Criteria waarop native applicaties ontwikkelen interessanter wordt dan het ontwikkelen van cross-platform applicaties.

De Catelle Jonathan.

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

Promotor: Dhr. G. Blondeel
Co-promotor: Dhr. A. Boel
Academiejaar: 2022–2023
Tweede examenperiode

Departement IT en Digitale Innovatie .

HO
GENT

Woord vooraf

Het schrijven van deze bachelorproef is een uitdaging die ik graag heb aangenomen. Het heeft mij de kans gegeven om mijn kennis en vaardigheden te verbeteren in een gebied van informatica dat me echt interesseert en waar ik een meerwaarde uit krijg voor mijn toekomstige carrière.

Mijn keuze voor dit onderwerp is gebaseerd op mijn interesse in ontwikkeling van mobiele applicaties. Tijdens mijn opleiding had een vak gevolgd dat gerelateerd was aan mobiele applicaties, en dit heeft mijn interesse alleen maar vergroot. Voor mij was de keuze dan snel gemaakt om mijn bachelorproef over mobiele applicaties te houden.

Met de verkregen kennis die ik doorheen dit onderzoek heb opgedaan, kan ik in de toekomst nog altijd iets doen. Het zal mijn beslissingen in verband met applicatieontwikkeling en de keuze van de gebruikte ontwikkelmethode efficiënter maken.

Ik wil graag mijn promotor G. Blondeel bedanken voor zijn hulp, feedback en steun tijdens het schrijven van deze bachelorproef. Daarnaast wil ik ook mijn co-promotor A. Boel bedanken voor zijn technische inzicht en feedback op mijn bachelorproef. En tot slot ook mijn ouders en vriendin voor hun steun en begrip tijdens het schrijven van deze bachelorproef.

Samenvatting

Bij het ontwikkelen van mobiele applicaties is de keuze tussen cross-platform of native ontwikkeling van cruciaal belang. Het maken van de verkeerde keuze kan een project maken of breken. Deze bachelorproef richt zich op het vergelijken van ontwikkeltijden, prestaties en schaalbaarheid van verschillende functionaliteiten tussen native en cross-platform ontwikkeling. Het onderzoek beantwoordt de vraag: "Hoe verschillen de ontwikkeltijden, prestaties en schaalbaarheid van functionaliteiten tussen native en cross-platform ontwikkeling van mobiele applicaties?"

Het onderzoek is uitgevoerd aan de hand van een specifiek stappenplan. Blanco projecten zijn aangemaakt om van daaruit de ontwikkeltijd, prestaties en schaalbaarheid van verschillende functionaliteiten te meten. De resultaten zijn als volgt:

Voor basisfunctionaliteiten presteert native ontwikkeling beter dan cross-platform ontwikkeling, met een snellere opstarttijd en lager geheugengebruik. Beide methoden zijn schaalbaar en kunnen bestaande schermen hergebruiken.

Bij audio- en videospelers presteren native applicaties over het algemeen beter dan cross-platform applicaties. Native applicaties hebben minder geheugengebruik en lager CPU-gebruik. Beide methoden kunnen de logica abstracteren en hergebruiken.

Bij het gebruik van sensoren presteren native applicaties gelijkaardig aan cross-platform applicaties. Native applicaties zijn wel sneller en hebben een lager geheugengebruik. Beide methoden zijn schaalbaar en kunnen sensoren gemakkelijk uitbreiden en bestaande logica hergebruiken.

Bij het aanmaken van notificaties is er een groot verschil tussen Android en React Native. Android heeft een kortere creatietijd en lager geheugengebruik. Beide methoden hebben vergelijkbare ontwikkeltijd en schaalbaarheid.

Uit het onderzoek blijkt dat native ontwikkeling over het algemeen betere prestaties levert dan cross-platform ontwikkeling voor de onderzochte functionaliteiten. Het onderzoek biedt applicatie-ontwikkelaars en andere belanghebbenden inzicht in de verschillen tussen native en cross-platform ontwikkeling, zodat zij een weloverwogen beslissing kunnen nemen bij het kiezen van de ontwikkelmethode.

Inhoudsopgave

Lijst van figuren

Woordenlijst

bottlenecks verwijst naar een onderdeel van een systeem, een proces of een stuk code dat de algehele prestaties, efficiëntie of snelheid beperkt. 21

chocolatey is een pakketbeheerder voor het Windows besturingssysteem. Het biedt een opdrachtregelinterface (CLI) waarmee u verschillende softwarepakketten op uw Windows-machine kunt installeren, bijwerken en beheren. Choco vereenvoudigt het installatieproces van software door de download-, installatie- en configuratietaken te automatiseren. 25

codebase is een verzameling broncode die wordt gebruikt om een bepaald softwaresysteem, toepassing of softwarecomponent te bouwen. 5

emulator is een softwareprogramma waarmee een computersysteem zich kan gedragen als een ander systeem. Hiermee kunnen ontwikkelaars een virtuele omgeving op hun computer creëren die zich gedraagt als een ander hardware- of softwaresysteem, zodat zij hun toepassingen op verschillende platforms kunnen testen en debuggen zonder fysieke toegang tot dat platform nodig te hebben. 13

garbage collector ofwel GC is een geheugenherstelfunctie ingebouwd in programmeertalen. 12

hot reload is een functie in sommige softwareontwikkelingsomgevingen waarmee ontwikkelaars snel de wijzigingen kunnen zien die zij aanbrengen in de code van een lopende applicatie of programma. Wanneer een ontwikkelaar de code wijzigt, worden de wijzigingen automatisch gecompileerd en vervolgens in het lopende programma geïnjecteerd, zonder dat het programma gestopt en opnieuw gestart hoeft te worden. 16

jetBrains is een bedrijf dat geïntegreerde ontwikkelingsomgevingen (IDEs) aanbiedt voor verschillende programmeertalen. 11

libraries zijn verzamelingen van vooraf geschreven code die programmeurs kunnen gebruiken om taken te optimaliseren. Deze verzameling herbruikbare code is meestal gericht op specifieke veel voorkomende problemen. Een bibliotheek bevat meestal een aantal verschillende voorgecodeerde componenten. 11

metro is de standaard JavaScript-bundler die wordt gebruikt tijdens de ontwikkeling. Het is verantwoordelijk voor het transformeren en bundelen van uw JavaScript-code en andere data (zoals afbeeldingen, lettertypen, enz.) in een formaat dat kan worden begrepen en uitgevoerd door het mobiele apparaat of de simulator. 27

node is een runtime-omgeving waarmee u JavaScript-code buiten een webbrowser kunt uitvoeren. In het geval van React Native wordt Node.js gebruikt als onderliggende runtime-omgeving voor het uitvoeren van JavaScript-code op de server en het bouwen van de React Native applicatie. Het biedt de nodige tools en bibliotheken om JavaScript-code uit te voeren en te communiceren met de native API's van het apparaat via het React Native-framework. 25

overhead verwijst naar de extra bronnen die nodig zijn om een bepaalde taak uit te voeren, maar die niet direct bijdragen aan de hoofdfunctie of het gewenste resultaat. Overhead kan verschillende vormen aannemen zoals extra code, berekeningen, geheugen, verwerkingskracht of netwerkverkeer. 30

superset is een programmeertaal die alle kenmerken/functionaliteiten van een andere taal bevat plus extra kenmerken. De term "superset" wilt zeggen dat de taal een grotere of uitgebreidere versie is van de andere taal. 13

Acroniemen

ANDK Android Native Development Kit, is een toolset waarmee u delen van uw app in native code kunt implementeren. 11

CMS Content Management System, is een systeem dat bedrijven helpt hun digitale inhoud te beheren. 7

CRM Customer Relationship Management, is een technologie voor het beheer van alle relaties en interacties van uw bedrijf met klanten en potentiële klanten. 7

IDE Integrated Development Environment, is software voor het bouwen van toepassingen die gemeenschappelijke developer tools combineert in een enkele grafische gebruikersinterface (GUI). 10

JDK Java SE Development Kit, is een softwareontwikkelingskit van Oracle met tools en bibliotheken voor de ontwikkeling van Java-toepassingen. React Native gebruikt Java voor het bouwen en uitvoeren van Android-toepassingen. 25

SDK Software Development Kit, is een verzameling van hulpmiddelen voor softwareontwikkeling, bibliotheken, documentatie en codevoorbeelden die ontwikkelaars gebruiken om toepassingen te maken voor een bepaald softwareplatform, zoals een specifiek besturingssysteem of een specifieke programmeertaal. 13

1

Inleiding

Bij het ontwikkelen van een mobiele applicatie is de keuze tussen native of cross-platform ontwikkeling altijd een belangrijke beslissing, aangezien de gekozen ontwikkelmethode veel invloed heeft op het ontwikkelproces. Indien er gekozen wordt om native te ontwikkelen zijn er twee teams nodig, één voor elk platform, of één team dat de vaardigheden en tijd beschikt voor beide platformen. Daarnaast moet de klant over een budget beschikken dat groot genoeg is om eventueel native te kiezen als ontwikkelmethode, aangezien native een hogere kostprijs met zich meebrengt in tegenstelling tot cross-platform.

Beide ontwikkelmethodes hebben hun voor- en nadelen, en de gekozen ontwikkelmethode hangt af van deze voor- en nadelen. Native ontwikkeling wordt vaak gebruikt wanneer performantie cruciaal is, omdat het platform-specifieke code en speciaal ontworpen frameworks gebruikt om de applicatie te ontwikkelen en runnen. In vergelijking met cross-platform ontwikkeling dat gebruik maakt van een ”write once, run anywhere-principe. Ontwikkelaars schrijven één applicatie die op IOS en Android werkt. Daarnaast wordt cross-platform vaak gebruikt voor simpele, niet-grafisch intensieve applicaties of bij applicaties met een hoge tijdsdruk.

Ondanks het feit dat native en cross-platform al vaak vergeleken zijn met elkaar wordt er nooit veel tijd gespendeerd om individuele functionaliteiten te vergelijken, wat in sommige gevallen belangrijk kan zijn. Daarom wordt er in deze bachelorproef gekeken naar de functionaliteiten die mobiele applicaties kunnen bevatten. Met andere woorden de functionaliteiten die in deze bachelorproef worden onderzocht, worden vergeleken op basis van hun ontwikkeltijd, performantie en schaalbaarheid tussen native en cross-platform ontwikkeling. Op die manier ken er een conclusie getrokken worden uit de resultaten om daarna op basis van de gewenste functionaliteiten een ontwikkelmethode te kiezen.

Om te kunnen focussen op de verschillen tussen native en cross-platform ontwikkeling en niet de verschillen tussen de native platformen zelf, wordt het onderzoek in deze bachelorproef langs de kant van native ontwikkeling uitgevoerd met één platform. Tijdens de literatuurstudie zal er besloten worden welk platform het meest geschikt is om het onderzoek op uit te voeren.

1.1. Probleemstelling

Voor veel bedrijven en ontwikkelaars is de beslissing tussen native of cross-platform om mobiele applicaties te ontwikkelen een moeilijke keuze. Er moeten verschillende factoren in overweging worden genomen zoals tijd, budget, performantie, schaalbaarheid en functionaliteiten. Indien de verkeerde keuze wordt gemaakt, kan dit een project maken of kraken. Bij native ontwikkelen kan een project snel veel tijd en geld kosten. Cross-platform daarentegen is een snellere en goedkopere oplossing en kan daardoor een betere keuze zijn.

1.2. Onderzoeksvragen

1.2.1. Hoofdonderzoeksvraag

- Hoe verschillen de ontwikkeltijden, prestaties en schaalbaarheid van functionaliteiten tussen native en cross-platform ontwikkeling van mobiele applicaties?

Om deze vraag te beantwoorden worden de functionaliteiten vergeleken op basis van hun ontwikkeltijd, performantie en schaalbaarheid gebruikmakend van native en cross-platform ontwikkelmethodes. Voor de performantie wordt er gekeken naar de tijdsduur die een bepaalde actie nodig heeft, het CPU en het geheugengebruik van een applicatie. Voor de schaalbaarheid wordt er eerst gekeken of de functionaliteit wel schaalbaar is. Indien deze schaalbaar is, wordt er gekeken hoe dit precies zou gebeuren en hoe gemakkelijk of moeilijk het is om de functionaliteit op te schalen. Voor de ontwikkeltijd wordt er in het groot gekeken naar hoeveel uren werk nodig zijn om een functionaliteit te implementeren. Daarnaast worden ook eventuele problemen of bugs gedocumenteerd en wordt er gekeken hoelang het duurt om deze op te lossen. Tot slot wordt er in het algemeen gekeken naar de compiletijd die de ontwikkelmethodes nodig hebben om een applicatie te bouwen. Met andere woorden hoelang duurt het om tijdens ontwikkeling een app te bouwen en/of veranderingen te zien.

1.2.2. Deelonderzoeksvragen

Daarnaast zijn er twee ondersteunende deelonderzoeksvragen die bij het onderzoek horen.

- Zijn er functionaliteiten die cross-platform niet ondersteunen?
- Zijn er functionaliteiten bij cross-platform waarbij de performantie de functionaliteit onbruikbaar maakt?

Dankzij deze deelonderzoeksvragen wordt er een beter beeld geschetst over mogelijke functionaliteiten die niet ondersteund worden of niet bruikbaar zijn bij cross-platform ontwikkeling. Het kan zijn dat cross-platform alle functionaliteiten ondersteunt, maar het kan ook zijn dat er bepaalde functionaliteiten zijn die cross-platform niet zal ondersteunen of waarbij het verschil in performantie de functionaliteit onbruikbaar maakt.

1.3. Onderzoeksdoelstelling

Dit onderzoek zal applicatie-ontwikkelaars, ondernemingen en andere geïnteresseerden een beter inzicht geven in de verschillen van functionaliteiten bij native en cross-platform ontwikkeling. Hierdoor zijn ze beter in staat om een beslissing te maken voor de te gebruiken ontwikkelmethode. Daarnaast zal het ook meer inzicht geven in de ontwikkeltijd, performantie en schaalbaarheid van functionaliteiten. Het onderzoek zal ook helpen bepalen of een functionaliteit ondersteund wordt/bruikbaar is bij cross-platform ontwikkeling ofwel onbruikbaar gemaakt wordt door de performantie van de functionaliteit.

1.4. Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:
In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.
In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.
In Hoofdstuk 4 wordt uitgelegd hoe de ontwikkelomgevingen van de gebruikte IDEs en frameworks werd opgesteld.
In Hoofdstuk 5 wordt uitgelegd hoe nieuwe projecten worden aangemaakt om de functionaliteiten te implementeren.
In Hoofdstuk 6 wordt het onderzoek naar basisfunctionaliteiten uitgevoerd.
In Hoofdstuk 7 wordt het onderzoek naar audio- en videospelers uitgevoerd.
In Hoofdstuk 8 wordt het onderzoek naar gebruik van sensoren uitgevoerd.
In Hoofdstuk 9 wordt het onderzoek naar push-notificaties uitgevoerd.
In Hoofdstuk 10, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen.

2

Stand van zaken

Voordat de bachelorproef van start kan gaan, is het belangrijk om inzicht te krijgen in de basis van mobiele applicaties en hoe ze precies ontwikkeld worden. Pas na deze uitleg kan het onderzoek worden uitgevoerd. Eerst worden mobiele applicaties algemeen uitgelegd alsook de verschillende soorten mobiele applicaties. Daarna worden de ontwikkelmethodes en hun verschillen uitgelegd die gebruikt worden tijdens het onderzoek.

2.1. Inleiding mobiele applicatie-ontwikkeling

De ontwikkeling van mobiele applicaties is het proces waarbij er software wordt gemaakt voor smartphones, tablets, televisies en digitale assistenten (Palko, 2021). Deze applicaties worden ontwikkeld voor de besturingssystemen Android en IOS. De software kan op verschillende manieren op een apparaat komen. Het kan gedownload worden uit een app store, vooraf op het apparaat worden geïnstalleerd of het kan worden geopend via een mobiele webbrowser (IBM, 2023).

Mobiele applicaties worden in veel sectoren gebruikt zoals telecommunicatie, e-commerce, verzekeringen, gezondheidszorg, overheid enz. Ze worden in deze sectoren veel gebruikt omdat het een gemakkelijke en populaire manier is om mensen en bedrijven in contact te brengen met elkaar en met het internet. Op deze manier kunnen bedrijven relevant, responsief en succesvol blijven.

Er bestaan verschillende soorten mobiele applicaties zoals native, web en hybride applicaties (AWS, 2023). Elke soort mobiele applicatie is een manier waarop applicaties op smartphones of computers worden geïnstalleerd of werken.

2.1.1. Native applicaties

Dit is de meest gekende en gebruikte soort mobiele applicaties. Hierbij wordt een applicatie gedownload en geïnstalleerd op een apparaat. Het is een applicatie die specifiek wordt ontwikkeld voor één besturingssysteem, zoals Android of IOS (van Laarhoven, 2021). Native applicaties zijn vaak sneller en gebruiksvriendelijker dan een web of hybride applicatie. Dit komt omdat de applicaties platformspecifieke software gebruiken.

Voordelen van native applicaties

Platformspecifieke code

Omdat er platformspecifieke code wordt gebruikt, heeft de applicatie direct toegang tot de interne APIs van een apparaat (AWS, 2023). Hierdoor zal ook de performantie hoger liggen dan bij web of hybride applicaties. Ook hebben ze daardoor direct toegang tot bepaalde functionaliteiten zoals camera, gps, versnellingsmeter, kompas, lijst van contacten, enz. Daarnaast hebben ze ook toegang tot het notificatiesysteem en werken ze offline in tegenstelling tot web applicaties (Budiu, 2016).

Fouten vermijden of oplossen

Omdat het niet nodig is om twee applicaties te onderhouden vanuit één codebase, kan het gemakkelijker zijn om fouten bij een bepaald platform op te sporen of compleet te vermijden (Koffer, 2023).

User interface

Nog een ander voordeel van platformspecifieke code en het gebruik van interne APIs is de UI. Omdat de applicatie gebruik maakt van de interne APIs zal de UI consistenter zijn. Dit zal voor een betere gebruikerservaring zorgen aangezien de applicatie gebruiksvriendelijker aanvoelt (Kotlin, 2023).

App store ondersteuning

Dankzij hun betere performantie en snelheid zullen native applicaties populairder zijn in de app store. Het is ook gemakkelijker om native applicaties te publiceren in de app store (Koffer, 2023).

Nadelen van native applicaties

Kost en onderhoudsprijs

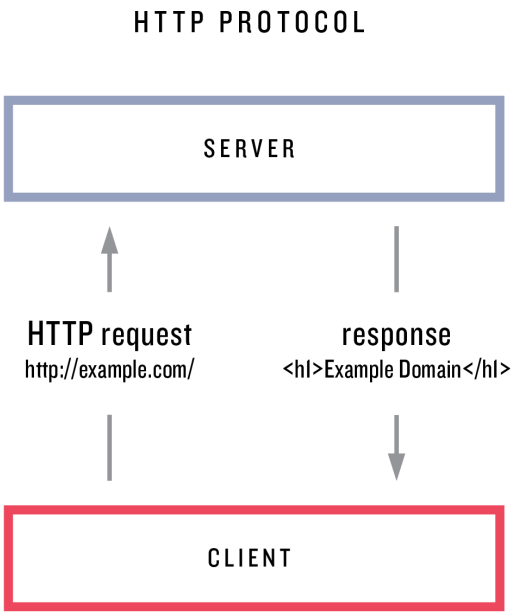
Aangezien er een applicatie ontwikkeld wordt voor twee platformen, zal de kostprijs en ontwikkeltijd van een applicatie hoger liggen dan wanneer er één applicatie ontwikkeld moet worden. Daarnaast moeten beide applicaties worden onderhouden nadat ze ontwikkeld zijn. Het is dus niet enkel de kostprijs die hoger zal zijn. Ook de onderhoudsprijs zal hoger liggen (AWS, 2023).

Verschillende logica
Doordat de applicatie twee keer ontwikkeld wordt, is het mogelijk dat de business logica op bepaalde plaatsen in de applicatie verschillend is van elkaar. Er moet dus altijd aandachtig getest worden zodat dit niet het geval is (Kotlin, 2023).

Downloaden
Zoals eerder gezegd, wordt de applicatie gedownload vanop een app store 2.1.1. Om die download te doen, is er een internetverbinding nodig, pas daarna kan de applicatie indien mogelijk offline werken.

2.1.2. Web applicaties

Web applicaties zijn geen echte applicaties maar het zijn mobiele versies van responsive websites. Een web applicatie simuleert een native applicatie en wordt niet geïnstalleerd op een apparaat (Beeproger, 2023). Ze zullen dus net zoals bij websites een HTTP request naar een server sturen die op zijn beurt dan een HTTP response zal terugsturen.



Figuur (2.1)
Request-response cycle van HyperText Transfer Protocol (Hartl, 2019).

Aangezien web applicaties niet geïnstalleerd worden, is het dus wel nodig om een internetverbinding te hebben bij het gebruik ervan. Ze worden vaak gebruikt door bedrijven om informatie en diensten aan hun klanten op een veilige manier aan te bieden (Nehra, 2023).

Een web applicatie is bereikbaar door een unieke URL in te voeren in een browser (Beeproger, 2023). Microsoft Word is een voorbeeld van een native applicatie die gebruikers moeten installeren op hun apparaat, in vergelijking met Google Docs dat een web applicatie is en dat niet geïnstalleerd moet worden (Nehra, 2023).

Web applicaties worden op dezelfde manier gemaakt als websites. Ze maken gebruik van HTML, CSS en JavaScript. De meeste web applicaties gebruiken een web server voor het verwerken en beheren van requests, een applicatieserver om de gevraagde taken te voltooien en een database om de gevraagde data op te halen (Varsha, 2023).

Voordelen van web applicaties

Kostprijs

Aangezien één web applicatie op alle platformen draait, zal dit veel goedkoper uitkomen dan bij native applicaties (van Laarhoven, 2021). Er moet namelijk maar één applicatie ontwikkeld worden die op alle platformen zal draaien.

Updates

Omdat web applicaties aan een URL gelinkt zijn, is het gemakkelijk om updates uit te voeren op de applicatie. De URL wordt vanzelf periodiek geüpdatet. Hierdoor wordt de applicatie automatisch ook up to date gehouden ongeacht het platform dat een gebruiker gebruikt (Varsha, 2023).

Installatie & Compatibiliteit

Daarnaast moet, zoals reeds vermeld, een web applicatie niet geïnstalleerd worden. Ze is beschikbaar op alle platformen en browsers. Hierdoor is ze zeer toegankelijk voor gebruikers. Er is enkel een actieve internetverbinding nodig.

Integratie

Web applicaties hebben daarnaast ook nog de mogelijkheid om gemakkelijk met andere web applicaties te integreren. Ze kunnen bijvoorbeeld met CMS of CRM systemen werken om data te beheren (Nehra, 2023).

App store

Er is geen goedkeuring nodig van een app store om de app te gebruiken of te updaten. Omdat een web applicatie via een browser wordt geopend, is er zelfs geen app store nodig (Varsha, 2023).

Data veiligheid

Tot slot heeft de nood aan een internetverbinding nog wel een voordeel. Indien een apparaat crasht of vastloopt, zullen de gegevens niet verloren gaan. Dankzij

de internetverbinding worden de gegevens automatisch opgeslagen op de server (Nehra, 2023).

Nadelen van web applicaties

Internetverbinding

Het grootste nadeel bij web applicaties is dat er ten alle tijde een internetverbinding nodig is om met de applicatie te kunnen werken (Varsha, 2023). Indien er geen actieve internetverbinding aanwezig is, zal het niet mogelijk zijn om de applicatie te runnen.

Functionaliteiten

In vergelijking met native applicaties, die gebruik kunnen maken van alle functionaliteiten van een apparaat, is het voor web applicaties niet mogelijk om alle functionaliteiten van een apparaat te gebruiken (van Laarhoven, 2021).

