



Faculteit Bedrijf en Organisatie

Kotlin Multiplatform Mobile als alternatief voor native applicaties: een vergelijkende studie en
proof-of-concept

Ziggy Moens

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

Promotor:
Ludwig Stroobant
Co-promotor:
Kenneth Saey

Instelling: Endare

Academiejaar: 2020-2021

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Kotlin Multiplatform Mobile als alternatief voor native applicaties: een vergelijkende studie en
proof-of-concept

Ziggy Moens

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

Promotor:
Ludwig Stroobant
Co-promotor:
Kenneth Saey

Instelling: Endare

Academiejaar: 2020-2021

Tweede examenperiode

Woord vooraf

Deze bachelorproef gaat dieper in op Kotlin Multiplatform Mobile en de mogelijkheden dit als alternatief te gebruiken voor de ontwikkeling van native applicaties. Deze studie werd uitgevoerd in het kader van de opleiding toegepaste informatica aan de Hogeschool Gent. Als eerste zou ik graag mijn promotor Ludwig Stroobant, lector aan de Hogeschool Gent, willen bedanken. De heer Stroobant stond altijd klaar om mij te helpen en nam een zeer actieve rol op als promotor. Ook heb ik deze bachelorproef gemaakt in samenwerking met het bedrijf Endare, vandaar wil ik graag Kenneth Saey, die de rol van co-promotor op zich heeft genomen, bedanken. De heer Saey stond steeds klaar als ik vragen of problemen had. Daarnaast wil ik nog Nicolas Loots, een werknemer van Endare die mijn stagebuddy was, bedanken voor het nalezen en de technische steun. Ook wil ik graag mijn vriendin bedanken voor het regelmatige nalezen en verbeteren van deze bachelorproef. Daarnaast wil ik ook mijn ouders en broer bedanken, voor de steun en motivatie tijdens het schrijfproces.

Samenvatting

Binnen deze bachelorproef werd nagegaan of Kotlin Multiplatform Mobile een alternatief kan bieden voor native applicatieontwikkeling. Cross-platform ontwikkeling wordt de laatste tijd steeds interessanter gezien de doorgaans lagere kosten en snellere ontwikkeltijden. Binnen deze studie werd aan de hand van twee native applicaties voor Android en iOS geschreven en daarnaast ook een Kotlin Multiplatform Mobile applicatie die zowel Android als iOS ondersteunt. Met deze applicaties werden de verschillen en gelijkenissen opgemeten tussen de native en cross-platform applicaties. Deze applicaties zijn vooral gefocust op het vergelijken van de core van de applicaties, hierdoor bevatten deze echter niet veel inhoud.

Uit deze resultaten is gebleken dat Kotlin Multiplatform Mobile veel potentieel toont als alternatief voor native applicaties. Bij het evalueren van deze resultaten dient wel het feit dat Kotlin Multiplatform Mobile nog in het alpha stadium is, in achterhoofd gehouden te worden. De cross-platform applicaties bezaten dezelfde look en feel van de native applicaties en hadden gelijke of zelfs betere resultaten op vlak van snelheid, performantie en ontwikkelingstijd.

Inhoudsopgave

1	Inleiding	17
1.1	Probleemstelling	17
1.2	Afbakening van het onderwerp	18
1.3	Onderzoeks vraag	18
1.4	Onderzoeksdoelstelling	19
1.5	Opzet van deze bachelorproef	21
2	Stand van zaken	23
2.1	Kotlin	23
2.2	Platformen en hun ontwikkelingsvormen	24
2.2.1	Platform	25
2.2.2	Ontwikkelingsvormen	26
2.3	Kotlin Multiplatform	31

2.4	Kotlin Multiplatform versus alternatieven	34
2.5	Testcriteria	35
3	Methodologie	37
3.1	Installatie	37
3.1.1	Hardware	37
3.1.2	Android Studio	38
3.1.3	Xcode	42
3.1.4	JDK	43
3.2	Basis Kotlin Multiplatform Mobile applicatie	44
3.3	Basis native applicaties	49
3.3.1	Android	50
3.3.2	iOS	52
3.4	Testen van de basis applicaties	55
3.4.1	Aantal lijnen code	56
3.4.2	Compileersnelheid	58
3.4.3	Voetafdruk	60
3.4.4	Ontwikkeltijd	63
3.4.5	Kostprijs	65
4	Conclusie	67
4.1	Resultaten van de testen	67
4.2	Verder onderzoek	68
4.3	Conclusie	69

