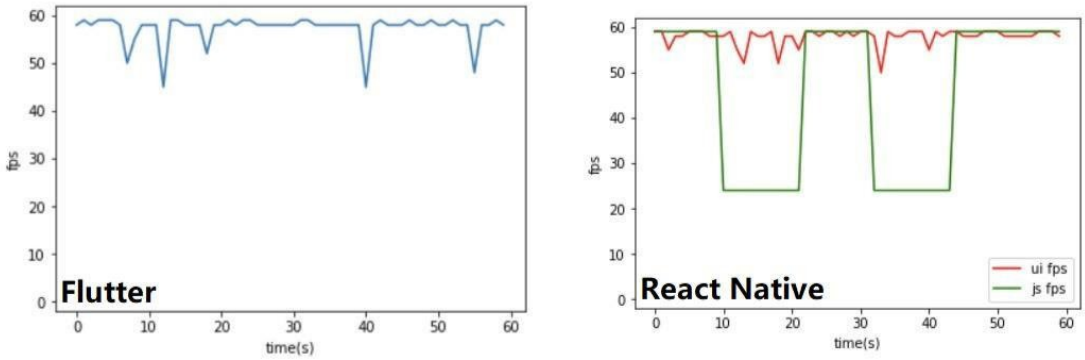
Mart 2017'de React Native, "ListView"ı resmi olarak kullanımdan kaldırdı ve kaydırılabilir liste oluşturmak için birincil bileşen olarak yeni "FlatList"i tanıttı. Yeni FlatList, bellek kullanımını optimize etmek ve basitleştirilmiş API olsa da Pull to Refresh gibi modern

özellikler sağlamak üzere tasarlanmıştır. Öte yandan Flutter, liste widget'ını sık sık yinelemedi. “ListView”, bir dizi veriyi doğrusal sırada görüntülemek için Özel “ScrollView”un özel bir alt widget'ı olarak oluşturulmuştur.

Diğer faktörlerin etkisini azaltmak için, yalnızca çerçevenin bileşeni/widget'ı kullanılarak son derece basitleştirilmiş bir örnek yazılmıştır. Örnek, esas olarak 1000 öğeli dikey kaydırma listesinden oluşur. Her öğe bir resim ve iki satır metin içerir. Aşağıdaki şekil, performansla ilgili en önemli rakamlardan bazılarını ortaya koymaktadır.



Şekil 62: Cihazdaki FPS monitörü



Şekil 63: Kaydırma için FPS takibi

Hem şekil 63'ten hem de şekil 64'ten Flutter ve React Native'in kaydırma konusunda mükemmel bir iş çıkardığını kolayca fark edebilirsiniz. Kaydırma sırasındaki ortalama fps baştan beri 60'ın üzerinde. Ancak, UI iş parçacığındaki FPS, React Native için performansı tam olarak ortaya çıkarmaz. Önceki bölümde belirtildiği gibi, React Native uygulamasının çoğu JavaScript iş parçacığında çalışır. Daha büyük sorun, React Native'deki FlatList'in bellek kullanımını nasıl optimize ettiğidir. Bellek alanından tasarruf etmek için, çok sayıda öğe içeren yoğun bir liste için, Düz liste yalnızca belirli bir öğeyi bir kerede işler. Aşırı hızlı kaydırma gibi belirli senaryolar için boş bloklar yer tutucu olarak işlenecek ve JavaScript iş parçacığındaki fps önemli ölçüde düşecektir. Öte yandan, Flutter'ın fps'si oldukça kararlı. Grafikteki bu pikler aracılığıyla, kullanıcı girdisini ele almanın ve animasyonu birlikte oluşturmanın yeniden kaynak gerektirdiğini gösterir.

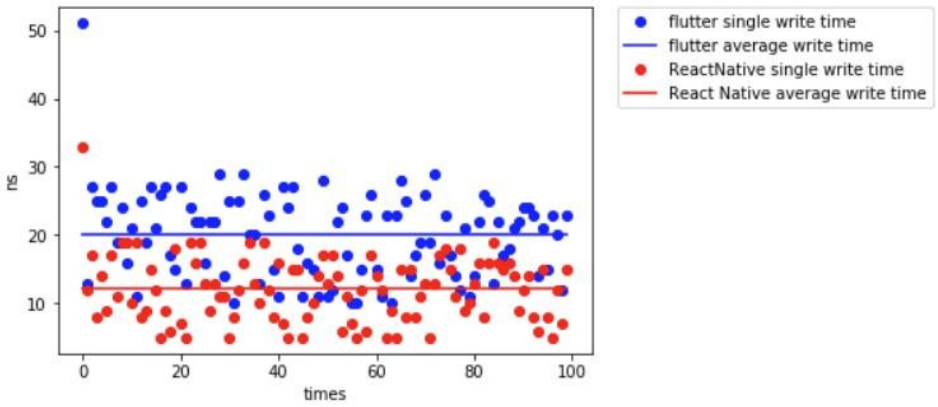
Disk G/Ç Performansla ilgili bir diğer kritik faktör de giriş ve çıkış (I/O) hızıdır. Ana cihaz içindeki dahili dosya alışverişi, disk IO'nun hızı olarak ölçülebilir. Mobil uygulamanın veri depolamak için ana cihazla iletişim kurması yaygın bir senaryodur. Sistem dosyası işlemlerinin çoğu, React Native'de "react-native-fs" adlı bir kitaplık tarafından gerçekleştirilir. Bu kitaplık, React Native uygulaması için yerel dosya sistemine erişim sağlar. Ayrıca, dosyayı eşzamansız olarak okumak ve yazmak için basitleştirilmiş API sağlar. Benzer bir eklenti Flutter topluluğunda “path_provider” adıyla bulunabilir. Her iki kitaplık da ana bilgisayarın cihaz dosya sisteminin yerel optimizasyonunu kullanır.

```
var before = new DateTime.now();
await (await _getLocalFile())
  .writeAsString('Lorem ipsum dolor sit amet')
  .whenComplete((){
    var after = new DateTime.now();
    var difference = after.difference(before);
    times.add(difference.inMilliseconds);
    int sum = 0;
    times.forEach((time){
      sum += time;
    }); // times.forEach
    var avg = sum / times.length;
    print("Average Time: $avg");
  }); // whenComplete
```

Şekil 64: Disk G/Ç hızını ölçmenin temel mantığı

Şekil 65, ölçümün nasıl ilerlediğine dair temel fikri ortaya koymaktadır:

1. Zaman sayacını başlatın.
2. Bu işlev tetiklendiğinde bir dosyayı eşzamansız olarak açın.
3. Daha önce açılan dosyaya bir cümle yazın.
4. Yazma başarılı olduktan sonra zaman sayacını durdurun.
5. Kaydedilen süreyi önceden kaydedilmiş süreler listesine ekleyin.
6. Her dönem için ortalama süreyi hesaplayın.



Şekil 65: Flutter(mavi) ve React Native(kırmızı) dosya yazma grafiği/zamana bağlı

Şekil 66'da gösterildiği gibi, React Native hem ortalama süre (düz çizgi) hem de tek bir zaman (noktalar) tüketme avantajına sahiptir. Bu, esasen “react-native-fs” kitaplığı tarafından yapılan yerel optimizasyon sürecine atfedilecektir. Aslında gerçek yazma hızıyla ilgili olarak, React Native'in performansı native bir uygulamanın ulaşabileceği kadar iyidir. Gözlemlenmesi gereken bir diğer ilginç nokta ise hem Flutter hem de React Native, ilk yazmayı gerçekleştirmek için nispeten uzun zaman kullanıyor. Naif bir varsayım, yerel dosya G/Ç örneğini başlatmak olabilir, yüksek hızlı iletişim gerçekleştirmek için bir ön koşuldur.

React Native, öncü çalışmalarıyla takipçilerini bir ölçüde aktif olarak etkilemiştir. Tek yönlü veri akışı ve JSX gibi React'ten çığır açan kavramlar, React Native'de iyi bir şekilde benimsenmiş ve sindirilmiştir. Güçlü topluluğu ile React Native, çapraz platform uygulamasını sıfırdan başlatmak için şüphesiz en iyi seçimdir. Flutter'ın parlak bir geleceği var. React Native'in sofistike tasarımı, Flutter'ın kendi gelişimi ile iyi korunmuştur. Sözdizimi ve SDK düzeyindeki tutarlılık ve düzenlilik, geliştiricilere neşe getiriyor. Widget'ları özel bir

motor aracılığıyla oluşturmak, performansı artırır ve OEM'lerden kaynaklanan kirlilikleri ortadan kaldırır. Sonuç olarak, hem React Native hem de Flutter, platformlar arası mobil uygulama çerçevesinin değerini büyük ölçüde kanıtlamıştır. Geliştirme ile ilgili verimlilik ve rahatlık, ürünün pazara sürülme hızını kesinlikle artırabilir. Tüm mobil platformlar için kaliteli ve güzel bir uygulama üretmek hiç bu kadar kolay olmamıştı. Bir değiş tokuş olarak, yerel uygulama ile karşılaştırıldığında belirli performans kaybı makul bir şekilde kabul edilir ve buna izin verilir.

5.3.4 Tartışma

React Native beklenenden daha iyi performans göstermesine rağmen, toplanan sonuçların çoğu yerel uygulamaların lehineydi. Performans değerlendirmesi, bu uygulama için farkın küçük olduğunu gösterdi. React Native uygulamasının her iki sürümünde de kodun yaklaşık% 75'inin kullanıldığı da gösterildi. Görünüm ve his, geliştirme sonuçlarının yanı sıra kullanıcı çalışmasında da gösterildiği gibi aynıydı. Aşağıdaki bölümler, farklı sonuçların teoriyle ve diğer çalışmalarla nasıl karşılaştırıldığını göstermektedir.

a) Geliştirme

React Native, Android ve iOS'ta yerel geliştirmeye kıyasla mevcut bileşen sayısı açısından sınırlıdır. Bununla birlikte, çerçeve, açık kaynaklı özel bileşenlerle katkıda bulunan aktif bir geliştirici topluluğuna sahiptir. Bu, React Native temel bileşenlerini oldukça geniş bir kitaplık oluşturacak şekilde genişletir. React Native'de eklentilerin kullanımı ve başlaması kolaydır, ancak olumsuz tarafları da vardır. Eklentilerin yerel kaynak kodu, herhangi biri katkıda bulunabildiğinde farklı kalitededir, bu bazen eklentinin özelleştirilmesini gerekli kılar. Yine, eklenti kaynak kodunun kalitesine bağlı olarak, onu özelleştirmek hızlı ve kolay veya zaman alıcı ve karmaşık olabilir. React Native uygulamasının Android sürümünü geliştirirken yerleşik kaydırıcı bileşeni yoktu. Ancak, birkaç ay sonra React Native 0.24 1 sürümünün yayınlanmasıyla birlikte piyasaya sürüldü. React Native hala genç bir çerçeve ve yeni bileşenlerle sık sık güncelleniyor. Android desteği şu anda iOS kadar kapsamlı değil, ancak hızla yetişiyor. Uygulama, örneğin kamera veya Bluetooth gibi platforma özgü daha fazla özellik kullanabilirdi. Uygulama spesifikasyonu sırasında, Attentec'in standart bir uygulama siparişinin hangi özellikleri içereceğine çok fazla odaklanıldı. Odak başka bir yere konulmuş olsaydı, bu, platforma özgü kod ve hangi özelliklerin mevcut olduğu konusunda daha fazla zayıflık gösterebilirdi. Ayrıca muhtemelen platformlar arasında paylaşılabilecek kod miktarını da düşürecektir.

