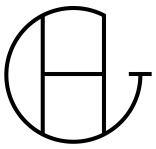


HoGent

Faculteit Bedrijf en Organisatie

Hoe beperken ontwikkelaars de schijfruimte ingenomen door mobiele applicaties voor Android Wear?

Thomas Ledoux

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Dr. Jens Buysse
Co-promotor:
Joris Missiaen

Instelling: —

Academiejaar: 2016-2017

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Hoe beperken ontwikkelaars de schijfruimte ingenomen door mobiele applicaties voor Android Wear?

Thomas Ledoux

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Dr. Jens Buysse
Co-promotor:
Joris Missiaen

Instelling: —

Academiejaar: 2016-2017

Tweede examenperiode

Samenvatting

Voorwoord

Ik zou graag deze gelegenheid nemen om enkele mensen te bedanken. Als eerste wil ik mijn promotor, Dr. Jens Buysse bedanken voor de kans die hij me bood om deze bachelorproef onder zijn promotorschap te volbrengen, voor de feedback & de aangename begeleiding tijdens het semester. En ook mijn co-promotor Joris Missiaen verdient zeker een dankjewel voor de inhoudelijke ondersteuning, waardoor mijn bachelorproef er zeker anders had uitgezien. Natuurlijk was ik nooit zo ver gekomen zonder de onvoorwaardelijke steun van mijn twee lieve ouders. Zowel financieel als emotioneel zijn zij er 3 jaar lang volledig voor mij geweest. Voor het nalezen van teksten, het warme weekendnest en het oneindige vertrouwen dat ze in mij stelden gedurende mijn volledige opleiding, dankjewel! Ook mijn vriendin Louise wil ik bedanken voor het geduld, de gerustellingen en de ondersteuning op elke manier. Louise geloofde er altijd in dat ik deze bachelorproef tot een goed einde zou brengen, en wist mij in dit geloof meer te trekken.

Inhoudsopgave

1. Inleiding

1.1 Probleemstelling en Onderzoeksvragen

De voorbije jaren hebben wearable devices steeds meer aan belang gewonnen in het dagelijks leven. Volgens een recent onderzoek werd het aantal verkochte wearable devices wereldwijd geschat op 20.1 miljoen, waarvan 22.9% Android Wear als operating system gebruikt. (IDC Research, 2016) voorspelt tegen 2020 dat er 54.6 miljoen wearable devices verkocht zullen worden, waarvan 41.8% Android Wear als operating system zal gebruiken. Elk noemenswaardig bedrijf heeft tegenwoordig een app laten ontwikkelen die ook op wearable devices kan gebruikt worden om zoveel mogelijk van deze mobiele gebruikers te bereiken. Hiermee gaat echter gepaard dat gebruikers meer en meer apps willen installeren op hun wearables, terwijl deze slechts over een beperkte schijfruimte beschikken. Daarom moeten de ontwikkelaars van deze mobiele applicaties ervoor zorgen dat de grootte van hun applicaties zoveel mogelijk beperkt wordt, zodat mobiele toestellen niet zonder vrije schijfruimte komen te staan. We kiezen voor Android Wear omdat dit een open-source platform is. De belangrijkste pijlers van open-source zijn dat open-source software vrij kan gedistribueerd worden en dat de source code beschikbaar wordt gesteld (Initiative, 2017). Dit vergemakkelijkt het onderzoek doordat er in de broncode kan gekeken worden welke technieken precies gebruikt worden en op welke manier deze geïmplementeerd worden. Daardoor kunnen er voor het testen van de verschillende technieken open-source apps gebruikt worden, die al verder uitgewerkt zijn dan de app die tijdens dit onderzoek gecreëerd werd. Android Wear toestellen beschikken vaak maar over een beperkte opslagruimte, waardoor elke megabyte vrije ruimte telt. De eindgebruikers van de apps merken hier dus het meest directe effect van, zij zullen kunnen profiteren van de vrijgekomen ruimte door de compressie van de apps. Maar ook voor bedrijven en hun app-ontwikkelaars is dit belangrijk: wanneer hun Android Wear app niet te veel

schrijfruimte inneemt op het toestel van de eindgebruikers, zullen zij minder snel geneigd zijn deze app te verwijderen. Wanneer ze de app wel zouden verwijderen ontloopt het bedrijf belangrijke inkomsten uit in-app purchases en eventuele reclame-inkomsten. Het vergelijken van de verschillende technieken zal gebeuren op verschillende vlakken: zijn er veel media-bestanden aanwezig in de applicatie? Is er veel code aanwezig in de applicatie? Zijn er meerdere APK's voorzien?

1.2 Opzet van deze bachelorproef

Om dit onderzoek te ondersteunen zal een proof-of-concept app gecreëerd worden voor Android Wear. Op deze app zullen dan de verschillende technieken toegepast worden en zal het effect van deze technieken vergeleken en besproken worden. In de literatuurstudie bespreken we wat Android Wear precies inhoudt en bespreken we de verschillende compressietechnieken. Binnen de literatuurstudie kijken we ook welke verschillende profielen van apps er onderzocht moeten worden. Zo zullen er apps zijn die veelvuldig gebruik maken van media-bestanden, bijvoorbeeld foto-apps, waarbij compressie dus vooral zal gericht zijn op de foto's. Anderzijds zullen er ook apps zijn die gebruikt worden tijdens het sporten, waarbij GPS-tracking een grote rol zal spelen. Als laatste categorie zullen apps onderzocht worden die CPU-intensieve taken zullen uitvoeren, bijvoorbeeld een app die muziek afspeelt en op het tempo ervan live visuele effecten genereert. Deze app-profielen werden geselecteerd na onderzoek van de Google Play Store voor Android Wear. In de top 5 van de populairste voor Android Wear zijn 3 apps aanwezig die gekoppeld zijn aan fitness en gezondheid. Daarnaast staan in de top 10 3 chat-apps aanwezig, deze apps zullen vaak gebruik maken van afbeeldingen, GIF's en andere mediabestanden. In de derde categorie vinden we apps terug zoals Spotify, Play Music en Shazam. Deze apps zullen muziek afspelen op het Android Wear toestel, en intussen visueel ofwel de albumcover weergeven, of grafische elementen genereren op het tempo van de muziek. Shazam is ook aanwezig in de top 10 van populairste apps voor Android Wear. Er zal ook gekeken worden of er al dan niet een verband is tussen de compressie van de apps en de prestatie ervan op vlak van geheugengebruik en CPU-gebruik. Nadien zullen de resultaten van het onderzoek besproken worden en vergeleken worden. Tot slot volgt een samenvattende conclusie van het onderzoek.