A	Onderzoeksvoorstel	73
A.1	Introductie	73
A.2	State-of-the-art	74
A.2.1	Kotlin	74
A.2.2	Cross-platform en native	74
A.2.3	Kotlin Multiplatform	75
A.2.4	Kotlin Multiplatform versus alternatieven	76
A.2.5	Testcriteria	77
A.3	Methodologie	78
A.4	Verwachte resultaten	79
A.5	Verwachte conclusies	80
B	Resultaten van compileersnelheid	81
B.1	Tabel	81
B.2	grafisch	84
C	Resultaten van aantal lijnen code	85
C.1	Aantal lijnen code - Android	86
C.2	Aantal lijnen code - iOS	87
C.3	Aantal lijnen code - KMM	88
	Bibliografie	89

Lijst van figuren

2.1 Grafische voorstelling Kotlin Multiplatform en de verschillende platformen (Kotlin & JetBrains, 2021b)	32
2.2 Grafische voorstelling Kotlin Multiplatform Mobile (Kotlin & JetBrains, g.d.)	33
2.3 Grafische voorstelling van de code binnen KMM met iOS en Android als voorbeeld (Kotlin & JetBrains, 2021b)	34
3.1 De ADV Manager in Android Studio met de Pixel 4a emulator geïnstalleerd	39
3.2 De opties voor een nieuw project binnen Android Studio, hierbij is een KMM applicatie ook beschikbaar	40
3.3 De standaard instellingen voor een KMM project	41
3.4 De instellingen van een KMM project voor de gedeelde logica en de native iOS en Android delen	41
3.5 De plug-in geïnstalleerd binnen Android Studio	42
3.6 De Xcode Command Line Tools correct geconfigureerd binnen Xcode 43	
3.7 De terminal bevestigd dat Java SE 16 correct is geïnstalleerd op de betreffende computer	44

3.8 Proces voor het opzetten van een KMM project	46
3.8.1 Startscherm Android Studio	46
3.8.2 Nieuw project met de KMM application optie	46
3.8.3 Standaard instellingen Android Project	46
3.8.4 Specifieke KMM project instellingen	46
3.9 Android en iOS emulators binnen Android Studio	47
3.9.1 Instellingen Android emulator	47
3.9.2 Instellingen iOS emulator	47
3.10 KMM basic applicatie op Android en iOS	47
3.10.1Android applicatie	47
3.10.2iOS applicatie	47
3.11 Projectstructuur van het KMM project	48
3.12 Instelling voor het native Android project	50
3.13 Interface van de basic native Android applicatie	52
3.14 De optie voor een nieuwe iOS App binnen Xcode	53
3.15 Instelling voor het native iOS project	53
3.16 Interface van de basic native iOS applicatie	55
3.17 Instellingen voor de Statics plug-in	57
3.18 Voetafdruk van de native Android applicatie	62
3.18.1Voetafdruk van de native Android met cache en lokale opslag	62
3.18.2Voetafdruk van de native Android zonder cache en met lokale opslag ..	62
3.18.3Voetafdruk van de native Android zonder cache en lokale opslag	62
3.19 Voetafdruk van de native iOS applicatie	63
3.20 Voetafdruk van de KMM applicaties	64
3.20.1Voetafdruk van KMM Android met cache en lokale opslag	64
3.20.2Voetafdruk van KMM Android zonder cache en met lokale opslag ..	64
3.20.3Voetafdruk van KMM Android zonder cache en lokale opslag	64
3.20.4Voetafdruk van KMM iOS kmm	64

A.1 Grafische voorstelling Kotlin Multiplatform Mobile (Kotlin & JetBrains, g.d.)	76
---	----

Lijst van tabellen

2.1 Vergelijking en samenvatting van de verschillende cross-platform categorieën	29
3.1 Statistiek resultaten voor KMM en de native projecten	57
3.2 Resultaten van de statistiek voor het KMM project	58
3.3 Resultaten van de compilatie twee tot honderd	59
3.4 Resultaten van de eerste compilatie	60
3.5 Resultaten van de compilatie twee tot honderd	60
3.6 Overzicht van de voetafdruk van de applicaties	61
3.7 Overzicht van de ontwikkeltijd van de applicaties	63
3.8 Overzicht van de ontwikkeltijd van de applicaties	65
B.1 Compileersnelheid resultaten van native en KMM projecten	81