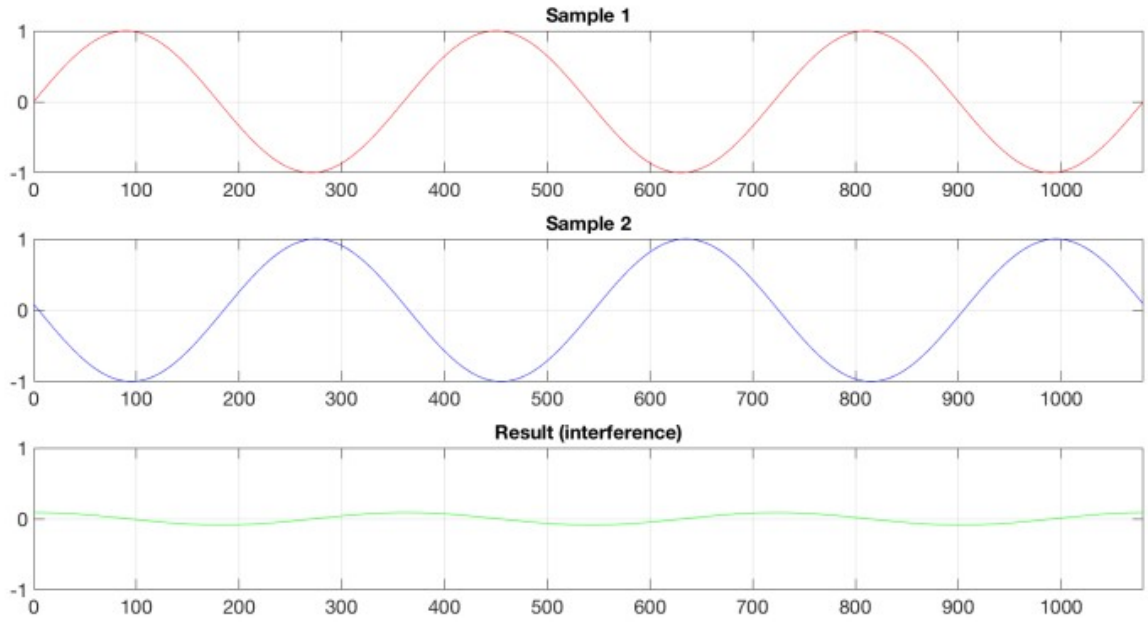
b) Performans Değerlendirmesi

Bu bölüm, performans sonuçlarının mevcut teorilerle nasıl karşılaştırıldığını ve gözlemlenen kalıplara ve düzensizliklere neyin sebep olduğunu tartışmaktadır. Bu bölüm ayrıca sonuçları, ölçümleri bu çalışmada gerçekleştirilen ölçümlerle bir şekilde karşılaştırılabilir olan diğer çalışmalarla, çoğunlukla karşılaştırır.

c) CPU Kullanımı

Yerel Android uygulaması, her kullanıcı giriş olayıyla birlikte HTTP istekleri gönderir. Yerel Android, kullanıcı giriş olaylarıyla bağlantılı olarak sürekli HTTP istekleri gönderirken bile, tutarlı bir şekilde React Native'den daha düşük bir CPU kullanımı gösterir. Başlangıçta tüm verilerin talep edildiği React Native sürümündeki uygulama yapısı nedeniyle, daha fazla veri almak için HTTP isteği gönderilmez. Bu olaylarda CPU kullanımında önemli farklılıkların olmaması, HTTP isteklerinin CPU yükünü önemli ölçüde etkilemediğini gösterir.

React Native uygulamasının, ana iş parçacığına ek olarak JavaScript iş parçacığını da başlatması gerekir. Bundan sonra, JavaScript iş parçacığını kullanarak sanal DOM oluşturur ve ortaya çıkan görünümü JavaScript köprüsü aracılığıyla ana iş parçacığına gönderir. Bu işlem, uygulamanın tüm ömrü boyunca başlangıçta ekstra CPU gerektiriyor gibi görünmektedir. Senaryo 2, uygulamada iki grafiğin açılmasıyla sona erer. İkinci büyük React Native ani artışından iki zaman adımı ötede yer değiştirmiş olmasına rağmen, yalnızca bir büyük sivri uç görülebilir. Bununla birlikte, rakamları oluşturmak için kullanılan toplanan veriler, yerel Android'de de sona doğru iki büyük artış olduğunu gösteriyor. Topladığı verileri manuel olarak incelediğinde, iki yerel zirvenin ortalama değeri% 7 ve% 8 olduğunu ve bu da her ikisinin de React Native emsallerinden yalnızca bir yüzde daha düşük olduğunu gösteriyor. Bu sivri uçlardan ilki, beş okumanın bazılarında sivri uç yer değiştirdiği için yerel arsada kaybolur. Bu, ortalama bir değer hesaplandığında, ani yükselmenin aralarında düşük değerlerin olduğu bir alana yayıldığı anlamına gelir. Bu, şekil 13'deki örneğe benzer bir girişim modelinin oluşmasına neden olur.



Şekil 66: İki örnek veri kümesi birbirine eklendiğinde parazit etkisi.

d) Hafıza Kullanımı

Bellek kullanımı çalışması, React Native için umut verici sonuçlar gösteriyor. Sonuçlar her iki platform için de çok benzer, React Native, bellek kullanımı açısından yerel ile neredeyse aynı. İOS sonuçları tamamen aynı sonuçları gösteriyor ve bu umut verici görünse de, aslında seçilen ölçüm yöntemiyle ilgili bir sorun olduğu anlamına gelebilir. . Ancak yöntem tarafından kullanıldı ve iOS'ta platformlar arası ve yerel arasında farklı sonuçlar elde ettiler. Bu, sonuçların geçerli olduğunu veya ölçüm aracının çalışma biçiminde bir değişiklik olduğunu gösterir. Araçta yapılan değişiklikler hakkında hiçbir bilgi bulunamadı ve bu nedenle sonuçların geçerli ve güvenilir olduğuna inanıyoruz.

Diğer Android çalışmaları ile ilgili olarak, PhoneGap uygulamalarının hem yerel uygulamalardan hem de Xamarin ile oluşturulan uygulamalardan önemli ölçüde daha fazla bellek kullandığını göstermektedir. React Native kullanan sonuçlarımız, Android'de Xamarin kullananlara benzer sonuçlar göstermektedir. Bu, React Native'in bellek kullanımı açısından Xamarin'e benzer olabileceğini gösterir.

e) Saniyedeki kare sayısı

Neredeyse tüm kare düşüşleri, bir animasyon tetiklendiğinde, örneğin, ana ekrandan oda ekranına bir geçiş başlatıldığında gerçekleşir. Testler arasındaki fark, bu animasyonlar sırasında yerel sürümün her zaman daha fazla kare düşürmesidir. Bazı animasyonlar native ve React Native'de aynı şekilde gerçekleştirilmedi ve bu, sonuçları etkilemiş olabilir. Geliştirme sırasında tamamen aynı animasyonlar kullanılarak daha fazla odaklanılsaydı daha güvenilir sonuçlar elde edilebilirdi. Bu yaklaşımla aynı sonuçlara ulaşılsaydı, yerel ve React Native için animasyonların uygulanmasındaki farklılıklar hakkında daha fazla sonuç çıkarılabilirdi. Ancak, düşme oranı tüm senaryolar için çok düşüktür ve ortalama atlanan kare sayısı hiçbir zaman% 1'i geçmez. Bu, yerel ve React Native uygulaması arasında farklılıklar tespit edilebilmesine rağmen, bunların kullanıcı deneyimi üzerinde önemli bir etkisi olmayacak kadar küçük oldukları anlamına gelir.

f) Tepki Süresi

Şekillerden görüldüğü gibi Android yanıt süresi sonuçları, React Native'in tüm test durumlarında daha hızlı veya yerelden daha hızlı olduğunu gösterdi. Aykırı değer, test numarası 1, uygulama başlangıcıdır. Bu testte React Native, yerel versiyondan belirgin şekilde daha hızlıdır. Bunun nedeni muhtemelen ilk oluşturma etkinliğindeki uygulama farklılıklarından kaynaklanmaktadır. Uygulamalara baktığınızda, başlamalarının yaklaşık olarak aynı miktarda zaman aldığını görürsünüz. Bu sonuç React Native için olumludur ve yanıt süreleri kullanıcı deneyimini büyük ölçüde etkilediği için önemli bir husustur.

g) Uygulama Boyutu

Uygulama boyutu, APK / IPA boyutu ve yüklü uygulama boyutu olmak üzere iki şekilde ölçülmüştür. APK / IPA boyutu, platforma özgü araçlardan oluşturulan dosyalarda ölçüldü. Ancak, bir kullanıcı bu yükleyicileri kullandığında, genellikle bir uygulama mağazasından indirilirler. Bu uygulama mağazaları, özellikle iOS için dosyalara genellikle daha fazla yük ekler. Apple uygulama mağazasına bir uygulama yüklemek için başvuruyu incelenmek üzere göndermeniz gerekir. Bu, zaman ve kaynak açısından pahalı bir girişimdir ve bu nedenle bu atıldı. Bunun yerine, montajcıları inşa etme ve boyutlarını ölçme yaklaşımı seçildi. Beklenen sonuç, çok benzer olduklarından, mağaza sahibinin uygulamanın her iki sürümüne de eşit miktarda ek yük eklemesidir.

Yüklenen uygulama boyutu, entegre işletim sistemi araçları kullanılarak ölçüldü. Bu yöntem, bu araçlar güvenilir bir kaynak olarak kabul edilen iOS ve Android geliştiricileri tarafından geliştirildiği için seçildi. Sonuçların yüksek bir geçerliliğe sahip olduğu ve mevcut en doğru sonuçlar olduğu kabul edilir.

h) Platform Kod Paylaşımı

Platform kod paylaşımının ölçülme şeklinin bazı dezavantajları vardır. Örneğin, her hattın paylaşılan veya platforma özgü olarak sınıflandırılabileceği gerçeğine dayanır. React Native platforma özel dosya uzantılarına izin verdiği için bu daha kolay hale geldi. ".Android.js" ile biten her dosya Android'e özgü olarak sayıldı ve ".ios.js" ile biten her dosya iOS'a özgü olarak sayıldı.

Javascript ile veya bizim tarafımızdan yazılmadıkları için eklentilerin hesaba katılması da zordu. Bu eklentileri tamamen sınıflandırmanın dışında tutmaya karar verdik. Mevcut olan eklentiler genellikle iOS'a özeldir. Bu nedenle, daha az kod satırı yazdığınız bir iOS uygulaması oluşturmak genellikle çok daha kolaydır. Bununla birlikte, eklentiler dahil edilirse daha fazla kod kullanmanız gerekebilir. Uygulamamız için olabildiğince az eklenti kullanıldı ve kullanılanların her iki platform için de mevcut olduğundan emin olduk.

Kodu sınıflandırmanın otomatik bir yolunu bulamadığımız için, gerçekten manuel sınıflandırmadan başka bir seçenek yoktu. Yerel ve React Native arasında herhangi bir kodu karşılaştırmadık, çünkü bunlar farklı dillerde yazıldı ve ondan güvenilir sonuçlar çıkarmak mümkün olmayacaktı.

j) Estetik

Görünüş ve hissetme kullanıcı çalışması, standartlaştırılmış bir kullanıcı deneyimi anketiyle gerçekleştirildi. Örneğin, kullanıcının bir anketle birlikte nasıl hissettiğini analiz etmeye çalıştığımız yüksek sesle düşünme protokolü kullansaydık, görünüm ve hissin daha kapsamlı bir analizini elde edebilirdik. Ancak, bu yaklaşım daha fazla zaman aldığından, sınırlı bir zaman çerçevesi içinde böyle bir kullanıcı çalışması yürütmenin mümkün olmadığını düşündük. Kullanıcıların uygulamayı kullanırken nasıl hissettiklerini açıklayabilecekleri, standartlaştırılmamış bir anket kullanmak da ilginç olabilirdi. Bu, bazı sonuçların sınıflandırılmasına da ihtiyaç duyacağından, verileri analiz etmeyi zorlaştıracaktır. Ayrıca

sonuçlar tarafımızca sınıflandırılırsa uygulamaları birbirleriyle karşılaştırmanın daha zor olacağını düşündük.

UEQ yaklaşımı, testler sırasında her zaman hazır bulunduğumuz için uygulamayı başka kişilere dağıtmak zorunda kalmadan çalışmayı yürütmeyi de kolaylaştırdı. Diğer bir yaklaşım, kullanıcıların uygulamaları daha uzun süre kullandıkları ve sonra fikirlerini verdikleri boylamsal bir test olabilirdi. Boylamsal bir testten sonra kullanıcıların farklı bir görüşe sahip olmaları mümkün olsa da, kullanıcıların anlık duygularını test etmek istediğimizi hissettik. Boylamsal test, belirli kullanıcı görevlerimiz tarafından değil, bir kullanıcının uygulamayı nasıl kullanacağıyla da kontrol edilecektir. Bu, her kullanıcı muhtemelen uygulama ile ilgili farklı bir deneyime sahip olacağından, çalışmanın tekrarlanmasını zorlaştırabilir. Çalışmayı tekrarlanabilir hale getirmeyi amaçladığımız için bu, standartlaştırılmış yaklaşımı seçmek için başka bir nedendi.