2. Methodologie

2.1 Literatuurstudie Android Wear

Android Wear is Google's platform voor wearables. Een wearable wordt beschreven als een computer of geavanceerd elektronisch toestel dat geïncorporeerd is in een accessoire gedragen op het lichaam of in een kledingstuk (Dictionary.com, 2017). We lijsten enkele van de belangrijkste kenmerken van Android Wear op (Techradar, 2017). Android Wear is gebouwd voor kleinere toestellen en met hands-free gebruik in het achterhoofd. Het maakt de toegang tot enkele van uw smartphone's makkelijkste functionaliteiten zo makkelijk als kijken naar uw pols. Het is vooral bedoeld om de notificaties van uw smartphone makkelijk te weergeven zodat niet elke keer gezocht moet worden naar de smartphone in de broekzak. Gebaseerd op uw Google zoekopdrachten zullen real-time scores van uw favoriete sportteam, verkeerscondities of afspraken in uw agenda weergeven worden. Als het Android Wear toestel NFC (Near Field Communication) ingebouwd heeft, kan er in verschillende landen (momenteel Australië, Hong Kong, Ierland, Japan, Nieuw-Zeeland, Polen, Singapore, Verenigd Koninkrijk en de Verenigde Staten) ook met de wearable betaald worden via Android Pay (Google, 2017) Ook spraakherkenning kon niet ontbreken in de Android Software. Door deze technologie kan je de op de wearable spraakherkenning activeren door "Okay, Google"te zeggen of door de aanknop ingedrukt te houden. Google Assistant zal u hierop verderhelpen. Android Wear is compatibel met smartphones die werken op Android 4.3 en hoger of iOS 9 en hoger. Eén van de belangrijkste app-categorieën op Android Wear zijn de gezondheidsapps. Hiervoor wordt standaard gebruik gemaakt van Google Fit. Met Google Fit kan je kiezen tussen verschillende workouts tijdens het sporten, of bijvoorbeeld een dagelijks doel instellen voor het aantal stappen dat u wil nemen.

Figuur 2.1: *Enkele voorbeelden van smartwatches gebruikmakend van Android Wear*

2.1.1 Wat is een APK en uit wat bestaat dit?

APK staat voor Android Application Package. Een APK is een applicatiebestand klaar voor installatie op een Android-toestel. Het gecompresseerde APK bestand, een ZIP archief in JAR-formaat, wordt gedistribueerd naar Android gebruikers voor de installatie op hun smartphone, tablet of wearable (PCMag, 2017). Een APK bestand bestaat normaliter uit volgende elementen :

Elementen APK

1. **classes.dex**
   Bevat gecompileerde applicatiecode, getransformeerd naar een DEX bytecode. Het is mogelijk dat er meerdere DEX bestanden in de APK staan als er multidex gebruikt wordt om de 65536 limiet te overkomen. Vanaf Android 5.0, met de introductie van ART runtime, worden deze bestanden gecompileerd naar OAT bestanden door de compiler bij de installatie en worden deze op de data partitie van het toestel geplaatst.

2. **res/**
   Deze folder bevat de meeste XML-bestanden en drawables (bijvoorbeeld PNG- of JPEG-bestanden) in mappen met verschillende kwalificaties, zoals -mdpi en -hdpi voor schermdichtheden, -sw600dp of -large voor schermgroottes, en -en, -de, -pl voor talen. De XML bestanden worden automatisch gecompileerd naar een compactere binaire representatie, dus deze kunnen niet met een teksteditor geopend worden vanuit de APK.

3. **resources.arsc**
   Sommige resources en identifiers worden gecompileerd naar dit bestand. Het wordt normaal ongecomprimeerd opgeslagen in de APK voor snellere toegang tijdens runtime. Dit bestand manueel comprimeren kan een makkelijke oplossing lijken, maar dit is geen goed idee door 2 redenen: Eén, Play Store comprimeert alle data voor transfer automatisch en twee, dit bestand comprimeren in de APK verspilt

systeem resources (RAM) en performantie (vooral de opstarttijd van de app).

4. **AndroidManifest.xml**
   Gelijklopend met andere XML resources wordt de applicatie Manifest getransformeerd naar een binair formaat tijdens de compilatie. De Play Store gebruikt bepaalde informatie in de AndroidManifest om te bepalen of een APK kan geïnstalleerd worden op een bepaald toestel. Zo zijn er controles op toegelaten schermdichtheden of schermgroottes, beschikbare hardware en kenmerken (bijvoorbeeld aanwezigheid van een touchscreen). De inhoud van het Manifest kan geïnspecteerd worden na compilatie. Gebruik hiervoor de aapt tool van de Android SDK:

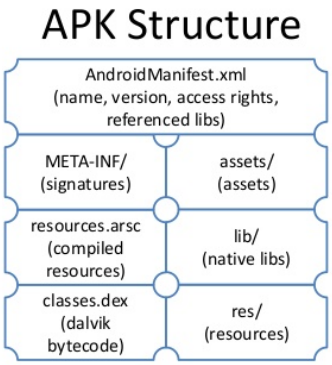
   ```
   $ aapt dump badging your_app.apk
   ```

5. **libs/**
   Alle native libraries (*.so bestanden) zullen in submappen geplaatst worden genaamd naar de ABI (CPU architectuur, bv. x86, `x86_64`, armeabi-v7a) die ze als doel hebben onder de libs/ map. Normaal worden deze uit de APK gekopieerd naar de datapartitie tijdens het installeren van de applicatie. Echter, sinds de APK zelf nooit aangepast wordt zolang deze op het toestel van de gebruiker staat, neemt dit dubbel zoveel plaats in voor elke native library.

6. **assets/**
   Deze map zal gebruikt worden voor alle bestandsonderdelen die niet als Android-type resources gebruikt zullen worden. Meestal zullen dit lettertypebestanden of game data zijn, zoals levels of textures, samen met andere applicatiedata die direct geopend moet kunnen worden als een file stream.

7. **META-INF/**
   Deze map is aanwezig in ondertekende APKs en bevat een lijst van alle bestanden in de APK met hun handtekening. Momenteel werkt het ondertekenen van bestanden in Android door het verifiëren van de handtekening tegen de ongecomprimeerde bestandsinhoud van het archief, één voor één. Dit heeft enkele interessante gevolgen. Doordat elke onderdeel van een ZIP bestand apart opgeslagen wordt, betekent dit dat je individuele bestanden hun compressieniveau kan aanpassen zonder ze opnieuw te ondertekenen. De verificatie van de handtekening zal echter falen wanneer een bestand verwijderd wordt uit het archief nadat het ondertekend werd.