Verschil met een website

Het verschil tussen een web applicatie en een website is niet altijd duidelijk. Zoals eerder gezegd is een web applicatie software die toegankelijk is door er naar te surfen via een browser. Enkele voorbeelden van web applicaties zijn: Google apps, Amazon en Youtube. Het verschil met een website is dat een website een collectie is van gerelateerde web applicaties. Het bevat statische data zoals foto's, tekst, audio, video's... en kan bestaan uit meerdere pagina's (sugandha, 2022). Een voorbeeld is de website van een bank waarop je de diensten en openingsuren kan zien.

2.1.3. Hybride applicaties

Het spreekt voor zich, dat een hybride applicatie een combinatie is van native en web applicaties (Blaž Denko, 2021). Een hybride applicatie kan er identiek uitzien als een native applicatie maar de manier waarop een hybride applicatie wordt opgebouwd is verschillend (Beeproger, 2023). Het heeft als basis een web applicatie die in een soort van container wordt gestoken. Deze container zorgt er dan voor dat in de applicatie een browser aanwezig is die de applicatie kan runnen. Hierdoor kan de applicatie ook in de app store komen waardoor het mogelijk is om hem te installeren.

Bedrijven maken vaak hybride applicaties als omhulsel voor een bestaande web applicatie. Op die manier proberen ze hun gebruikersaantal te vergroten door een aanwezigheid te hebben in de app store zonder al te veel moeite (Budiu, 2016).

Voordelen van hybride applicaties

Ontwikkelingskosten

Omdat hybride applicaties platform onafhankelijke ontwikkeling mogelijk maakt, kunnen bedrijven op die manier de ontwikkelingskosten laag houden (Budiu, 2016).

Offline gebruik & installatie

Eigenlijk hebben hybride applicaties alle voordelen van web applicaties, alleen krijgen ze nu nog eens de extra mogelijkheid om applicaties offline te gebruiken en om ze te installeren op een apparaat.

Nadelen van hybride applicaties

UI verschil

Aangezien hybride applicaties op verschillende platformen kunnen werken, is het mogelijk dat de GUI er verschillend uitziet (sgshradha, 2019).

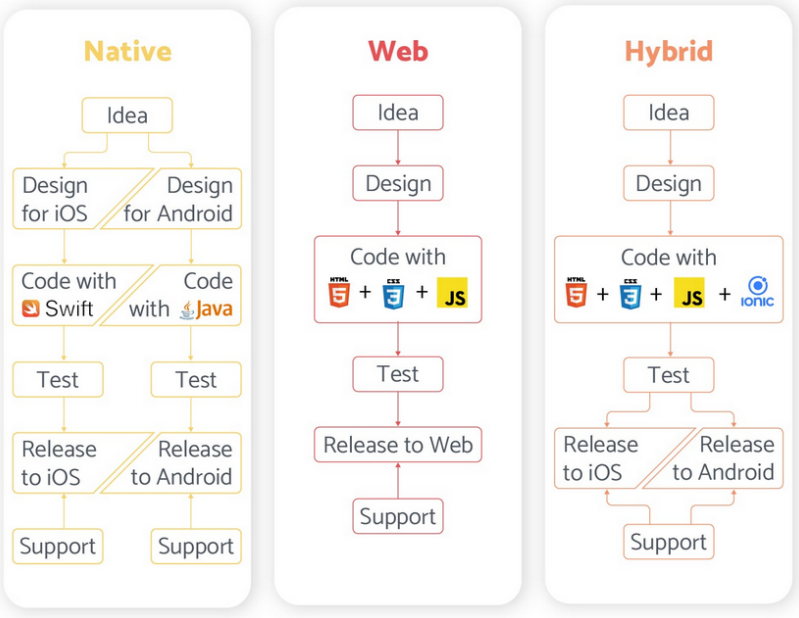
Testing

De mogelijkheid die hybride applicaties met zich meebrengen om te werken op alle platformen, zorgt ervoor dat een applicatie op al deze platformen getest moet worden (sgshradha, 2019).

Offline gebruik

Dankzij de combinatie van native en web applicaties, kunnen hybride applicaties ook offline en online werken (Khan, 2021). De applicaties kunnen wel enkel offline werken als de applicatie niet afhankelijk is van data op een database (sgshradha, 2019). Indien dit het geval is, kan het zijn dat de applicatie niet naar behoren werkt als er geen internetverbinding aanwezig is.

2.1.4. Samenvatting soorten mobiele applicaties



Figuur (2.2)

Native vs Web vs Hybride applicaties (Merenych, 2021).

Op bovenstaande figuur is een mooi overzicht te zien hoe de verschillende soorten applicaties werken. Elke soort applicatie heeft een andere manier waarop ze op een platform geïmplementeerd worden.

2.2. Native ontwikkeling

2.2.1. Wat is native ontwikkeling?

Native ontwikkeling is het proces om native 2.1.1 applicaties te maken voor een specifiek besturingssysteem, zoals Android of IOS. Hierbij wordt er gebruik gemaakt van de bijhorende programmeertalen en ontwikkelingsomgevingen die door de platformen worden aangeboden (Meirelles, 2019). Ook kunnen ontwikkelaars zoals eerder gezegd gebruik maken van platform specifieke APIs die toegang verlenen tot de camera, gps, versnellingsmeter, kompas, lijst van contacten, enz...

2.2.2. Voordelen van native ontwikkeling

Schaalbaarheid

Dankzij de flexibiliteit bij het ontwikkelen van native applicaties en het scheiden van de ontwikkeling ervan, zijn native applicaties zeer schaalbaar (Koffer, 2023). Ook hebben ontwikkelaars de mogelijkheid om elk platform individueel te schalen (Sakovich, 2023a).

2.2.3. Nadelen van native ontwikkeling

Ontwikkelingsteams

Een reden van de hoge kost of onderhoudsprijs bij native mobiele applicaties zullen de ontwikkelingsteams zijn. Bij de ontwikkeling van een applicatie voor beide platformen, zal er gewerkt moeten worden met ofwel één team met de kennis om voor beide platformen te ontwikkelen ofwel zal er met twee teams gewerkt moeten worden die elk een applicatie voor één platform maken (Kotlin, 2023).

Onderhoudstijd

Na het ontwikkelen van een native applicatie kan het zijn dat er wijzigingen komen of updates moeten gebeuren. Aangezien er niet wordt gewerkt met één centraal codebase, maar met twee onafhankelijke projecten, zullen de wijzigingen of updates op beide projecten uitgevoerd moeten worden (Kotlin, 2023).

2.2.4. Programmeertalen en frameworks voor native ontwikkeling

Zoals er reeds werd besproken, gaat native ontwikkeling een applicatie ontwikkelen specifiek voor één besturingssysteem. Op die applicatie kunnen verscheidene programmeertalen, frameworks en IDE gebruikt worden. Deze zijn allemaal specifiek voor ofwel Android of IOS. In de volgende secties worden enkele programmeertalen overlopen die gebruikt kunnen worden om het onderzoek uit te voeren.

Android programmeertalen

Java

Java is een all-round programmeertaal met meerdere doeleinden. Eén van die doeleinden is onder andere Android applicatie ontwikkeling. Het was de eerste officiële programmeertaal voor Android applicatie ontwikkeling. Maar het blijft vandaag de dag nog steeds de meestgebruikte (harkiran, 2022). Aangezien het de meest gebruikte programmeertaal is, is het soms gemakkelijker om er mee te werken aangezien eventuele bugs meestal al zijn opgelost door de community (Thorndyke, 2021). Dat wil niet zeggen dat het een gemakkelijke programmeertaal is om mee te werken. Java kan soms zeer complex zijn waardoor onervaren ontwikkelaars het lastig kunnen krijgen (Kesavan, 2021). Daarnaast is Java mede door zijn complexiteit een zeer robuust programma dat een heleboel voordelen met zich meebrengt zoals flexibiliteit, portabiliteit en herbruikbaarheid (Kesavan, 2021).

Kotlin

Momenteel is Kotlin de officiële programmeertaal voor Android applicatie ontwikkeling. Het is ontwikkeld door Google en JetBrains als lichtere en meer gebruiksvriendelijke manier om Android applicaties te maken (Thorndyke, 2021). In vergelijking met Java is er geen nood aan puntkomma's en is er minder code nodig om dezelfde applicatie te maken. Daarnaast laat Kotlin toe om variabelen te maken zonder ze op voorhand te moeten definiëren (Thorndyke, 2021). Het wordt door veel bedrijven gebruikt omwille van de herbruikbaarheid van code alsook het gebruik van externe libraries (Kesavan, 2021).

C++

C++ is een alternatieve methode voor Android applicatie ontwikkeling. Om C++ te gebruiken moet de ANDK gebruikt worden. De applicatie kan echter niet compleet met C++ worden gemaakt, maar C++ en de ANDK kunnen wel gebruikt worden om libraries te ontwikkelen waarvan een applicatie gebruik maakt (harkiran, 2022). Het gebruik van C++ voor libraries wordt vaak toegepast voor de performantie die C++ met zich meebrengt. Er moet wel rekening mee worden gehouden dat het een moeilijke programmeertaal is om te leren en dus niet voor iedereen weggelegd is (Designveloper, 2022).

C#

Origineel is C# ontwikkeld als object-georiënteerde programmeertaal, die gebruikt zou worden om desktop, mobiele en web applicaties te maken. Het was een taal die voortkwam vanuit C en werd ontwikkeld door Microsoft (Designveloper, 2022). Het is niet alleen omdat het al 20 jaar bestaat dat het een populaire keuze is voor ontwikkelaars maar C# heeft toegang tot het .NET framework (Kesavan, 2021). Het .NET framework biedt heel wat tools en libraries aan die helpen bij het ontwikkelen

van Android applicaties. Daarnaast heeft C# nog een aantal andere voordelen. C# heeft cross-platform capaciteiten waardoor er code over platformen heen gedeeld kan worden en er is een garbage collector, die ervoor zorgt dat er geen geheugenlekken zijn (Patel, 2023).

Dart

Dart is een open-source programmeertaal gemaakt door Google, die werd ontworpen om op Android platformen te runnen (Kesavan, 2021) met de bedoeling om klanten een geoptimaliseerde manier te geven snel applicaties op het Android platform te krijgen (harkiran, 2022). Dit wordt gerealiseerd door te focussen op UI development. De wijzigingen worden snel aan de ontwikkelaar getoond dankzij de hot-reload. Tot slot is Dart ook gekend voor zijn performantie en de mogelijkheid om te compilen tot ARM en x64 code (harkiran, 2022).

Corona

Ondanks het feit dat het minder populair is, is Corona zeker geen slecht alternatief. Het beschikt over de capaciteit om de meeste Android applicaties te runnen (Kesavan, 2021). Corona is ook geen programmeertaal op zich, maar een development kit die gebruikt kan worden om Android applicaties te maken door gebruik te maken van Lua. Lua is een lichte, efficiënte en open-source programmeertaal die object-georiënteerd, functioneel, data-driven... programmeren ondersteunt (Lua, 2021). Om met Corona te werken, worden er twee operationele modes gebruikt namelijk de Corona Simulator en Corona Native. De Corona Simulator wordt gebruikt om applicaties te maken en Corona Native wordt gebruikt om de Lua code te integreren met een Android Studio project om applicaties te maken die native functionaliteiten gebruiken (harkiran, 2022).

IOS programmeertalen

Swift

Swift wordt niet meer enkel en alleen gebruikt om mobiele applicaties te ontwikkelen. Swift wordt ook gebruikt voor besturingssystemen zoals Windows en Linux. Het werd ontwikkeld door Apple als een open-source opvolger van alle C-gebaseerde programmeertalen zoals Objective-C, C++ en C (Coursera, 2022). Er zijn 3 hoofdzakelijke voordelen waardoor Swift zeer populair is geworden bij ontwikkelaars: snelheid, veiligheid en introductie niveau (yuvraj, 2022). Als opvolger van alle C gebaseerde programmeertalen, heeft het ook een betere performantie bij het uitvoeren van de meeste taken. Daarnaast kunnen variabelen ook aangemaakt worden zonder op voorhand een type te definiëren en puntkomma's zijn niet verplicht (Thorndyke, 2021).

C#

Net zoals bij Android kan C# ook gebruikt worden om IOS applicaties te ontwikkelen. En zoals eerder gezegd, is C# een object-georiënteerde programmeertaal die geïntegreerd is met het .NET framework 2.2.4 (yuvraj, 2022). Sinds zijn ontstaan in 2000 heeft C# doorheen de jaren heel wat aan populariteit gewonnen dankzij de eenvoudige en hoogwaardige architectuur (yuvraj, 2022). Momenteel is C# gerangschikt als de zevende populairste programmeertaal (Johns, 2023). C# gebruikt niet alleen de tools en libraries die .NET te bieden heeft, maar het maakt ook gebruik van de .NET runtime omgeving (Pruciak, 2022).

Objective-C

Objective-C was de originele programmeertaal voor Apple en de fundering van MacOS en IOS (Johns, 2023). Het is een superset van C, het breidt C dus uit en het voegt nieuwe functionaliteiten eraan toe (Johns, 2023). Net zoals C# is Objective-C een object-georiënteerde programmeertaal (Pruciak, 2022). Daarnaast bestaat het wel al langer dan C#, Objective-C is ontwikkeld in de vroege 1980's (Pruciak, 2022). Aangezien het van C komt, is het ook mogelijk om C code in een Objective-C klasse toe te voegen. Daardoor worden Objective-C klassen simpeler, flexibeler en meer schaalbaar voor mobiele applicaties (yuvraj, 2022). Dankzij de eenvoud en betere runtijd is Objective-C een van de meest gekozen programmeertalen, vooral dankzij de krachtige SDK die worden gebruikt.

2.2.5. Tools en IDEs voor native ontwikkeling

Android Studio

Android Studio is een populaire IDE die vaak wordt gebruikt in combinatie met Kotlin om native Android applicaties te ontwikkelen. Het wordt vaak door ontwikkelaars gebruikt omwille van de ingebouwde emulator die Android Studio bevat. Hiermee kunnen ontwikkelaars verschillende apparaten met verschillende Android versies uittesten (Medewar, 2022). Daarnaast ondersteunt de emulator ook verschillende features die normaal enkel toegankelijk zijn bij fysieke apparaten, zoals toegang tot de camera, locatie, batterij instellingen, enz. (Okeke, 2022b). Tot slot biedt Android Studio ook een krachtige manier om layouts te maken. Ontwikkelaars kunnen met een visuele design editor schermen opbouwen in plaats van code te schrijven, die schermen worden dan automatisch door Android Studio omgezet naar xml bestanden (Medewar, 2022). Het ondersteunt ook GitHub integratie om gemakkelijk aan grote projecten te werken (Studio, 2023).

Eclipse

Eclipse is ontwikkeld in 2001 maar over de jaren is het uitgegroeid tot een IDE dat meerdere programmeertalen ondersteunt (Medewar, 2022). Ondanks de ondersteuning van meerdere programmeertalen wordt het nog altijd vaak gebruikt om

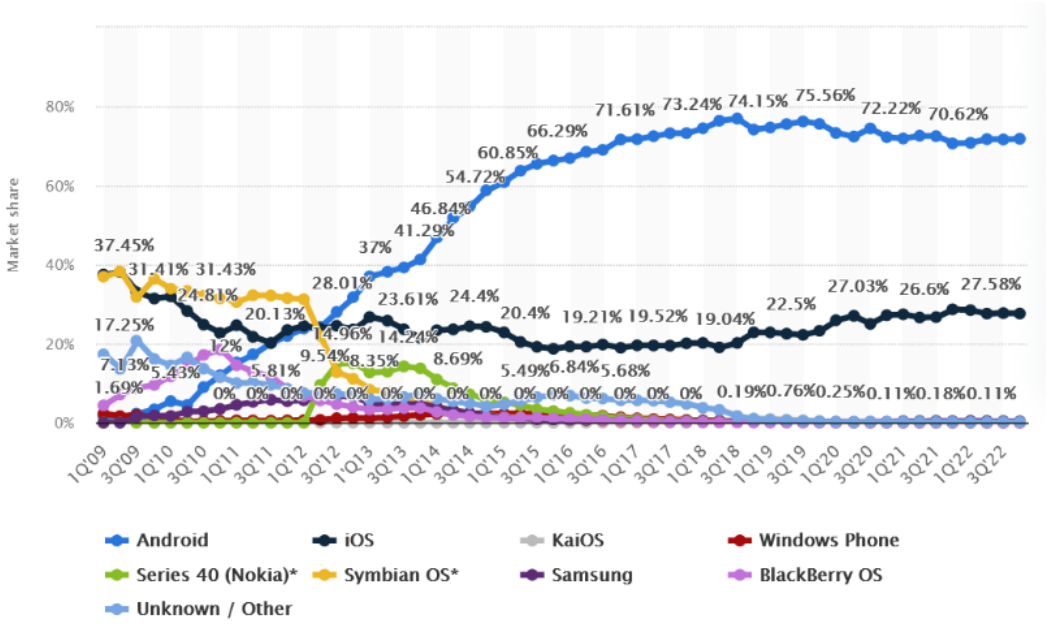
native Android applicaties te ontwikkelen. Het laat ook net zoals Android Studio ontwikkelaars toe om gemakkelijk te werken aan projecten dankzij de GitHub integratie (Okeke, 2022b), maar het ondersteunt ook andere integratiemethoden zoals Maven (Medewar, 2022). Tot slot bestaat er dankzij de populariteit een grote community rondom Eclipse die meewerkt aan de verbetering ervan (Medewar, 2022).

Xcode

Xcode is een IDE dat wordt gebruikt om software en applicaties te maken voor IOS, macOS, iPadOS, watchOS en tvOS (jahnavisarora, 2020). Net zoals bij Android Studio en Eclipse bevat ook Xcode een Source Control menu waarmee ontwikkelaars gemakkelijk met GitHub kunnen werken (Medewar, 2022). Om software voor alle Apple gerelateerde besturingssystemen te schrijven, wordt er gebruik gemaakt van Swift, C, C++ en Objective C compilers (jahnavisarora, 2020). Wat Xcode zo populair maakt, is de geïntegreerde workflow voor coderen, testen, debuggen en ontwerpen van gebruikersinterfaces (jahnavisarora, 2020).

Samenvatting native ontwikkelomgeving

Omdat dit onderzoek vooral focust op het verschil tussen native en cross-platform en niet op de verschillen tussen de native platformen onderling, wordt er langs de native kant één platform gekozen.



Figuur (2.3)
Grafiek marktaandeel native platformen (Monus, 2023).

Op bovenstaande foto is duidelijk te zien dat Android het grootste marktaandeel heeft bij mobiele applicaties. Daarom wordt Android gekozen als platform voor het uitvoeren van het onderzoek.

Om de functionaliteiten van native Android applicaties te testen, wordt er gebruik gemaakt van de officiële programmeertaal van Android applicaties, namelijk Kotlin. Dit wordt gedaan omdat Kotlin ontworpen is om veiliger te zijn dan sommige andere programmeertalen (Kesavan, 2021), waardoor de kans op bugs en fouten kleiner is. Daarnaast is Kotlin gemakkelijk te integreren in bestaande Java-projecten (Kesavan, 2021). Dit bespaart tijd en verhoogt de efficiëntie van de ontwikkeling. Tot slot heeft Kotlin een actieve gemeenschap van ontwikkelaars die regelmatig updates en nieuwe functies uitbrengen, waardoor de taal zich snel blijft ontwikkelen en verbeteren (Patel, 2023).

De gebruikte IDE om applicaties te ontwikkelen is Android Studio. Bij Android Studio wordt Kotlin standaard ondersteund terwijl dit bij Eclipse niet het geval is. Daarnaast wordt Kotlin door Google aangeraden aan ontwikkelaars (Medewar, 2022).

2.3. Cross-platform ontwikkeling

2.3.1. Wat is cross-platform ontwikkeling?

Cross-platform ontwikkeling is het proces om native applicaties 2.1.1 te maken voor een specifiek besturingssysteem, zoals Android en IOS. Cross-platform wordt gebruikt om het platformspecifiek probleem van native applicaties op te lossen door beide applicaties te ontwikkelen vanuit één codebase (Khan, 2021).

2.3.2. Voordelen van cross-platform ontwikkeling

Net zoals bij web applicaties 2.1.2, zal ook bij cross-platform ontwikkeling de kostprijs van applicaties lager liggen dan bij native ontwikkeling. Dit komt omdat er maar één applicatie ontwikkeld moet worden in plaats van twee. Ook zullen de onderhoudskosten lager liggen voor dezelfde reden (Terekhov, 2022). Nog een ander groot voordeel van de gedeelde codebase voor twee applicaties is de gedeelde logica ervan. Aangezien er maar één codebase is, zal de logica voor beide applicaties altijd gelijk zijn waar dat bij native ontwikkeling soms niet het geval is (Kotlin, 2023).

2.3.3. Nadelen van cross-platform ontwikkeling

Bij cross-platform ontwikkeling is het niet mogelijk om gebruik te maken van sommige functionaliteiten van een apparaat (Terekhov, 2022). Daarnaast kan native ontwikkeling indien een apparaat een nieuwe functionaliteit aanbiedt hier onmiddellijk gebruik van maken in vergelijking met cross-platform dat moet wachten op updates voor cross-platform ontwikkeling (Sakovich, 2023b).

Verschil met native ontwikkeling

Het grootste verschil met native ontwikkeling is dat er bij cross-platform maar één programmeertaal en IDE nodig zijn om mobiele applicaties voor zowel Android als

IOS te ontwikkelen (Hu, 2021). Bij native ontwikkeling is er voor elk platform een programmeertaal en IDE nodig.

2.3.4. Frameworks voor cross-platform ontwikkeling

Om cross-platform te ontwikkelen, kunnen verschillende frameworks en IDEs gebruikt worden. In de volgende secties worden er enkele overlopen die gebruikt kunnen worden om het onderzoek uit te voeren.

Flutter

Flutter is een open-source framework ontwikkeld door Google dat de ontwikkeling toelaat van niet alleen Android en IOS applicaties, maar ook Linux, macOS, Fuchsia en Windows vanuit één codebase (Okeke, 2022a). Het is vooral populair bij ontwikkelaars dankzij zijn eenvoudige gebruik en de performantie (Sakovich, 2023b). De hoge performantie komt door het gebruik van C en C++ (Terekhov, 2022). Daarnaast heeft het ook een hot reload waardoor ontwikkelaars snel resultaten kunnen zien bij veranderingen in de code (Sakovich, 2023b).

React Native

React Native is net zoals Flutter een open-source framework ontwikkeld door Facebook. Het laat toe om IOS en Android applicaties te ontwikkelen (Terekhov, 2022). De grootste voordelen van React Native zijn de ontwikkeltijd waarmee applicaties ontwikkeld kunnen worden (Terekhov, 2022) en de grote community die meehelpt aan het ontwikkelen van externe libraries (Okeke, 2022a). Dankzij die externe libraries hebben ontwikkelaars toegang tot native features die normaal voor cross-platform ontwikkeling niet toegankelijk zijn. Deze externe libraries kunnen geschreven zijn in Objective-C, Swift of Java (Okeke, 2022a). Tot slot heeft React Native ook een hot reload functie net zoals flutter waardoor ontwikkelaars snel resultaten kunnen zien bij veranderingen in de code (Terekhov, 2022).

.NET MAUI (vroeger Xamarin)

.NET MAUI is net zoals Flutter en React Native een open-source framework ontwikkeld door Microsoft. Het maakt gebruik van C# als programmeertaal en het .NET framework voor native libraries (Sakovich, 2023b). Net zoals bij Flutter en React Native bezit ook .NET MAUI over een hot reload functie om snel te kunnen ontwikkelen en debuggen. Dankzij het gebruik van C# beschikt ook .NET MAUI net zoals Flutter en React Native over een hoge performantie (Okeke, 2022a). Een belangrijk nadeel is echter dat applicaties niet altijd compatibel zijn met de nieuwste versies van Android of iOS (Terekhov, 2022).