El kitabı izin verse bile UEQ'yu değiştirmemeyi seçtik. Geliştirdiğimiz uygulamamız açısından gerçekten ilginç olmayan bir ölçeği, örneğin yenilikleri hariç tutabilirdik. Ancak, sadece aynı olması gereken iki uygulamayı karşılaştırmaya çalıştığımız için, her bakımdan eşit olmaları gerektiğini hissettik. Bir sorunun ifadesini değiştirme olasılığı da vardı. En çok sorun yaşadığımız soru güvenli veya güvenli olmayan 17. soruydu. Bu çalışmayı tekrar yaparsak, muhtemelen bu sorunun ifadesini değiştirirdik. Birkaç kullanıcı, bunun nerede cevap vereceğini bilmedikleri bir soru olduğunu söyledi ve bu, düşük bir korelasyona ve güvenilirlik ölçeği için Cronbach katsayısına yol açtı.

Kullanıcı testlerinin yürütülmesi ile ilgili olarak, her bir kullanıcının belirli bir işletim sistemi için her iki sürümü de test etmesine izin vermeyi de düşündük. Bu, daha fazla kullanıcı testi yapmadan karşılaştırmak için bize daha fazla sonuç verecektir. Ancak, daha sonra bazı kullanıcıların önce yerel sürümü kullanacağını ve bazı kullanıcıların önce React Native sürümünü kullanacağını düşünmemiz gerekir. Her iki versiyona da önyargı olmadığından emin olmak için bu farklılıkların araştırılması gerekecekti. Testin zor olduğunu hisseden ve çok benzer oldukları için ikinci uygulamanın değerlendirilmesi sırasında ilk sürümün nasıl değerlendirildiğini hatırlamaya çalışan bir iOS kullanıcısı için de böyle bir test yapmayı denedik. Bu nedenle, seçilmiş yaklaşımımızı kullanarak daha dürüst sonuçlar alacağımızı hissettik. Bu ikinci yaklaşım aynı zamanda testi iki kat daha fazla zaman alır ve bu da kullanıcı adaylarını caydırabilir.

k) Daha kapsamlı perspektif

Mobil uygulamalar hayatın merkezi bir parçası haline geldi ve insanlar bunları günlük olarak ve birçok günlük etkinlik için kullanıyor. Mobil uygulamaları ve bunların kolayca erişilebilir olmasını sağlamak, bu nedenle toplum için büyük önem taşımaktadır. Profesyonellerden birden fazla platform için yerel uygulamalar satın almak genellikle pahalı olarak kabul edilir ve bu da bir yazılım geliştirme şirketi için potansiyel müşteri sayısını büyük ölçüde sınırlar. React Native gibi bir çapraz platform çerçevesi, geliştirme süresini ve dolayısıyla hem Android hem de iOS cihazları için uygun olan mobil uygulamaların maliyetini azaltabilir. Bir yazılım geliştirme şirketi React Native uygulamalarını yerel uygulamalara kıyasla yeterince iyi düşünürse, yerel uygulamaları hiç geliştirmemeyi seçebilirler. Şirketin daha önce yerel Android geliştiren bir çalışanı ve yerel iOS uygulamaları geliştiren bir çalışanı olabilir. React Native'e geçmenin ve bununla yerel geliştirmenin atılmasının acil kaygısı, önceki iki yerel uygulama geliştiricisinin geçişten sonra her ikisinin de hala bir işi olması için yeterli iş olup olmayacağıdır. React Native aslında web JavaScript çerçevesi ReactJS üzerine inşa edildiğinden, bu mantık, web geliştiricilerini de dahil etmek için bir adım daha ileri götürülebilir. Teoride, React Native ve ReactJS arasındaki benzerlikler nedeniyle bu, her iki yerel uygulama geliştiricisinin de bir web geliştiricisi tarafından değiştirilmek üzere kovulabileceği anlamına gelir. Web geliştiricisi, mobil uygulamaları geliştirmeye başlamak için kendi ReactJS bilgilerini tercüme edebilmelidir. React Native, Facebook tarafından geliştirilen açık kaynaklı bir çerçevedir. Açık kaynaklı bir çerçeve, topluluk bunu keşfetmeden kodun herhangi bir gizli amaç için kullanılmasına izin vermediğinden etik açıdan avantajlıdır. Bunun dışında geliştiriciler ve kullanıcılar uygulamalarının çalışması için asla bir şirkete bağımlı olmayacaklar. Başvuruları aracılığıyla gönderilen bilgilerin ifşa edilmediğine veya kişinin dürüstlüğüne hakaret edecek şekilde kullanılmadığına her zaman güvenebilirler.

5.4 FLUTTER

Flutter, Google'da Chrome tarayıcı ekibinin üyeleri tarafından gerçekleştirilen bir deney olarak başladı. Geleneksel düzen modelini göz ardı ederek hızlı bir rendering engine oluşturmanın mümkün olup olmadığını görmek istediler. Birkaç hafta içinde, önemli performans kazanımları elde edildi ve keşfedilen şey buydu:

• Çoğu layout nispeten basittir, örneğin: kayan bir sayfadaki metin, boyutu ve konumu yalnızca ekranın boyutuna bağlı olan sabit dikdörtgenler ve belki bazı tablolar, kayan öğeler vb.

• Çoğu layout, bir widget alt ağacı için yereldir ve bu alt ağaç tipik olarak bir layout modeli kullanır, bu nedenle bu widget'lar tarafından yalnızca az sayıda kuralın desteklenmesi gerekir. Tüm bu bilgileri inceledikten sonra, Flutter ekibi, büyük ölçüde değiştirilirse layoutun önemli ölçüde basitleştirilebileceği fikrini ortaya attı:

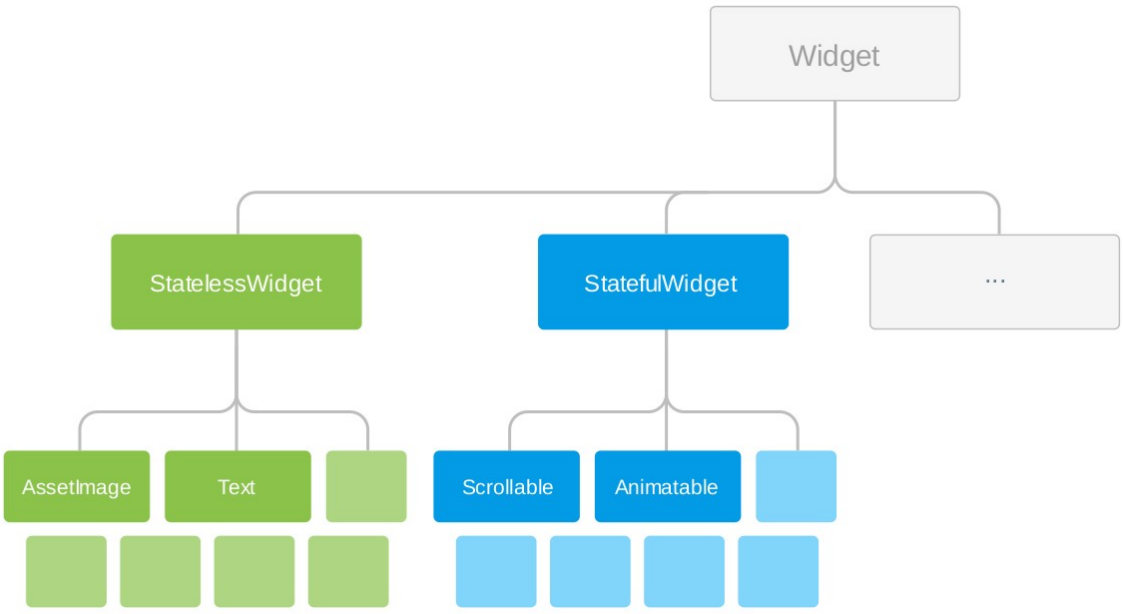
• Herhangi bir widgeta uygulanabilecek geniş bir düzen kuralları kümesine sahip olmak yerine, her widget kendi basit düzen modelini belirler.

• Her widgetın dikkate alınması gereken çok daha küçük bir düzen kuralları kümesi olduğundan, düzen büyük ölçüde optimize edilebilir.

• Düzeni daha da basitleştirmek için neredeyse her şey bir widget'a dönüştürüldü.

5.4.1 Widget'lar

Widget'lar, bir Flutter uygulaması kullanıcı arayüzünün temel yapı taşlarıdır. Her pencere öğesi, kullanıcı arayüzünün bir kısmının değişmez bir beyanıdır. Görünümleri, görünüm denetleyicileri, düzenleri ve diğer özellikleri ayıran diğer çerçevelerden veya yerel platform araçlarından farklı olarak, Flutter tutarlı, birleşik bir nesne modeline sahiptir - pencere öğesi Bir pencere öğesi şunları tanımlayabilir: • yapısal bir öğe (bir düğme veya menü gibi) • biçimsel bir öğe (yazı tipi veya renk şeması gibi) • düzenin bir yönü (dolgu gibi) •ve benzeri... Başka bir deyişle, Flutter'da her şey widget'tır! Pencere öğeleri, görünümü, uygulama hissini ve görünümünü etkileyen ve kontrol eden öğelerdir. Widget'ların bir mobil uygulamanın en önemli parçalarından biri olduğunu söylemek abartı olmaz.



Şekil 67: Flutter Widget sınıf hiyerarşisi

Widget'lar çok önemli olduğundan, çeşitli ekran boyutları da dahil olmak üzere iyi görünmeleri gerekir. Ayrıca doğal hissetmeleri gerekir. Dahası, widget'lar olabildiğince hızlı çalışmalıdır. Widget ağacının oluşturulması, widget'ların şişirilmesi (alt öğelerinin somutlaştırılması), ekranda görüntülenmesi, render edilmesi veya widget'ların canlandırılması, tüm bunların yapılması için zaman gerekir ve tutarlı 60FPS'ye sahip olmak için mümkün olduğunca düşük olmalıdır.

Modern uygulamalar için, widget'lar genişletilebilir ve özelleştirilebilir olmalıdır. Geliştiriciler, hoş yeni widget'lar ekleyebilmek ve tüm widget'ları uygulama / şirket markasına uyacak şekilde özelleştirebilmek istiyor.

Flutter, iyi görünen ve iyi hissettiren, hızlı, özelleştirilebilir ve genişletilebilir widget'lar içeren yeni bir mimariye sahiptir. Esas nokta, Flutter'in platformu veya OEM widget'larını kullanmaması, kendi widget'larını sağlamasıdır (Şekil 14).

5.4.2 Layout

Flutter'daki en büyük iyileştirmelerden biri, layoutun nasıl yapıldığıdır. Layout, bir dizi kurala göre widget'ların boyutunu ve konumunu belirler.

Geleneksel olarak, düzen, herhangi bir widgeta (sanal olarak) uygulanabilen geniş bir kurallar kümesi kullanır. Kurallar birden çok düzen yöntemi uygular. Örnek bir Android XML almak için. Tüm görünüm öğelerine uygulanan birçok özelliğe ve niteliğe sahiptir. Her widgetin kendi nitelikleri olabilir. Dahası, ana layout modelleri önceden tanımlanmıştır ve kurallara uymanız gerekir. Bu, optimizasyon için daha az alan ve kendi layoutunu yazmak sorunlu olduğundan ve buna değmeyebileceğinden çok sayıda hile ile sonuçlanır.

Geleneksel düzen ile ilgili bir başka sorun da, kuralların birbiriyle etkileşime girebilmesi (ve hatta çatışması) ve öğelerin genellikle kendilerine uygulanan düzinelerce kurala sahip olmasıdır. Bu, layoutu yavaşlatır. Daha da kötüsü, düzen performansı tipik olarak N-kare düzeyindedir, bu nedenle widgetların sayısı arttıkça düzen daha da yavaşlar.

Flutter basit ve okuyucu dostudur. Oldukça basit bir pencere öğesi ağacı aşağıda sunulmuştur.

```
Card(child: ListTile(
```

```
leading: new CachedNetworkImage(
placeholder: new Icon(Icons.attach_money),
imageUrl: currency.getImageUrl(),
),
title: new Text(currency.getText(),
style: new TextStyle(fontWeight: FontWeight.bold)),
));
```

Bu kod semantik, ne üreteceğini kolayca hayal etmek için yeterlidir, sonuç Şekil 9'da görülebilir. Bu kodda TextStyle hariç her şey bir widget'tır. Card widgeti, childını cardın içine wrap yapar. ListTile layout widgeti, children parçalarını düzenler, böylece baştaki widget solda çizilir ve ondan sonra title widgetı görüntülenir.