Figuur 2.2: *De structuur van de APK*



2.1.2 Conclusie APK-structuur

Een APK bestaat dus uit 7 onderdelen. Deze onderdelen zullen echter wel verschillen van grootte. Indien de app veel afbeeldingen bevat zal de map res/ het grootste deel van de APK-grootte innemen. Indien de app veel code bevat zal het classes.dex bestand groter worden. In de proof-of-concept app, die later in deze studie wordt toegelicht, konden we volgende verhoudingen waarnemen tussen de groottes van de verschillende onderdelen:

1. classes.dex: 0.4%
2. res/: 98.3%
3. resources.arsc: 1.1%
4. AndroidManifest.xml: kleiner dan 0.01%
5. libs/: kleiner dan 0.01%
6. assets/: kleiner dan 0.01%
7. META-INF/: 0.2%

2.1.3 Verschillende app-profielen Android Wear

Verder in deze studie zullen de verschillende compressietechnieken toegepast worden op een proof-of-concept app binnen elk van de 3 profielen (foto-apps, fitness-apps, CPU-intensieve apps). Hierdoor kan binnen elk app-profiel gekeken worden welke technieken het meeste invloed zullen hebben op de grootte van de APK. Ook zal voor elk app-profiel onderzocht worden of er al dan niet negatieve bijwerkingen zijn voor de prestaties van de app na compressie en hoe groot deze bijwerkingen zijn. Op deze manier kan dit onderzoek voor de ontwikkelaars van elk type app nagaan welke technieken voor hun app in ontwikkeling het meest effect zullen hebben op de grootte van de APK. Aan de hand van de resultaten van het onderzoek kan dan ook de afweging gemaakt worden of er voor hun applicatie niet te veel moet ingeboet worden op vlak van CPU- en geheugenprestaties van de app.

Foto-apps

Apps die veel gebruikmaken van foto's zullen in het algemeen vrij veel opslagruimte in beslag nemen, aangezien elke foto in de app meegeleverd wordt en de grootte van een foto snel kan oplopen tot 1MB. Indien de foto's niet meegeleverd worden in de APK van de app, dienen de foto's gedownload te worden van het internet en zal er dus meer netwerkactiviteit gebruikt worden en CPU verbruik om de foto te genereren op de juiste plaats in de app.

GPS-intensieve apps

Deze apps zullen de gebruiker vooral helpen bij het opvolgen van lichamelijke inspanningen en beweging. Bij deze apps wordt meestal gebruikgemaakt van de ingebouwde GPS in het Android Wear toestel. Doordat deze GPS constant actief is, zal het batterijgebruik van deze apps zeer hoog liggen.

CPU-intensieve apps

Apps die veel (visuele) informatie moeten genereren tijdens runtime, zullen in het algemeen een hoog CPU verbruik hebben. Een voorbeeld hiervan is een app die muziek afspeelt en intussen visuele effecten afbeeldt die gegenereerd worden op het tempo van de afgespeelde muziek.

2.2 Verschillende compressietechnieken

Uit de richtlijnen van Google voor het verkleinen van de grootte van de APK werden volgende technieken gekozen om te onderzoeken (Google, 2015):

Gebruik Proguard en Dexguard

Uit een interview met Cozmos, het bedrijf waarbij mijn co-promotor Joris Missiaen werkzaam is blijkt dat zij deze tools ook gebruiken :

> "De belangrijkste tool om de grootte van een APK te beperken is Proguard. Proguard zorgt er niet enkel voor dat je code moeilijker leesbaar wordt bij reverse engineering maar het verwijdert ook al je ongebruikte code en resources. Bij de grotere klanten waar ook security van groot belang is maken we gebruik van Dexguard. Dexguard is de commerciële variant van Proguard en gaat een stap verder met zijn compressietechnieken maar biedt daarnaast ook tal van andere features aan."

Deze andere features zijn dan bijvoorbeeld bescherming tegen static analysis, beveiliging tegen klonen van de APK/SDK en bescherming tegen piraterij. Ook beschermt Dexguard tegen dynamic analysis (APK beveiligen tijdens runtime) door omgeving- en certificaatcontroles.

Verwijder ongebruikte resources

De ingebouwde lint tool in Android Studio, een statische code analyzer, detecteert resources in de res/ map die in de code niet gerefereerd worden. Android Studio zal hiervan een melding maken, maar zal deze niet automatisch verwijderen. Libraries die u toevoegt kunnen ongebruikte resources meebrengen. Gradle kan deze automatisch verwijderen als "shrinkResources" in de build.gradle wordt toegevoegd.

Verklein resource gebruik van libraries

Wanneer u een Android app ontwikkelt, worden vaak externe libraries gebruikt om de gebruiksvriendelijkheid te verhogen. Deze libraries bevatten vaak elementen die voor desktops of servers bedoeld zijn, deze elementen kunnen uit de libraries verwijderd worden indien de licentie dit toestaat. Deze techniek wordt ook gebruikt door Cozmos volgens Joris Missiaen:

> "Bij het gebruik van libraries kan het handig zijn om enkel de modules in je project te importeren die je ook daadwerkelijk nodig hebt. Ook kan het nuttig zijn om verschillende libraries tegenover elkaar af te wegen en te kijken welke de laagste method count hebben, het best met het geheugen omgaan, het minste plaats innemen, enz. Stel dat je een library aan je project zou willen toevoegen om images in te laden dan zou je de afweging kunnen maken tussen Glide en Picasso. Picasso is bijvoorbeeld 3,5 keer kleiner dan Glide, maar anderzijds gebruikt Glide een pak minder geheugen om een image weer te geven. Dat zijn afwegingen die je moet maken."