Samenvatting frameworks

De drie besproken frameworks zijn allen vergelijkbaar qua prestaties. React Native en .NET MAUI hebben daarnaast ook gemakkelijk toegang tot native features dankzij de native libraries. Uit deze drie frameworks wordt React Native gekozen om cross-platform te ontwikkelen dankzij de performantie, native libraries en de grote community er rond. React Native bevat voor dit onderzoek de beste eigenschappen om het onderzoek uit te voeren.

2.3.5. IDEs voor cross-platform ontwikkeling

Visual Studio Code

Er is maar één IDE die eruit springt en dat is Visual Studio Code. Het is een universele IDE die gebruikt kan worden voor alle programmeertalen. Daarnaast is er een groot aanbod aan extensies die het mogelijk maken om de IDE aan te passen aan de specifieke behoefte van de ontwikkelaar (Heller, 2022). Daarom wordt Visual Studio Code gebruikt als IDE om cross-platform te ontwikkelen.

2.4. Samenvatting stand van zaken

Nu er een inzicht verkregen is in mobiele applicaties en de ontwikkeling ervan, kan het onderzoek van start gaan. Alle functionaliteiten worden voor zowel native als cross-platform ontwikkeld. Bij native ontwikkeling wordt Kotlin en Android Studio gebruikt om native applicaties te maken. Bij cross-platform wordt React-native en Visual Studio Code gebruikt om de native applicaties te maken.

3

Methodologie

In dit hoofdstuk wordt uitgelegd hoe de bachelorproef er zal uitzien, wat de volgorde is van het onderzoek en waar er precies op getest wordt.

Vervolgens wordt er in hoofdstuk 4 gekeken hoe de ontwikkelomgeving wordt opgesteld om met Kotlin in Android Studio te werken en met React Native in Visual Studio Code.

Na het opstellen van de ontwikkelomgeving wordt er in hoofdstuk 5 uitgelegd hoe de blanco projecten worden aangemaakt. Deze blanco projecten worden dan doorheen de bachelorproef gebruikt om functionaliteiten uit te werken.

Daarna zullen alle functionaliteiten uitgewerkt worden tot een project. Deze functionaliteiten zijn de basisfunctionaliteiten, audio- en videospelers, sensoren en push-notificaties. Ze zijn gekozen omdat ze veel gebruikt worden in mobiele applicaties en omdat ze een goede basis vormen om de verschillen tussen native en cross-platform te tonen.

1. De basisfunctionaliteiten maken deel uit van elke mobiele applicatie
2. De audio- en videospelers geven inzicht in de gebruikersinterface van de applicatie
3. De sensoren geven inzicht in hardwarespecifieke performantie
4. De push-notificaties geven inzicht in het uitvoeren van taken op de achtergrond

Bij het uitwerken van de functionaliteiten wordt de ontwikkeltijd van een functionaliteit gedocumenteerd met daarbij eventuele bugs of problemen. Ook wordt de

performantie van de functionaliteiten gemeten en de eventuele mogelijkheid om op te schalen zal ook bekeken worden. Elke functionaliteit wordt opgedeeld in zijn eigen hoofdstuk. Op die manier kunnen de resultaten per functionaliteit duidelijk teruggevonden worden.

- Hoofdstuk 6 voor de basisfunctionaliteiten
- Hoofdstuk 7 voor de audio- en videospelers
- Hoofdstuk 8 voor het gebruik van sensoren
- Hoofdstuk 9 voor de push-notificaties

Tot slot worden de resultaten en individuele conclusies van alle functionaliteiten uit de vorige hoofdstukken opgelijst en samengevat in hoofdstuk 10 om een antwoord te geven op de onderzoeksvragen uit hoofdstuk 1.

3.1. Volgorde onderzoek

Om de functionaliteiten te onderzoeken, wordt hetzelfde stappenplan gevolgd per functionaliteit.

1. Aanmaken van blanco project
2. Ontwikkeltijd meten
   (a) Duurtijd van de implementatie van een functionaliteit
   (b) Duurtijd van eventuele bugs of problemen tijdens het implementeren
3. Performantie meten
   (a) Tijdsduur van uit te voeren code meten
   (b) CPU en Geheugen gebruik meten
4. Mogelijkheid om op te schalen onderzoeken
   (a) Mogelijkheid tot abstractie bekijken
   (b) Herbruikbaarheid van code
5. Conclusie formuleren

Het onderzoek start altijd met het aanmaken van een blanco project. Daarna wordt de functionaliteit in dit blanco project geïmplementeerd. Hierbij wordt de tijd nodig om de functionaliteit te implementeren bijgehouden alsook eventuele bugs of problemen. Na het implementeren van de functionaliteit wordt de performantie van beide applicaties gemeten. Daarna wordt er gekeken naar de complexiteit van de code en of de functionaliteit opgeschaald kan worden. Tot slot wordt er dan per functionaliteit een conclusie geformuleerd op basis van de verkregen resultaten.

3.2. Hoe wordt er getest

3.2.1. Ontwikkeltijd

Hoe wordt de ontwikkeltijd van applicaties gemeten doorheen het onderzoek?

Algemene ontwikkeltijd

Om de algemene ontwikkeltijd van een functionaliteit te meten, wordt bijgehouden hoeveel tijd er besteed wordt aan het implementeren. Dit kan dan verder opgedeeld worden in de opzet en implementatie van een functionaliteit. Daarnaast wordt na het onderzoek de algemene compiletijd besproken en vergeleken. Aangezien dit een enorm verschil kan geven op de ontwikkeltijd bij een applicatie. Een verschil van vijf minuten voor het zien van veranderingen in de applicatie, in vergelijking met twee minuten, kan de algemene ontwikkeltijd veel beïnvloeden.

Bugs

Indien er eventuele bugs voorkomen tijdens het implementeren van een functionaliteit, dan zal deze ook beschreven worden: wat is de oorzaak, hoe is het opgelost en hoeveel tijd is er hieraan gespendeerd.

3.2.2. Performantie

Hoe wordt de performantie van applicaties gemeten doorheen het onderzoek?

Android Studio

Het is mogelijk om de performantie van native applicaties te meten met de Android profiler tool binnen Android Studio *View > Tool Windows > Profiler*. Deze tool wordt gebruikt om inzicht te verkrijgen in verschillende prestatie-aspecten zoals CPU-gebruik, geheugengebruik, netwerkactiviteit en energieverbruik. De Android Profiler genereert realtime grafieken om de gegevens te bekijken.

React Native

Om de performantie van de React Native applicaties te meten, kan de Performance Monitor tool gebruikt worden. Deze is beschikbaar via de React Native Developer Tools en is geoptimaliseerd om de werking en performantie van React Native componenten te meten. Naast de Performance Monitor bieden de React Native Developer Tools ook nog andere tools aan zoals Element Inspector, Network Inspector, Console Logging en Redux Debugger. De Performance Monitor tool is specifiek ontworpen voor het meten en analyseren van de performantie.

Probleem performantietools Normaal gezien worden React Native applicaties getest met de Performance Monitor tool. Maar ondanks het feit dat deze inzicht geeft in de performantie van de applicatie, zijn er een aantal verschillen met de Android profiler. De Performance Monitor tool van React Native is namelijk zodanig

ontworpen om enkel de performantie van de React applicaties te meten en analyseren. Op die manier kunnen eventuele bottlenecks ontdekt worden binnen de applicatie om zo de performantie te verbeteren.

Het probleem bij dit onderzoek is dat de performantie van beide applicaties met elkaar vergeleken moet worden. En met de Performance Monitor tool van React native is dit niet mogelijk. Daarom dient er gezocht te worden naar een externe/alternatieve manier om de performantie op een eerlijke manier te vergelijken.

Firebase Performance Monitoring Eén manier om de performantie te meten, is de Firebase Performance Monitoring tool. Deze is ontwikkeld door Google en kan de duur van een stuk code van beide applicaties onafhankelijk meten. Dankzij deze tool kan bijvoorbeeld opstarttijd, HTTP requests, laadtijd van scherm... gemeten worden. De implementatie van deze library voor beide ontwikkelmethodes wordt besproken in hoofdstuk 5.

Android profiler Ondanks het feit dat de Firebase Performance Monitoring gebruikt kan worden om de tijdsduur van een stuk code te meten, is dit niet genoeg om een volwaardige vergelijking te maken. Daarom wordt de Android profiler ook voor de React Native applicaties gebruikt. Dit is mogelijk als de emulator binnen Android Studio eerst wordt opgestart en daarna pas de React Native commandos om een applicatie te starten worden uitgevoerd 4.2. Hierdoor wordt de applicatie in de emulator binnen Android Studio opgestart en kan de Android profiler gebruikt worden.

Door het gebruik van zowel de Android Profiler als Firebase Performance Monitoring kunnen de prestaties van beide applicaties op een eerlijke manier worden vergeleken.

3.2.3. Schaalbaarheid

Hoe wordt de schaalbaarheid van applicaties gemeten doorheen het onderzoek?

Complexiteit

Eerst wordt de complexiteit van de code geanalyseerd om te kijken hoe goed deze gestructureerd en georganiseerd is. Daarnaast wordt er ook gekeken of de functionaliteit in kleine herbruikbare componenten opgedeeld kan worden aangezien een goed gestructureerde codebase, opgedeeld in kleine componenten, gemakkelijker is om aan te passen en uit te breiden.

Herbruikbaarheid

Tot slot wordt ook gekeken in hoeverre de code van de functionaliteiten hergebruikt kan worden in andere delen van de applicatie. Want als componenten gemakkelijk hergebruikt kunnen worden, vergroot dit de schaalbaarheid van de applicatie omdat toekomstige aanpassingen of uitbreidingen gemakkelijker geïmplementeerd kunnen worden.

4

Opstellen ontwikkelomgeving

Vooraleer het onderzoek van start kan gaan, moet de ontwikkelomgeving voor zowel native als cross-platform opgesteld worden.

Zoals besproken in hoofdstuk 2 zal voor native ontwikkeling Kotlin met Android Studio gebruikt worden om applicaties te ontwikkelen. Voor cross-platform zal React Native met Visual Studio Code gebruikt worden.

4.1. Opstellen native ontwikkeling

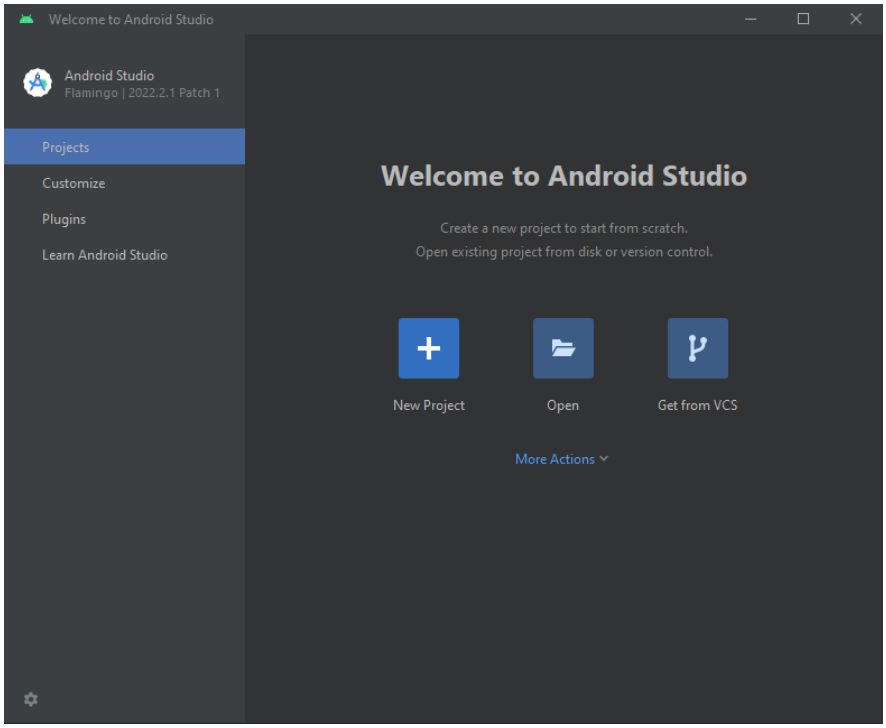
Om met native ontwikkeling van start te gaan, wordt Android Studio eerst gedownload.

1. Components
Bij het eerste scherm van het installatieprogramma is het belangrijk om ook **Android Virtual Device** te selecteren. Dit is de emulator die gebruikt zal worden om de applicaties te runnen.

2. Installatie locatie
Op het volgende scherm wordt de locatie waar Android Studio geïnstalleerd wordt gekozen. Het is aan te raden om dit niet te vervangen omdat React Native gebruik maakt van de standaard locatie om bepaalde componenten op te zoeken.

3. Startmenu map
Deze is zelf te kiezen. Hiermee wordt een shortcut gecreëerd vanwaar Android Studio kan worden opgestart.

4. Android SDK
Als de installatie goed is verlopen, wordt volgend scherm getoond.



Figuur (4.1)
Startscherm na Android Studio installatie.

Standaard wordt de laatste Android SDK geïnstalleerd door Android Studio. Deze wordt ook tijdens de duur van het onderzoek gebruikt. Om met deze laatste Android SDK te werken, is Android 13 (Tiramisu) nodig. Deze wordt geïnstalleerd door op *More Actions* te drukken. Op het nieuwe verkregen scherm kan Android 13 (Tiramisu) dan worden aangevinkt. Ook moet onderaan rechts het vakje **Show Package Details** aangevinkt worden, om dan te verifiëren dat de **Android SDK Platform 33**, **Sources for Android 33** en **Google APIs Intel x86_64 Atom System Image** zijn aangevinkt.

Daarna wordt er genavigeerd naar SDK Tools en wordt **Show Package Details** en **33.0.0** onder Android SDK Build-Tools aangevinkt. Tot slot wordt er op **Apply** gedrukt om de Android SDK en gerelateerde tools te downloaden en installeren. Daarna kan een dummy project worden aangemaakt (Empty Activity) om Android Studio van start te krijgen.

5. Emulator
Om de applicaties die doorheen het onderzoek ontwikkeld worden te runnen, is een emulator nodig. Deze wordt aangemaakt bovenaan rechts *Device Manager* > *Create device*. Eerst en vooral wordt de device geselecteerd die gesimuleerd wordt.

Voor het onderzoek wordt een Pixel 3 als emulator gebruikt. Na het selecteren van de device wordt de Android versie gekozen. Hier wordt Tiramisu met API Level 33 (die daarnet is geïnstalleerd) geselecteerd. Tot slot zijn er nog een aantal configuratie instellingen voor het apparaat. Voor het onderzoek worden de standaard instellingen gebruikt.

Nu is Android Studio klaar om native applicaties te ontwikkelen en runnen.

4.2. Opstellen cross-platform ontwikkeling

Om cross-platform te ontwikkelen door gebruik te maken van React Native is er wat meer werk om de omgeving op te stellen. Eerst moet Visual Studio Code worden gedownload. Na het installeren van Visual Studio Code kan de React Native ontwikkelomgeving worden opgesteld.

1. React Native CLI Quickstart

Voor de React Native ontwikkelomgeving kan gebruik worden gemaakt van Expo. Dit is een simpele manier om snel applicaties visueel werkend te krijgen. Maar voor dit onderzoek moeten de applicaties op dezelfde emulator runnen waarop de native ontwikkelde applicaties runnen. Daarom wordt de React Native CLI gebruikt in plaats van Expo.

2. Besturingssysteem en doelsysteem

Afhankelijk van het gebruikte besturingssysteem is er een andere handleiding. Doorheen dit onderzoek zal gewerkt worden op een Windows laptop en zal er gefocust worden op Android applicaties.

3. Node & JDK

Om te beginnen moeten Node en JDK geïnstalleerd worden. Dit kan gemakkelijk worden gedaan met Chocolatey. Eerst wordt een opdrachtprompt (powershell) als administrator geopend waaraan volgend commando wordt meegegeven:

```
choco install -y nodejs-lts microsoft-openjdk11
```

Voor Node wordt de laatste versie gebruikt. De minimale versie waarmee React Native kan werken is 14. Voor de JDK wordt versie 11 gebruikt aangezien hogere versies voor problemen kunnen zorgen.

4. Android ontwikkelomgeving

Net zoals bij native ontwikkeling hebben de React Native applicaties ook Android Studio nodig. Zo kan de emulator in Android Studio gebruikt worden. Android Studio zelf wordt niet direct gebruikt maar er wordt wel gebruik gemaakt van de emulator die Android Studio aanbiedt.

Bij het opstellen van de React Native omgeving zijn er een paar instellingen waar er rekening mee moeten worden gehouden. Hiermee is al rekening gehouden bij het installeren van Android Studio in paragraaf 4.1.

5. ANDROID_HOME omgevingsvariabelen
De React Native tools hebben enkele omgevingsvariabelen nodig om native applicaties te bouwen. Om deze omgevingsvariabelen in te stellen, wordt het configuratiescherm van Windows geopend. Daarna wordt er genavigeerd naar *Gebruikersaccounts > Gebruikersaccounts > Mijn omgevingsvariabelen veranderen > Nieuw...*. In het nieuwe verkregen scherm wordt de naam van de omgevingsvariabel **ANDROID_HOME** en de path naar de Android SDK ingevuld. De standaard path naar de Android SDK is **%LOCALAPPDATA%\Android\Sdk**. De locatie van de Android SDK is ook te vinden via Android Studio via *File > Settings > Appearance & Behavior > System Settings > Android SDK*.

Om te controleren of de omgevingsvariabel correct is toegevoegd, wordt een nieuwe opdrachtprompt (powershell) geopend en wordt volgend commando ingevoerd:

```
Get-ChildItem -Path Env:\
```

In de teruggegeven lijst kan dan gecontroleerd worden of **ANDROID_HOME** effectief werd toegevoegd.

Tot slot moet de **Path** omgevingsvariabel aangepast worden. Hiervoor wordt opnieuw genavigeerd naar het overzicht met alle omgevingsvariabelen in het configuratiescherm *Gebruikersaccounts > Gebruikersaccounts > Mijn omgevingsvariabelen veranderen*. Daarna wordt de Path omgevingsvariabel geselecteerd en wordt deze bewerkt met de *Bewerken...* knop. Bij het nieuw verkregen scherm wordt op *Nieuw* gedrukt en wordt de path naar platform-tools toegevoegd aan de lijst. Het path van de platform-tool is standaard **%LOCALAPPDATA%\Android\Sdk\platform-tools**.

6. React Native Command Lin Interface (CLI)
React Native heeft een ingebouwde command line interface (CLI) om te werken met React Native. Deze kan gebruikt worden dankzij Node.js dat daarnet werd geïnstalleerd.

```
npx react native [commando]
```

7. Emulator gebruiken
Om nu effectief applicaties te laten runnen, wordt de emulator gebruikt die Android Studio aanbiedt. Aangezien deze al opgesteld is bij het opzetten van Android

Studio 4.1, moeten deze niet meer voor React Native worden opgezet. Om de applicatie te starten, zijn er twee terminals nodig binnen Visual Studio code. Deze kunnen geopend worden via *Terminal > New Terminal*. In de eerste terminal wordt Metro gestart met volgend commando:

```
npx react-native start
```

In de tweede terminal wordt de applicatie opgebouwd en geïnstalleerd op de emulator met volgend commando:

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

4.3. Gebruikte hardware tijdens onderzoek

Doorheen het onderzoek zullen alle projecten starten vanuit een blanco project en op dezelfde emulator getest worden. Deze emulator zal altijd runnen op hetzelfde apparaat. Dankzij volgend commando kan de informatie van een apparaat verkregen worden:

```
npx react-native info
```

Deze geeft bij het gebruikte apparaat voor dit onderzoek volgende output:

```
System:
    OS: Windows 10 10.0.19044
    CPU: (12) x64 AMD Ryzen 5 5600H with Radeon Graphics
    Memory: 2.90 GB / 15.86 GB
Binaries:
    Node: 18.16.0 - C:\Program Files\nodejs\node.EXE
    Yarn: Not Found
    npm: 9.5.1 - C:\Program Files\nodejs\npm.CMD
    Watchman: Not Found
SDKs:
    Android SDK: Not Found
    Windows SDK: Not Found
IDEs:
    Android Studio: AI-222.4459.24.2221.9971841
    Visual Studio: Not Found
Languages:
    Java: 11.0.18
npmPackages:
    @react-native-community/cli: Not Found
    react: 18.2.0 => 18.2.0
    react-native: 0.71.7 => 0.71.7
    react-native-windows: Not Found
```

```
    npmGlobalPackages:
    *react-native*: Not Found
```

Voor dit onderzoek wordt een apparaat met een AMD Ryzen 5 5600H processor met 12 cores, geïntegreerde Radeon Graphics en 16GB RAM geheugen gebruikt, wat sterk en snel genoeg is om het onderzoek uit te voeren.

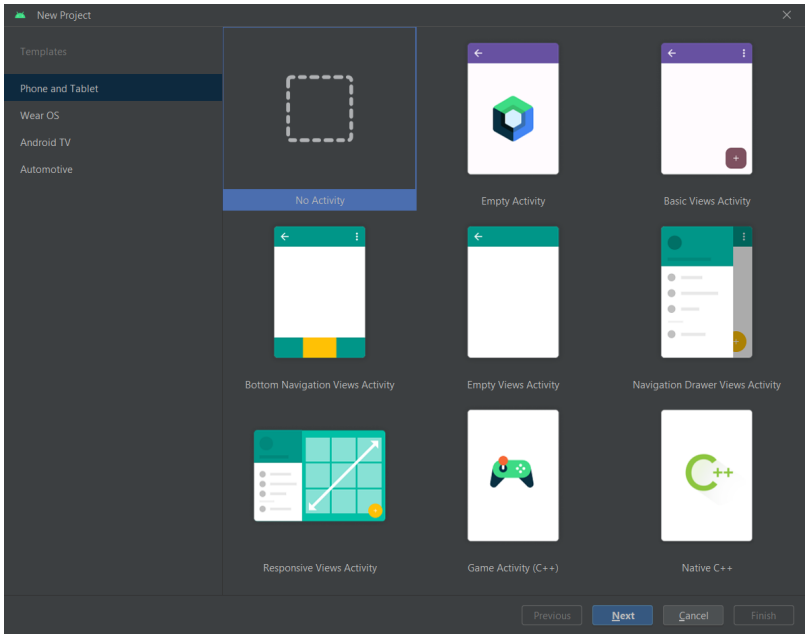
5

Aanmaken blanco project

Na het opzetten van de ontwikkelomgeving voor zowel native als cross-platform wordt in dit hoofdstuk uitgelegd hoe een blanco project wordt aangemaakt.

5.1. Android Studio

Android Studio kan als IDE zelf nieuwe projecten aanmaken met *File* > *New* > *New Project*.... Hierna wordt een overzicht getoond van projecten die als basis kunnen worden gebruikt. Ook kan er op dit scherm gekozen worden voor welk apparaat de applicatie is.



Figuur (5.1)
Overzicht startprojecten Android Studio.

Afhankelijk van de functionaliteit die wordt geïmplementeerd, zal een ander startproject gekozen worden. Na het kiezen van het startproject kan een naam, programmeertaal en de minimaal ondersteunde Android API worden ingegeven. Voor dit onderzoek wordt er gekozen voor Kotlin. De naam en minimale ondersteunde Android API spelen hierbij geen rol.

5.1.1. Basisfunctionaliteiten

Bij de basisfunctionaliteiten wordt **Bottom Navigation Views Activity** als startproject gebruikt om te kunnen navigeren tussen verschillende schermen. Daardoor moeten er geen extra libraries worden geïmplementeerd.