Flutter'da merkezleme ve dolgu widget'lardır. Temalar, çocukları için geçerli olan widget'lardır. Ve hatta uygulamalar ve gezinme widget'lardır.

Flutter, sadece sütunlar değil, aynı zamanda satırlar, ızgaralar, listeler vb. De yerleşim yapmak için epeyce widget içerir. Buna ek olarak, Flutter, kaydırma için kullanılan "şerit yerleşim modeli" olarak adlandırdığımız benzersiz bir düzen modeline sahiptir. Flutter'da düzen o kadar hızlı ki kaydırma için kullanılabilir. Bunu bir an düşünün. Kaydırma o kadar anlık ve pürüzsüz olmalıdır ki, kullanıcı fiziksel ekranda sürüklerken ekran görüntüsünün parmağına takılı olduğunu hisseder.

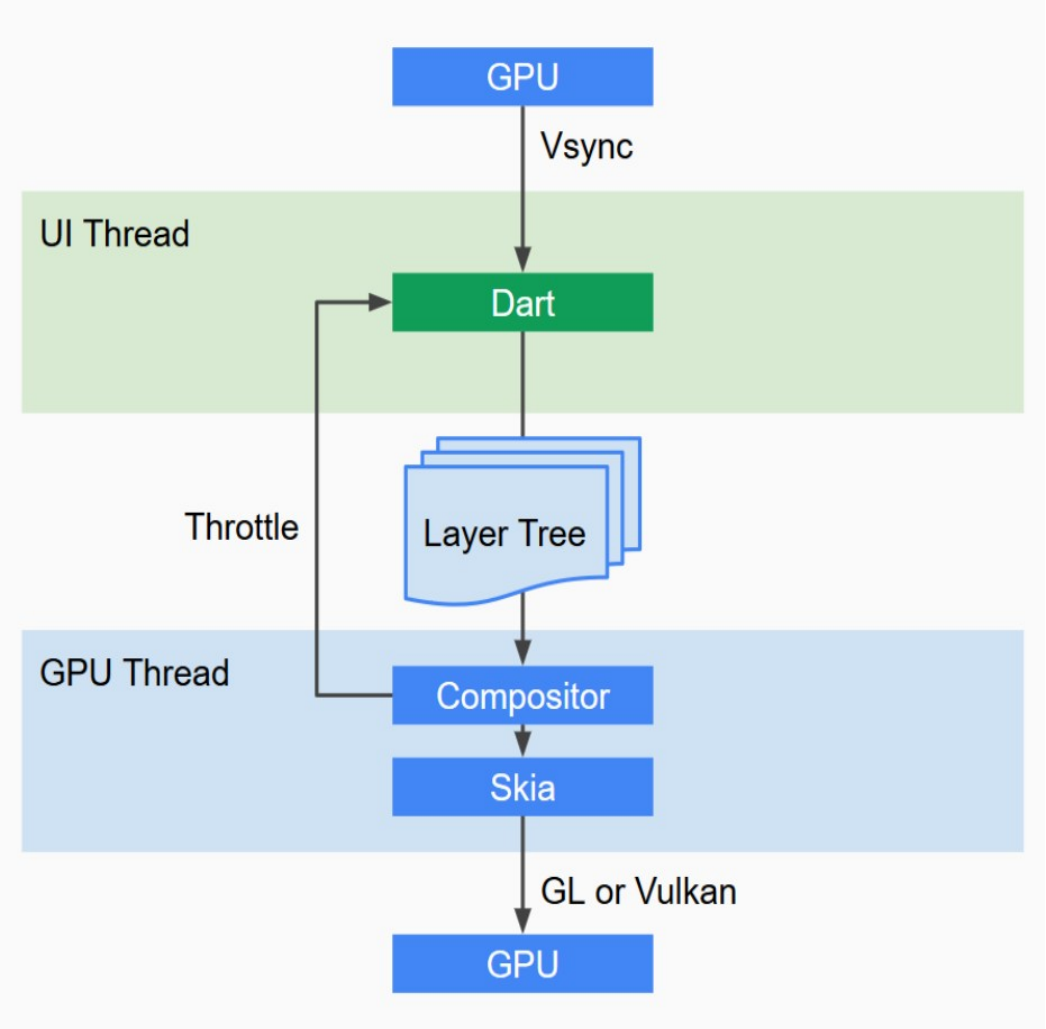
Kaydırma için düzen kullanarak, Flutter çok sayıda animasyonla gelişmiş kaydırma türleri uygulayabilir.

5.4.3 Arka Plan

Flutter, C, C ++, Dart ve Skia grafik motoru ile oluşturulmuştur (Şekil 69).

Şekil 68: Flutter framework ve engine

Flutter, bileşenleri oluşturmak için Dart'ı kullanır ve kaputun altında, kodu hayata geçirmek için Skia 2D grafik motorunu kullanır. Flutter ayrıca modern bir tepki tarzı çerçeve içerir. En düşük seviyede tüm UI kodu, uygulama UI'sını oluşturmak için Skia'yı kullanır (Şekil 70). Flutter, çerçevesinin ve uygulama kodunun çoğunu hafif bir Dart sanal makinesinde çalıştırır. Çerçeve kodu Dart'ta yazılırken, oluşturma motoru C ++ ile uygulanır.

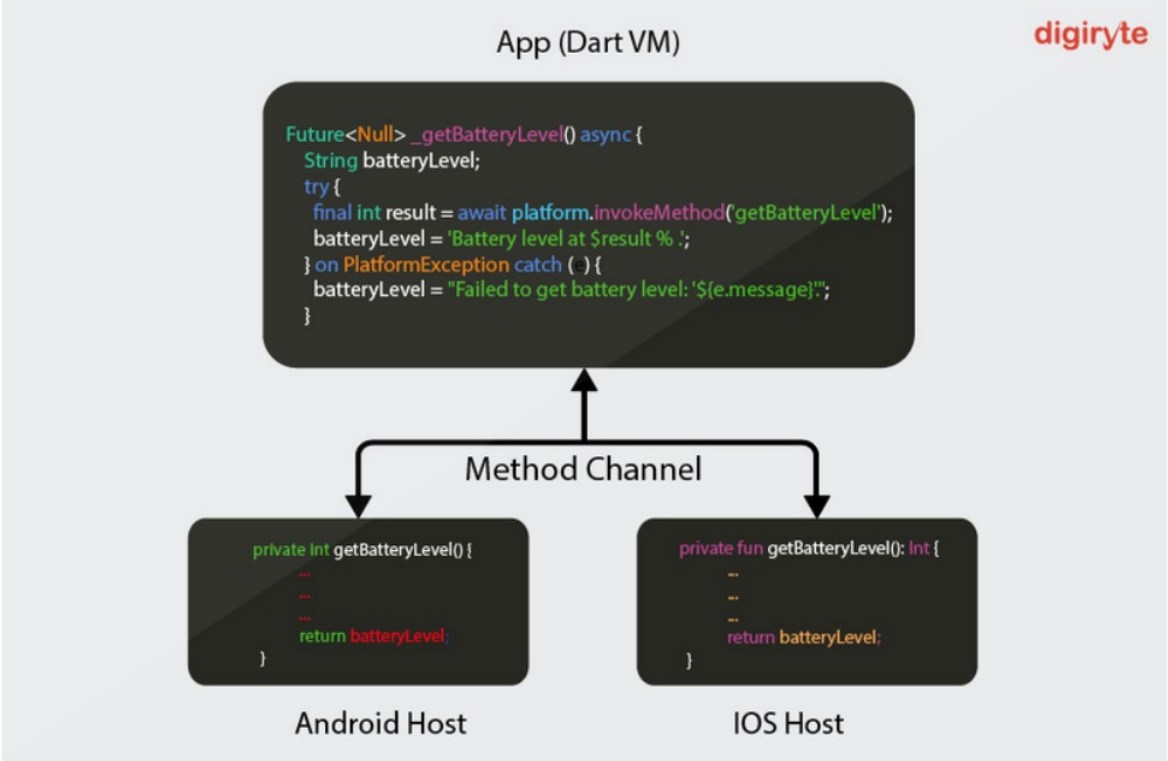


Şekil 69: Flutter drawing pipeline(grafik arayüzü çizme veri hattı)

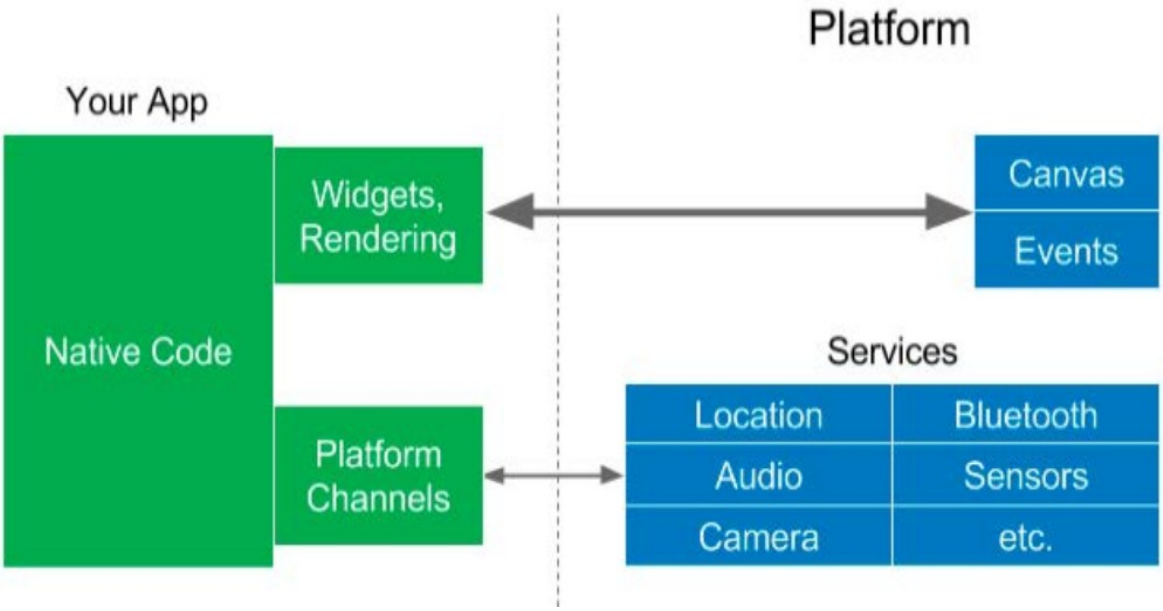
Dart kaynak kodu, Dart'ın AoT (Ahead of Time) derleme özelliği kullanılarak yerel koda derlenir. Ancak, çalışması için yine de Dart VM'ye (Sanal Makine) ihtiyaç duyar. Android'de motorun C / C ++ kodu Android'in NDK ile derlenir ve tüm Dart kodu AOT ile yerel kodda derlenir. İOS'ta, motorun C / C ++ kodu LLVM (Düşük Seviyeli Sanal Makine) ile derlenir ve tüm Dart kodu da AOT ile yerel koda derlenir. Uygulama, her iki durumda da yerel yönerge kümesini kullanarak çalışır.

Flutter, sensörler ve depolama gibi tüm platform hizmetlerine de erişebilir. Flutter, paketler aracılığıyla zaten çok sayıda platform hizmeti ve API sağlıyor. Bununla birlikte, ek yerel işlevselliğe ihtiyaç duyulursa, bir platform kanalının uygulanabileceği Flutter hizmetleri kitaplığını kullanmak mümkündür ve bu, Dart'tan platforma özgü işlevleri çağırmak için kullanılabilir ve bunun tersi de geçerlidir (Şekil 70). Örneğin, Android'de Java işlevlerine

erişmek mümkündür ve iOS'ta Objective-C işlevlerine erişim mümkündür. Flutter ayrıca yerel platform koduna çağrı yapılmasına izin veren özel eklentiler oluşturmayı da destekler (Flutter 2018.).



Şekil 70: Flutter kullanarak platform özelliklerine erişim örneği



Şekil 71: Flutter'ın platformla etkileşmesi (Wm L. 2018)

Tüm platform özelliklerine eşzamansız olarak erişilir ve bu nedenle kullanıcı arayüzünü yavaşlatmaz.