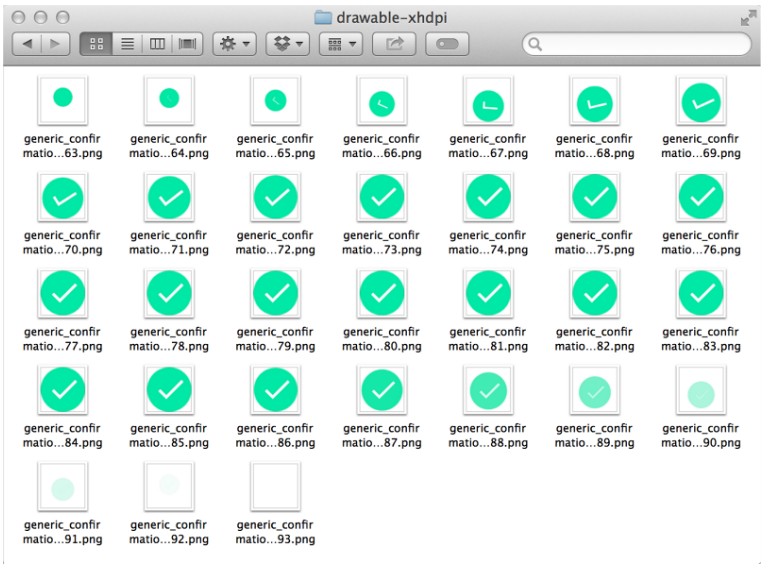
Ondersteun enkele specifieke schermdichtheden

Vanaf Android 4.4 worden verschillende schermdichtheden ondersteund (ldpi, mdpi, tvdpi, hdpi, xhdpi, xxhdpi en xxxhdpi). Het is niet nodig om elke afbeelding te exporteren naar elke dichtheid. Android zal automatisch de afbeelding schalen naar andere schermdichtheden als er geen specifieke export voorzien is.

Verminder animatieframes

Voor elke frame in een animatie wordt een aparte afbeelding opgeslagen. Als er bijvoorbeeld een animatie aanwezig is met 30 FPS (frames per second) zullen er 30 afbeeldingen opgeslagen worden, maar vaak is bijvoorbeeld 15 FPS meer dan voldoende. Op deze manier zijn er dus maar half zoveel animatieframes nodig.

Figuur 2.3: *Door vermindering FPS zullen minder aparte afbeeldingen per animatie opgeslagen worden*



Hergebruik resources

Het is mogelijk om voor elke variatie op een afbeelding (getint, met schaduw of geroteerd) een aparte resource op te slaan. Google raadt echter aan om één afbeelding te voorzien, en deze dynamisch tijdens runtime aan te passen. Android voorziet verschillende hulpprogramma's om de kleur van een afbeelding aan te passen.

Comprimeer afbeeldingen

De Elinux, 2010 tool kan de afbeelding resources in res/drawable/ optimaliseren met lossless compressie. De tool kan bijvoorbeeld een true-color PNG bestand dat niet meer dan 256 kleuren bevat omzetten naar een 8-bit PNG met een kleurenpallet. Dit resulteert in een afbeelding van eenzelfde kwaliteit maar met een kleinere geheugenvoetafdruk. Deze tool kan geen PNG-bestanden in de asset/ folder verkleinen. Deze techniek wordt ook bij Cozmos gebruikt volgens Joris Missiaen:

> "Je kan al ruimte besparen door een juiste keuze te maken tussen JPG/PNG en verdere compressie toe te passen. Soms is ook geen volledige image nodig maar kan je opteren voor een 9-patch image of om een drawable in XML te definieren."

Gebruik WebP bestandformaat

In plaats van PNG- of JPEG-bestanden, kan er ook gebruik gemaakt worden van het WebP formaat voor afbeeldingen. Dit formaat voorziet lossless compressie (zoals JPEG) en

transparantie (zoals PNG) maar voorziet betere compressie dan beide andere formaten. Het gebruik van WebP heeft echter ook enkele nadelen. WebP-ondersteuning is niet beschikbaar bij versies lager dan Android 3.2 (API level 13). Het neemt ook een langere tijd voor het systeem om WebP bestanden te decoderen dan PNG bestanden. Bestaande BPM, JPG, PNG of statische GIF afbeeldingen kunnen naar het WebP formaat omgezet worden door Android Studio.

Gebruik vector graphics

Vector graphics kunnen gebruikt worden om resolutie-onafhankelijke iconen of andere schaalbare media te creëren. Deze afbeeldingen worden in Android gerepresenteerd als VectorDrawable objecten. Met een VectorDrawable object kan een 100-byte bestand een scherpe afbeeldingen genereren op de grootte van het scherm. Echter, het neemt een significante tijd voor het systeem om elk VectorDrawable object te genereren. Grote afbeeldingen kunnen zelfs nog meer tijd in beslag nemen om op het scherm te verschijnen. Daarom wordt aangeraden de VectorDrawable objecten enkel voor kleine afbeeldingen te gebruiken.

2.2.1 Verminder gebruik code

Verwijder automatisch gegenereerde code

Zorg dat u verstaat wat elke lijn gegenereerde code inhoudt en wat de voetafdruk ervan is. Zo zullen veel protocol buffer tools een groot aantal methodes en klassen genereren, die de grootte van de app kunnen verdubbelen of verdriedubbelen.

Verwijder enumeraties

Elke enum kan 1 tot 1.4 KB grootte toevoegen aan uw app's classes.dex bestand. Deze toevoegingen kunnen snel oplopen voor complexe systemen of gedeelde libraries. Indien mogelijk gebruikt u best de @IntDef annotatie en ProGuard om enumeraties te strippen en om te zetten in integers. Deze conversie behoudt alle voordelen van enums.

Verklein grootte native binaries

Bij het gebruik van native code en de Android NDK (Native Development Kit, hierdoor kan je bepaalde delen van je app implementeren in native-code talen zoals C of C++) kan je de grootte van de app verkleinen door het optimaliseren van de code. Hiervoor zijn 2 technieken bruikbaar :

1. **Debug symbolen verwijderen**
   Wanneer de app niet meer in development is, zijn debug symbolen overbodig. Deze kan je uit de applicatie verwijderen door de "arm-eabi-strip" tool te gebruiken, deze zit ingebouwd in Android NDK.
2. **Vermijd het extraheren van native libraries**

Bewaar .so bestanden ongecompresseerd in de APK, en stel de “android:extractNativeLibs“ vlag op false in het “<application>“ element van je app manifest. Dit zal vermijden dat de PackageManager de .so bestanden uit de APK naar het bestandssysteem zal kopiëren tijdens de installatie, en zal als bijkomend voordeel hebben dat delta updates van de app kleiner zullen zijn.

2.2.2 Voorzie meerdere APK’s

De app kan inhoud bevatten die gebruikers downloaden maar nooit gebruiken, zoals regio- of taalgerelateerde informatie. Om een minimale download voor de gebruikers te voorzien, kan de app gesegmenteerd worden in verschillende APKs, gedifferentieerd door factoren zoals schermgrootte of GPU structuur. Wanneer gebruikers de app downloaden zal hun toestel de correcte APK ontvangen gebaseerd op het toestel zijn instellingen en kenmerken. Op deze manier ontvangen hun toestellen geen inhoud bedoeld voor opties die het toestel niet heeft.