5.1.2. Overige functionaliteiten

Voor de overige functionaliteiten (gebruik van sensoren, push-notificaties en audio- en videospelers) wordt **Empty Views Activity** als startproject gebruikt. Op die manier zal er zo weinig mogelijk overhead zijn die de resultaten kan beïnvloeden.

5.1.3. Firebase Performance Monitoring

Om de performantie van de applicaties te meten, meer specifiek de tijdsduur die een stuk code nodig heeft om uit te voeren, wordt de Firebase Performance Monitoring tool in het project geïmplementeerd.

Firebase aan project toevoegen

Vooraleer de Performance Monitoring tool geïmplementeerd kan worden, moet Firebase aan het project worden toegevoegd.

1. Configuratie bestanden toevoegen In het Firebase project wordt een nieuwe Android applicatie toegevoegd. Om deze aan te maken, moet de package naam worden meegegeven die te vinden is in het **build.gradle(module)** bestand als **applicationId**. Optioneel kan de naam van de applicatie worden meegegeven.

Nadat de Android applicatie is aangemaakt, moet het **google-services.json** bestand worden gedownload. Deze wordt dan geplaatst in de *app* folder.

2. Firebase plugins configureren Om Firebase het configuratiebestand nu te laten gebruiken, moet de google-services plugin worden toegevoegd aan de dependancies in het **build.gradle(project)** bestand boven de plugins.

```
buildscript {
    repositories {
        google()
        mavenCentral()
    }
```

```
    dependencies {
        // andere dependancies
        classpath "com.google.gms:google-services:4.3.15"
    }
}
```

Daarna moet de plugin aan het **build.gradle(module)** bestand worden toegevoegd.

```
plugins {
    // andere plugins
    id "com.google.gms.google-services"
}
```

Tot slot moeten de Firebase SDKs worden toegevoegd aan het **build.gradle(module)** bestand.

```
dependencies {
    // andere dependancies
    implementation platform("com.google.firebase:firebase-bom:32.0.0")
}
```

Firebase is nu volledig aan ons project toegevoegd.

Implementatie Performance Monitoring tool
Nu Firebase aan ons project is toegevoegd, kan de Performance Monitor worden toegevoegd.

1. Performance Monitoring SDK aan applicatie toevoegen Eerst moet de Performance Monitor SDK aan het project worden toegevoegd in het **build.gradle(module)** bestand.

```
dependencies {
    // andere dependancies
    implementation "com.google.firebase:firebase-perf-ktx"
}
```

2. Performance Monitoring Gradle plugin toevoegen Na het toevoegen van de SDK moet de Performance Monitoring Gradle plugin worden toegevoegd aan het **build.gradle(project)** bestand.

```
dependencies {
    // andere dependancies
    classpath "com.google.firebase:perf-plugin:1.4.2"
}
```

Tot slot moet de plugin worden toegevoegd aan het **build.gradle(module)** bestand.

```
plugins {
    // andere plugins
    id "com.google.firebase.firebase-perf"
}
```

Nu is het blanco project klaar om vanuit dit project alle functionaliteiten te implementeren en onderzoeken, buiten de basisfunctionaliteiten. Hiervoor wordt er vanuit een ander project gestart, maar zal de Performance Monitoring tool op dezelfde manier worden geïmplementeerd.

3. Performance meten Om de performantie te meten doorheen de applicaties zal dit er als volgt uitzien:

```
val trace = FirebasePerformance.getInstance().newTrace("naam_trace")

trace.start()
// uit te voeren code
trace.stop()
```

De trace wordt automatisch naar de Firebase console gestuurd, van waarop de resultaten kunnen worden bekeken.

5.2. Cross-platform project

Om bij React Native een project op te starten, wordt volgend commando gebruikt:

```
npx react-native init <projectnaam>
// Of
npx react-native@X.XX.X init <projectnaam> --version X.XX.X
```

Deze wordt uitgevoerd in de terminal van Visual Studio Code in de gewenste map waar het project moet worden aangemaakt.

Het eerste commando zal een blanco React Native project aanmaken met de laatste versie. Bij het tweede commando kan er een versie worden meegegeven door **X.XX.X** in bovenstaand commando te vervangen. In dit onderzoek wordt er gebruik gemaakt van de laatste beschikbare versie namelijk **0.71.7**.

5.2.1. Firebase Performance Monitoring

Om de performantie van de applicaties te meten, moet de Firebase Performance Monitoring tool in het project worden geïmplementeerd.

Firebase project aanmaken
Vooraleer de Performance Monitoring tool geïmplementeerd kan worden moet Firebase aan het project worden toegevoegd.

1. Dependancy installeren Eerst moet de React Native Firebase app module aan de root van het React Native project worden toegevoegd. Dit wordt gedaan met volgend commando:

```
npm install --save @react-native-firebase/app
```

2. Configuratie bestanden toevoegen Nadat de dependancy is toegevoegd, moet er een nieuw Firebase project worden aangemaakt. Om deze aan te maken, moet de package naam worden meegegeven die te vinden is in het *android/app/-build.gradle* bestand als **applicationId**. Optioneel kan de naam van de applicatie worden meegegeven.

Nadat de Android applicatie is aangemaakt, moet het **google-services.json** bestand worden gedownload. Dit wordt dan geplaatst in de *android/app* folder.

3. Firebase configureren Om Firebase het configuratiebestand nu te laten gebruiken, moet de google-services plugin worden toegevoegd aan de dependancies binnen het *android/build.gradle* bestand.

```
buildscript {
    dependencies {
        // andere dependencies
        classpath "com.google.gms:google-services:4.3.15"
    }
}
```

Daarna moet de plugin aan het *android/app/build.gradle* bestand worden toegevoegd.

```
apply plugin: "com.google.gms.google-services"
```

Tot slot wordt deze ook aan de dependancies toegevoegd in hetzelfde *android/app/build.gradle* bestand.

```
dependencies {
    // andere dependancies
    implementation platform("com.google.firebase:firebase-bom:32.0.0")
}
```

De React Native Firebase app module is nu volledig geïmplementeerd.

Implementatie Performance Monitoring tool
Nu de React Native Firebase app module is geïmplementeerd, kan de Performance Monitoring tool worden geïmplementeerd.

1. Dependancy installeren Eerst moet de Performance Monitoring tool aan de root van ons project worden toegevoegd. Dit wordt gedaan met volgend commando:

```
npm install --save @react-native-firebase/perf
```

2. Performance Monitoring tool configureren Daarnaast moet de plugin worden toegevoegd aan de dependancies binnen het */android/build.gradle* bestand.

```
buildscript {
    dependencies {
        // andere dependencies
        classpath "com.google.firebase:perf-plugin:1.4.2"
    }
}
```

En moet de plugin worden uitgevoerd door deze aan het */android/app/build.gradle* bestand toe te voegen.

```
apply plugin: "com.google.firebase.firebase-perf"
```

Tot slot wordt deze ook toegevoegd aan de dependancies in hetzelfde */android/app/build.gradle* bestand.

```
dependencies {
    // andere dependencies
    implementation "com.google.firebase:firebase-perf-ktx"
}
```

Nu is het blanco project klaar om vanuit dit project alle functionaliteiten te implementeren en onderzoeken.

3. Performance meten Om de performantie te meten doorheen de applicaties zal dit er als volgt uitzien:

```
import perf from '@react-native-firebase/perf';

const trace = await perf().newTrace('naam_trace');

trace.start();
// uit te voeren code
trace.stop();
```

De trace wordt automatisch naar de Firebase console gestuurd, van waarop de resultaten kunnen worden bekeken.

5.3. Android Profiler

Om de Android profiler te gebruiken, zijn er geen extra stappen nodig. Deze is standaard beschikbaar voor applicaties die op de emulator binnen Android Studio runnen.

6

Basisfunctionaliteiten

In dit hoofdstuk worden de basisfunctionaliteiten van native en cross-platform vergeleken met elkaar. Met de resultaten kan dan een gepaste conclusie worden gevormd.

6.1. Native

Wat is er nodig

Normaal gezien wordt er voor de functionaliteiten altijd een of andere library of API gebruikt. Enkel bij de navigatie is dit niet nodig. Voor de navigatie wordt het startproject gebruikt dat Android Studio aanbiedt, hierin zit de navigatie al geïmplementeerd 5.1.1. Voor het laadscherm dat getoond wordt bij het opstarten van de applicatie zal de SplashScreen API gebruikt worden die wordt aangeboden door Android Studio.

Uitvoering

Voor de navigatie is het voldoende om het nieuwe project aan te maken. Hierbij zit de navigatie al geïmplemnteerd.

1. Gradle instellingen aanpassen

Om de SplashScreen API te gebruiken, moeten deze aan de dependancies worden toegevoegd. Dit wordt gedaan in het **build.gradle(module)** bestand.

```
dependencies {
    // andere dependacies
    implementation("androidx.core:core-splashscreen:1.0.0")
}
```

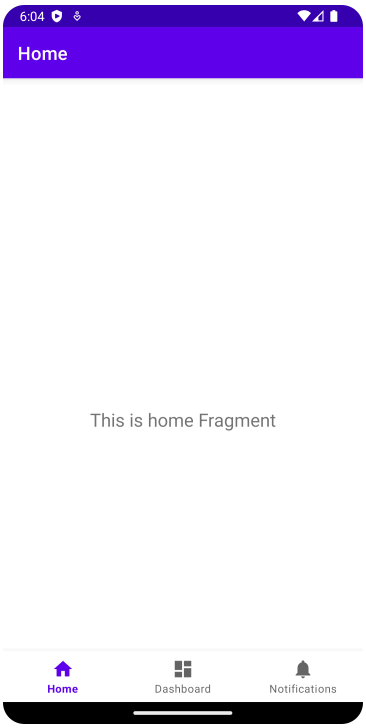
2. SplashScreen instellen

Na het toevoegen van de dependancy kan het laadscherm aangepast worden. Hiervoor wordt een nieuw **splash.xml** bestand in de **res/values** map aangemaakt. In dit bestand kan het laadscherm gecustomized worden. Ook wordt hier de icoon toegevoegd die wordt getoond tijdens het laden. Dit icoon wordt in de **res/drawable** map geplaatst.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="Theme.MyApp.MySplash" parent="Theme.SplashScreen">
        <item name="windowSplashScreenBackground">@color/black</item>
        <item name="windowSplashScreenAnimatedIcon">
            @drawable/bank_svgrepo_com</item>
        <item name="postSplashScreenTheme">@style/Theme.Basis</item>
    </style>
</resources>
```

3. Applicatie maken

Dankzij deze informatie wordt een applicatie opgezet die een laadscherm toont dat weer zal verdwijnen van zodra de applicatie zijn eerste frame tekent. Daarna is er onderaan een navigatiebar te zien die ervoor zorgt dat er genavigeerd kan worden tussen de verschillende schermen.



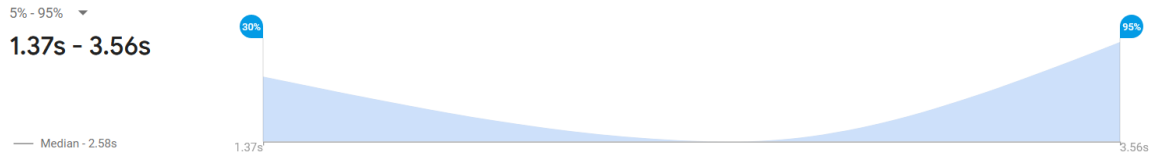
Figuur (6.1)
Layout van applicatie voor de basisfunctionaliteiten bij Android.

Ontwikkeltijd

Dankzij het startproject dat Android Studio aanbiedt, is het mogelijk om snel een applicatie op te zetten die gebruik maakt van de navigatiecomponent. Daarnaast kan het laadscherm ook snel geïmplementeerd worden. Er moet enkel al een icoon aanwezig zijn om te tonen. Om de volledige applicatie aan te maken inclusief opzoekwerk en het aanmaken van de icoon is er 1 uur gespendeerd.

Performantie

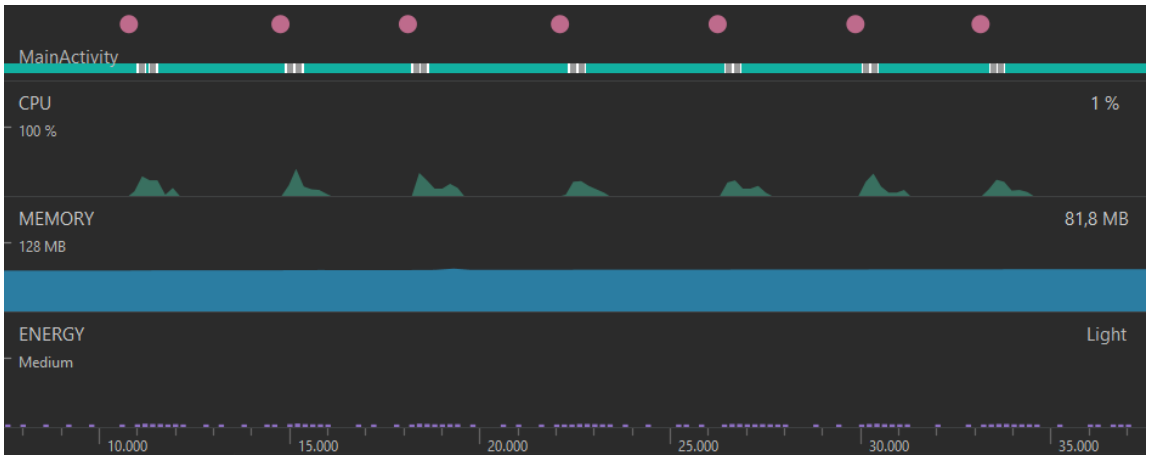
Tijdsduur



Figuur (6.2)
Overzicht tijdsduur opstarten van applicatie met basisfunctionaliteiten bij Android.

Tijdens het meten van de duur nodig voor het opstarten van de applicatie, is de applicatie meerdere keren opgestart. Op de grafiek is te zien dat de applicatie gemiddeld 2,58 seconden nodig heeft om op te starten. Het minimum en maximum liggen op 1,37 en 3,56 seconden.

CPU & geheugen



Figuur (6.3)
Overzicht CPU en geheugen gebruik tijdens het navigeren tussen schermen bij Android.

Op de grafiek is te zien dat het CPU gebruik van de applicatie wanneer deze inactief is, rond de 0 - 1% ligt en dat het duidelijk is wanneer er genavigeerd wordt tussen de schermen. De piek van het CPU gebruik lag gemiddeld op 31% met een minimum en maximum van 25% en 42%. Het geheugen blijft in tegenstelling tot de CPU wanneer de applicatie inactief en actief is, rond de 81MB hangen, met verschillen

van maximum 2-3MB. Er is geen merkbaar verschil in het geheugen wanneer er genavigeerd wordt tussen de schermen.

Schaalbaarheid

Complexiteit

De navigatie is niet complex om op te zetten aangezien een startproject kan worden gebruikt. Daarnaast is het ook niet complex om schermen toe te voegen of aan te passen. Dankzij de Splashscreen API is ook het laadscherm niet complex om op te zetten. Het enige dat nodig is, is een icoon om te tonen.

Herbruikbaarheid

De navigatie kan zoals hierboven vermeld gemakkelijk hergebruikt worden om extra schermen toe te voegen en is dus zeer schaalbaar. Het is ook gemakkelijk om de bestaande schermen te hergebruiken in andere applicaties. Bij het laadscherm is er geen sprake van opschaling. Dit is enkel nodig bij het opstarten van de applicatie en zal dus maar één keer geïmplementeerd moeten worden.

6.2. Cross-platform

Wat is er nodig

Bij React Native is er voor de navigatie een library nodig. Hiervoor wordt React Navigation gebruikt. Voor het laadscherm wordt react-native-bootsplash library gebruikt. Met behulp van deze libraries kan er genavigeerd worden in de applicatie en kan er een laadscherm getoond worden.

Uitvoering

1. Library toevoegen

Om de navigatie en het laadscherm te kunnen gebruiken, moeten de juiste libraries aan de root van het project worden toegevoegd. Deze worden toegevoegd met volgende commando's:

```
npm install @react-navigation/native
npm install react-native-screens
npm install react-native-safe-area-context
npm install @react-navigation/bottom-tabs
```

2. onCreate methode toevoegen

Om de navigatie te gebruiken, moet een **onCreate** methode worden toegevoegd aan het *android/app/src/main/java/com/project/MainActivity.java* bestand.

```
import android.os.Bundle;
// ...
public class MainActivity extends ReactActivity {
```

```
  // ...
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(null);
  }
  // ...
}
```

3. Dependencies toevoegen

Om het laadscherm te gebruiken, moeten de juiste dependencies worden toegevoegd aan het *android/app/build.gradle* bestand.

```
dependencies {
  // Andere dependencies
  implementation("androidx.core:core-splashscreen:1.0.0")
}
```

4. Laadscherm toevoegen

Om een eerste laadscherm te genereren, kan volgend commando worden gebruikt:

```
npx react-native generate-bootsplash path/to/icon.png
    --background-color "#000000"
    --logo-width=100
    --assets-path=assets
    --flavor=main
    --platforms=android,ios
```

Daarna moet er een stijl worden aangemaakt om het laadschem te tonen. Dit gebeurt in het *android/app/src/main/res/values/styles.xml* bestand.

```
<style name="BootTheme" parent="Theme.SplashScreen">
    <item name="windowSplashScreenBackground">
      @color/bootsplash_background</item>
    <item name="windowSplashScreenAnimatedIcon">
      @mipmap/bootsplash_logo</item>
    <item name="postSplashScreenTheme">@style/AppTheme</item>
</style>
```

Om bij het opstarten van de applicatie het laadscherm eerst te tonen, moet de volgende lijn in het *android/app/src/main/AndroidManifest.xml* bestand worden aangepast.

```
<application
  android:name=".MainApplication"
```

```
  android:label="@string/app_name"
  android:icon="@mipmap/ic_launcher"
  android:roundIcon="@mipmap/ic_launcher_round"
  android:allowBackup="false"
  android:theme="@style/BootTheme"> <!-- Verander deze lijn -->
  <!-- ... -->
</application>
```

Tot slot kan het laadscherm geïnitialiseerd worden door volgende regels toe te voegen aan het *android/app/src/main/java/com/project/MainApplication.java* bestand.

```
import com.zoontek.rnbootsplash.RNBootSplash;

public class MainApplication extends Application implements ReactApplication {
  // ...
  @Override
  public void onCreate(Bundle savedInstanceState) {
    RNBootSplash.init(this); // Voeg deze lijn toe
    super.onCreate(savedInstanceState);
  }
  // ...
}
```

5. Navigatie toevoegen

Om de navigatie te gebruiken, moeten alle componenten in een **NavigationContainer** component worden gestoken. Dit wordt gedaan door volgende regels toe te voegen aan het *App.tsx* bestand:

```
import { NavigationContainer } from '@react-navigation/native';

export default function App() {
  return (
    <NavigationContainer>
      {/* ... */}
    </NavigationContainer>
  );
}
```

6. Bottom Tab navigatie toevoegen

Om de bottom tab navigatie te gebruiken, moet een **createBottomTabNavigator** functie worden aangemaakt. Dit wordt gedaan door de volgende regels toe te voegen aan het *App.tsx* bestand:

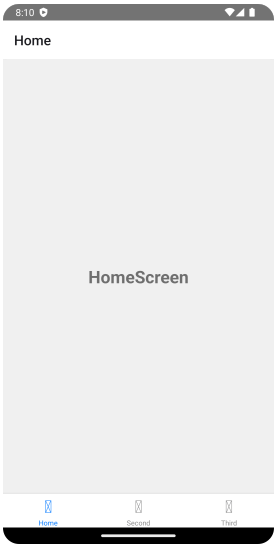
```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

const Tab = createBottomTabNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <Tab.Navigator>
        {/* ... */}
      </Tab.Navigator>
    </NavigationContainer>
  );
}
```

Daarna kunnen de verschillende schermen worden toegevoegd aan de **Tab.Navigator** component.