5.4.4 Android ile Karşılaştırılması

Tablo 12: Kaynak Kodu Kod Satırları, Dosya Sayısı ve Bağımlılıklar

Uygulama	Satır Sayısı	Dosyalar	Bağımlılıklar
Android Native SDK	1,87	24	4
Flutter	1,27	16	7

a) Kod satırları

Bir uygulama için yazılan kod satırı sayısı, bir yazılım parçası yazmak için harcanan karşılaştırılabilir çabayı değerlendirmek için bir ölçü sağlar. Veriler, yazılımın kullanım ömrü boyunca bakımının ne kadar kolay olacağını belirlemek için de kullanılabilir. Programlar arasındaki sayımdaki bazı farklılıklar çerçevenin kod stilinden kaynaklanabileceğinden, kod satırları mükemmel bir ölçüm değildir. Genel olarak konuşursak, bir kod tabanı diğerinden daha fazla kod satırına sahipse, yazmak için daha fazla geliştirici çabası gerektirdiği ve bakımı için daha fazla kod olduğu için uzun vadede kodu revize etmek veya güncellemek için daha fazla çaba gerektirebileceği söylenebilir.

Tablo 12'deki verilere göre, yerel Android uygulaması 1.870 satırla yazılmış en fazla kod satırına ihtiyaç duyuyordu. Bu kısmen, Android'in öğeleri kullanıcı arayüzünden uygulama koduna bağlamak için daha fazla kod satırı gerektirmesine ve Android'in düzen işlevselliğini uygulamanın her sayfası için ayrı XML dosyalarına ayırmasına bağlanabilir. Flutter, yerel platform muadillerinden daha az kod satırı gerektirmektedir.

b) Kullanıcı Tarafından Yönetilen Dosyalar

Her geliştirme çerçevesinin kendi proje yapısı vardır. Yerel Android SDK kullanılarak geliştirilen uygulamalar, Java sınıfı dosyalarına ve XML düzeni ve kaynak dosyalarına bölünür. Flutter uygulamaları, programın işlevselliğini ve düzenini Dart sınıfı dosyalarında birleştirir. Ionic/Cordova programları TypeScript sınıf dosyalarına, düzenler için HTML dosyalarına ve stil ayrıntıları için CSS dosyalarına bölünmüştür. Bir çerçeve, geliştiricilerin daha fazla sayıda dosya tutmasını gerektiriyorsa, endişelerin daha net bir şekilde ayrılmasına yol açabilir, ancak bu aynı zamanda projenin yönetilmesinin daha karmaşık olduğu anlamına da gelebilir. Bu tezin amaçları doğrultusunda, kullanıcı tarafından yönetilen bir dosya, tipik

bir geliştiricinin bir projede oluşturacağı veya düzenleyeceği bir dosyadır. İncelenen tüm geliştirme çerçeveleri, oluşturma işlemi sırasında otomatik olarak oluşturulan veya değiştirilen bir dizi dosyaya sahipti.

Tablo 12, 2 uygulama prototipinin her biri için kullanıcı tarafından yönetilen dosya sayısını içerir. Karşılaştırıldığında, Flutter projesi 16 dosya gerektiriyordu.

c) Bağımlılıklar

Çok sayıda dış bağımlılığa dayanan uygulamaların uzun vadede sürdürülmesi potansiyel olarak daha zor olabilir. Yazılım kitaplıkları gelişmeye devam ettikçe, bunları kullanan uygulamalarda hatalara, hatalara veya uyumsuzluklara neden olan değişiklikler ortaya çıkabilir. Ayrıca, harici kitaplıkların geliştiricinin farkında olmadığı güvenlik açıklarını ortaya çıkarma potansiyeli de vardır. Bu deney, her Görev İstasyonu prototipinin kod tabanlarının gerekli bağımlılıklarının sayısının bir değerlendirmesini içeriyordu.

Task Station'ın her sürümü için bağımlılık sayısı Tablo 1'de bulunabilir. Yerel Android ve iOS uygulamaları çoğunlukla yalnızca Google Firebase kitaplıklarının kod tabanlarına entegre edilmesini gerektiriyordu. Flutter, Firebase kitaplıklarına ve cihaz kamera erişimi için "image_picker" eklentisi, cihaz dosya sistemi erişimi için "path_provider" eklentisi ve tarihlerle çalışmak için "intl" eklentisi dahil olmak üzere birkaç eklentiye ihtiyaç duyuyordu.

Kaynak kodu değerlendirmeleri, geliştirme çerçevesi seçiminin bir mobil uygulamanın geliştirme ve bakım deneyimi üzerinde ne gibi bir etkisi olduğu sorusunu yanıtlamaya yardımcı olacak bilgiler sağladı. Flutter, ayrı yerel uygulamalar yazmaktan daha az kod satırı gerektiriyordu. Platformlar arası uygulamalar ayrıca daha az dosyanın yönetimini gerektiriyordu. Flutter uygulaması, yerel uygulamalardan birkaç ek bağımlılık gerektiriyordu.

d) Uygulama Özellikleri ve Performans Profili Oluşturma

Uygulama performans profili değerlendirmelerinin amacı, bir geliştirme çerçevesi seçiminin, oluşturulan uygulamanın performansı ve algılanan kalitesi üzerinde ne gibi bir etkisi olduğu sorusunun yanıtlanmasına yardımcı olmaktı. Kaynak kodu değerlendirme sonuçları, platformlar arası bir geliştirme çerçevesinin kullanılmasının, birden çok platform için yazılım oluşturmak ve sürdürmek için gereken çabayı önemli ölçüde azaltabileceğini göstermektedir. Bu azaltılmış çaba, dağıtılan uygulamanın kalitesinde bir ödünleşimle mi geliyor? Yazılımın

performansını değerlendirmek için gerçekleştirilebilecek bir dizi kıyaslama testi vardır. Bu deneyde gerçekleştirilen testler, bir uygulamanın derlenmiş paket boyutuna, kurulu bir uygulamanın bellek ayak izine, çalışan bir uygulamanın ömrü boyunca iki farklı noktada RAM kullanımına ve bir uygulamanın başlatılması için geçen zamana bir bakış içerir.

e) Uygulama Paketi Boyutu ve Kurulu Ayak İzi

Flutter, Android cihazlar için uygulama yazmak için gereken kod miktarını azalttı. Kullanıcı tarafından yönetilen kod tabanları daha küçük olabilir, ancak dağıtılan uygulamalara ne kadar ek yük dahildir? Flutter uygulamaları, uygulamaların kullanıcı arayüzlerini oluşturmak için Flutter motorunu içerir. Flutter, dosya sistemi veya aygıtların donanımı ile etkileşimi yönetmek için kitaplıkların veya eklentilerin eklenmesini gerektirir. Bu ek yükün tümü, derlenen uygulamaların boyutunu potansiyel olarak etkileyebilir. Genel olarak, kullanıcılar indirilmeleri daha az zaman aldığından ve kurulduğunda daha küçük bir cihaz belleği ayak izi gerektirdiğinden daha küçük paketlere sahip uygulamaları tercih eder.

Tablo 3 ve Tablo 4, Task Station'ın dağıtılan Android ve iOS sürümlerinin paket boyutunu ve kurulu boyutunu görüntüler. Flutter uygulamaları daha büyük paketlere sahipti ve yerel platform uygulamalarından daha fazla yüklemek için daha fazla bellek gerektiriyordu. Flutter SDK'nın bir ek yükü vardır ve Flutter motorunun bildirilebilir bir etkisi vardır. Native paketinin boyutu bir aykırı değer gibi görünüyor ve bu, projeye dahil edilen Firebase pod dosyalarının büyük boyutundan kaynaklanıyor olabilir.

Tablo 13: Android Uygulama Paket Boyutu ve Yüklenmiş Boyutu

Uygulama	Paket (MB)	Paket (MB)
Android Native	5.70	5.69
Android Flutter	16.9	41.29

f) Bellek Kullanımı Profili Oluşturma

Paket boyutu ve kurulum belleği ayak izi değerlendirmeleri, bir geliştirme çerçevesinin konuşlandırılmış uygulamaların özellikleri üzerindeki etkisi hakkında bazı bilgiler sağladı. Bir platformlar arası çerçevenin bellek yükünün bir miktar ölçümü, potansiyel olarak bu değerlendirmeler yoluyla yapılabilir, ancak aynı zamanda, çalışan bir uygulamanın ömrü boyunca gereken bellek miktarıyla da ölçülebilir. Her bir uygulama türünün RAM kullanımını ölçmeye çalışmak için, uygulama ömrü boyunca iki noktada platforma özel profil oluşturma araçları kullanılarak ölçümler yapılmıştır. İlk nokta, uygulamanın başlatılmasının

tamamlanmasından ve menü sayfasının kullanıcı etkileşimi için hazır olmasından sonraydı. İkinci nokta, uygulama görevleri görüntüleme sayfasına geçtikten ve görev listesi yüklendikten sonraydı. Ölçüm alma işlemi toplam üç kez tekrarlanmıştır. Bu ölçümler, Flutter uygulamalarının performans yüküyle ilgili daha fazla ayrıntı sağladı.

Android uygulamaları için RAM kullanım sonuçları Tablo 5 ve Tablo 6'da bulunabilir. En belirgin gözlem Flutter uygulamalarının yerel android uygulamasına göre daha yüksek RAM kullanımına sahip olmasıdır.

Tablo 14: Android Uygulama Menüsü RAM Kullanımı

Uygulama	Deneme#1 (MB)	Deneme#2 (MB)	Deneme#3 (MB)
Android Native	97	99	88
Android Flutter	150	143	139

Tablo 15: Android Uygulama Görünümü Görevleri RAM Kullanımı

Uygulama	Deneme #1 (MB)	Deneme#2 (MB)	Deneme#3 (MB)
Android Native	115	101	103
Android Flutter	176	174	180

Android uygulamaları için RAM kullanım verilerini inceledikten sonra, platformlar arası çerçevelerin ek yükünün bellek gereksinimleri üzerinde gözle görülür bir etkisi olduğu görülüyor. Bu değerlendirmenin sonuçları, platformlar arası uygulamaların etkinken daha yüksek bellek kullanımına sahip olduğu önceki diğer çalışmaların bulgularıyla tutarlıydı. Android için Flutter prototipleri en yüksek miktarda bellek gerektirmektedir. Genel olarak, Flutter uygulama gereksinimlerinin daha yüksek iş yükleriyle (işlenecek daha fazla widget veya işlenecek veri gibi) arttığı görülüyor. Task Station prototipleri veri yoğun uygulamalar değildir, bu nedenle aynı tür değerlendirmeler yapılırsa diğer uygulama türlerinin farklı davranması olasıdır.

g) Uygulama Başlama Süresi ve Geçiş Zamanını Görüntüle

Bir uygulamanın kullanıcı açısından algılanan kalite açısından en önemli yönlerinden biri, uygulamanın yanıt verebilirliğidir. Uygulama hızlı yükleniyor mu? Kullanıcı uygulamanın arayüzüyle etkileşime girdiğinde uygulama hemen yanıt veriyor mu? Mobil uygulama

kullanıcıları, bir uygulamanın etkileşime anında yanıt vermesini beklemekte ve bu beklentiyi karşılamayan uygulamaları kalite açısından daha düşük olarak algılamaktadır [6].

Task Station'ın her bir sürümünün yanıt verebilirliğini ölçmeye çalışmak için üç farklı değerlendirme yapıldı. İlk değerlendirme, kullanıcının cihazındaki uygulamanın simgesine dokunduğu an ile uygulamanın menü sayfasının tam olarak oluşturulduğu an arasında geçen sürenin bir ölçümüydü. İkinci değerlendirme, kullanıcının menü sayfasındaki görev ekle düğmesine dokunması ile görev ekle sayfasının tam olarak oluşturulması arasında geçen süreyi ölçmüştür. Son test, kullanıcının uygulamanın menü sayfasına geri dönmesini istediği nokta ile menü sayfasının tekrar tamamen görünür olduğu nokta arasındaki süreyi ölçtü.
Android uygulamaları için uygulama başlatma süreleri Tablo 16'da ve Android için geçiş süreleri Tablo 17'da bulunabilir. Flutter uygulaması, yerel Android uygulamasından sürekli olarak biraz daha hızlı başlatıldı. Sayfa geçiş süreleri açısından, Flutter sürümü, görev ekleme sayfasını yerel sürümden ancak yalnızca küçük bir farkla görüntüledi. Tüm Android uygulamaları için yanıt süresi sürekli olarak yarım saniye civarında olduğundan, geriye doğru gezinme süreleri burada bir tabloda görüntülenmez. Genel olarak, Flutter Android sürümü en hızlı başladı ve en kısa sürede görev ekleme sayfasına gitti.