1. Inleiding

Cross-platform ontwikkeling is een gegeven dat in de laatste jaren aan populariteit wint. Dit komt vooral omdat er voordelen aan verbonden zijn die, voor bedrijven gespecialiseerd in software voor verschillende platformen, zeer interessant kunnen zijn. Hierbij treden vooral de snellere ontwikkelingstijd en de lagere ontwikkelingskosten naar de voorgrond. Deze twee zaken spelen voor die bedrijven een grote rol binnen hun dagelijkse werking. Ondanks deze voordelen is nog niet elk bedrijf te vinden voor cross-platform ontwikkeling. Het beeld dat deze cross-platform applicaties ‘beter’ zouden zijn dan de native oplossingen op de markt is nog niet overal aanvaard. Kotlin Multiplatform Mobile¹ (KMM), de nieuwe cross-platform software development kit van JetBrains², zou hier verandering in kunnen brengen. KMM gaat voor een andere aanpak voor hun cross-platform applicaties en zal vooral focussen op een gedeelde business logica en een platformafhankelijke en dus native user interface.

1.1 Probleemstelling

Veel bedrijven die zich specialiseren in applicatieontwikkeling voor mobile toestellen zoals Android³ en iOS⁴ toestellen hebben verschillende mogelijkheden om dit aan te pakken. Zo is er bijvoorbeeld de keuze tussen een native applicatie of een cross-platform-applicatie. Anderzijds zijn er nog mogelijkheden zoals een progressive web application

¹kotlinlang.org/lp/mobile

²jetbrains.com

³android.com

⁴apple.com/benl/ios/ios-14

(PWA) geschreven in React⁵, Angular⁶, Vue.js⁷... De PWA's vallen echter buiten de scope van deze bachelorproef. Er zal dus vooral gefocust worden op bedrijven die momenteel native applicaties ontwikkelen en eventueel willen overschakelen naar cross-platform.

1.2 Afbakening van het onderwerp

JetBrains heeft Kotlin Multiplatform⁸ (KM) uitgebracht met Kotlin Multiplatform Mobile (KMM) als specialisatie voor de mobile toestellen. KMM is vooral gericht op iOS en Android in tegenstelling tot KM dat zich ook zal toespitsen op macOS⁹, Windows¹⁰... Voor deze bachelorproef zal er dus vooral gefocust worden op KMM aangezien de onderzoeksraag zich toespitst op de mobiele toestellen. KMM heeft dezelfde functies als KM aangezien KMM een aftakking is van KM. Voor besturingssystemen van mobile toestellen zal er vooral gekeken worden naar Android en iOS aangezien deze de meest gebruikte besturingssystemen zijn op de markt.

1.3 Onderzoeksraag

De opzet van deze bachelorproef is na te gaan of native applicatieontwikkeling voor Android en iOS apparaten nog steeds de beste keuze is of KMM een sneller en efficiënter alternatief kan vormen binnen het huidige beeld van applicatieontwikkeling. Om dit te onderzoeken zijn er verschillende testcriteria opgesteld. Deze testcriteria zijn:

- het aantal lijnen code
- de compileersnelheid
- de voetafdruk
- de ontwikkeltijd
- de kostprijs

Aan de hand hiervan kan besloten worden of de KMM-applicaties een beter alternatief kunnen vormen voor native applicaties. Daarnaast zal ook de vraag gesteld worden of het huidige beeld van de SWOT-analyse voor cross-platform applicaties ook toepasbaar is voor KMM-applicaties. Indien niet het geval zou zijn, zal gekeken worden hoe deze dient aangepast te worden.

Hieronder wordt een overzicht van de onderzoeksraag gegeven

- Is native applicatieontwikkeling nog steeds de beste optie voor Android?
- Is native applicatieontwikkeling nog steeds de beste optie voor iOS?

⁵reactjs.org

⁶angular.io

⁷vuejs.org

⁸kotlinlang.org/docs/mpp-intro.html

⁹apple.com/benl/macos/big-sur

¹⁰microsoft.com/nl-be/windows

- Is KMM een sneller alternatief voor native applicaties?
- Is KMM een efficiënter alternatief voor native applicaties?
- Is het huidige beeld op de SWOT-analyse van cross-platform applicaties toepasbaar op KMM?