```
<Tab.Navigator>
    <Tab.Screen name="Home" component={HomeScreen} />
    <Tab.Screen name="Second" component={SecondScreen} />
    <Tab.Screen name="Third" component={ThirdScreen} />
</Tab.Navigator>
```

7. Applicatie maken

Met deze informatie wordt een applicatie met een laadscherm aangemaakt. Het laadscherm zal verdwijnen zodra de applicatie gerenderd is. Daarna is er een bottom tab navigatie die navigeert tussen de drie verschillende schermen.



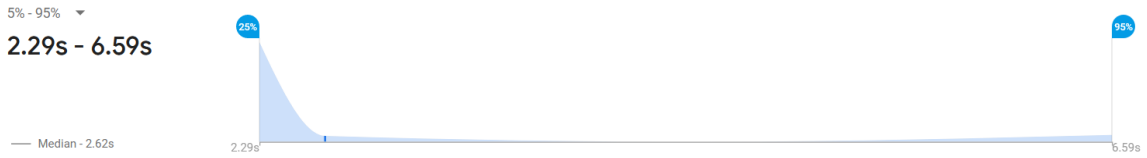
Figuur (6.4)
Layout van applicatie voor de basisfunctionaliteiten bij React Native.

Ontwikkeltijd

Het implementeren van beide libraries is niet moeilijk. Voor de implementatie van de navigatie is ongeveer 45 minuten nodig en voor het laadscherm ongeveer 30 minuten, inclusief het opzoekwerk. In totaal wordt er dus 1 uur en 15 minuten gespendeerd om beide libraries te implementeren. Het is wel belangrijk om aandachtig de installatiestappen te volgen. Indien dit niet gebeurt, kunnen er gemakkelijk fouten optreden waardoor tijd verloren gaat aan het oplossen van deze fouten.

Performantie

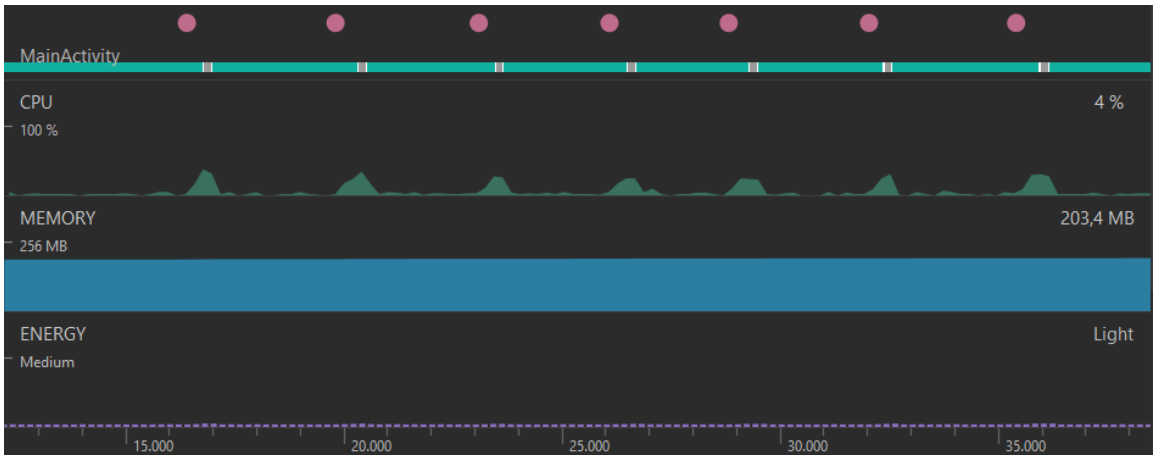
Tijdsduur



Figuur (6.5)
Overzicht tijdsduur opstarten van applicatie met basisfunctionaliteiten bij React Native.

Tijdens het meten van de duur voor het opstarten van de applicatie, is de applicatie meerdere keren opgestart. Op de grafiek is te zien dat de applicatie gemiddeld 2,84 seconden nodig heeft om op te starten. Het minimum en maximum liggen op 2,29 en 6,59 seconden.

CPU & geheugen



Figuur (6.6)
Overzicht CPU en geheugen gebruik tijdens het navigeren tussen schermen bij React Native.

Net zoals bij native is op de grafiek te zien dat het CPU gebruik van de applicatie rond de 4% ligt wanneer deze inactief is. Bovendien is het duidelijk zichtbaar wanneer er genavigeerd wordt tussen de schermen. De piek van het CPU gebruik lag gemiddeld op 32% met een minimum en maximum van 27% en 40%. Wat bij Re-

act Native opvalt, is dat de eerste keer navigeren naar een scherm een grotere piek geeft. Eenmaal het scherm al werd ingeladen, is de piek minder groot. Dit wijst erop dat React Native gebruik maakt van lazy loading. Het geheugen blijft in tegenstelling met de CPU rond de 202MB hangen wanneer de applicatie inactief en actief is, met verschillen van maximum 4-5MB. Er is geen merkbaar verschil in het geheugen wanneer er genavigeerd wordt tussen de schermen.

Schaalbaarheid

Complexiteit

Het gebruik van de libraries is niet complex. De navigatie is snel om op te zetten dankzij de documentatie van de library. Het laadscherm is ook niet complex om op te zetten. Het enige dat nodig is, is een icoon dat getoond zal worden en de initiële setup van de library.

Herbruikbaarheid

Het toevoegen van extra schermen is niet complex en kan dus gemakkelijk hergebruikt worden. Het is ook gemakkelijk om de bestaande schermen te hergebruiken in andere applicaties. Bij het laadscherm is er geen sprake van opschaling. Dit is enkel nodig bij het opstarten van de applicatie en zal dus maar 1 keer geïmplementeerd moeten worden.

6.3. Conclusie

Op basis van de verzamelde gegevens blijkt dat native ontwikkeling een betere performantie heeft dan cross-platform ontwikkeling voor de basisfunctionaliteiten. De gemiddelde opstarttijd van de native applicatie is sneller, met een gemiddelde van 2,58 seconden in vergelijking met 2,84 seconden bij cross-platform. Daarnaast heeft de native applicatie een lager gemiddeld geheugengebruik van ongeveer 81MB, terwijl de cross-platform applicatie gemiddeld 202MB aan geheugen vereist. Desondanks dat beide applicaties een vergelijkbaar CPU-gebruik hebben, toont dit onderzoek aan dat native applicaties voor basisfunctionaliteiten beter zijn op vlak van opstartsnelheid en geheugenverbruik.

	Native (Android)	Cross-platform (React Native)
	Tijdsduur	
Minimaal Maximaal Gemiddeld	1,37s 3,56s 2,58s	2,29s 6,59s 2,84s
	Geheugen	
Offset Gemiddeld	2-3MB 81MB	4-5MB 202MB
	CPU	
Minimaal Maximaal Gemiddeld	25% 42% 31%	27% 40% 32%

Met behulp van Android Studio en het startproject dat Android Studio aanbiedt, kunnen de basisfunctionaliteiten sneller worden geïmplementeerd bij native applicaties dan bij cross-platform. Dit komt omdat het installatieproces van de gebruikte libraries bij cross-platform meer werk vraagt dan bij native applicaties. Daarnaast moet de navigatie bij cross-platform zelf worden opgebouwd wat opnieuw meer tijd vraagt.

Op het gebied van schaalbaarheid zijn beide ontwikkelmethodes een goede keuze aangezien het bij beide ontwikkelmethodes gemakkelijk is om de navigatiecomponenten uit te breiden met extra schermen. Daarnaast is het ook gemakkelijk om bestaande schermen te hergebruiken.

Op vlak van performantie gaat de voorkeur naar native. Op vlak van ontwikkeltijd gaat de voorkeur naar cross-platform als er rekening wordt gehouden met het feit dat native enkel Android bevat. Op vlak van schaalbaarheid is er geen voorkeur tussen beide ontwikkelmethodes.

7

Audio- en videospelers

In dit hoofdstuk worden de audio- en videomogelijkheden van native en cross-platform vergeleken met elkaar. Met de resultaten kan dan een gepaste conclusie worden gevormd.

7.1. Native

Wat is er nodig

Om audio en video af te spelen binnen een applicatie wordt gebruik gemaakt van het MediaPlayer framework en de VideoView klasse. Deze stellen ons in staat om gemakkelijk audio en video af te spelen binnen onze applicatie. Er kunnen zowel lokale files als streams van het internet worden gebruikt.

Uitvoering

1. Permissions toevoegen

Indien het scherm van de applicatie niet mag uitvallen als er audio of video afspeelt, dan moet volgende permission worden toegevoegd aan het **AndroidManifest.xml** bestand.

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

2. Variabelen initialiseren, vinden en linken

Om de MediaPlayer en VideoView te gebruiken, moeten er eerst variabelen worden aangemaakt. Daarna moeten de variabelen gelinkt worden met het juiste element in de layout.

```
private var mediaPlayer: MediaPlayer? = null
private var videoView: VideoView? = null
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    // R.raw.sample is een audio file in de raw folder
    mediaPlayer = MediaPlayer.create(this, R.raw.sample)
    videoView = findViewById(R.id.videoView)
    videoView?.setVideoPath("Video URL")
}
```

3. Audio en video afspelen, pauzeren en stoppen

Om audio en video af te spelen, pauzeren en stoppen worden de **start()**, **pause()** en **stop()** methodes van de MediaPlayer en de VideoView gebruikt.

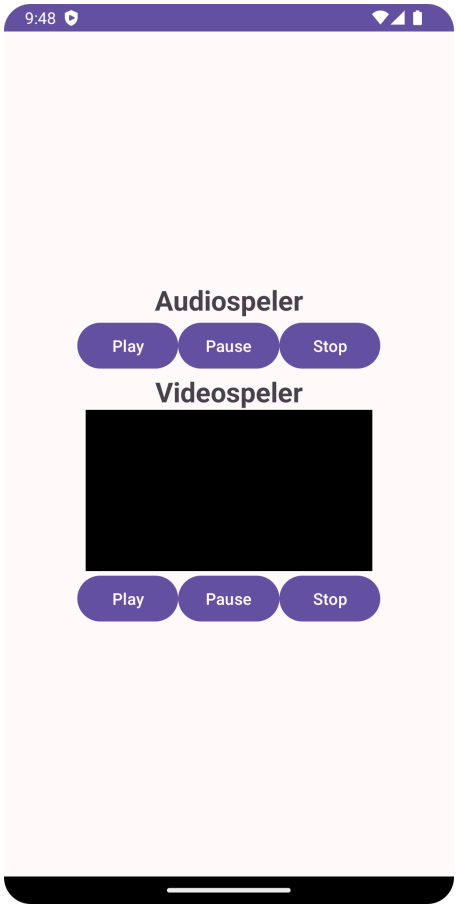
```
// Audio
mediaPlayer?.start()
mediaPlayer?.pause()
mediaPlayer?.stop()

// Video
videoView?.start()
videoView?.pause()
videoView?.stopPlayback()
```

Deze worden dan gekoppeld aan een knop in de layout om de audio en video af te spelen, pauzeren en stoppen.

```
val playButton = findViewById<Button>(R.id.playButton)
playButton.setOnClickListener {
    mediaPlayer?.start()
    videoView?.start()
}
```

4. Applicatie maken

Nu wordt de applicatie aangemaakt om audio en video af te spelen, pauzeren en stoppen. Deze bestaat uit een layout met telkens drie **Button** componenten voor het starten, pauzeren en stoppen van zowel audio als video. De video wordt afgespeeld in een **VideoView** component. De audio wordt afgespeeld met een **MediaPlayer** object.



Figuur (7.1)
Layout van applicatie voor het afspelen van audio en video bij Android.

Ontwikkeltijd

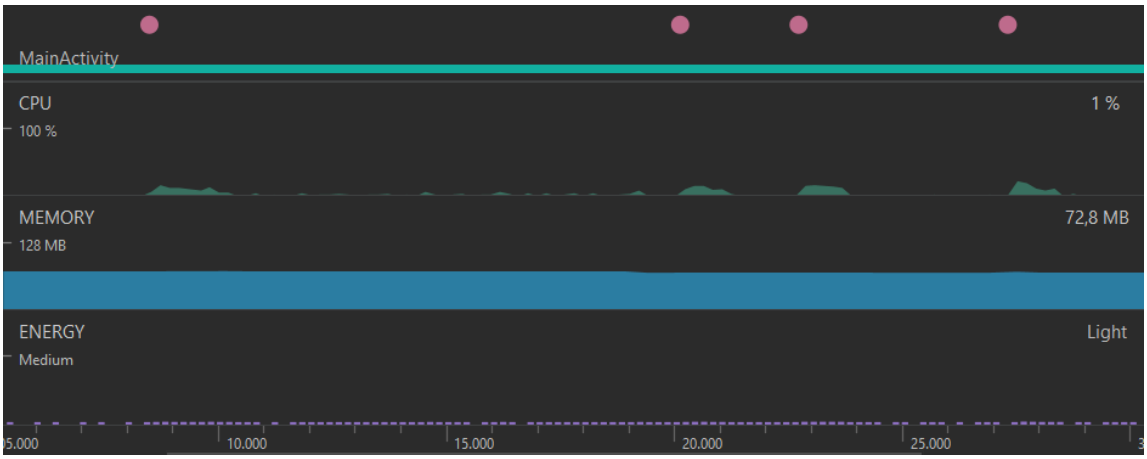
Het implementeren van een audio- en videospeler duurt niet lang. Er moet enkel een variabel worden aangemaakt die dan gelinkt wordt aan de juiste elementen in de layout. Daarna kunnen de gepaste methodes worden aangeroepen om het afspelen te regelen. Daardoor was de applicatie zeer snel gemaakt. Het duurde ongeveer 45 minuten om informatie op te zoeken en de functionaliteiten te implementeren.

Performantie

Tijdsduur

Bij het afspelen van audio of video is het niet mogelijk om een meting te doen. De audio of video die afspeelt, is ofwel spelend ofwel niet spelend. Daarom wordt er enkel gekeken naar het CPU en geheugen gebruik tijdens het afspelen van audio en video.

CPU & geheugen



Figuur (7.2)
Overzicht CPU en geheugen gebruik tijdens het afspelen van audio en video bij Android.

De eerste twee klik events zijn voor het starten en pauzeren van de video. Op de grafiek is te zien dat het CPU gebruik stijgt tot 17% en daarna afzwakt naar een wisselend gebruik van 0 - 7%. Bij het pauzeren van de video stijgt het CPU gebruik tot 16% en daarna zakt het terug naar 0%. Het geheugen gebruik tijdens het afspelen van de video komt overeen met het geheugen gebruik bij het inactief zijn van de applicatie. Er is dus geen verschil in geheugen gebruik bij het afspelen van de video.

De laatste twee klik events zijn voor het starten en pauzeren van de audio. Op de grafiek is te zien dat het CPU gebruik opnieuw stijgt tot 17%. In tegenstelling tot de video is er geen wisselend CPU gebruik. Het CPU gebruik blijft constant rond 0 - 1%. Bij het pauzeren van de audio stijgt het CPU gebruik tot 23% en zakt daarna terug naar 0%. Net zoals bij de video is er geen verschil in geheugen gebruik bij het afspelen van de audio. Het geheugen gebruik tijdens het afspelen van de audio komt opnieuw overeen met het geheugen gebruik bij het inactief zijn van de applicatie.

Schaalbaarheid

Complexiteit

Het implementeren van een audio en video speler in Android is zeer simpel. Er moet geen extra library of tool worden gebruikt. Er moet enkel een variabel worden aangemaakt en gelinkt worden aan de juiste elementen in de layout.

Herbruikbaarheid

Aangezien de code zeer simpel is en er geen extra library of tool wordt gebruikt, is de code zeer herbruikbaar. De code kan gebruikt worden in elke applicatie of klasse die audio en video moet afspelen. In verband met de schaling, is mogelijk

om de audio en video speler te abstracteren en deze dan vanuit één centrale plaats op te schalen.

7.2. Cross-platform

Wat is er nodig

Om audio en video af te spelen binnen een cross-platform applicatie wordt gebruik gemaakt van de react-native-track-player en react-native-video libraries. Deze stellen ons in staat om gemakkelijk audio en video af te spelen binnen onze applicatie. Er kunnen zowel lokale files als streams van het internet worden gebruikt.

Uitvoering

1. Library toevoegen

Eerst moeten de libraries aan de root van ons project worden toegevoegd. Deze worden toegevoegd met volgende commando's:

```
npm install --save react-native-track-player
npm install --save react-native-video
```

2. Package teruggeven

Normaal gezien moet de package dan worden toegevoegd aan het *android/app/src/main/java/com/project/MainApplication.java* bestand. Maar dit is niet meer nodig bij React Native 0.60+.

3. Audio library initialiseren

Om de audio speler te kunnen gebruiken, moet een service bestand worden geregistreerd in de main component van de applicatie, meestal is dit het *index.js* bestand.

```
import TrackPlayer from 'react-native-track-player';
// AppRegistry.registerComponent( ... );
TrackPlayer.registerPlaybackService(() ⇒ require('./service.js'));
```

Daarna moet er een service worden aangemaakt in een apart *service.js* bestand.

```
module.exports = async function() {
    // ...
}
```

Tot slot moet de TrackPlayer worden opgezet met de **setupPlayer()** methode. Deze methode moet worden aangeroepen voor de **TrackPlayer** kan worden gebruikt.

```
import TrackPlayer from 'react-native-track-player';

await TrackPlayer.setupPlayer()
```

4. Audio en video speler gebruiken

4.1. Audio Om de audio speler te gebruiken moet eerst een **Track** object worden aangemaakt.

```
const track = {
    url: require(''),
    title: 'Title',
    artist: 'Artist',
    artwork: require(''),
    duration: 100
};
```

Daarna wordt deze aan het **TrackPlayer** object toegevoegd.

```
await TrackPlayer.add(track);
```

Nu is het mogelijk om de audio te starten, pauzeren en stoppen met behulp van een aantal **<Button>** componenten.

```
<Button title="Play" onPress={() => TrackPlayer.play()} />
<Button title="Pause" onPress={() => TrackPlayer.pause()} />
<Button title="Stop" onPress={() => TrackPlayer.stop()} />
```

4.2. Video Om de video speler te gebruiken, moet eerst een **<Video>** component worden aangemaakt.

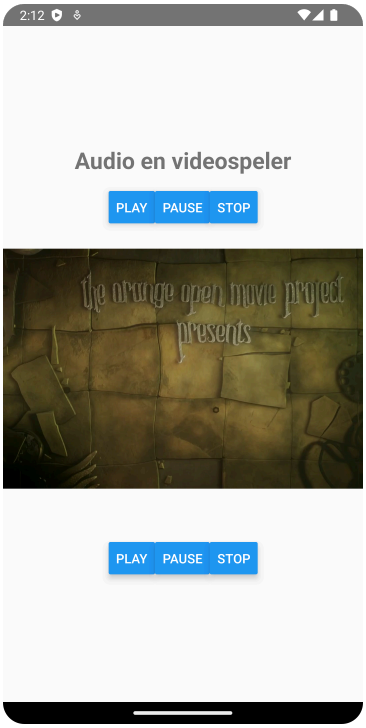
```
<Video source={require('')} />
```

Nu is het mogelijk om de video te starten, pauzeren en stoppen met behulp van een aantal **<Button>** componenten.

```
<Button title="Play" onPress={() => this.video.play()} />
<Button title="Pause" onPress={() => this.video.pause()} />
<Button title="Stop" onPress={() => this.video.stop()} />
```

5. Applicatie maken

Net zoals bij native bestaat de applicatie voor elke audio en video speler uit een **<View>** component met daarin de **<Button>** componenten om de audio of video te starten, te pauzeren of te stoppen. Daarnaast bevat de applicatie ook een **<View>** component met daarin een **<Video>** component voor het afspelen van de video.



Figuur (7.3)
Layout van applicatie voor het afspelen van audio en video bij React Native.

Ontwikkeltijd

De library react-native-track-player die wordt gebruikt is een uitgebreide library die complex kan overkomen. Daarom is de ontwikkeltijd hoger dan bij native. Ook moeten er meer stappen ondernomen worden om de TrackPlayer werkend te krijgen. De extra complexiteit van react-native-track-player kan ook een voordeel zijn. Indien de extra functionaliteit van react-native-track-player nodig is, kunnen deze gemakkelijk geïmplementeerd worden. Terwijl dit bij andere libraries of bij native niet altijd even gemakkelijk is.

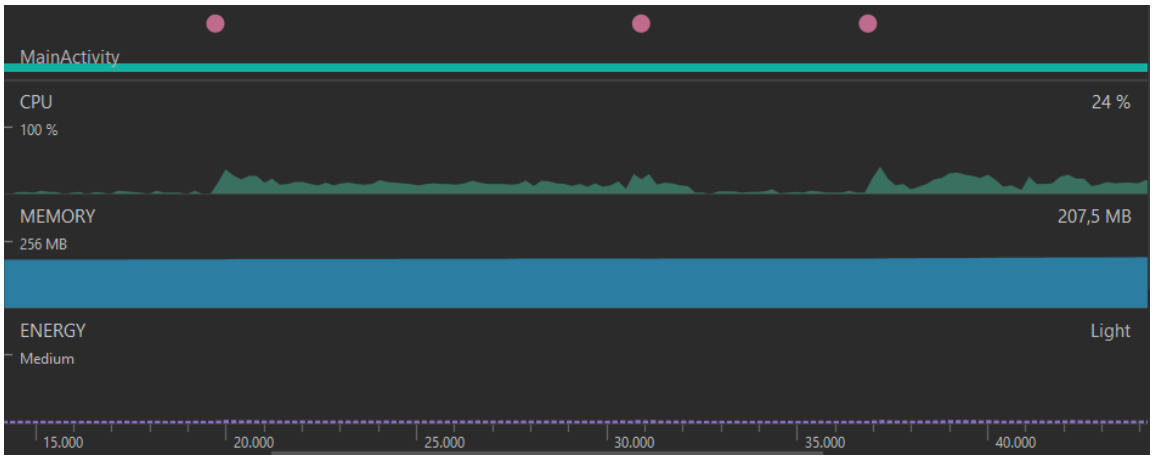
Om video's af te spelen, moest er terug een externe library geïmplementeerd worden, wat ook weer tijd in beslag neemt. Deze library is wel gemakkelijk om te implementeren en te gebruiken. In totaal heeft het ongeveer 1 uur en 45 minuten geduurd om de audio en video speler te implementeren.

Performantie

Tijdsduur

Net zoals bij native wordt er enkel naar het CPU en geheugen gebruik tijdens het afspelen van audio en video gekeken.

CPU & geheugen



Figuur (7.4)
Overzicht CPU en geheugen gebruik tijdens het afspelen van audio en video bij React Native.

Bij het eerste klik event voor het starten van de audio stijgt het CPU gebruik tot 42% en zakt daarna af tot een wisselend gebruik van 15 - 25%. Bij het tweede klik event voor het pauzeren van de audio stijgt het CPU gebruik tot 35%. Bij het derde klik event voor het starten van de video stijgt het CPU gebruik tot 43% en blijft dan schommelen tussen 10 - 35%. Tijdens het afspelen van de audio en video blijft het geheugen gebruik rond de 207MB hangen. Er is dus geen verschil in geheugen gebruik bij het afspelen van audio of video en het inactief zijn van de applicatie.

Schaalbaarheid

Complexiteit

De gebruikte library react-native-track-player kan nogal complex overkomen. De library biedt heel wat mogelijkheden aan die niet altijd nodig zijn. Ook zijn er redelijk wat stappen nodig om de TrackPlayer werkend te krijgen. In sommige situaties kan react-native-sound of react-native-sound-player een betere keuze zijn maar in sommige gevallen kan de extra complexiteit van react-native-track-player ook een voordeel zijn.
De gebruikte library react-native-video is gemakkelijk om te implementeren en te gebruiken. Na het installeren van de library is er direct toegang tot de **<Video>** component die dan gebruikt kan worden om video's af te spelen.

Herbruikbaarheid

Het is mogelijk om alle functionaliteiten van bijvoorbeeld de audio en video speler elk in een aparte component te steken, zodat deze gemakkelijk op andere plaatsen in de applicatie kunnen worden gebruikt. Maar dit is niet altijd nodig. In sommige gevallen is het ook mogelijk om de audio en video speler in één component te steken. Dit is bijvoorbeeld het geval wanneer de audio en video speler eenzelfde functionaliteit hebben. Desondanks is het wel mogelijk om de audio en video speler in

aparte componenten te steken en deze dan op meerdere plaatsen in de applicatie te gebruiken. Ook is het mogelijk om ze dan individueel op te schalen.

7.3. Conclusie

Bij het geheugen is te zien dat er geen verschil is tussen het afspelen van audio of video en het inactief zijn van de applicatie. Desondanks is het geheugengebruik bij cross-platform veel hoger dan bij native. Er wordt bij native maar 73MB gebruikt in vergelijking met 207MB bij cross-platform, een verschil van 183,56%.

Bij het CPU gebruik scoort native opnieuw beter dan cross-platform. Zowel bij het afspelen van audio als video is het CPU gebruik bij native lager. Bij het afspelen van audio bij native is de beginpiek 17% en de eindpiek 23%. Bij cross-platform is de beginpiek 42% en de eindpiek 35% terwijl het gemiddeld CPU gebruik bij native 0-1% is en bij cross-platform 20%. Bij het afspelen van video bij native is de beginpiek 17% en de eindpiek 16%. Bij cross-platform is de beginpiek 43% en de eindpiek 28% terwijl het gemiddeld CPU gebruik bij native 4% is en bij cross-platform 22%.

Audio		
	Native (Android)	Cross-platform (React Native)
	Geheugen	
Offset Gemiddeld	2-3MB 73MB	1-2MB 207MB
	CPU	
(Start)Piek (Eind)Piek Offset Gemiddeld	17% 23% / 0-1%	42% 35% 5% 20%
Video		
	Native (Android)	Cross-platform (React Native)
	Geheugen	
Offset Gemiddeld	2-3MB 73MB	1-2MB 207MB
	CPU	
(Start)Piek (Eind)Piek Offset Gemiddeld	17% 16% 3% 4%	43% 28% 12% 22%

De ontwikkeltijd bij native ligt veel lager dan bij cross-platform. Dit komt omdat native gemakkelijker is om te implementeren en te gebruiken. En omdat er geen extra libraries nodig zijn. Terwijl dit bij cross-platform wel het geval is. Het kan echter de moeite waard zijn om extra tijd te investeren in cross-platform ontwikkeling aangezien de gebruikte bibliotheek extra functionaliteit biedt, zoals bijvoorbeeld de mogelijkheid om audio en video in de achtergrond af te spelen. Dit kan helpen bij eventuele uitbreidingen in de toekomst.

Op basis van de verzamelde gegevens kan geconcludeerd worden dat voor het afspelen van audio en video native de beste keuze is. Dit komt omdat native beter scoort op vlak van performantie. Ook is native gemakkelijker te implementeren en te gebruiken. Daarnaast zijn er geen extra libraries nodig voor de implementatie. Let wel op, dit is enkel het geval wanneer er geen extra functionaliteit nodig is. Indien er extra functionaliteit nodig is, kan cross-platform een betere keuze zijn. Je zal in dat geval wel met een groot performantieverschil zitten.

8

Gebruik van sensoren

In dit hoofdstuk wordt het gebruik van sensoren bij native en cross-platform vergeleken met elkaar. Met de resultaten kan dan een gepaste conclusie worden gevormd.

8.1. Native

Wat is er nodig

Om toegang te krijgen tot de sensoren bij native ontwikkeling moeten er geen extra libraries of tools worden gebruikt. Android studio biedt een aantal klassen aan die gebruikt kunnen worden: SensorManager, SensorEventListener en SensorEvent. Dankzij deze klassen kan de data van sensoren zoals accelerometer en gyroscoop worden opgevraagd.

Uitvoering

1. Permission toevoegen

Om toegang te krijgen tot de sensoren moet er een user-permission toegevoegd worden aan het AndroidManifest.xml bestand.

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

2. SensorManager initialiseren

Om toegang te krijgen tot de sensoren moet er een instantie van de SensorManager klasse aangemaakt worden.