Tablo 16: Android App başlatma süreleri (s)

Uygulama	Deneme #1 (s)	Deneme #2 (s)	Deneme #3 (s)
Android Native	1.3	1.9	1.5
Android Flutter	1.2	1.0	1.1

Tablo 17: Android App uygulama geçiş süreleri (s)

Uygulama	Deneme #1 (s)	Deneme #2 (s)	Deneme #3 (s)
Android Native	0.7	0.8	0.7
Android Flutter	0.5	0.4	0.5

Başlangıç, geçiş ve geriye doğru gezinme testlerinin sonuçları, Android ve iOS cihaz testleri arasında tutarlıydı. Flutter sürümleri en hızlı ve en duyarlı sürümlerdi. Hibrit uygulamaların başlatma ve yanıt süresi davranışı, önceki araştırmalarla tutarlıydı.

Tüm uygulama özellikleri ve performans profilleme değerlendirmeleri için veriler göz önüne alındığında, daha yüksek cihaz belleği veya RAM gereksinimlerine karşılık gelen bir yanıt süresi cezasının zorunlu olarak olduğu görülmemektedir. Flutter uygulamaları en büyük yüklü boyutlara ve en yüksek bellek gereksinimlerine sahipti, ancak aynı zamanda Android'den daha hızlı başlatıldı ve görünümler arasında daha hızlı geçiş yapıldı. Bir uygulamanın yanıt verebilirliği, bir kullanıcının bir uygulamanın kalitesini nasıl algıladığı açısından önemli bir faktördür ve Flutter sürümlerinin bu kategoride sürekli olarak daha iyi performans gösterdiği ortaya çıktı. Bir uygulamanın performansı, bir mobil uygulamanın kalitesini belirleyen tek faktör değildir, kullanıcı arabirimi öğeleri ve stil gibi çeşitli niteliksel alanlar bu bölümün sonraki bölümünde ayrıntılı olarak incelenecektir.

h) Kullanıcı arayüzü tasarımı

Mobil uygulama geliştirmenin en çok zaman alan yönlerinden biri, uygulamanın kullanıcı arayüzünü tasarlamak ve sürdürmektir. Yönetilmesi gereken birçok önemli ayrıntı vardır ve bir geliştirme çerçevesi bunu yapmak için kullanıcı dostu bir yöntem sağlamalıdır. Google'ın Flutter ve Ionic çerçeveli Apache Cordova, kullanıcı arayüzü tasarımını çok farklı şekilde ele alıyor.

Flutter, hem bir sayfanın düzenini hem de davranışlarını aynı Dart sınıfı dosyasında düzenler. Arayüz, sayfanın kendisi de dahil olmak üzere arayüzün her öğesinin bir pencere öğesi olduğu iç içe bir widget ağacı olarak tanımlanır. Widget'lar durum bilgisi olan veya olmayan olarak sınıflandırılır. Sayfanın durumu değiştiyse, durum bilgisi olan bir widget yeniden oluşturulacaktır. Durum bilgisi olmayan bir widget, etkin sayfa başlangıçta çizildiğinde oluşturulur ve sayfanın ömrü boyunca değişmez. Bir Flutter widget ağacı örneği Şekil 12'de gösterilmektedir.

Flutter'daki bir sayfanın düzeni, sayfanın build function'ında tanımlanır. Build function, sayfanın düzenini oluşturan tüm widget'ları içeren iç içe geçmiş ağacı döndürür. Yukarıdaki örnek, Task Station Flutter prototipinin menü sayfasının düzenini göstermektedir. Ağacın ana parçacığı, tek bir Sütun içeren bir Scaffold widgetidir. Sütun, Düğme, Metin veya Görüntü

pencere öğelerini saran birkaç container widget'i barındırır. Her bir widget türünün kendi özellikleri vardır, ancak bunlar ayrıca sarılabilirler.

Kenar boşluğu ve dolgu bilgileri gibi ek özellikler eklemek için Kapsayıcı veya Dolgu widget'larında. İşlev çağrıları, sayfanın mevcut durumuna göre hangi pencere öğelerinin görüntüleneceğini yönetmek için ağacın öğeleri olarak kullanılabilir. Sayfayla ilgili veriler değiştiğinde, verilere bağlı tüm widget'ların yeniden oluşturulabilmesi için sayfanın durumu güncellenmelidir.

```
Widget build(BuildContext context) {
 return Scaffold(
  appBar: AppBar(
   title: Text(widget.title),
  ),
  body: Column(
   mainAxisAlignment: MainAxisAlignment.start,
   crossAxisAlignment: CrossAxisAlignment.center,
   children: <Widget>[
    Container(
       margin: const EdgeInsets.only(top: 20.0, bottom: 10.0),
       child: Image.asset('images/taskstationlogo.png'))
    ,
    Container(
       margin: const EdgeInsets.only(top: 10.0, bottom: 10.0),
       child: RaisedButton(
          child: Text("View Tasks"),
          onPressed: user != null ? (){
            Navigator.push(context, MaterialPageRoute(
               builder: (context) => ViewTasksPage(user: user), settings: RouteSettings(name: "/viewtasks")));
          }
          : null))
    ,
    Container(
       margin: const EdgeInsets.only(top: 10.0, bottom: 10.0),
       child: RaisedButton(
          child: Text("Add Task"),
          onPressed: user != null ? (){
             Navigator.push(context, MaterialPageRoute(
                builder: (context) => AddTaskPage(addMode: true, taskClass: null, user: user)));
           }
           : null))
    ,
    _authButtons(),
    user != null ? Container(
           margin: const EdgeInsets.only(top: 10.0, bottom: 10.0),
           child: Text(user.displayName)) : Container()
     ],
    ),
 );
}
```

Şekil 72: Flutter widget ağacı örneği

Flutter'ın arayüz tasarımına yaklaşımı, yeni geliştiriciler için ilk başta sezgisel olmayabilir, ancak Flutter'ın canlı yeniden yükleme yetenekleriyle eşleştirildiğinde, ağaçla deneme yapmak ve anında güncellenmiş bir düzeni görmek kolaydır. Düzenler büyüdükçe ve daha karmaşık hale geldikçe, bir sayfanın tüm öğelerini organize bir şekilde yönetmek zorlaşabilir, ancak özel widget'ların kullanımı yapının daha iyi sağlanmasına yardımcı olabilir. Flutter widget'ları, kalıtım yerine kompozisyon kavramına odaklanarak tasarlandı, diğer widget

gruplarından oluşan özel widget'ların kullanımı, bir geliştiricinin arayüz tasarımına modüler bir yaklaşım benimsemesini sağlar.

Hangi çerçevenin daha iyi bir kullanıcı arayüzü tasarımı deneyimi sağlayacağının seçimi, geliştiricinin teknik geçmişine ve programlama dili tercihine bağlıdır. Flutter yaklaşımı, Android uygulamaları için arayüz tasarlamak için kullanılan XML yerleşim sistemine benzer görünebilir.

j) Öğrenme eğrisi

Bir geliştirme çerçevesinin en önemli yönlerinden biri, çerçevenin nasıl kullanılacağını hızlı bir şekilde öğrenme ve mobil yazılım oluşturmaya başlama yeteneğidir. Bir çerçeveyi öğrenmek ne kadar uzun sürerse, insan çabası, maaşlar ve potansiyel kayıp gelir açısından bir geliştirme ekibine o kadar pahalıya mal olur. Bir çerçevenin belgelerinin anlaşılması zorsa veya çerçeve birden fazla destekleyici teknolojinin kullanılmasını gerektiriyorsa, geliştiriciler için daha az çekici olabilir. Daha yüksek öğrenme eğrisine sahip bir çerçeve, uygulamaların tasarımı sırasında ortaya çıkan sorunlara çözüm bulmayı da zorlaştırabilir. Task Station prototiplerinin geliştirilmesi, Cordova/Ionic çerçevesinin ve Flutter'ın öğrenme eğrileri hakkında birçok fikir verdi. Programlama dillerine ve her çerçeve için ortak test araçlarına genel bir bakış Tablo 13'te bulunabilir.

Tablo 18: Diller ve Test araçları

Framework	Front-End Dilleri	Back-End Dilleri	Testing Araçları
Native Android	XML	Java / Kotlin	JUnit Espresso
Native iOS	XML	Objective-C / Swift	XCTest XCUITest
Flutter	Dart	Dart	Dart Test Package Flutter_Test Package Flutter WidgetTest

Flutter için gereken tek programlama dili Google'ın Dart'tır. Flutter eklentileri tek bir depoda bulunabilir ve Ionic çerçevesinde bulunanlar gibi sarmalayıcı eklentilerin kullanımını

gerektirmez. Flutter'ın hem kullanıcı arabirimi öğelerine hem de SDK'nın geliştirme özelliklerine ilişkin belgeler aynı web sitesinde bulunur. Flutter'ın tek bir programlama diline dayanması, daha az harici teknoloji gerektirmesi ve diğer Google ürünleriyle birlikte geliştirilmesi gerçeği, başlamanın ve sorunlara çözüm bulmanın çok daha kolay olduğu anlamına geliyor. Flutter, bu makalenin yazıldığı sırada henüz çok genç bir çerçeveydi, ancak Task Station prototiplerinin geliştirilmesi sırasında ortaya çıkan sorunlara hızlı bir şekilde yanıt bulmak zor değildi. Flutter uygulamaları, platform cihazı özellikleri açısından yerel platform uygulamalarına karşılaştırılabilir erişime sahiptir. Bu, Android ve/veya iOS programlama geçmişine sahip geliştiricilerin Flutter'ı öğrenmeyi ve kullanmayı daha kolay bulacağı anlamına gelir.

k) Hata Ayıklama Stratejileri ve Araçları

Her yazılım projesi kaçınılmaz olarak hatalar ve programlama hataları içerecektir. Bu nedenle hata ayıklama, yazılım tasarım ve bakım süreçlerinin son derece önemli bir parçasıdır. Flutter SDK, mobil uygulama geliştiricilerine projelerinde hata ayıklamak için farklı stratejiler sağlar.

Flutter SDK kullanılarak geliştirilen yazılım, çerçeveye dahil olan Dart observatory aracı kullanılarak hata ayıklanabilir. Flutter hata ayıklama araçları, Android Studio ve Xcode hata ayıklama araçları tarafından sağlanan özelliklerle karşılaştırılabilir özellikler sağlar. Bu, yalnızca bir geliştiricinin öğrenmesi için gereken araç sayısını basitleştirmekle kalmaz, aynı zamanda bir proje geliştirirken ortaya çıkan sorunları izlemek için aynı aracın kullanılabileceği ve aracın Dart tabanlı uygulamalarda hata ayıklamak için özel olarak tasarlandığı anlamına gelir.

l) Canlı Yeniden Yükleme(Hot Reloading)

Bir uygulamada hata ayıklama veya test etme işlemi sırasında, genellikle düzenler veya kaynak kodunda değişiklik yapmak ve ardından değişikliklerin istenen etkiye sahip olup olmadığını belirlemek gerekir. Uygulamanın durdurulması, yeniden derlenmesi ve yeniden başlatılması gerekiyorsa, oluşturma süreleri değişebileceğinden ve uygulamanın durumu kaybolduğundan ve yeniden oluşturulması gerektiğinden bu işlem sıkıcı olabilir. Mobil geliştirme araçlarının en yeni özelliklerinden biri, uygulamanın durumunu kaybetmeden veya projeyi yeniden derlemek zorunda kalmadan bir uygulamanın görünümü veya davranışındaki değişiklikleri canlı olarak yeniden yükleme yeteneğidir.