2.3 Verband tussen compressie en prestaties apps

Bij Cozmos werd het volgende opgemerkt als verband tussen compressie en de prestaties van de apps :

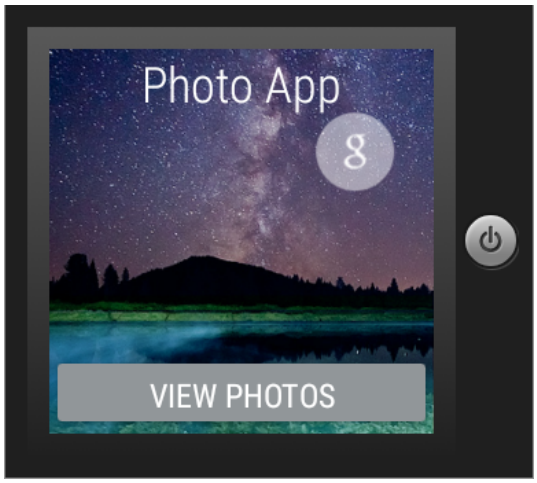
> "Er wordt zeker naar gekeken dat de performantie van de app goed zit. Bij het uitvoeren van network calls moet alles zo compact mogelijk zijn zodat een gebruiker ook onder een slechte verbinding kan blijven werken. Bij het tonen van meerdere afbeeldingen is het memory management zeer belangrijk om o.a. out of memory exceptions te vermijden. Op welk vlak de performantie een invloed heeft hangt af van ieder geval op zich. Je kan stellen dat er op één of meerdere van volgende punten verbetering merkbaar is":
> 1. **Grootte beperken**
> 2. **Geheugen gebruik beperken**
> 3. **Sneller laden**

2.4 Creatie proof-of-concept foto-app

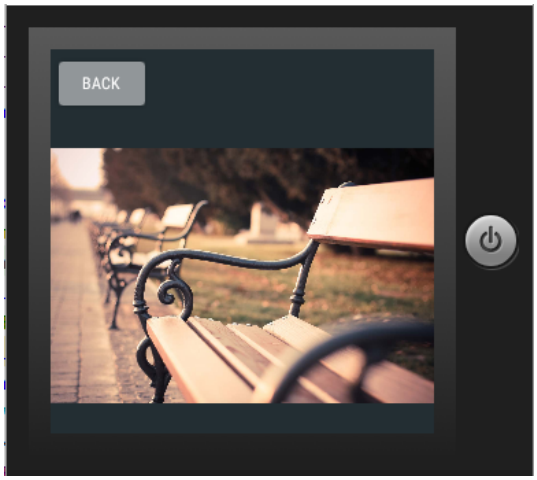
Om het onderzoek te onderbouwen, werd een Android Wear proof-of-concept app gecreëerd. Dit is een algemene applicatie, waarin verschillende elementen toegevoegd worden zoals media-bestanden en veel code. Het doel van de creatie van deze app is om de verschillende compressietechnieken hierop toe te passen en de resultaten grafisch voor te stellen. De applicatie draait op Android Wear 2.0. Deze versie werd op 09/02/2017 uitgebracht door Google (David Singleton, 2017). Doordat voor deze versie gekozen werd, kan de app als standalone app gebruikt worden. Dit houdt in dat de app op een smartwatch

kan gebruikt worden onafhankelijk van een smartphone. De gebruikers kunnen meer taken, zoals muziek afspelen of foto's bekijken, op hun smartwatch uitvoeren zonder toegang tot een Android of iOS smartphone. De app zal als functie hebben om foto's weer te geven op de smartwatch, waar de gebruiker tussen kan wisselen door te tikken op het scherm. Op deze afbeeldingen werd vooraf nog geen compressie toegepast er werden afbeeldingen van een standaardgrootte gebruikt die dan automatisch geschaald werden naar de grootte van de image views. In de app zit ook een splashscreen verwerkt (scherm dat weergeven wordt tijdens het opstarten van de applicatie). Deze app zal dus als testapp gebruikt worden voor het app-profiel "Foto-apps".

Figuur 2.4: *Hoofdscherm foto-app*



Figuur 2.5: *Foto view*



3. Resultaten

3.1 Vergelijking verschillende technieken

Verschillende van de voorgestelde technieken worden reeds automatisch toegepast door Android Studio bij het compileren van de applicatie. Zo zit Proguard ingebouwd in Android Studio, worden ongebruikte resources verwijderd door de ingebouwde lint tool en kunnen enumeraties automatisch omgezet worden naar integers. Hierdoor zal het handmatig uitvoeren van deze technieken geen extra effect teweeg brengen. Daarnaast zal het effect van de technieken afhankelijk zijn van het app-profiel. Legende technieken:

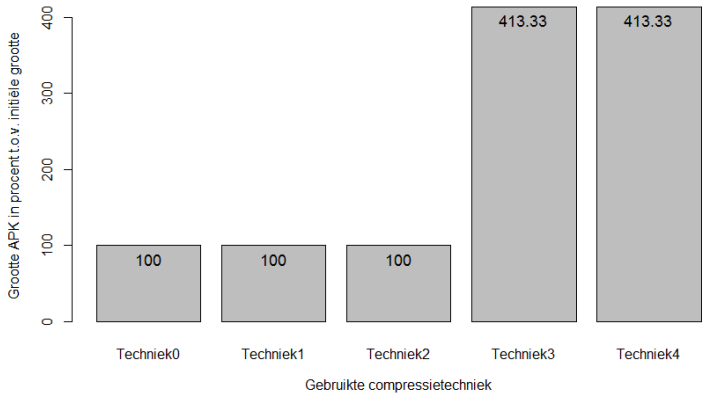
0. Techniek0 = Geen compressietechniek toegepast
1. Techniek1 = Verwijder ongebruikte resources
2. Techniek2 = Comprimeer afbeeldingen
3. Techniek3 = Gebruik WebP-formaat
4. Techniek4 = Verklein grootte native binaries

3.1.1 Foto-app

Bij de eerste applicatie die getest werd, een foto-app die veel mediabestanden bevat, had vooral het gebruik van het WebP-formaat voor alle afbeeldingen een grote invloed op de grootte van de APK. De afbeeldingen werden omgezet van het PNG-formaat naar het WebP-formaat met lossless compressie, waardoor er dus niet in kwaliteit van de afbeeldinge ingeboet moet worden. Door het toepassen van deze compressietechniek werd de grootte van de APK meer dan vervierdubbeld. In de gedetailleerde analyse van de APK in Android Studio kon waargenomen worden dat vooral de res/ folder, die de afbeeldingen die in de app gebruikt worden bevat, al de extra opslagruimte die werd ingenomen inneemt. Dit is

een onverwacht resultaat, aangezien er door Google wordt aangeraden om deze techniek te gebruiken om de grootte van de APK te verkleinen. De andere technieken brachten op dit vlak geen verandering. Op vlak van het geheugengebruik van de applicatie tijdens runtime kon een minieme verandering van 0.03MB meer geheugen gebruikt opgemerkt worden na het toepassen van de techniek "Verklein resource gebruik van libraries". Na de omzetting van de afbeeldingen naar het WebP-formaat verhoogde het CPU-gebruik met 9 procent. Dit wordt veroorzaakt doordat het genereren van de WebP-afbeeldingen meer CPU-kracht vergt.