Deze bachelorproef tracht bovenstaande vragen te beantwoorden. Het proces van testen wordt beschreven in 1.4 ‘Onderzoeksdoelstelling’.

1.4 Onderzoeksdoelstelling

De onderzoeksdoelstelling van deze bachelorproef is na te gaan of Kotlin Multiplatform Mobile (KMM) een goed alternatief kan bieden voor native ontwikkeling. Hiervoor zal een vergelijkende studie tussen KMM en native applicaties uitgevoerd worden. Voor deze studie zullen meerdere applicaties geschreven worden voor zowel Android als iOS. Eerst worden er twee native applicaties geschreven. De native iOS applicatie zal geschreven worden in Swift 5.3¹¹ of recenter met iOS 14 in gedachten en voor de user interface zal er gebruik gemaakt worden van UIKit¹². Aan de Android kant van het native verhaal zal er een applicatie geschreven worden met behulp van Kotlin 1.4.20¹³ of een recentere versie. Voor de native Android user interface zal gebruik gemaakt worden van de standaard Android en AndroidX¹⁴ bibliotheek. Naast de native applicaties zal er nog een derde applicatie geschreven worden, namelijk een Kotlin Multiplatform Mobile (KMM) applicatie. Deze applicatie zal werken op zowel Android als iOS toestellen. Hierbij zal voor het iOS-deel gebruik gemaakt worden van Swift en SwiftUI¹⁵ voor de user interface en voor het Android deel Kotlin en Jetpack Compose¹⁶.

Om deze manier van werken correct te beoordelen is het belangrijk dat zowel de native applicaties als KMM dezelfde functionaliteiten aanbieden. Dit geldt ook voor de native applicaties voor Android en iOS onderling. De geschreven applicaties zouden dus als het ware niet van elkaar te onderscheiden mogen zijn. Hierbij dient echter rekening gehouden te worden met het feit dat de user interface wel enigszins kan verschillen.

Voor de applicaties zelf zal er gekeken worden naar de voorbeeldapplicaties die JetBrains aanbiedt en voorstelt. Dit zijn onder andere de PeopleInSpace applicatie (O'Reilly, 2021) en de SpaceX launches applicatie (Kotlin & JetBrains, 2020). Deze kunnen gebruikt worden om benchmarks op te testen en als inspiratiebron voor de native applicaties en andere KMM-applicaties.

Om uiteindelijk de vergelijkende studie te maken tussen de native applicaties en de KMM-applicatie zal er gekeken worden naar verschillende testcriteria. Hieronder volgt

¹¹apple.com/benl/swift

¹²developer.apple.com/documentation/uikit

¹³kotlinlang.org

¹⁴developer.android.com/jetpack/androidx

¹⁵developer.apple.com/xcode/swiftui

¹⁶developer.android.com/jetpack/compose

een overzicht van deze gekozen testcriteria voor deze studie.

- Aantal lijnen code
- Kostprijs
- Ontwikkeltijd
- Compileersnelheid
- Voetafdruk

Voor dit onderzoek zal volgende hardware en software gebruikt worden:

Hardware:

- MacBook Pro 16 inch 2,3GHz 8-core i9 met 16GB RAM geheugen uit 2019¹⁷
- iPhone 11 Pro Max uit 2019¹⁸
- Huawei P9 lite uit 2016¹⁹

Deze laatste twee toestellen zijn echter minder van belang en komen enkel ter sprake indien er testen gebeuren op fysieke toestellen. Het grotere deel van de testen worden uitgevoerd op emulators die ingebouwd zijn in de gekozen ontwikkelingssoftware.

Software:

- Xcode²⁰ versie 12.3 of recenter
- Android studio²¹ versie 4.1.1 of recenter

Voor de KMM-applicaties zal er echter nog de Kotlin Multiplatform Mobile plugin²² versie 2.4 of recenter geïnstalleerd moeten worden.

Deze software zal gebruikt worden op de MacBook Pro die hierboven reeds werd beschreven. Voor dit onderzoek geldt de beperking dat er enkel gebruik gemaakt kan worden van toestellen die macOS gebruiken als besturingssysteem, aangezien dit een vereiste is voor de ontwikkeling van iOS-applicaties.