Flutter'ın en etkileyici yönlerinden biri, çerçevenin sıcak yeniden yükleme özelliğidir. Çalışırken yeniden yükleme, geliştiricinin bir uygulamanın Dart kaynak kodunda değişiklik yapmasına ve çalışan programdaki etkileri anında görmesine olanak tanır. Bu özellik, widget ağacındaki her değişiklik anında cihazda görülebildiği için bina yerleşimlerini çok daha kolay hale getirir. Flutter uygulamalarının hata ayıklama yapıları Dart Sanal Makinesi kullanılarak cihazlarda çalıştırıldığından, çalışırken yeniden yükleme mümkündür. Çalışırken yeniden yükleme talep edildikten sonra, kaynak kodu değişiklikleri analiz edilir ve sanal makineye enjekte edilir ve değişiklikler çoğu durumda uygulamanın yeniden başlatılmasına gerek kalmadan uygulanır. Flutter'ın çalışırken yeniden yükleme özelliği, mobil yazılımın geliştirme ve test süreçlerini önemli ölçüde kısaltabilir.

Flutter'ın araçları, yerel Android ve iOS geliştirme çerçevelerinin bir parçası olarak mevcut olan canlı yeniden yükleme araçlarını bile geride bırakır. Mobil yazılımı test ederken hızlı bir şekilde değişiklik yapma yeteneği, genel geliştirme deneyimi üzerinde büyük bir etkiye sahip olabilir ve Flutter çerçevesi, bu yeteneği diğer geliştirme çerçeveleri arasında öne çıkaracak şekilde sağlar.

m) Cihaz Dosya Sistemi ve Donanım Erişimi

Pek çok mobil uygulama, çok basit olmadıkça, dosya sistemi, kamera, ağ bağlantısı, GPS veya ivmeölçer gibi cihaz özelliklerine erişim gerektirir. Yerel platform SDK'ları, bu özelliklere tam erişim sağlar. Bir geliştirici, platformlar arası geliştirme çerçevesi kullanarak birden çok platformu hedeflemek istiyorsa, çerçeve, bu özellikleri tüm hedef cihazlarda kullanmak için basit bir uygulama programlama arabirimi (API) sağlamalıdır.

Task Station, cihaz kamerasının kullanımını, cihazın ağ bağlantısını ve yerel dosya sistemine erişimi gerektiriyordu. Hem Cordova/Ionic hem de Flutter, eklentilerin kullanımı yoluyla bu özelliklere erişim sağladı. Eklentiler, bir geliştiricinin, derleme işlemi sırasında uygun platform kodunun dahil edileceğini bilerek, cihaz donanımıyla ilgili kod yazmasına izin verir. Çoğunlukla, cihaz özelliği eklentilerinin davranışı her iki çerçeve için de karşılaştırılabilirdi, ancak İyonik çerçevenin birkaç ek komplikasyonu vardı.

Flutter bu deney için gerekli olan tüm cihaz özellikleri için API'ler sağladı, ancak Flutter daha basit bir geliştirme deneyimi sağladı. Flutter, uygulamaları tamamen yerel uygulamalarda

derlendiğinden, gömülü web uygulamalarının kullanımına kıyasla daha az komplikasyon vardı. Genel olarak, Flutter'ın cihaz özelliklerine erişimi daha sorunsuzdu ve daha az engel içeriyordu.

n) Google Firebase Desteği

Google Firebase paketi [32], mobil uygulama geliştiricileri için bir dizi faydalı araç sağlar. Mayıs 2018 itibariyle, Firebase SDK'sını [33] yaklaşık 1,2 milyon uygulama kullanıyordu. Task Station, görev bilgilerini depolamak ve yerel veriler ile bulut verileri arasında otomatik olarak eşitlemek için Cloud Firestore'a güvenir. Proje ayrıca görevlerle ilişkili görüntüleri depolamak için Cloud Storage'ı kullanır. Firebase Kimlik Doğrulama modülü, kullanıcıların çeşitli cihazlarda uygulamada oturum açmasına olanak tanıyan bir açılır kullanıcı kimlik doğrulama arabirimi sağlar. Firebase modülleri, yerel Android ve Flutter'ı tam olarak destekler.

Flutter, Firebase paketini de geliştiren Google tarafından geliştirilmekte olan bir projedir. Yerel cihaz desteği, Flutter Paketleri deposunda bulunan birinci taraf eklentilerin kullanımıyla sağlanır. Bu denemenin tamamlandığı sırada eklentilerin tümü geliştirmenin ilk aşamalarındaydı ve mevcut tüm Firebase özellikleri için tam destek sağlamıyordu. Eklentiler, henüz erken beta aşamasında olmalarına rağmen, yerel platform uygulamalarının kullandığı tüm özellikler için benzer destek sağladı.

Firebase, mobil geliştirme için kesinlikle gerekli bir araç değildir, ancak geliştiricilere veri depolama modellerini, kullanıcı kimlik doğrulamasını ve analitik araçlarını hızla kurmak için bir dizi araç sağlar. Genel olarak Flutter, yerel platform API'lerinde bulunan özelliklerin aynısını kullanma yeteneği sağladı. Firebase ve Flutter aynı şirketin ürünleri olduğundan, Flutter'ın Firebase desteğinin zaman içinde gelişmeye devam etmesi muhtemeldir.

o) Lisanslama ve Maliyet

Mobil geliştirme çerçeveleri pazarı düzinelerce seçenekten oluşur. Bazı çerçeveler ücretsiz olarak kullanılabilir ve kaynak kodları halka açık depolarda bulunur. Diğer çerçeveler ücretli lisanslar gerektirir ve uygun teknolojiler kullanılarak oluşturulur. İdeal geliştirme çerçevesinin, bir geliştirme ekibinin gereksinimlerine ve kaynaklarına uyması gerekir. Bu deneyin bir parçası olarak değerlendirilen iki çerçeve, ücretsiz olarak kullanılabilen açık kaynaklı projelerdir.

p) Uygulama Açılış Ekranları ve Simge Desteği

Herhangi bir mobil uygulamanın ilk izlenimlerinin anahtarı olan iki önemli yönü, uygulamanın simgeleri ve açılış ekranıdır. Desteklenecek birden çok mobil cihaz türü ve çeşitli boyut ve yapılandırmalar vardır. Bu, bu unsurları yönetme sürecini bir meydan okuma haline getirebilir. Flutter, açılış ekranının ve simgelerin tasarımını ve yönetimini farklı şekillerde ele alır.

Flutter, bir projenin simgelerini veya açılış ekranlarını yönetmede gerçekten herhangi bir doğrudan destek sağlamaz. Bir projeye (Android veya iOS gibi) bir platform eklendiğinde, Flutter SDK projeye platforma özel dosyalardan oluşan bir klasör ekler. Bir Flutter projesinin açılış ekranı ve simgelerinin yönetimi, yerel platform SDK'larını kullanmakla aynı şekilde işlenir. Geliştirici, bunları projenin platform klasörlerinde her platform için ayrı ayrı yapılandırmaktan sorumludur.

q) Uygulama Derleme ve Dağıtım

Mobil yazılım yazma, hata ayıklama ve çalıştırma yeteneği, herhangi bir geliştirme çerçevesinin çok önemli bir parçasıdır, ancak herhangi bir projenin nihai hedefi, uygulamayı platform cihazlarına ve pazar yerlerine dağıtmaktır. İdeal çerçeve, platform uygulama paketlerini kolayca üretme yeteneğine izin vermelidir. Flutter bunu kolayca yapabilir.

Her iki çerçeve de dağıtım için uygulamaları paketlemek ve imzalamak için yerel Android ve iOS SDK'larına güvenir. Platformlar arası projeler için, platforma özel proje dosyalarını yapılandırmak ve uygulama imzalama profillerini veya anahtar depolarını ayarlamak gereklidir. Bu işlem her iki çerçeve için de aynıdır. Uygun imzalama koşulları yerine getirildikten ve yerel platform bildirimleri uygun şekilde yapılandırıldıktan sonra Flutter ve Ionic komut satırı arabirimleri veya IDE eklentileri, uygulama paketi dağıtımını basit komutlarla yönetebilir.

5.4.5 Karşılaştırma Sonucu

Herhangi bir mobil uygulama geliştiricisinin istek listesi, tek bir kod tabanı ile birden fazla platform için yazılım yazabilme ve yazılımı kolayca paketleyip dağıtma becerisini içerir. Flutter, bu istekleri karşılayabiliyor ve hem Android hem de iOS platformlarını desteklemek için gereken çabayı önemli ölçüde azaltıyor. Her iki platformlar arası çerçevenin kullanılması, ayrı yerel platform uygulamaları üretmekten daha az kod yazılması, daha az dosyanın

yönetilmesi ve daha az geliştirme süresi gerektiriyordu. Bu gerçeği göz önünde bulundurarak bile, her çerçeve farklı bir geliştirme deneyimi sağladı ve oluşturulan uygulamalar bir takım farklı özelliklere sahipti.

Her iki çerçeve de platformlar arası uygulama geliştirme sürecini basitleştirirken, Flutter daha iyi bir genel geliştirme deneyimi sağladı. Öğrenilecek daha az teknoloji var ve belgeler tek bir yerde bulunabilir. Daha az gerekli dış bağımlılık, yönetimi ve bakımı daha basit projelerle sonuçlanır. Flutter'ın çalışırken yeniden yükleme özelliği, hızlı kullanıcı arayüzü tasarımına olanak tanır ve birinci şahıs eklentilerinin kullanılabilirliği, Firebase'i bir projeye entegre etmeyi oldukça zahmetsiz hale getirir.

Flutter, yerel uygulamalara kıyasla boyut olarak daha büyük ve aktif ömürleri boyunca daha fazla bellek gerektiren uygulamalar üretti.

BÖLÜM 6: KAPANIŞ

6.1 GELECEKTEKİ ARAŞTIRMALAR İÇİN SINIRLAMALAR VE ÖNERİLER

Herhangi bir araştırma projesinde olduğu gibi, bu tezde de bulguların bir takım sınırlamaları vardır. Araştırma sorularını daha ayrıntılı ve daha kesin olarak cevaplamak için bulguları genişletmek için daha fazla araştırma yapılabilir. Bu bölümde, bu sınırlamalardan bazıları, gelecekteki araştırma alanları için önerilerle birlikte detaylandırılacaktır.

Bu deneyde yalnızca bir tür uygulama, veriye dayalı görev yönetimi uygulaması geliştirildi ve değerlendirildi. Bir dizi farklı uygulama türü vardır ve her bir türün gereksinimleri, onları farklı geliştirme stratejileri ve araçları için daha uygun hale getirebilir. Gelecekteki araştırmalar için bir alan, çeşitli uygulama türlerini içerecek şekilde deneyi genişletmek olacaktır. Örneğin, animasyonlar, yoğun grafik kullanımı veya yoğun işlemci veya ağ iş yükleri içeren bir oyun veya başka bir uygulama, farklı bir geliştirme stratejisinden büyük olasılıkla faydalanacaktır.

Bu belgedeki kaynak kodu değerlendirmeleri yalnızca kod satırlarını, dosya sayısını ve dış bağımlılıkları saymaktan oluşuyordu. Gelecekteki değerlendirmelere dahil edilebilecek ek kaynak kodu ölçümleri vardır. Örneğin, bir uygulamadaki yöntemlerin döngüsel karmaşıklığı, uygulamayı test etmenin zorluğunu tahmin etmeye yardımcı olabilir. Kaynak kodun bir bölümündeki lineer bağımsız yolların sayısı, tatmin edici kod kapsamına ulaşmak için gerekli birim testlerinin sayısını belirlemeye yardımcı olur. Kod satırlarının ölçütü oldukça yüzeyseldir ancak karmaşıklıkla ilişkili olduğu bulunmuştur. Her prototipin kaynak kodunun daha fazla değerlendirilmesi, her çerçevenin geliştirme ve bakım çalışmaları üzerindeki etkisi hakkında ek bilgi sağlamaya yardımcı olabilir.

Dhillon ve Mahmoud'un çalışmalarında belirttiği gibi, birden çok platform için yazılmış uygulamaları birden çok çerçeve kullanarak kıyaslamak ve değerlendirmek, tutarlı bir değerlendirme stratejisinin mevcut olmayabileceği gerçeği nedeniyle zordur. Bu belgedeki araştırma, her bir prototipin uygulama yaşam döngüsüne ve yerel kayıt sistemlerine farklı erişimlere sahip olması gerçeğiyle sınırlıdır. Hem Android hem de iOS ayrıca farklı donanım ve cihazlarda çalışır, bu nedenle farklı prototipleri platformlar arasında tutarlı bir şekilde değerlendirmek imkansızdır. Bu nedenle, nicel değerlendirmelerin sonuçlarını Android ve iOS platformlarında karşılaştırmamak önemlidir.