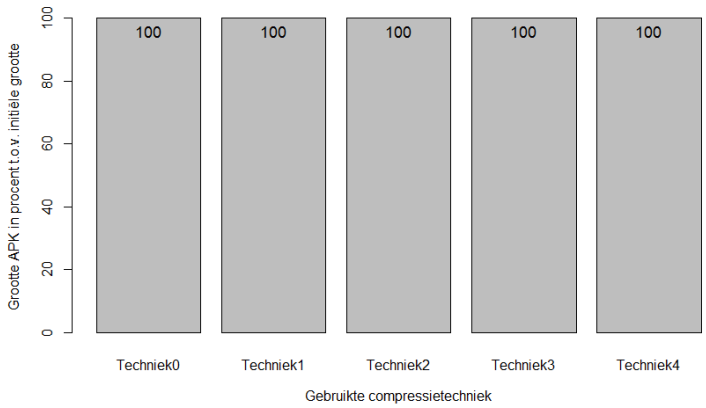
Figuur 3.1: *Resultaten verschillende compressietechnieken op APK-grootte foto-app*



3.1.2 GPS-intensieve app

De tweede applicatie die getest werd is een app gemaakt om op Swarm te kunnen inchecken via een Android Wear toestel. Dit toestel moet dan natuurlijk wel een GPS antenne bevatten. Deze app is open-source, en werd gecreëerd door (Viviani, 2014). De app heet Wear for Swarm. Bij deze app werden op het vlak van de grootte van de APK geen wijzigingen opgemerkt. De grootte van de APK werd na het toepassen van elke compressietechniek geanalyseerd, maar deze bleef op hetzelfde niveau. Bij het geheugengebruik werd een minieme verandering waargenomen na het toepassen van de 2e techniek, het comprimeren van de afbeeldingen. Het geheugengebruik lag 0.01MB lager na het toepassen van deze techniek. De overige technieken hadden geen invloed op het geheugengebruik. Het CPU-gebruik bleef constant na het toepassen van de verschillende technieken.

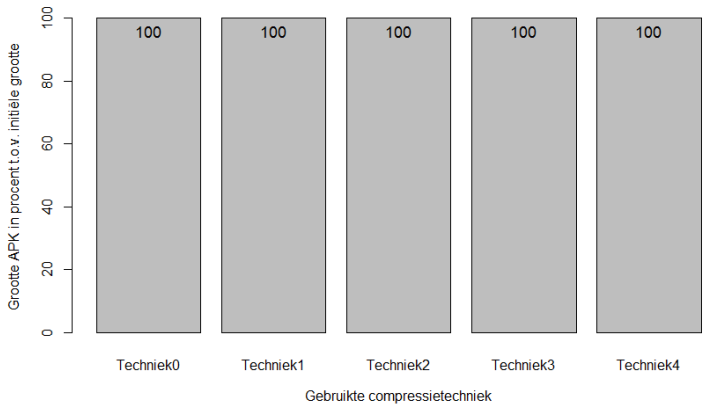
Figuur 3.2: *Resultaten verschillende compressietechnieken op APK-grootte GPS-intensieve app*



3.1.3 CPU-intensieve app

De derde en laatste app die getest werd is een timer-app die de CPU intensiever zal gebruiken. Er werd gekozen om een open-source app te kiezen om de verschillende technieken op toe te passen, zodat er toegang is tot de bronbestanden. De gekozen app heet WearTimer en werd gecreëerd door (Khmelenko, 2015). Er werden bij deze app op het vlak van de grootte van de APK geen wijzigingen opgemerkt. De grootte van de APK werd na het toepassen van elke compressietechniek geanalyseerd, maar deze bleef op hetzelfde niveau. Bij het geheugengebruik werd een verandering van 0.03MB meer geheugen waargenomen na het toepassen van de eerste techniek, het verwijderen van ongebruikte resources. Na het toepassen van de tweede techniek werd er nog een extra toevoeging van 0.01 gebruikt geheugen genoteerd. Het CPU-gebruik was constant na het toepassen van de verschillende technieken.

Figuur 3.3: *Resultaten verschillende compressietechnieken op APK-grootte CPU-intensieve app*

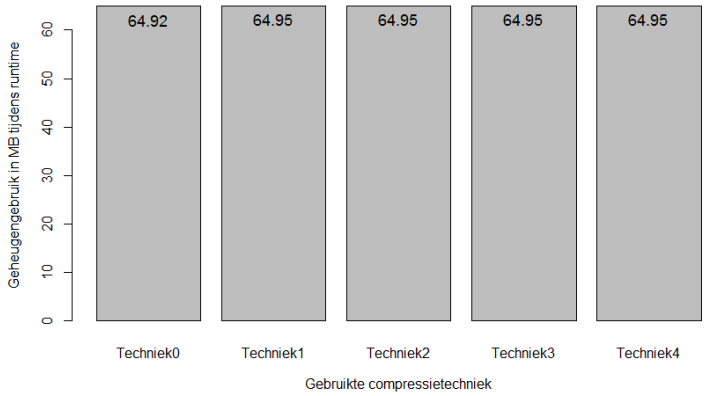


3.2 Invloed compressie op prestaties apps

3.2.1 Foto-app

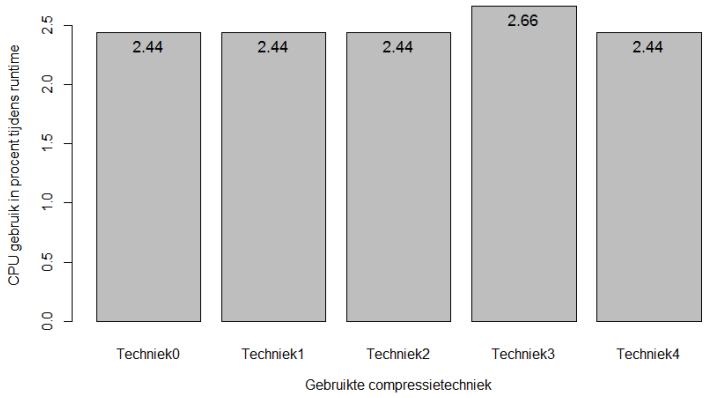
Geheugengebruik

Figuur 3.4: *Resultaten verschillende compressietechnieken op geheugengebruik foto-app*