KMM is een zeer recente technologie die volop in ontwikkeling is. Daardoor wordt verwacht dat de resultaten voor deze studie nog niet het volle potentieel zullen aantonen. Dit wil echter niet zeggen dat de technologie geen meerwaarde kan bieden voor bedrijven die zich momenteel bezighouden met native applicatieontwikkeling voor zowel Android als iOS. Enkele componenten van de SDK staan momenteel nog niet op punt, dit zal waarschijnlijk nog verbeteren naar de toekomst toe. Het onderzoek kan, gezien de prematuriteit van KMM, een tijdelijke referentie bieden omtrent KMM en de mogelijkheid dit als een sneller en efficiënter alternatief te gebruiken voor native applicaties. Dit is vooral interessant voor bedrijven die zich momenteel vooral toespitsen op native ontwikkeling en eventueel naar de toekomst toe de overstap willen maken naar KMM.

¹⁷ support.apple.com/kb/SP809?viewlocale=nl_NL&locale=en_US

¹⁸ support.apple.com/kb/SP806?viewlocale=nl_NL&locale=en_US

¹⁹ consumer.huawei.com/be/support/phones/p9-lite

²⁰ developer.apple.com/xcode

²¹ developer.android.com/studio

²² kotlinlang.org/docs/mobile/kmm-plugin-releases.html

1.5 Opzet van deze bachelorproef

Het vervolg van de bachelorproef wordt als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomain, 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 onderzoeks vragen.

In Hoofdstuk 4, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeks vragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

Binnen de stand van zaken zal er verder ingegaan worden op de theoretische kant van deze bacheloproef. Hierbij komen volgende thema's aanbod: Kotlin¹, de mogelijke ontwikkelingsvormen voor mobiele applicaties, Kotlin Multiplatform² en Kotlin Multiplatform Mobile³, de vergelijking met andere alternatieven en de gekozen testcriteria voor deze bacheloproef. De stand van zaken bouwt verder op de informatie uit de inleiding en wordt versterkt aan de hand van de literatuurstudie.

2.1 Kotlin

In het eerste deel van deze stand van zaken zal Kotlin besproken worden. Kotlin vormt de basis van Kotlin Multiplatform. Eerst zal kort de geschiedenis rond Kotlin bespreken om vervolgens over te gaan naar de technische kant van de programmeertaal.

Kotlin werd uitgebracht door JetBrains⁴ in juli 2011 en is binnen de informaticawereld een vrij recente programmeertaal.(Jemerov, 2011) JetBrains is een software ontwikkelingsbedrijf dat afkomstig uit Tsjechië en gekend is voor applicaties zoals IntelliJ IDEA⁵, PyCharm⁶, WebStorm⁷... Het bedrijf heeft ondertussen al meer dan 30 producten en meer dan tien miljoen gebruikers.(JetBrains, 2021) In juli 2011 was Kotlin echter al meer dan

¹kotlinlang.org

²kotlinlang.org/docs/mpp-intro.html

³kotlinlang.org/lp/mobile

⁴jetbrains.com

⁵jetbrains.com/idea

⁶jetbrains.com/pycharm

⁷jetbrains.com/webstorm

een jaar in ontwikkeling en evolueert tot de dag van vandaag nog steeds. JetBrains heeft dan ook beslist om van Kotlin een open-source project te maken. Dit zorgt ervoor dat iedereen de broncode kan bekijken en eventueel bewerken. Hierdoor hebben ze een zeer grote en actieve community gecreëerd rond Kotlin. In 2017 werd Kotlin door Google⁸ als de hoofdtaal gekozen voor de ontwikkeling van Android⁹ applicaties.(Shafirov, 2017) Dit alles heeft ertoe geleid dat Kotlin de snelst groeiende programmeertaal was op Github¹⁰ in 2018, zoals te zien was in GitHubs The State of the Octoverse 2018.(GitHub, 2018)

Na de korte geschiedenis rond Kotlin wordt er nu overgegaan naar de technische zijde van Kotlin. Kotlin is een programmeertaal met een aantal typerende factoren zoals het feit dat het een algemene programmeertaal is. Daarnaast is Kotlin een statische programmeertaal met type-interferentie. Ook is deze programmeertaal gecreëerd met cross-platform in het achterhoofd.(Oliveira e.a., 2020)