İki Görev İstasyonu prototipi, her platform için yalnızca bir cihazda test edildi, bu araştırmanın bulguları yalnızca deneyde kullanılan belirli cihazlara dağıtılan uygulamalar için geçerli olabilir. Gelecekte, yalnızca aynı türdeki birden fazla cihaza değil, aynı zamanda Tablotler, akıllı saatler veya TV bağlantılı cihazlar gibi birden çok farklı türdeki cihazlara dağıtılan uygulamalar üzerinde yapılan değerlendirmelerle benzer araştırmalar yapılabilir. Cihazların çeşitliliği, dağıtılan uygulamaların performans özelliklerinin yanı sıra her bir platformlar arası geliştirme çerçevesinin bireysel platformların dahili parçalanmasının üstesinden gelme yeteneği hakkında daha fazla bilgi sağlayacaktır.

Geliştiriciler, aynı uygulamanın birden çok sürümünü üretmek için çabalarını çoğaltmak zorunda kaldıklarında, muhtemelen uygulamalar arasında tutarsızlıklar ortaya çıkacaktır. Bu, iki Görev İstasyonu prototipinin geliştirilmesi sırasında doğruydu. Her prototip, uygulamanın sayfalarının kullanıcı arayüzlerinde ve uygulamanın farklı özelliklerinin uygulanmasında küçük farklılıklara sahiptir. Bu varyasyonlardan bazıları, prototipleri üretmek için kullanılan stratejilerdeki farklılıklardan kaynaklanmaktadır, ancak bazıları aynı anda dört prototipi yönetmenin karmaşıklığından ve her ayrıntıyı mükemmel bir şekilde tutarlı tutmanın zorluğundan kaynaklanmaktadır. Bu farklılıkların bu makaledeki bulguları etkilemiş olması muhtemeldir.

Birden çok uygulama prototipi geliştirme süreci, çaba ve zaman yoğundur. Bu tezin kapsamı, projeyi tek bir geliştirici için yönetilebilir kılmak için yalnızca Flutter ve Android Native'i değerlendirmekle sınırlıydı. Piyasada düzinelerce farklı platformlar arası geliştirme çerçevesi mevcuttur ve bunlar zaman içinde gelişmeye devam etmektedir. Bu makalenin bulgularını genişletmek için daha fazla çerçeve benzer bir şekilde değerlendirilebilir veya çerçevelerin nasıl değiştiğini ve bir çerçevenin sınırlamalarının ele alınıp alınmadığını görmek için gelecekte aynı çerçeveler kullanılarak benzer araştırmalar tekrar yapılabilir. . Özellikle Google'ın Flutter'ı, bu araştırmanın yapıldığı sırada hala beta aşamasındaydı ve çerçevenin nihai yayın sürümünün burada değerlendirilen sürümden önemli ölçüde farklı olması mümkündür.

Birim testi, bakımı yapılabilir yazılım geliştirmenin çok önemli bir parçasıdır. Otomatik birim testi olmadan, yoğun çaba gerektiren ve potansiyel olarak hataların ve hataların gözden kaçmasına neden olabilecek manuel testler yapılmalıdır. Her bir geliştirme stratejisinin birim

test yetenekleri bu tezin bir parçası olarak değerlendirilmemiştir. Her çerçeve için birim test yeteneklerini ve sınırlamalarını keşfetme konusunda gelecekteki araştırmalar yapılabilir.

Mobil uygulama geliştirme ve platformlar arası geliştirme çerçeveleri alanları, alanların karmaşıklığı ve mobil cihaz yazılımı için mevcut ve gelecekteki pazar potansiyeli nedeniyle daha fazla araştırma fırsatlarıyla zengindir. Bir uygulamayı, kolayca korunabilen tek bir kod tabanından birden çok platformda birden çok türde cihaza dağıtma yeteneği, herhangi bir mobil geliştirici için ideal olmaya devam etmektedir. Geliştirme çabası ve süresinde azalmaya izin veren, sorunsuz bir geliştirme deneyimi sağlayan ve yüksek kaliteli uygulamalar dağıtan bir geliştirme çerçevesi arayışı, mobil yazılım pazarı genişlemeye ve gelişmeye devam ettikçe devam edecektir.

6.2 KİŞİSEL YORUMUM

Android dışındaki bu iki platform; React Native ve Flutter, gerçekten de kolay olsa da, gördüğüm kadarıyla program karmaşıklığın arttıkça maintainability azalmaktadır. Kodların işlenmesi lineer değildir. Main.dart içinde bulunan anlaşılmaz komutlar ve fonksiyonlar vardır, bunların tam olarak ne yaptığı anlaşılamamaktadır. Bu da eğer ki gelecekte çok karmaşık bir program yazacaksak, kirlilik yaratmaktadırlar. Her halükarda, incelediğim kadarıyla, kimse bu kadar karmaşık programları bu iki frameworkle yazmamaktadırlar. Daha çok; basit ticari uygulamalar, yemek söyleme uygulamaları gibi nispeten basit uygulamalarda bu tip frameworkler kullanılmaktadır. Fakat, bir bankacılık uygulamasında kullanmak için yeterince güvenli gözükmemektedir.

Ayrıca, Flutter ve React Native, Android kadar geniş kapsamlı araçlar içermemektedir. Bu da kompleks uygulama geliştirmenin önündeki bir kısıttır. Fakat, ticari perspektiften bakıldığında, ticari basit uygulamalar ile pazarı neredeyse iki katına çıkarıp, kar marjını artırmak mümkün görünmektedir. Fakat kompleksite arttıkça, bakım maliyetinin artacağı da göz önünde bulundurulmalıdır.

KAYNAKLAR

1.The Android Source Code, https://source.android.com/source/index.html, Accessed: 2016-02-15

2.Android Interfaces and Architecture,
https://source.android.com/devices/index.html, Accessed: 2016-02-15

3.Android Binder, http://elinux.org/Android_Binder, Accessed: 2016-02-15

4.Android Application Fundamentals,
http://developer.android.com/guide/components/fundamentals.html, Accessed: 2016-02-15

5.Android Activity Lifecycle,
http://developer.android.com/guide/components/activities.html#Lifecycle, Accessed: 2016-02-15

6.Ohad Rodeh, Josef Bacik, and Chris Mason. Btrfs: The linux b-tree filesystem. ACM Transactions on Storage (TOS), 9(3):9, 2013.

7.Jffs2. http://www.linux-mtd.infradead.org/doc/jffs2.html

8.Xiang Gao, Mingkai Dong, Xie Miao, Wei Du, Chao Yu, and Haibo Chen. EROFS: A compression-friendly read-only file system for resource-scarce device. In Proceedings of Annual Technical Conference (USENIX ATC). USENIX Association, 2019.

9.Xuebin Zhang, Jiangpeng Li, Hao Wang, Danni Xiong, Jerry Qu, Hyunsuk Shin, Jung Pill Kim, and Tong Zhang. Realizing transparent os/apps compression in mobile devices at zero latency overhead. IEEE Transactions on Computers, 66(7):1188–1199, 2017.

10.Younghwan Go, Nitin Agrawal, Akshat Aranya, and Cristian Ungureanu. Reliable, consistent, and efficient data sync for mobile apps. In Proceedings of USENIX Conference on File and Storage Technologies (FAST 15), pages 359–372, 2015.

11.Kisung Lee and Youjip Won. Smart layers and dumb result: IO characterization of an android-based smartphone. In Proceedings of the tenth ACM international conference on Embedded software (EMSOFT), pages 23–32. ACM, 2012.

12.Sangwook Shane Hahn, Sungjin Lee, Cheng Ji, Li-Pin Chang, Inhyuk Yee, Liang Shi, Chun Jason Xue, and Jihong Kim. Improving file system performance of mobile storage systems using a decoupled defragmenter. In Proceedings of Annual Technical Conference (USENIX ATC 17), pages 759–771. USENIX Association, 2017.

13.S. Bhattacharya A. Dilger A. Tomas A. Mathur, M. Cao and L. Vivier. The new ext4 filesystem: current status and future plans. In Proceedings of the Linux Symposium, pages 21–33. Citeseer, 2007.

14.Xuebin Zhang, Jiangpeng Li, Hao Wang, Danni Xiong, Jerry Qu, Hyunsuk Shin, Jung Pill Kim, and Tong Zhang. Realizing transparent os/apps compression in mobile devices at zero latency overhead. IEEE Transactions on Computers, 66(7):1188–1199, 2017.

15.Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. F2FS: A new file system for flash storage. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST), 2015.

16.Ming-Chang Yang, Yuan-Hao Chang, Chei-Wei Tsao, and Chung-Yu Liu. Utilization-aware self-tuning design for TLC flash storage devices. IEEE Trans. VLSI Syst., 24(10):3132–3144, 2016.

17.Sangwook Shane Hahn, Sungjin Lee, Inhyuk Yee, Donguk Ryu, and Jihong Kim. Fasttrack: Foreground app-aware I/O management for improving user experience of android smartphones. In Proceedings of Annual Technical Conference (USENIX ATC 18), pages 15–28. USENIX Association, 2018.

18.Samsung Semiconductors. 3D TLC NAND to beat MLC as top flash storage. EETimes, 2015.

19.Jffs2. http://www.linux-mtd.infradead.org/doc/jffs2.html

20.Xiang Gao, Mingkai Dong, Xie Miao, Wei Du, Chao Yu, and Haibo Chen. EROFS: A compression-friendly read-only file system for resource-scarce device. In Proceedings of Annual Technical Conference (USENIX ATC). USENIX Association, 2019.

21.https://android.googlesource.com/kernel/common.

22.https://github.com/makomk/aeskeyfind.

23.https://github.com/504ensicsLabs/LiME.

24.https://github.com/difcareer/BootImgTool.

25.https://magiskmanager.com.

26.T. Occhino, “Keynote.” React.js Conf, 2015.

27.React: A JavaScript Library For Building User Interfaces, https://facebook.github.io/react/ , Accessed: 2016-02-11

28.React: Releases, https://github.com/facebook/react/releases?after=v0.9.0-rc1 , Accessed: 2016-02-10

29.React: A JavaScript Library For Building User Interfaces, https://facebook.github.io/react/ Accessed: 2016-02-11

30.T. Occhino, “React native: Bringing modern web techniques to mobile,” March 2015. Accessed: 2016-02-08.

31.Vue.js Overview, http://vuejs.org/guide/overview.html , Accessed: 2016-02-10

32.Babel the compiler for writing next generation JavaScript, http://babeljs.io/, Ac-

cessed: 2016-02-11

33.React Native: JavaScript Environment, https://facebook.github.io/react-native/docs/javascript-environment.html#content, Accessed: 2016-02-10

34.React Native: Native Android Modules, http://facebook.github.io/react-native/docs/native-modules-android.html, Accessed: 2016-02-11

35.React Native: Native iOS Modules, https://facebook.github.io/react-native/docs/native-modules-ios.html , Accessed: 2016-02-11

36.React Native: Native Communication between native and React Native, https://facebook.github.io/react-native/docs/communication-ios.html, Accessed: 2016-02-11

37.React Native: Performance, https://facebook.github.io/react-native/docs/performance.html , Accessed: 2016-02-11

38.iOS Developer Library: Dispatch Queues, https://developer.apple.com/library/ios/documentation/General/Conceptual/ConcurrencyProgrammingGuide/OperationQueues/OperationQueues.html, Accessed: 2016-02-11

39.Android Develop Reference: Handler, http://developer.android.com/reference/

40.Using CSS exible boxes, https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Using_CSS_flexible_boxes, Accessed: 2016-02-11

41.Official documentation of Flutter Navigator URL:https://docs.flutter.io/flutter/widgets/Navigator-class.html Accessed November 03, 2017

42.http://csuchico-dspace.calstate.edu/bitstream/handle/10211.3/211157/Gonsalves_Thesis_Fall18.pdf?sequence=1

43.https://www.usenix.org/system/files/fast21-ji.pdf

44.https://arxiv.org/pdf/2103.05292.pdf

45.https://www.diva-portal.org/smash/get/diva2:946127/FULLTEXT01.pdf

46.https://www.theseus.fi/bitstream/handle/10024/148975/thesis_Jakhongir_Fayzullaev.pdf?sequence=1&isAllowed=y

47.https://reader.elsevier.com/reader/sd/pii/S266628172100007X?token=75457C4CBD18F8440130610233BAE716E107F3B3E1FB3E29CF1A2016EAC676388C5DA1CC245F47666C2C4DB37F650AF2&originRegion=eu-west-1&originCreation=20210605144833