CPU-gebruik

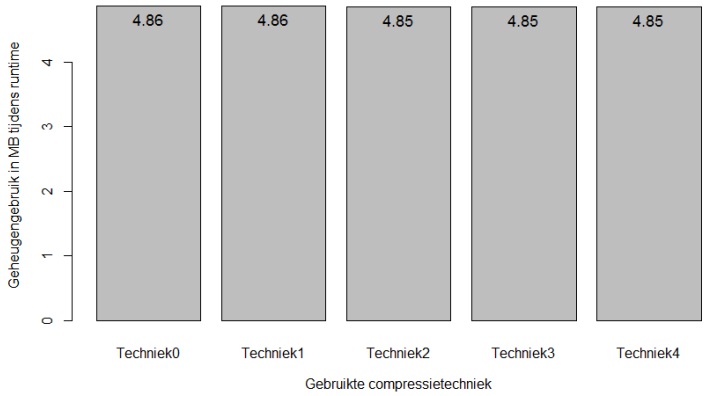
Figuur 3.5: *Resultaten verschillende compressietechnieken op CPU-gebruik foto-app*



3.2.2 GPS-intensieve app

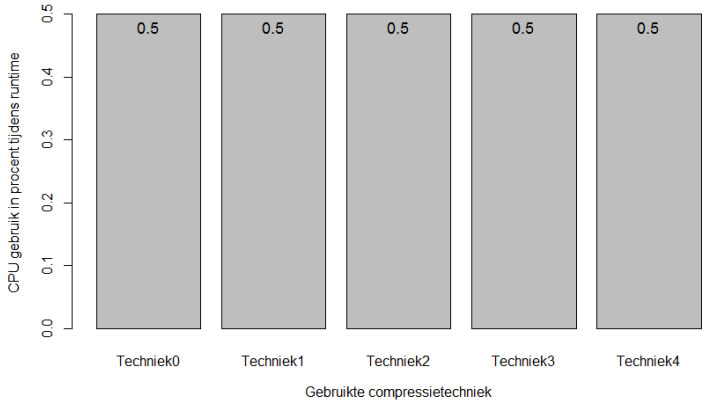
Geheugengebruik

Figuur 3.6: *Resultaten verschillende compressietechnieken op geheugengebruik GPS-intensieve app*



CPU-gebruik

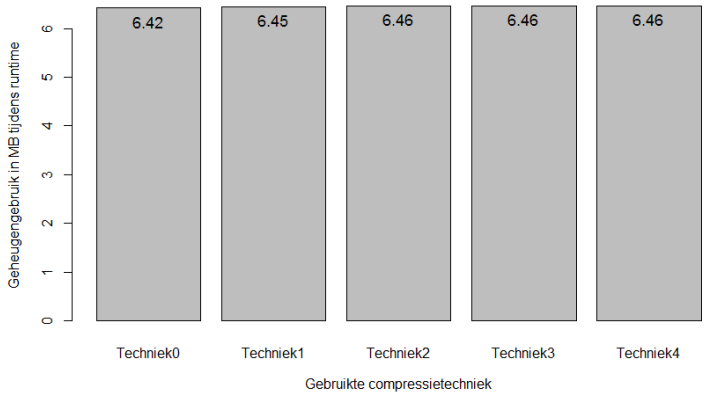
Figuur 3.7: *Resultaten verschillende compressietechnieken op CPU-gebruik GPS-intensieve app*



3.2.3 CPU-intensieve app

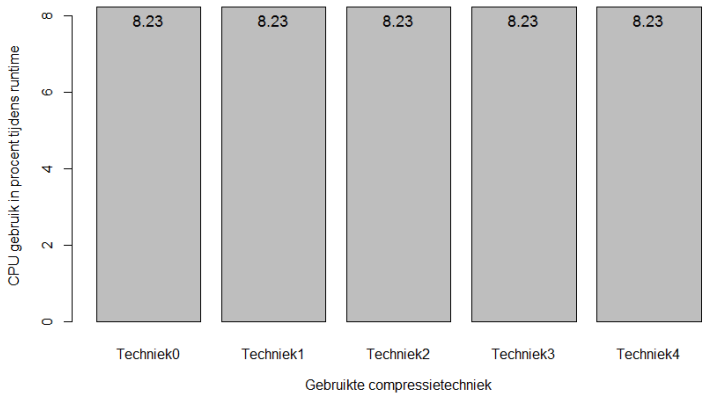
Geheugengebruik

Figuur 3.8: *Resultaten verschillende compressietechnieken op geheugengebruik CPU-intensieve app*



CPU-gebruik

Figuur 3.9: *Resultaten verschillende compressietechnieken op CPU-gebruik CPU-intensieve app*



3.3 Testresultaten framework Brian Pinsard

4. Conclusie

In dit onderzoek werden de verschillende compressietechnieken vergeleken die kunnen gebruikt worden door ontwikkelaars van Android Wear apps. Er werden 10 technieken besproken, waarbij het effect en nut werd toegelicht. Uit deze 10 technieken werden 4 technieken gekozen om toe te passen op 3 apps, die allemaal behoren tot een verschillend app-profiel. Uit de resultaten kunnen we concluderen dat Android Studio al veel van de besproken technieken automatisch toepast. Bij het toepassen van de technieken die niet automatisch uitgevoerd werden, werd er bij de CPU-intensieve app en de GPS-intensieve app geen verschil in de grootte van de APK opgemeten. Enkel bij de proof-of-concept foto-app die gebouwd werd om dit onderzoek te ondersteunen werd een groot verschil in de grootte van de APK opgemerkt. Dit verschil was echter niet zoals we verwacht hadden een verkleining van de APK, maar een vergroting. Dit verschil in grootte van de APK kwam tot stand door het omzetten van de aanwezige afbeeldingen in de res/ folder naar het WebP-formaat. Deze omzetting werd via Android Studio uitgevoerd. Hierbij werd er gekozen om de omzetting lossless te laten gebeuren, zodat we zeker niet moeten inboeten op de kwaliteit van de foto's. De grootte van de APK was hierna gegroeid met 413,33%, wat een zeer ingrijpende verhoging is. Ook het effect op de CPU- en geheugenprestaties van de app werd gecontroleerd voor en na het toepassen van elke techniek. Hierbij werden slechts enkele minieme verschillen opgemerkt van 0.01MB verschil in gebruikt RAM-geheugen en tot maximum 0.22% CPU-gebruik verhoging. Dit ligt in lijn van de verwachtingen die we op voorhand hadden bij het toepassen van de technieken, er moet ergens ingeboet worden om te compenseren dat er compressie wordt toegepast op de bronbestanden. De hoofdconclusie is dus dat de compressietechnieken in het algemeen weinig tot geen effect hebben gehad op de grootte van de APK's.