```
private lateinit var sensorManager: SensorManager
private var accelerometer: Sensor? = null
private var gyroscope: Sensor? = null
```

Daarna worden de variabelen met behulp van de sensorManager gelinkt aan een sensor met de onCreate methode.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
    accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
    gyroscope = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE)
}
```

3. SensorEventListener initialiseren

Daarna wordt de SensorEventListener aangemaakt, hiervoor wordt de onSensorChanged methode gebruikt.

```
private val sensorEventListener = object : SensorEventListener {
    override fun onSensorChanged(event: SensorEvent) {
        if (event.sensor.type == Sensor.TYPE_ACCELEROMETER) {
            val x = event.values[0]
            val y = event.values[1]
            val z = event.values[2]
            // Doe iets met de accelerometerwaarden (x, y, z)
        } else if (event.sensor.type == Sensor.TYPE_GYROSCOPE) {
            val x = event.values[0]
            val y = event.values[1]
            val z = event.values[2]
            // Doe iets met de gyroscoopwaarden (x, y, z)
        }
    }

    override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
        return // Niet nodig bij dit onderzoek
    }
}
```

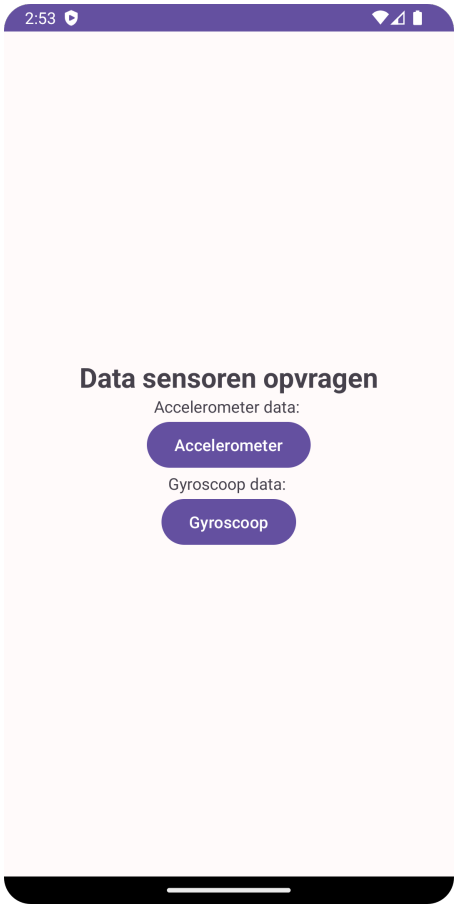
Nu kan de data van de sensoren worden uitgelezen in de onSensorChanged methode.

```
fetchButton.setOnClickListener {
    sensorManager.registerListener(
        sensorEventListener,
        sensor, // Veranderen door gyroscope of accelerometer
        SensorManager.SENSOR_DELAY_NORMAL
```

```
    )
}
```

4. Applicatie maken

Met deze informatie wordt een applicatie opgebouwd die de data van sensoren ophaalt. De applicatie bestaat uit twee **TextView** componenten voor de data van de accelerometer en gyroscoop en tot slot twee **Button** componenten om de data op te halen. Als de knoppen ingedrukt worden, dan worden de setOnClickListener methodes aangeroepen. In de onSensorChanged methode wordt de data van de sensoren opgehaald en in de TextViews geplaatst.



Figuur (8.1)
Layout van applicatie voor data van sensoren op te halen bij Android.

Ontwikkeltijd

Aangezien er geen extra libraries of tools gebruikt worden om de sensoren te gebruiken, kunnen de sensoren snel geïmplementeerd worden. Wat enkel dient te gebeuren, is het toevoegen van de juiste permission aan het AndroidManifest.xml bestand, het importeren van de juiste klasse en het aanroepen van de juiste methode. Daarom is de tijd nodig om de sensoren te gebruiken zeer laag. Er is ongeveer 45 minuten gespendeerd om de sensoren te implementeren, inclusief op-

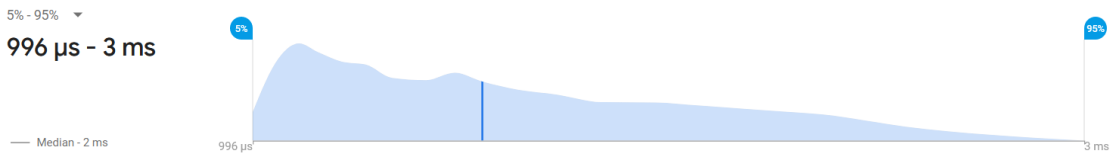
zoekwerk. Er zijn ook geen grote problemen of bugs voorgekomen tijdens de implementatie.

Performantie

Tijdsduur

Figuur (8.2)
Overzicht tijdsduur ophalen van accelerometer data bij Android.

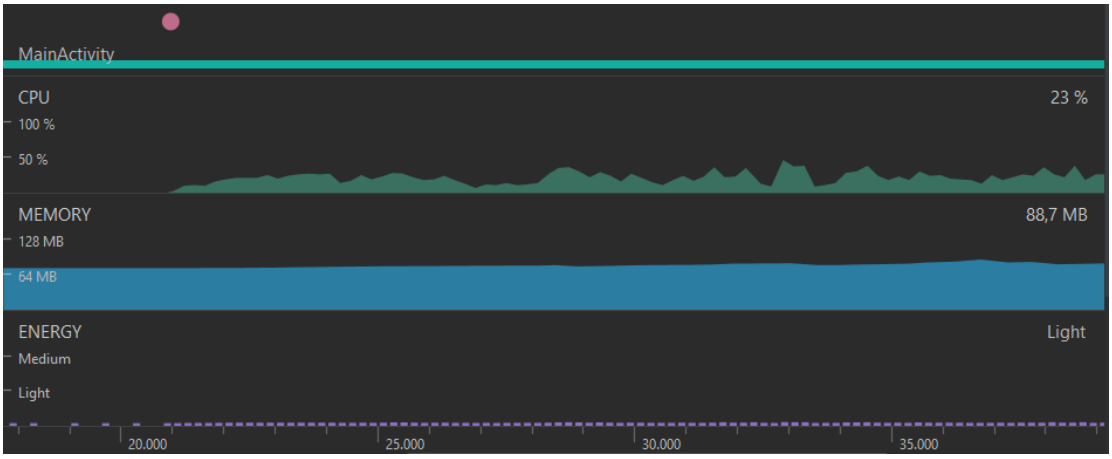
Van zodra er op de knop wordt gedrukt om gegevens op te halen van de accelerometer blijft de applicatie dit continu doen. Hierdoor worden er honderden metingen gedaan die ons vertellen dat het ophalen van de accelerometer data gemiddeld 2ms duurt. De minimum en maximum waarden liggen op 736µs en 7ms.



Figuur (8.3)
Overzicht tijdsduur ophalen van gyroscoop data bij Android.

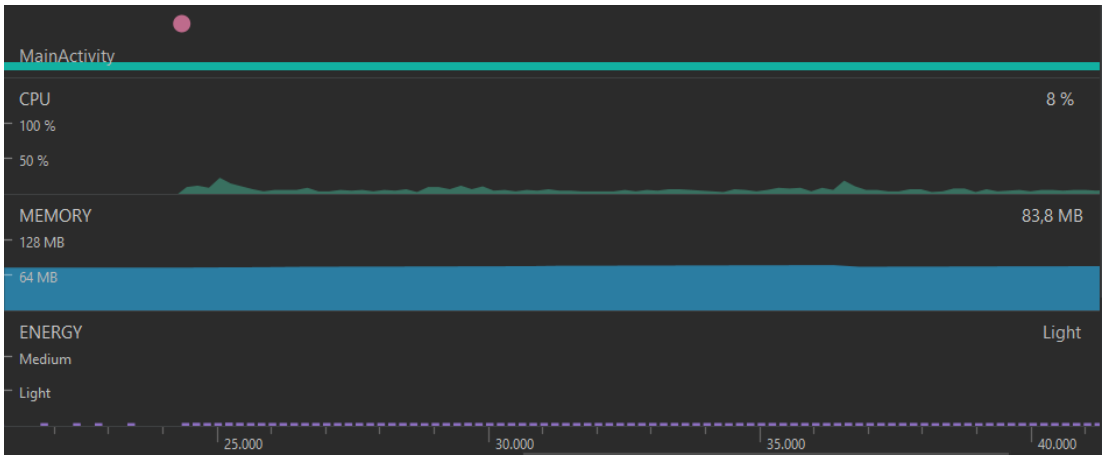
Net zoals bij de accelerometer worden er constant gegevens opgehaald van zodra er op de knop wordt gedrukt. Hierdoor worden er opnieuw honderden metingen gedaan die ons vertellen dat het ophalen van de gyroscoop data gemiddeld 2ms duurt. De minimum en maximum waarden liggen op 996µs en 3ms.

CPU & geheugen



Figuur (8.4)
Overzicht CPU en geheugen gebruik tijdens het ophalen van accelerometer data bij Android.

Op de grafiek is te zien dat het CPU gebruik van de applicatie bij het ophalen van de accelerometer data, gemiddeld 23% is met toch wel redelijke schommelingen. Wanneer er nog geen data wordt opgehaald, wordt de CPU niet gebruikt. Het geheugen blijft rond de 88MB hangen, met verschillen van maximum 4-5MB. Er is geen merkbaar verschil in het geheugen wanneer er data wordt opgehaald of wanneer er geen data wordt opgehaald.



Figuur (8.5)
Overzicht CPU en geheugen gebruik tijdens het ophalen van gyroscoop data bij Android.

Net zoals bij de accelerometer is op de grafiek te zien dat het CPU gebruik van de applicatie bij het ophalen van de gyroscoop data, gemiddeld 8% is. Wat opvalt, is dat er bij het ophalen van de gyroscoop data in het begin een piek is van 30%. Maar dat dit daarna terug zakt naar 8%. Bij de accelerometer was er geen piek te zien maar was het gemiddelde CPU gebruik wel hoger. Net zoals bij de accelerometer is er ook geen CPU gebruik wanneer er geen data wordt opgehaald. Het geheugen

blijft terug rond de 88MB hangen, met verschillen van maximum 4-5MB. Opnieuw is er geen merkbaar verschil in het geheugen wanneer er data wordt opgehaald of wanneer er geen data wordt opgehaald.

Schaalbaarheid

Complexiteit

Het proces om de data van sensoren te verkrijgen is vrij simpel. De ontwikkelaar moet enkel de juiste klasse importeren, de juiste methode aanroepen en de juiste permissies toevoegen aan de AndroidManifest.xml file.

Herbruikbaarheid

Het is gemakkelijk om de gegevens van de sensoren te hergebruiken. De gegevens kunnen bijvoorbeeld worden opgeslagen in een object in plaats van te tonen in een TextView component. Dat object kan dan vanuit andere klassen worden opgevraagd. Het is ook mogelijk om de logica voor het ophalen van de gegevens in een aparte klasse te plaatsen. Zo kan de logica gemakkelijk worden hergebruikt in andere klassen en kan deze ook aangepast of opgeschaald worden.

8.2. Cross-platform

Wat is er nodig

Om bij React Native toegang te krijgen tot de sensoren moet er gebruik worden gemaakt van de react-native-sensors library. Deze library biedt onder andere toegang tot de accelerometer en gyroscoop. Het is een wrapper voor de native sensoren van Android en iOS.

Uitvoering

1. Library toevoegen

Eerst moet de React Native Sensors library worden toegevoegd aan de root van ons project. Deze wordt toegevoegd met volgend commando:

```
npm install react-native-sensors --save
```

2. Package teruggeven

Normaal gezien moet de package dan worden toegevoegd aan het *android/app/src/* **main/java/com/project/MainApplication.java** bestand. Maar dit is niet meer nodig bij React Native 0.60+.

3. Gradle instellingen aanpassen

Eerst moet volgende regel worden toegevoegd aan het *android/app/build.gradle* bestand.

```
implementation project(':react-native-sensors')
```

Daarna moet volgende regel worden toegevoegd aan het *android/settings.gradle* bestand.

```
include ':react-native-sensors'
project(':react-native-sensors').projectDir =
    new File(rootProject.projectDir
        , '../node_modules/react-native-sensors/android')
```

De package is nu volledig geïnstalleerd en klaar voor gebruik.

4. Sensor gebruiken

Eerst wordt de library in het bestand geïmporteerd.

```
import { accelerometer, gyroscope } from "react-native-sensors";
```

Daarna worden twee variabelen die de data van de sensoren zullen bewaren aangemaakt.

```
const [accelerometerData, setAccelerometerData] = useState({});
const [gyroscopeData, setGyroscopeData] = useState({});
```

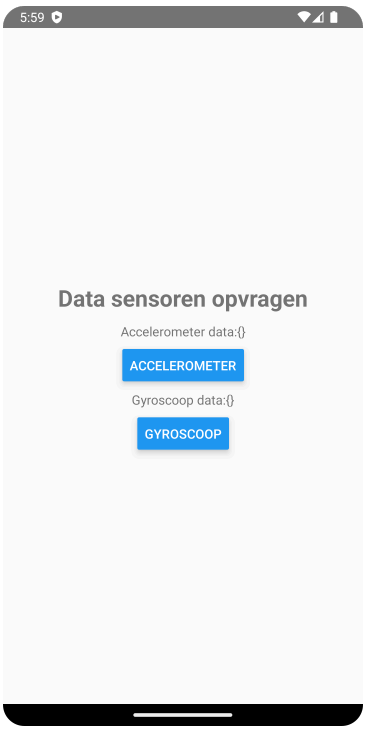
Tot slot kunnen deze variabelen gebruikt worden om de data van de sensoren op te slaan.

```
const getData = () => {
    setIsFetchingData(true);

    startFetchingData();
};

const startFetchingData = () => {
    if (isFetchingData) {
        const accelerometerSubscription = new Accelerometer({
            updateInterval: 100, // Verander dit indien nodig
        }).subscribe(({ x, y, z }) => {
            // Doe iets met de accelerometerwaarden (x, y, z)
        });

        const gyroscopeSubscription = new Gyroscope({
        updateInterval: 100, // Verander dit indien nodig
        }).subscribe(({ x, y, z }) => {
            // Doe iets met de accelerometerwaarden (x, y, z)
        });

        // Unsubscribe van de sensoren wanneer je klaar bent
```

```
        // met het ophalen van de data
        return () => {
            accelerometerSubscription.unsubscribe();
            gyroscopeSubscription.unsubscribe();
        };
    }
};
```

4. Applicatie maken

Net zoals bij de native applicatie wordt een applicatie gemaakt die de accelerometer en gyroscoop data opvraagt en weergeeft. Deze bestaat uit twee **<Text>** componenten om de data van de accelerometer en gyroscoop weer te geven en tot slot twee **<Button>** componenten om de data op te halen. Als de knoppen ingedrukt worden, dan wordt ofwel de **getAccelerometerData** of **getGyroscopeData** methode aangeroepen. In deze methodes wordt de data van de sensoren opgehaald en in de **<Text>** componenten geplaatst.



Figuur (8.6)
Layout van applicatie voor data van sensoren op te halen bij React Native.

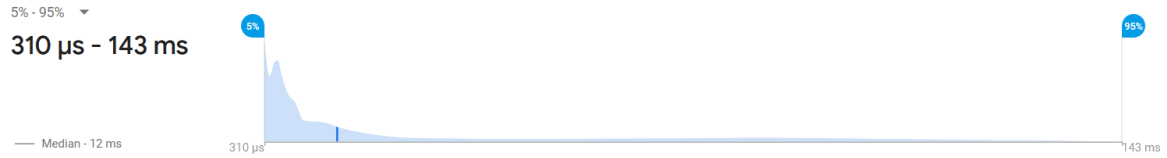
Ontwikkeltijd

In vergelijking met native ontwikkeling moeten er meer stappen ondernomen worden om de sensoren te gebruiken. Eerst en vooral moet er een externe library geïmplementeerd worden om de sensoren te gebruiken. Pas daarna kunnen de sensoren worden gebruikt. Daardoor is de tijd die nodig is om de sensoren te gebruiken

hoger dan bij native ontwikkeling. Er is ongeveer 1 uur en 15 minuten gespendeerd om de sensoren te gebruiken. Er zijn ook geen grote problemen of bugs voorgekomen tijdens de implementatie.

Performantie

Tijdsduur



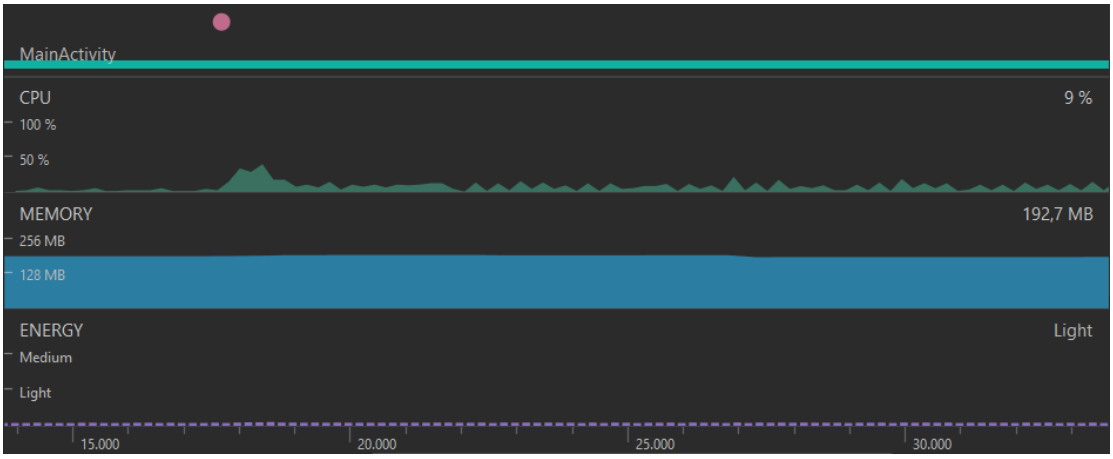
Figuur (8.7)
Overzicht tijdsduur ophalen van accelerometer data bij React Native.

Net zoals bij native worden er constant gegevens opgehaald van zodra de knop wordt ingedrukt. Hierdoor worden er opnieuw honderden metingen gedaan die ons vertellen dat het ophalen van de accelerometer data gemiddeld 12ms duurt. De minimum en maximum waarden liggen op 310µs en 143ms. Wat wel opvalt bij React Native, is dat de eerste meting altijd langer duurt dan de rest. Hierdoor is het maximum veel hoger dan bij native. Bij native liggen de metingen dichter bij elkaar.

Figuur (8.8)
Overzicht tijdsduur ophalen van gyroscoop data bij React Native.

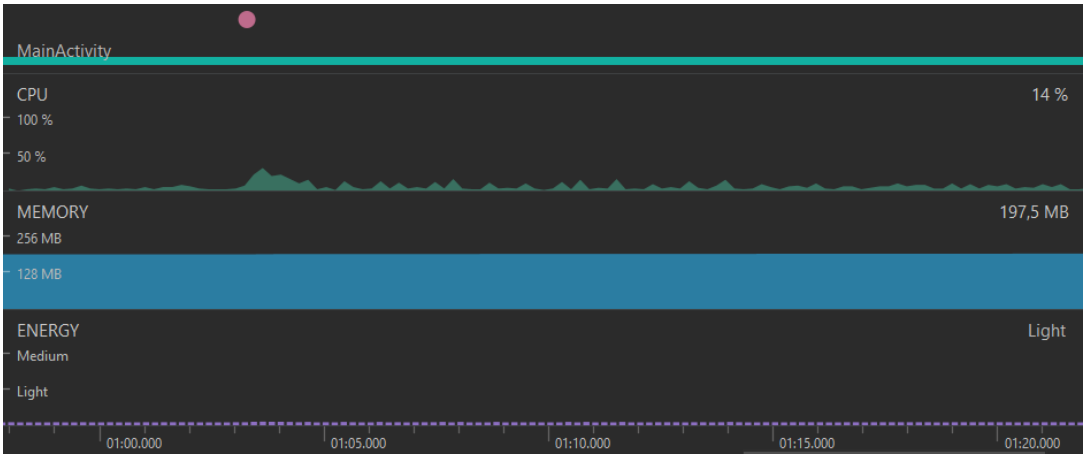
Net zoals bij de accelerometer worden er constant gegevens opgehaald van zodra de knop wordt ingedrukt. Hierdoor worden er opnieuw honderden metingen gedaan die ons vertellen dat het ophalen van de gyroscoop data gemiddeld 11ms duurt. De minimum en maximum waarden liggen op 203µs en 163ms. Maar zoals bij de accelerometer is de eerste meting opnieuw veel trager dan de rest.

CPU & geheugen



Figuur (8.9)
Overzicht CPU en geheugen gebruik tijdens het ophalen van accelerometer data bij React Native.

Op de grafiek is te zien dat het CPU gebruik van de applicatie bij het ophalen van de accelerometer data, piekt tot net geen 50% en gemiddeld 9% is na de piek. Wat opvalt, is dat het CPU gebruik tijdens het ophalen van de data schommelt tussen 0% en 20%. Het geheugengebruik blijft stabiel rond de 192MB hangen, met verschillen van maximum 3-4MB. Ook is er opnieuw geen merkbaar verschil in het geheugen wanneer er data wordt opgehaald of wanneer er geen data wordt opgehaald.



Figuur (8.10)
Overzicht CPU en geheugen gebruik tijdens het ophalen van gyroscoop data bij React Native.

Net zoals bij native is er een piek in het CPU gebruik bij het starten van het ophalen van de gyroscoop data. De piek ligt hier wel hoger dan bij native, namelijk 40%. Na de piek ligt het gemiddeld CPU gebruik rond de 10%. Net zoals bij de accelerometer schommelt het CPU gebruik tussen 0% en 20%. Het geheugen gebruik blijft terug stabiel rond de 197MB hangen, met verschillen van maximum 3-4MB. Net zoals bij de accelerometer is er geen merkbaar verschil in het geheugen wanneer er data

wordt opgehaald of wanneer er geen data wordt opgehaald.

Schaalbaarheid

Complexiteit

Het gebruik van de sensoren is net zoals bij native vrij simpel, ondanks het feit dat er een extra library moet worden geïmplementeerd. De package zorgt ervoor dat het ophalen van de data van de sensoren overeenkomt met het ophalen van de data van de sensoren bij native.

Herbruikbaarheid

Net zoals bij native is het gemakkelijk om de gegevens van de sensoren te hergebruiken. De gegevens van de sensoren kunnen bijvoorbeeld in een globale variabel of context worden gestoken. Andere componenten kunnen daardoor de gegevens opvragen. Het is ook mogelijk om de logica voor het ophalen van de gegevens in een aparte useContext te plaatsen. Zo kan de logica gemakkelijk worden hergebruikt in andere componenten en kan deze ook aangepast of opgeschaald worden.

8.3. Conclusie

De tijdsduur van het ophalen van de data van de sensoren is bij native veel sneller dan bij cross-platform. Dit komt omdat de library die gebruikt wordt bij cross-platform een wrapper is voor de native sensoren. Hierdoor moet de data van de sensoren eerst naar de wrapper gestuurd worden en pas daarna naar de applicatie. Bij native is dit niet het geval: de data van de sensoren wordt rechtstreeks naar de applicatie gestuurd. Het ophalen van data is daardoor gemiddeld 10ms trager bij cross-platform dan bij native.