Appendices

1. Worden bij Cozmos apps gecreëerd voor Android Wear? Zoja, zijn die dan standalone-apps of vooral apps die afhankelijk van de smartphone werken?

De projecten die we bij klanten doen starten tot nu toe altijd vanuit een gewone Android app. Klanten zoeken vaak een oplossing om hun interne processen te vereenvoudigen of ze willen aan hun klanten een app aanbieden. Door het grotere scherm, meer rekenkracht en simpelweg het feit dat bijna iedereen een smartphone bezit ligt de focus volledig daarop. Uiteraard hebben we al een aantal Android Wear projecten gehad maar dit was steeds ter aanvulling op een smartphone app. Voorbeelden hiervan zijn o.a. het tonen van je saldo op je watch of het slagen/falen van een transactie weergeven op je watch.

2. Welke compressietechnieken worden gebruikt bij Cozmos om de grootte van de APK zo klein mogelijk te houden?

De belangrijkste tool om de grootte van een APK te beperken is Proguard. Proguard zorgt er niet enkel voor dat je code moeilijker leesbaar wordt bij reverse engineering maar het verwijdert ook al je ongebruikte code en resources. Bij de grotere klanten waar ook security van groot belang is maken we gebruik van Dexguard. Dexguard is de commerciële variant van Proguard en gaat een stap verder met zijn compressietechnieken maar biedt daarnaast ook tal van andere features aan.

Eén van de andere manieren om de grootte van je APK te gaan beperken is kijken naar de manier waarop je images en andere resources gaat gebruiken. Je kan al ruimte besparen door een juiste keuze te maken tussen jpg/png en verdere compressie toe te passen. Soms is ook geen volledige image nodig maar kan je opteren voor een 9-patch image of om een drawable in XML te definieren. Bij animaties kan je dan weer het aantal frames gaan beperken.

Bij het gebruik van libraries kan het handig zijn om enkel de modules in je project te importeren die je ook daadwerkelijk nodig hebt. Ook kan het nuttig zijn om verschillende libraries tegenover elkaar af te wegen en te kijken welke de laagste method count hebben, het best met het geheugen omgaan, het minste plaats innemen, enz. Stel dat je een library aan je project zou willen toevoegen om images in te laden dan zou je de afweging kunnen maken tussen Glide en Picasso. Picasso is bijvoorbeeld 3,5 keer kleiner dan Glide maar anderzijds gebruikt Glide een pak minder geheugen om een image weer te geven. Dat zijn afwegingen die je moet maken.

Een andere optie is om niet essentiële zaken achterwege te laten uit je APK. Afbeeldingen of data die maar voor een beperkte groep gebruikers van belang zijn kan je beter in realtime downloaden ipv. deze al in je APK te voorzien.

Verder kan je ook door middel van je code zuiniger omspringen met storage. Zou dien je enums te vermijden, kan je (eenvoudige) images renderen, enzovoort.

Deze vraag kan natuurlijk vrij breed beantwoord worden. Er zijn waarschijnlijk nog wel andere dingen die ik nu over het hoofd zie maar ik denk dat bovenstaande al een goed beeld geeft.

3. Wordt bij het gebruiken van compressietechnieken bij Cozmos ook gecontroleerd of deze compressie een effect heeft op de performantie van de app? Zoja, heeft u enig idee op welk vlak van performantie dit een invloed heeft en hoeveel dit een invloed heeft?

Er wordt zeker naar gekeken dat de performantie van de app goed zit. Bij het uitvoeren van network calls moet alles zo compact mogelijk zijn zodat een gebruiker ook onder een slechte verbinding kan blijven werken. Bij het tonen van meerdere afbeeldingen is het memory management zeer belangrijk om o.a. out of memory exceptions te vermijden.

Op welk vlak de performantie een invloed heeft hangt af van ieder geval op zich. Je kan stellen dat er op 1 of meerdere van volgende punten verbetering merkbaar is:

1. Grootte beperken
2. Geheugen gebruik beperken
3. Sneller laden

Echt actief gaan monitoren welke prestatieverschillen een compressietechniek met zich meebrengt daar hebben we meestal de tijd niet voor. Het is pas wanneer er bij het testen dingen naar boven komen dat er echt wordt gekeken welke extra stappen we kunnen ondernemen en welke winst er geboekt wordt. Uiteraard worden bovenstaande technieken tijdens het developen in het achterhoofd gehouden zodat er een optimaal resultaat bekomen wordt.

Bibliografie

David Singleton, V. O. e. A. W. (2017). Android Wear 2.0: Make the most of every minute. https://blog.google/products/android-wear/android-wear-20-make-most-every-minute/.

Dictionary.com. (2017). Wearable definition. http://www.dictionary.com/browse/wearable.

Elinux. (2010). Android aapt. http://elinux.org/Android_aapt.

Google. (2015). Reduce APK size. https://developer.android.com/topic/performance/reduce-apk-size.html.

Google. (2017). Participating banks and supported cards for Android Pay. https://support.google.com/androidpay/answer/6314169?hl=en.

IDC Research, I. (2016). Worldwide Smartwatch Market Will See Modest Growth in 2016 Before Swelling to 50 Million Units in 2020. https://www.idc.com/getdoc.jsp?containerId=prUS41736916.

Initiative, O. S. (2017). The Open Source Definition (Annotated). https://opensource.org/osd-annotated.

Khmelenko, D. (2015). WearTimer. https://github.com/dkhmelenko/WearTimer.

PCMag. (2017). Definition of: APK. http://www.pcmag.com/encyclopedia/term/64037/apk.

Techradar. (2017). Android Wear: everything you need to know. http://www.techradar.com/news/wearables/google-android-wear-what-you-need-to-know-1235025.

Viviani, M. (2014). Wear For Swarm. https://github.com/sealskej/wear-for-swarm.

Lijst van figuren

Lijst van tabellen