Bij het geheugengebruik is er geen verschil tussen het ophalen van sensoren data en evenmin wanneer er niks gebeurt in de applicatie. Ondanks het feit dat het ophalen van data niet beïnvloed wordt, wordt er 121.59% meer geheugen gebruikt bij React Native dan bij Android (88MB in vergelijking met gemiddeld 195MB). Dit komt omdat cross-platform een extra library moet implementeren. Deze library neemt ook geheugen in beslag.

Bij het CPU gebruik zijn er onderling tussen de accelerometer en gyroscoop ook verschillen. Bij native heeft de accelerometer geen piek maar blijft deze constant op gemiddeld 23% CPU gebruik. Bij cross-platform heeft de accelerometer een piek van 50% CPU gebruik waarna deze naar 9% zakt, een groot verschil. De gyroscoop heeft bij native een piek 30% CPU gebruik waarna deze naar 8% zakt. Bij cross-platform is de piek 40% CPU gebruik waarna deze naar 10% zakt. Daarnaast is er bij alle sensoren een schommeling van 5 tot 20%. Dit komt omdat de sensoren

data constant wordt opgehaald. Normaal zou er verwacht worden dat de sensoren bij native minder CPU gebruiken, aangezien de data eerst naar de wrapper wordt gestuurd en dan pas naar de applicatie. Maar bij de accelerometer is dit niet het geval. Hier is het CPU gebruik bij cross-platform lager dan bij native.

Accelerometer		
	Native (Android)	Cross-platform (React Native)
	Tijdsduur	
Minimaal Maximaal Gemiddeld	736µs 7ms 2ms	310µs 143ms 12ms
	Geheugen	
Offset Gemiddeld	2-3MB 88MB	3-4MB 192MB
	CPU	
(Begin)Piek Offset Gemiddeld	/ 20% 23%	50% 10% 9%
Gyroscoop		
	Native (Android)	Cross-platform (React Native)
	Tijdsduur	
Minimaal Maximaal Gemiddeld	996µs 3ms 2ms	203µs 163ms 11ms
	Geheugen	
Offset Gemiddeld	4-5MB 88MB	3-4MB 197MB
	CPU	
(Begin)Piek Offset Gemiddeld	30% 5% 8%	40% 10% 10%

De ontwikkeltijd voor het implementeren van sensoren bij native en cross-platform is ongeveer hetzelfde. Buiten het feit dat cross-platform een extra library moet implementeren waardoor de ontwikkeltijd iets langer wordt.

Op vlak van schaalbaarheid is er geen verschil tussen native en cross-platform. Beide applicaties kunnen gemakkelijk uitgebreid worden met extra sensoren en de bestaande logica van de sensoren kan gemakkelijk hergebruikt of opgeschaald worden.

Uit deze resultaten kan geconcludeerd worden dat applicaties die intensief gebruik maken van de sensoren beter native gebruiken aangezien deze sneller gegevens kan ophalen. Daarnaast is cross-platform beter in gebruik voor eenvoudige applicaties die de sensoren niet intensief gebruiken.

9

Notificaties

In dit hoofdstuk worden de lokale notificaties van native en cross-platform vergeleken met elkaar. Met de resultaten kan dan een gepaste conclusie worden gevormd.

9.1. Native

Wat is er nodig

Om lokale notificaties bij Android te kunnen aanmaken, wordt de NotificationCompat API aangeboden door de Android support library. Hierdoor kunnen titel, tekst, pictogram en andere inhoud van de notificatie ingesteld worden. Notificaties kunnen ook worden ingesteld om speciale acties te ondernemen bij het openen van de applicatie via de notificatie. Er kan eventueel ook een prioriteit worden meegegeven.

Uitvoering

1. Dependancy toevoegen

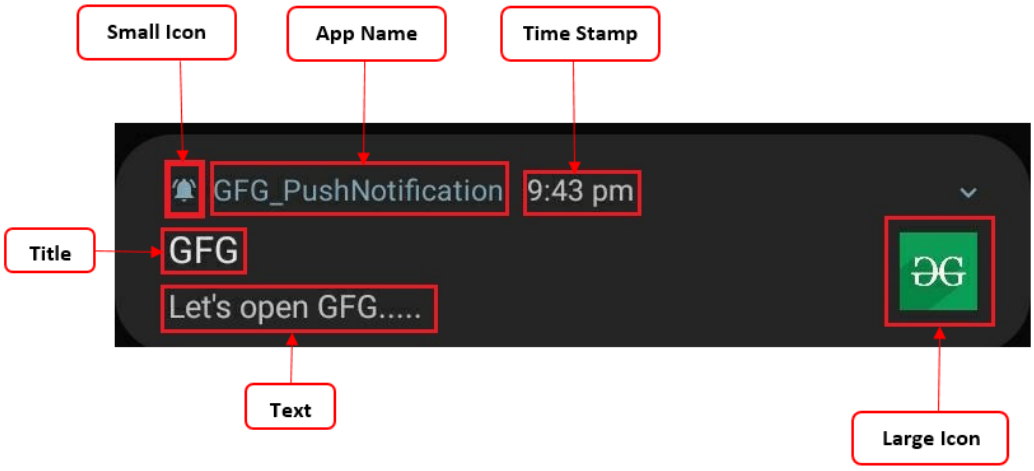
Om de NotificationCompat API te kunnen gebruiken, moeten er normaal gezien geen extra dependancies worden toegevoegd. Maar er moet wel gecontroleerd worden of de volgende dependancy aanwezig is.

```
val core_version = "1.6.0"
dependencies {
    implementation("androidx.core:core-ktx:$core_version")
}
```

2. Notificatie aanmaken

Om een notificatie aan te maken, moet de NotificationCompat.Builder object worden gebruikt.

```
  private fun createNotification(title: String, body: String) {
    val builder = NotificationCompat.Builder(this, CHANNEL_ID)
      .setSmallIcon(icon)
      .setContentTitle(title)
      .setContentText(body)
      .setPriority(NotificationCompat.PRIORITY_DEFAULT)

    val notificationManager: NotificationManager =
      getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    notificationManager.notify(notificationId, builder.build())
}
```



Figuur (9.1)
Anatomy standaard notificatie (the dir one, 2020).

3. Channel aanmaken

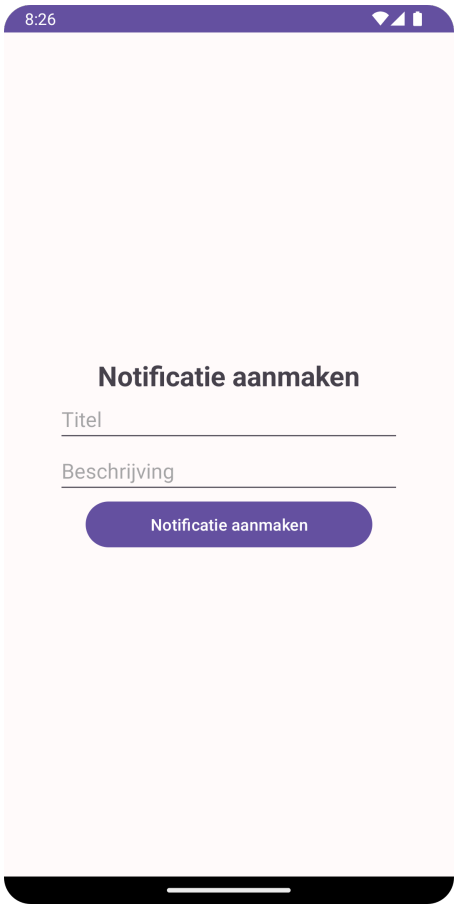
Het laatste dat nodig is vooraleer de notificatie getoond kan worden, is een channel. Deze wordt gebruikt om notificaties te groeperen volgens hun belang. Om een channel aan te maken, wordt de .createNotificationChannel() methode gebruikt. Het maakt niet uit wanneer of waar deze wordt uitgevoerd. Het is enkel belangrijk dat de channel wordt aangemaakt vooraleer notificaties worden getoond. Dus best zo vroeg mogelijk in de applicatie.

```
private fun createNotificationChannel() {
  if (Build.VERSION.SDK_INT ≥ Build.VERSION_CODES.O) {
    val name = "bachproef"
    val descriptionText = "bachproef notificaties"
    val importance = NotificationManager.IMPORTANCE_DEFAULT
    val channel = NotificationChannel(CHANNEL_ID, name, importance).apply {
```

```
      description = descriptionText
    }
    // Channel bij het systeem registreren
    val notificationManager: NotificationManager =
      getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    notificationManager.createNotificationChannel(channel)
  }
}
```

4. Applicatie maken

Met deze informatie kan er een applicatie worden opgebouwd die notificaties zal sturen. De applicatie bestaat uit twee **EditText** componenten voor de titel en beschijving van de notificatie en tot slot een **Button** om de notificatie te triggeren. Als de knop wordt ingedrukt dan wordt de **createNotification** methode aangeroepen en wordt de waarde van de inputs meegegeven.



Figuur (9.2)
Layout van applicatie voor notificaties te sturen bij Android.

Ontwikkeltijd

Ondanks het feit dat er zich een probleem heeft voorgedaan bij het implementeren van de functionaliteit, was het implementeren van de notificaties vrij simpel. Als de tijdsduur van het probleem niet wordt meegerekend dan kostte het 1 uur en 15 minuten om notificaties te sturen.

Problemen

Naast de tijd die nodig was om de functionaliteit te implementeren, is er ook 2 uur verloren gegaan aan een probleem. Waarom het probleem zich voordeed is niet duidelijk maar de applicatie vroeg, ondanks het feit dat de **POST_NOTIFICATIONS** bij user-permissions in het AndroidManifest.xml bestand stond, niet om toestemming om notificaties te sturen. Hierdoor leek het alsof er geen notificaties werden aangemaakt terwijl het probleem was dat de applicatie geen toestemming had gegeven om notificaties te ontvangen.

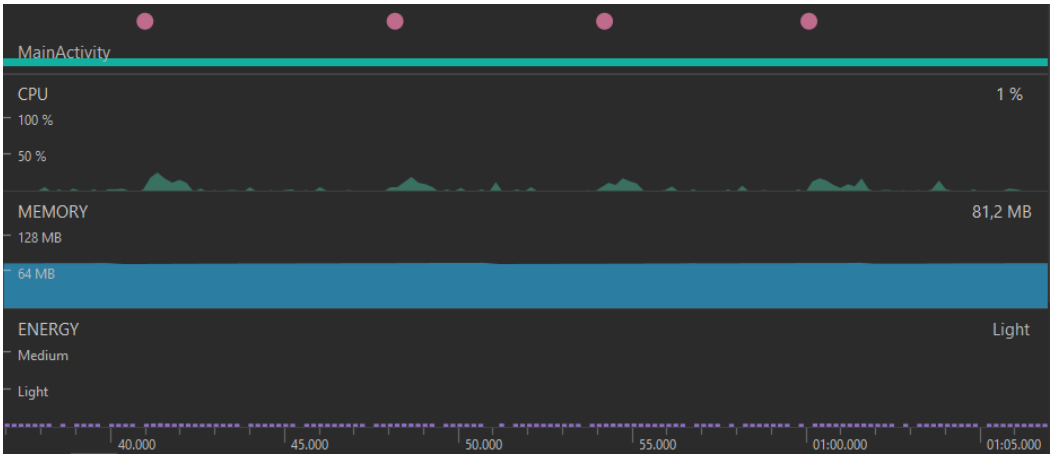
Performantie

Tijdsduur

Figuur (9.3)
Overzicht tijdsduur aanmaken notificaties bij Android.

Tijdens het meten van de duur voor het aanmaken van een notificatie, is er 10 keer een notificatie aangemaakt. Na 10 keer een notificatie aan te maken, is er een gemiddelde duur van 12ms en met een minimum en maximum van 4ms en 88ms.

CPU & geheugen



Figuur (9.4)
Overzicht CPU en geheugen gebruik tijdens aanmaken notificaties bij Android.

Op de grafiek is te zien dat het CPU gebruik van de applicatie rond de 4% ligt wanneer deze inactief is. Het is ook duidelijk zichtbaar wanneer een notificatie wordt aangemaakt. De piek van het CPU gebruik lag gemiddeld op 21% en schommelde tussen de 19% en 27%. Het geheugen blijft in tegenstelling tot de CPU rond de 81MB hangen, wanneer de applicatie inactief en actief is, met verschillen van maximum 2-3MB. Er is geen merkbaar verschil in het geheugen wanneer een notificatie wordt aangemaakt.

Schaalbaarheid

Complexiteit

De NotificationCompat API die Android aanbiedt, is gemakkelijk in gebruik. Er moeten normaal gezien geen extra stappen ondernomen worden om de notificaties te gebruiken.

Herbruikbaarheid

Het is mogelijk om alle delen van de logica weg te steken in methodes. Op die manier is het gemakkelijk om de notificaties later uit te breiden door bijvoorbeeld meerdere channels aan te maken of op andere plaatsen in het programma een notificatie aan te maken. Aangezien deze zich allemaal op één plaats bevinden, is het ook gemakkelijk om de logica van een notificatie aan te passen.

9.2. Cross-platform

Wat is er nodig

Om lokale notificaties bij React Native te tonen, wordt de library Notifee gebruikt. Deze library geeft toegang om titel, tekst, pictogram en andere inhoud van de notificatie in te stellen, net zoals bij Android. Daarnaast kunnen er ook speciale acties of prioriteiten worden ingesteld.

Uitvoering

1. Library toevoegen

Eerst moet de Notifee library aan de root van ons project worden toegevoegd. Deze wordt toegevoegd met volgend commando:

```
npm install --save @notifee/react-native
```

2. Toestemming vragen & channel aanmaken

Nog voor een notificatie kan getoond worden, moet er eerst toestemming worden gevraagd. Aangezien dit bij React Native niet automatisch gebeurt zoals bij Android moet de toestemming programmatisch gevraagd worden. Daarnaast zal ook een channel moeten aangemaakt worden. Net zoals bij Android maakt het niet uit wanneer dit gebeurt maar het moet wel gebeuren voordat er notificaties worden gestuurd. Dus best zo vroeg mogelijk in de applicatie.

```
export const notificationsSetup = async () => {
  const PERMISSION = await notifee.requestPermission();

  if(PERMISSION.authorizationStatus === AuthorizationStatus.DENIED) {
    return;
  }

  await notifee.createChannel({
    id: 'bachproef',
    name: 'BachproefNotificaties',
  });
}
```

3. Notificatie aanmaken

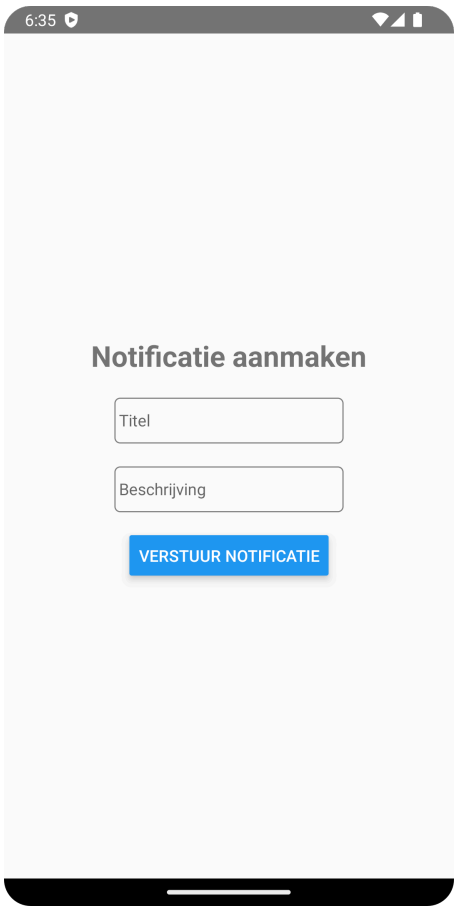
Bij React Native worden notificaties direct getoond bij het maken ervan. Net zoals bij Android wordt er een naam, description, channelId en optioneel een icoon (default ic_launcher) meegegeven.

```
export const createNotification = async (title: string, body: string) => {
  await notifee.displayNotification({
```

```
    title,
    body,
    android: {
      channelId: 'bachproef',
      pressAction: {
        id: 'default',
      },
    },
  });
}
```

4. Applicatie maken

Net zoals bij native zal de applicatie bestaan uit twee **<TextInput/>** componenten voor een titel en beschijving van de notificatie en tot slot een **<Button/>** om de notificatie te triggeren. Als de knop wordt ingedrukt dan wordt de **createNotification** methode aangeroepen en wordt de waarde van de inputs meegegeven.



Figuur (9.5)
Layout van applicatie voor notificaties te sturen bij React Native.

Ontwikkeltijd

Ondanks dat er een externe library gebruikt en geïmplementeerd moest worden, ging implementeren van de notificaties vlot. Inclusief het opzoekwerk en de implementatie is er 1 uur gespendeerd om de notificaties te implementeren. Er zijn ook geen grote problemen of bugs voorgekomen tijdens de implementatie.

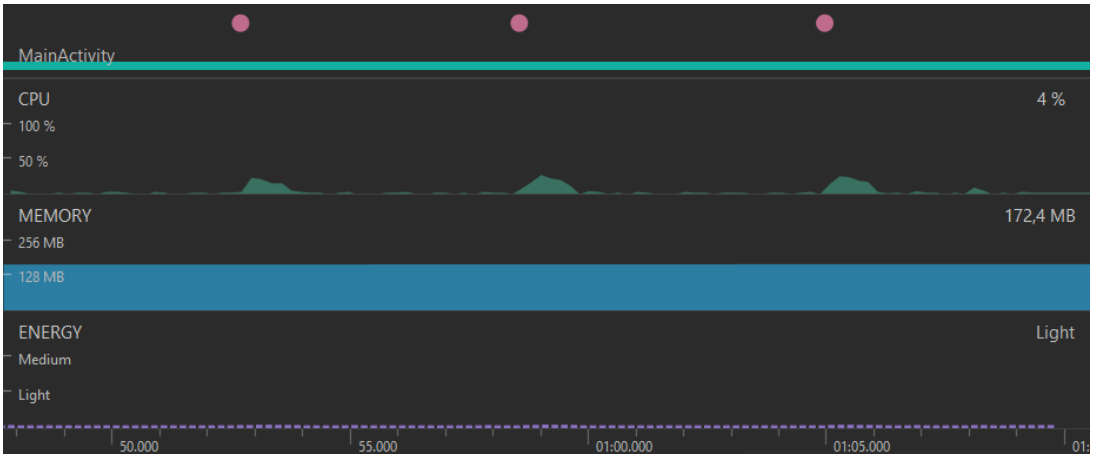
Performantie

Tijdsduur

Figuur (9.6)
Overzicht tijdsduur aanmaken notificaties bij React Native.

Tijdens het meten van de duur voor het aanmaken van een notificatie, is er net zoals bij native 10 keer een notificatie aangemaakt. Na 10 keer een notificatie aan te maken is er een gemiddelde duur van 166ms en met een minimum en maximum van 141ms en 297ms voor het aanmaken van een notificatie.

CPU & geheugen



Figuur (9.7)
Overzicht CPU en geheugen gebruik tijdens aanmaken notificaties bij React Native.

Op de grafiek is te zien dat het CPU gebruik van de applicatie wanneer deze inactief is, rond de 4% ligt. Daarnaast is het duidelijk zichtbaar wanneer een notificatie wordt aangemaakt. De piek van het CPU gebruik lag gemiddeld op 28% en schommelde tussen de 25% en 30%. Het geheugen blijft in tegenstelling tot de CPU wanneer de applicatie inactief en actief is, rond de 172MB hangen, met verschillen van

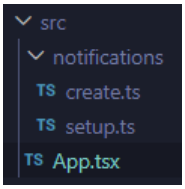
maximum 0,5MB. Er is geen merkbaar verschil in het geheugen wanneer een notificatie wordt aangemaakt.

Schaalbaarheid

Complexiteit

De complexiteit van de library valt goed mee, dankzij de beschikbare documentatie van de library is het zeer gemakkelijk om alle mogelijkheden van de library terug te vinden en zelf in een project te implementeren.

Herbruikbaarheid



Figuur (9.8)
Structuur notificaties implementatie React Native.

Door de logica van de library op te delen, is het gemakkelijk om later opnieuw dezelfde logica op andere plaatsen in de code te gebruiken. Er moet enkel een methode geïmporteerd worden om een notificatie aan te maken. Aan die methode wordt dan de titel en beschrijving meegegeven. Het is ook mogelijk om de logica voor het aanmaken van een notificatie aan te passen. Dit kan dan gemakkelijk gebeuren in het **create.ts** bestand. Of er kan een nieuwe methode worden aangemaakt om bijvoorbeeld de notificatie in te plannen op een later tijdstip.

9.3. Conclusie

Bij de tijdsduur die nodig is om een notificatie aan te maken, is er een duidelijk verschil tussen Android en React Native. Gemiddeld is het aanmaken van een notificatie bij Android 154ms sneller dan bij React Native. Bij React Native duurt het gemiddeld 166ms om een notificatie aan te maken terwijl dit bij Android maar 12ms duurt. Dit kan liggen aan het feit dat de externe library de code moet omzetten naar platformspecifieke code waardoor er veel tijd wordt verloren.

Bij het geheugen is er geen verschil tussen het aanmaken van een notificatie en wanneer er niks gebeurt in de applicatie. Ondanks het feit dat er geen invloed is bij het aanmaken van een notificatie, gebruikt React Native 112,35% meer geheugen dan Android (81MB in vergelijking met 172MB). Dit kan opnieuw liggen aan het feit dat de externe library de code moet omzetten naar platformspecifieke code waardoor er veel geheugen wordt gebruikt.

Bij het CPU gebruik is er wel een verschil tussen het aanmaken van een notificatie en wanneer de applicatie inactief is. Daarnaast is er ook een verschil tussen Android en React Native: bij React Native springt het CPU gebruik gemiddeld naar 28% terwijl dit bij Android naar 21% springt.

	Native (Android)	Cross-platform (React Native)
	Tijdsduur	
Minimaal Maximaal Gemiddeld	4ms 88ms 12ms	141ms 166ms 297ms
	Geheugen	
Offset Gemiddeld	2-3MB 81MB	0,5MB 172MB
	CPU	
Minimaal Maximaal Gemiddeld	19% 27% 21%	25% 30% 28%

De ontwikkeltijd van notificaties met beide ontwikkelmethodes blijft relatief gelijk, bij beide applicaties moet er toestemming gevraagd worden. Bij Android gebeurt dit door de permission aan het **AndroidManifest.xml** bestand toe te voegen, en bij React Native door de **notifee.requestPermission()** methode aan te roepen. Hierna moet er bij beide applicaties een channel worden aangemaakt en tot slot de notificatie.

Op vlak van schaalbaarheid zijn beide ontwikkelmethodes net zoals bij de ontwikkeltijd gelijk. Allebei geven ze de mogelijkheid om de logica weg te steken, waardoor deze later opnieuw gebruikt of opgeschaald kan worden.

Over het algemeen is het aanmaken van notificaties bij Android sneller dan bij React Native. Dat wil niet zeggen dat het de betere keuze is. Aangezien de schaalbaarheid en ontwikkeltijd vergelijkbaar zijn en de performantie niet zoveel verschilt, is het aangeraden om cross-platform te gebruiken voor het aanmaken van notificaties.

10

Conclusie

10.1. Hoofdonderzoeksvraag

In dit onderzoek werd er gezocht naar een antwoord op de hoofdonderzoeksvraag:

- Hoe verschillen de prestaties, schaalbaarheid en ontwikkeltijd van functionaliteiten tussen native en cross-platform ontwikkeling van mobiele applicaties?

Om een antwoord te vinden op deze vraag werden er twee applicaties ontwikkeld, één native en één cross-platform. Bij deze applicaties werden dan verschillende functionaliteiten geïmplementeerd en is de performantie getest. Daarnaast is er ook gekeken naar de ontwikkeltijd en schaalbaarheid van de applicaties.

10.1.1. Ontwikkeltijd

Algemene ontwikkeltijd

Over het algemeen is de ontwikkeltijd bij cross-platform langer dan bij native. Dit komt door de libraries die geïnstalleerd moeten worden, wat extra tijd in beslag neemt en meer ruimte geeft voor fouten. Deze fouten moeten dan ook nog eens opgelost worden, wat ook weer extra tijd in beslag neemt.

De ontwikkeltijd is ook langer omdat de React Native libraries de bestaande implementatie van de native platformen gebruiken en op verder bouwen. Hierdoor is er soms meer code nodig bij cross-platform om dezelfde functionaliteit te krijgen.

Compiletijd

De compiletijd bij cross-platform is altijd langer dan bij native. Dit komt omdat er bij cross-platform een extra stap is bij het compileren. De code wordt eerst gecompileerd naar Javascript, dan pas naar native. Bij native is er maar één stap, de code wordt gecompileerd naar native.

Ondanks het feit dat de compiletijd langer is bij cross-platform, is het niet altijd nodig om de applicatie te compileren. React Native heeft een hot reload functie, waardoor de applicatie niet gecompileerd moet worden bij het testen. Eenmaal de applicatie gecompileerd is, kunnen er veranderingen worden aangebracht aan de code en deze worden dan automatisch doorgevoerd in de applicatie. Dit is een groot voordeel bij het ontwikkelen van de applicatie, omdat er geen tijd verloren gaat aan het compileren van de applicatie.

10.1.2. Performantie

Tijdsduur

Op het vlak van tijdsduur is er enkel een groot verschil bij het aanmaken van notificaties. Bij cross-platform duurt het aanmaken van een notificatie 14 keer langer dan bij native. Ook al is dit een groot verschil, het aanmaken van een notificatie duurt bij cross-platform nog steeds maar 166ms. Dit is nog steeds snel genoeg om de gebruiker niet te storen aangezien dit gebeurt op de achtergrond. Bij de andere functionaliteiten is er wel een verschil in tijdsduur, maar dit verschil is niet zo groot dat het storend is voor de gebruiker of dat het een reden is om cross-platform niet te gebruiken.

CPU gebruik

Bij het CPU gebruik zijn de resultaten wisselvallig. Over het algemeen is het CPU gebruik bij cross-platform hoger dan bij native, maar dit is niet altijd het geval. Bij het gebruik van sensoren bijvoorbeeld is het CPU gebruik bij cross-platform lager dan bij native. Maar als we dan kijken naar het CPU gebruik bij het afspelen van audio en video, dan is het CPU gebruik bij cross-platform hoger dan bij native. Bij het kiezen van een ontwikkelmethode is het dus belangrijk om te kijken naar de functionaliteiten die gebruikt zullen worden in de applicatie. Deze zullen bepalend zijn voor de uiteindelijke keuze van de ontwikkelmethode.

Geheugengebruik

Op het vlak van geheugengebruik is nooit een merkbaar verschil geweest bij het uitvoeren van functionaliteiten of wanneer de applicatie inactief is. Daarnaast was het gemiddelde geheugengebruik bij alle functionaliteiten bij cross-platform veel hoger dan bij native. Het geheugengebruik bij cross-platform is gemiddeld 2,5 keer hoger dan bij native. Dit is een groot verschil, als gevolg van de libraries die gebruikt worden bij cross-platform. Alle extra libraries die gebruikt worden bij cross-platform nemen extra geheugen in beslag. Ook de React Native library die altijd standaard is geïnstalleerd, neemt extra geheugen in beslag. Dit is een nadeel bij cross-platform, maar het is niet zo groot dat het een reden is om cross-platform niet te gebruiken aangezien de meeste smartphones tegenwoordig genoeg geheugen hebben om cross-platform applicatie te runnen zonder dat de gebruiker hier iets van merkt.

10.1.3. Schaalbaarheid

Complexiteit

Over het algemeen is de complexiteit gelijkaardig bij native en cross-platform. Bij cross-platform is er wel meer complexiteit bij het opzetten van de applicatie, maar dit is een eenmalige complexiteit. Er bestaan wel complexe libraries bij cross-platform, maar deze zijn niet verplicht om te gebruiken.

Herbruikbaarheid

Zowel native als cross-platform hebben een hoge herbruikbaarheid. Bij beide methodes is het mogelijk om de code te centraliseren en te hergebruiken op andere plaatsen binnen de applicatie of vanop de gecentraliseerde plaats de code op te schalen. Het is ook mogelijk om de code te hergebruiken in andere applicaties.

10.2. Deelonderzoeksvragen

Naast de hoofdonderzoeksvraag zijn er twee deelonderzoeksvragen waar er een antwoord op gezocht werd bij het uitvoeren van dit onderzoek.

- Zijn er functionaliteiten die cross-platform niet ondersteunen?
- Zijn er functionaliteiten bij cross-platform waarbij de performantie de functionaliteit onbruikbaar maakt?

Bij de vraag of er functionaliteiten zijn die cross-platform niet ondersteunt, is er gebleken dat dit niet het geval is. Alle functionaliteiten die in dit onderzoek zijn geïmplementeerd waren mogelijk bij cross-platform. Bij sommige functionaliteiten had cross-platform meerdere libraries die de functionaliteit ondersteunden, terwijl er bij native altijd maar één implementatie mogelijk was.

Bij de vraag of er functionaliteiten onbruikbaar zijn bij cross-platform ontwikkeling vanwege de performantie, is de conclusie getrokken dat dit niet het geval is. Alle functionaliteiten die in dit onderzoek zijn geïmplementeerd, waren bruikbaar bij cross-platform. Er was wel altijd een verschil in performantie tussen native en cross-platform, maar dit verschil was nooit zo groot dat de functionaliteit onbruikbaar was.

10.3. Wat nu?

Op vlak van performantie is er duidelijk een verschil tussen native en cross-platform. Dit roept de vraag op of dit bij alle functionaliteiten het geval zal zijn, en of er zelfs functionaliteiten zijn die beter presteren bij cross-platform. Het kan ook zijn dat cross-platform frameworks altijd met een achterstand zullen zitten op native frameworks. Daarom zijn dit soort onderzoeken belangrijk, om een inzicht te krijgen in de vooruitgang van cross-platform frameworks.

Het onderzoek kan ook uitgebreid worden naar andere cross-platform frameworks zoals Flutter of .NET MAUI. Het kan zijn dat deze frameworks beter presteren op vlak van performantie zonder te comprimeren op de ontwikkeltijd en schaalbaarheid. Het onderzoek kan ook uitgebreid worden naar andere native frameworks zoals Java of zelfs andere native platformen zoals IOS. Eigenlijk zin de mogelijkheden om dit onderzoek uit te breiden of voort te zetten zijn eindeloos.

Het debat over native en cross-platform zal altijd blijven bestaan. Er komen steeds nieuwe frameworks en technologieën bij die het debat weer doen oplaaien. Een antwoord op de vraag welke ontwikkelingsmethode/framework uiteindelijk het beste is, zal er nooit komen. Dit soort onderzoeken echter kan een ontwikkelaar wel helpen om een doordachte keuze te maken bij het kiezen van een technologie voor zijn eigen specifieke project.

A

Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

Samenvatting

In deze paper gaan we de performantie, schaalbaarheid en functionaliteit van mobiele applicaties onderzoeken om te bepalen of het interessanter is om native te ontwikkelen in vergelijking met cross-platform. Langs de native kant gaan we Kotlin in Android Studio en Swift in Xcode gebruiken. Voor cross-platform zullen we Javascript in React Native gebruiken. Zo kunnen we ook een eventueel verschil tussen Android en IOS ontdekken. Op deze manier zullen we kleine tot middelgrote organisaties of bedrijven helpen met het kiezen van een gebruikte ontwikkelingsmethode. We zullen in het onderzoek met kleine en grote applicaties rekening houden om te bepalen vanaf wanneer het interessanter wordt.

A.1. Introductie

In dit onderzoek zullen we nagaan vanaf wanneer het interessanter is om native te gaan ontwikkelen in plaats van cross-platform. Hiervoor gaan we kijken naar de performantie, functionaliteit, schaalbaarheid en kost tussen native en cross-platform applicaties.

Bij native applicatieontwikkeling zullen we 2 ontwikkelingsmethodes kiezen en voor cross-platform 1 ontwikkelingsmethode gebaseerd op hun populariteit. Dit gaan we doen om het onderzoek te baseren op de meest gebruikte en relevante ontwikkelingsmethode. Voor native gebruiken we 2 ontwikkelingsmethodes om zo ook het verschil tussen Android en IOS in kaart te brengen. Voor cross-platform

zullen we 1 ontwikkelingsmethode gebruiken aangezien deze over allebei de platformen zal werken. We zullen meerdere applicaties gebruiken om zo ook rekening te houden met de grote en complexiteit van een applicatie.

Hierdoor kunnen we kleine tot middelgrote organisaties of bedrijven helpen met het kiezen van een gebruikte ontwikkelingsmethode.

A.2. State-of-the-art

A.2.1. Keuze ontwikkelingssoftware

Om te bepalen met welke software dat we ons onderzoek gaan uitvoeren zullen we een van de meest populaire en relevante ontwikkelingsmethode kiezen.

Android

Android applicaties worden allemaal ontwikkeld in Java of Kotlin (Faisal, 2021). Daarnaast moeten we ook kijken naar welke IDE we kunnen gebruiken om in Java of Kotlin te ontwikkelen. Hier hebben we keuze uit tientallen verschillende IDE's namelijk: Eclipse, Visual Studio, Android Studio, IntelliJ, NetBeans, Komodo, Cordova en PhoneGap (Harvey, 2021).

We zullen kiezen voor Android Studio aangezien dat dit de officiële IDE is voor Android applicaties. (JavaTpoint, 2019)

Android Studio

Android Studio is een IDE dat op 1 mei 2013 ontwikkeld is. Het wordt door Google gebruikt als hun officiële IDE en het heeft heel wat functionaliteiten dat het ontwikkelen van Android applicaties ondersteund. In Android Studio maken ze gebruik van Kotlin voor hun applicaties te maken. (JavaTpoint, 2019)

IOS

Ondanks dat Android meer gebruikt wordt dan IOS zijn er heel wat programmeertalen die gebruikt kunnen worden voor het ontwikkelen van IOS applicaties. We hebben keuzen uit: Objective-C, Swift, C++, Python (Sahu, 2020).

Objective-C was de meest gebruikte tot 2014. Het is een object -georiënteerde programmeertaal met heel wat functies. Swift is in 2014 op de markt gebracht door Apple als het beste van het best en biedt heel van voordelen bij het ontwikkelen van applicaties. C++ wordt gemixt met Objective-C om IOS applicaties te maken en Python kan gebruikt worden om IOS applicaties te schrijven, maar het moet daarna omgezet worden naar Objective-C. (Sahu, 2020)

Naast de programmeertaal zullen we opnieuw kijken naar welke IDE we zullen ge-

bruiken. Hiervoor kiezen we voor Xcode aangezien dat deze door Apple gemaakt is om IOS applicaties te maken. (TechCommuters, 2020)

Xcode

Zoals net gezegd, Xcode is de officiële tool voor IOS applicaties te maken, ontwikkeld door Apple. Het maakt gebruik van Swift en je kan er applicaties mee ontwikkelen voor Iphone, Ipad, AppleTV of zelf MAC applicaties. Het wordt ook gezien als de meest krachtige en betrouwbare tool nu op de markt(TechCommuters, 2020). Daarom zullen we deze gebruiken in ons onderzoek.

Cross-platform

Voor onze cross-platform ontwikkelingsmethode gaan we terug kijken naar een populair en relevante framework. We hebben keuze uit: Xamarin, Codename One, Flutter, React native en NativeScript (Zubair, 2022). Elke van deze frameworks is een open-source cross-platform framework waarmee je applicaties kan maken. Xamarin gebruikt C# en het .Net framework om apps te maken. Het wordt gebruikt door UPS en Microsoft. Codename One gebruikt Java of Kotlin en wordt gebruikt door Muving en HBZ. Flutter, dat gemaakt is door Google gebruikt Dart en wordt gebruikt door Google, eBay, Alibaba en BMW. React native is gemaakt door Facebook en het gebruikt Javascript en React.js om applicaties te maken. Het wordt onder andere gebruikt door Facebook, Bloomberg, Walmart, Uber en Shopify. NativeScript gebruikt Javascript, TypeScript, Angular of Vue.js en wordt gebruikt door Strudel en BitPoints. (Thaker, 2022)

Voor onze cross-platform ontwikkelingsmethode gaan we kiezen voor React native. We doen dit omdat React native een van de meest populaire en ondersteunde frameworks is. Daarnaast maakt React native ook gebruik van Javascript, een van de meest populaire programmeertalen.

Overzicht keuze

We zullen voor native ontwikkelen bij Android gebruik maken van Kotlin in Android Studio en voor IOS van Swift in Xcode. Voor cross-platform ontwikkelen gaan we gebruik maken van Javascript in React native.

A.3. Methodologie

We zullen in het onderzoek de geselecteerde ontwikkelingsmethodes af toetsen op basis van drie categorieën die belangrijk zijn bij het gebruik ervan. Zo gaan we kijken of dat native ontwikkelingsmethodes beter zijn dan cross-platform en indien ze beter zijn zullen we kijken vanaf wanneer het interessanter is om native te ontwikkelen in plaats van cross-platform op basis van de grote en de complexiteit van een applicatie. De drie categorieën bestaan uit:

1. Hoe snel start de applicatie op en hoe responsive is hij? (Performantie)

2. Hoe gemakkelijk kunnen we ons applicatie uitbreiden zonder veel code te moeten wijzigen en/of toevoegen? (Schaalbaarheid)

3. Laat cross-platform even veel functionaliteiten toe als native? (Functionaliteit)

Na het testen van elke ontwikkelingsmethode op elk van deze categorieën zullen we een besluit kunnen nemen of dat het interessanter is om native te ontwikkelen in plaats van cross-platform. Dit zullen we doen door te kijken of dat een eventueel verschil van performantie, schaalbaarheid of functionaliteit het waard is om te investeren in native ontwikkeling in plaats van cross-platform afhankelijk van de grootte en complexiteit van de applicatie.

A.3.1. Performantie

De performantie van een framework is van groot belang binnen applicaties, daarom zullen we deze categorie als eerst gebruiken om de ontwikkelingsmethodes op te testen.We zullen elk ontwikkelingsmethode testen op hun opstartsnelheid en eventuele connectie met een database (ophalen van data). We zullen dan alle applicaties laten verbinden met dezelfde database, waardoor de performantie zal getest worden.

A.3.2. Schaalbaarheid

De schaalbaarheid van een ontwikkelingsmethode is niet voor elke programmeur of applicatie van zeer groot belang, maar in dit onderzoek is het wel een zeer belangrijk punt. Als je native zou werken moet je namelijk twee keer de applicatie gaan uitbreiden en bij cross-platform maar één keer. We zullen dus kijken hoe makkelijk functionaliteiten bij een bestaande applicatie kunnen toevoegen of integreren, zonder dat er al te veel code moet aangepast worden.

A.3.3. Functionaliteit

Tot slot zullen we onderzoeken of dat cross-platform alle functionaliteiten of toch de minimaal nodige functionaliteiten ondersteund dat een applicatie nodig heeft. We zullen dus kijken vanaf welke complexiteit dat de cross-platform ontwikkelingsmethode niet meer alle functionaliteiten heeft of ondersteund dat native wel ondersteund.

A.4. Verwacht resultaat, conclusie

Er wordt verwacht dat het beter is om simpele en kleine applicaties die niet veel functionaliteiten nodig hebben te maken met cross-platform ontwikkelingsmethodes. Daarnaast verwachten we dat naarmate een applicatie groter en complexer wordt het interessanter is om te investeren in native ontwikkelingsmethodes om zo volledig gebruik te kunnen maken van alle functionaliteiten. Ook omdat deze

efficiënter zullen zijn aangezien dat ze volledig gebruik kunnen maken van het apparaat waarop de applicatie draait.

Bibliografie

AWS. (2023). *What is Mobile Application Development?* (AWS, Red.). https://aws.amazon.com/mobile/mobile-application-development/

Beeproger. (2023). Native apps vs Web apps vs Hybride apps - wat is de beste keuze? *Beeproger*. https://beeproger.com/blog/native-apps-vs-web-apps-vs-hybride-apps-is-beste-keuze/

Blaž Denko, I. F. J., Špela Pečnik. (2021, januari 4). *A Comprehensive Comparison of Hybrid Mobile Application Development Frameworks* [masterscriptie, University of Maribor, Slovenia]. https://www.researchgate.net/publication/348132143_A_Comprehensive_Comparison_of_Hybrid_Mobile_Application_Development_Frameworks

Budiu, R. (2016). Mobile: Native Apps, Web Apps, and Hybrid Apps. *Nielsen Norman Group*. https://www.nngroup.com/articles/mobile-native-apps/

Coursera. (2022). Programming in Swift: Benefits of This Popular Coding Language. *Coursera*. https://www.coursera.org/articles/programming-in-swift

Designveloper. (2022). Top 5 Best Android App Development Languages for 2023. *Designveloper*. https://www.designveloper.com/blog/android-app-development-languages/

Faisal, F. (2021). What Makes Native App Development a Great Approach for Building Mobile Apps. https://mindster.com/mindster-blogs/native-app-development/#Native_App_Development_for_Android

harkiran. (2022). Top Programming Languages for Android App Development. *GeeksForGeek*. https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/

Hartl, M. (2019). Real-Time Apps With WebSockets and Action Cable. *LearnEnough*. https://news.learnenough.com/real-time-apps-with-websockets-and-action-cable

Harvey, C. (2021). 11 Best Android IDEs for Developers of 2022. https://www.developer.com/mobile/top-android-ides-for-developers/

Heller, M. (2022). What is Visual Studio Code? Microsofts extensible code editor. *InfoWorld*. https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html

Hu, M. (2021, juli 1). *A Comparative Study of Cross-platform Mobile Application Development* [tech. rap.]. Whitireia Polytechnic, New Zealand. https://www.

researchgate.net/publication/357898491_A_Comparative_Study_of_Cross-platform_Mobile_Application_Development

IBM. (2023, april 14). *What is mobile application development?* Verkregen april 14, 2023, van https://www.ibm.com/topics/mobile-application-development

jahnavisarora. (2020). Top 10 iOS App Development Tools That You Can Consider. *GeeksForGeek*. https://www.geeksforgeeks.org/top-10-ios-app-development-tools-that-you-can-consider/

JavaTpoint. (2019, augustus 1). *Android Studio*. https://www.javatpoint.com/android-studio

Johns, R. (2023). 5 Best iOS Programming Languages for App Development [2023]. *hackr.io*. https://hackr.io/blog/ios-programming-languages

Kesavan, M. (2021). Top 10 Programming Languages for Android App Development in 2023. *Ways2Smile Solutions*. https://www.way2smile.com/blog/programming-languages-for-android/

Khan, F. Q. (2021, maart 1). *A Comparative Analysis of Mobile Application Development Approaches* [onderzoeksrap.]. University of Swat. https://www.researchgate.net/publication/354199009_A_Comparative_Analysis_of_Mobile_Application_Development_Approaches

Koffer, P. (2023). What is a Native Mobile App Development? *mDev Talks*. https://mdevelopers.com/blog/what-is-a-native-mobile-app-development-

Kotlin. (2023, april 14). *Native and cross-platform app development: how to choose?* https://kotlinlang.org/docs/native-and-cross-platform.html

Lua. (2021, oktober 24). *What is Lua?* https://www.lua.org/about.html

Medewar, S. (2022). 7 Best IDEs for Mobile App Development. *Geekflare*. https://geekflare.com/best-ide-for-mobile-app-development/

Meirelles, P. R. M. (2019, juni 1). *A Students Perspective of Native and Cross-Platform Approaches for Mobile Application Development* [masterscriptie, University of São Paulo]. https://www.researchgate.net/publication/334086675_A_Students'_Perspective_of_Native_and_Cross-Platform_Approaches_for_Mobile_Application_Development

Merenych, S. (2021). What Is Cross-Platform Mobile App Development? *Clockwise Software*. https://clockwise.software/blog/cross-platform-app-development/

Monus, A. (2023). The 2023 guide to native app development. *Raygun*. https://raygun.com/blog/native-app-development/

Nehra, M. (2023). What is Web Application (Web Apps) And Its Benefits. *Decipherzone*. https://www.decipherzone.com/blog-detail/web-apps

Okeke, F. (2022a). What are the top cross-platform app development frameworks in 2022? *TechRepublic*. https://www.techrepublic.com/article/top-development-frameworks/

Okeke, F. (2022b). The 12 best IDEs for programming. *TechRepublic*. https://www.techrepublic.com/article/best-ide-software/

Palko, I. (2021). What is Mobile App Development? Development Process Explained. *Shoutem*. https://shoutem.com/blog/mobile-app-development/

Patel, A. (2023). Top 7 Programming Languages To Develop Native Android Apps. *ReadWrite*. https://readwrite.com/top-programming-languages-to-develop-native-android-apps/

Pruciak, M. (2022). Picking The Best Language For iOS App Development In 2022. *IdeaMotive*. https://www.ideamotive.co/blog/picking-the-best-language-for-ios-app-development

Sahu, A. (2020). Top 6 Platforms for iOS App Development. https://www.westagilelabs.com/blog/top-platforms-for-ios-app-development/

Sakovich, N. (2023a). Native Mobile App Development: Five Key Benefits. *Sam Solutions*. https://www.sam-solutions.com/blog/native-mobile-app-development/

Sakovich, N. (2023b). Cross-Platform Mobile Development: Five Best Frameworks. *Sam Solutions*. https://www.sam-solutions.com/blog/cross-platform-mobile-development/

sgshradha. (2019). Hybrid Application. https://www.geeksforgeeks.org/hybrid-application/

Studio, A. (2023, april 28). *Meet Android Studio* (A. Studio, Red.). https://developer.android.com/studio/intro/

sugandha. (2022). Difference Between Web application and Website. https://www.geeksforgeeks.org/difference-between-web-application-and-website/

TechCommuters. (2020). Top 15 iOS Development Tools in 2022. https://www.techcommuters.com/top-ios-development-tools/

Terekhov, V. (2022). 5 Best Cross-Platform App Development Frameworks in 2022. *Attract Group*. https://attractgroup.com/blog/best-cross-platform-app-development-frameworks/#

Thaker, H. (2022). Cross-Platform App Development: A Complete Guide. https://www.biztechcs.com/blog/cross-platform-app-development-complete-guide/

the dir one. (2020). How to Push Notification in Android? *GeeksForGeek*. https://www.geeksforgeeks.org/how-to-push-notification-in-android/

Thorndyke, K. (2021). What is the Best Language for App Development? *Code Cademy*. https://www.codecademy.com/resources/blog/whats-the-best-language-for-app-development/

van Laarhoven, V. (2021). Native app, web app of hybride app? Waar kies je voor en waarom? *Scrumble*. https://scrumble.nl/blog/innovatie-strategie/native-web-hybrid/

Varsha. (2023). What is a Web Application? How Does It Work? *Techjokey*. https://www.techjockey.com/blog/what-is-web-application

yuvraj. (2022). Top Programming Languages For iOS App Development. *GeeksForGeek*. https://www.geeksforgeeks.org/top-programming-languages-for-ios-app-development/

Zubair. (2022). Top 10 Best Cross Platform App Development Frameworks in 2022. https://www.codenameone.com/blog/top-10-best-cross-platform-app-development-frameworks-in-2022.html