1. Algemene programmeertaal

- Hiermee wordt bedoeld dat Kotlin een programmeertaal is die ontwikkeld is voor allerlei verschillende soorten software en dat de programmeertaal gebruikt kan worden binnen verschillende situaties.(Skeen & Greenhalgh, 2018)

2. Statische programmeertaal.

- Dit gegeven wijst op het feit dat de programmeertaal de types van de objecten zal controleren tijdens het compileren van de code en niet tijdens het uitvoeren. Andere voorbeelden van statische talen met type-interferentie zijn Java¹¹ en C¹². Talen die geen statische typering gebruiken zijn dynamische typerende talen zoals Perl¹³, PHP¹⁴...

3. Type-interferentie

- Dit zordt ervoor dat Kotlin zelf onderscheid kan maken tussen de datatypes van bepaalde expressies.(Meijer & Drayton, 2004)

4. Cross-platform

- hiermee wordt bedoeld dat de software kan bestaan en werken op verschillende versies. Deze versies kunnen ook draaien op verschillende platformen en zijn dus niet noodzakelijk gebonden aan één specifiek platform zoals bijvoorbeeld Android.(Bishop & Horspool, 2006)

2.2 Platformen en hun ontwikkelingsvormen

Nadat de basis rond Kotlin is besproken, zullen de mogelijkheden binnen het ontwikkelen van mobiele applicaties besproken worden. Allereerst wat gezien wordt als een platform en daarna wat de ontwikkelingsmogelijkheden zijn voor dat specifieke platform of voor

⁸about.google/intl/nl

⁹android.com

¹⁰github.com

¹¹java.com/nl

¹²en.cppreference.com/w/c

¹³perl.org

¹⁴php.net

meerdere platformen tegelijk.

2.2.1 Platform

Eerst zal er bekeken worden wat de term platform inhoudt, daarvoor wordt gebruikt gemaakt van een artikel van Bishop en Horspool (2006). Belangrijk te vermelden is dat de term platform nog niet strikt gedefinieerd is, er kunnen echter wel enkele zaken gelinkt worden aan de term. Een platform zal meestal een bepaalde programmeertaal, bepaald besturingssysteem of bepaalde hardware beschrijven. Hierbij is belangrijk te vermelden dat het niet enkel over een van deze zaken kan gaan maar ook over een combinatie van meerdere factoren. Een voorbeeld hiervan is een platform dat beschreven wordt door een bepaald besturingssysteem in combinatie met specifieke hardware. Enkele voorbeelden van platformen in de praktijk:

- Programmeertaal als platform:

Hierbij zal een specifieke programmeertaal en eventueel bijhorende libraries dienen als het platform waarvoor er bepaalde software voor gemaakt wordt. Enkele voorbeelden hiervan zijn Java SE 16¹⁵, AdoptOpenJDK 11¹⁶... Hiervan zijn nog vele voorbeelden op te sommen, ook de voorgaande versies van deze software worden als andere platformen gezien.

- Besturingssysteem als platform:

Hierbij zal een bepaald besturingssysteem gebruikt worden als platform. Hierbij kunnen al 3 grote platformen worden opgehaald namelijk Windows¹⁷, macOS¹⁸ en Linux¹⁹. Deze zijn echter te globaal en zullen hierbij bijvoorbeeld Windows 10²⁰, macOS Big Sur²¹ en Ubuntu 20.04²² gekozen worden als een platform voor ontwikkeling. Afhankelijk van de gekozen versie van het besturingssysteem als platform zal de ontwikkelde software bepaalde apparaten al dan niet ondersteunen.

- Hardware als platform:

Hierbij zal een specifiek deel van de hardware binnenin bepaalde apparaten fungeren als platform. Dit kan echter zeer ruim bekeken worden. Een bepaalde processor of CPU (central processing unit), een grafische processor of GPU (graphics processing unit)... dit zijn allemaal voorbeelden van hardware die gekozen kunnen worden als platform. Enkele praktijkvoorbeelden zijn andere andere een bepaalde CPU van Intel²³ of AMD²⁴ zoals de Intel Core i9 11900K²⁵ of de AMD Ryzen Threadripper

¹⁵oracle.com/java/technologies/javase-downloads.html

¹⁶adoptopenjdk.net/index.html

¹⁷microsoft.com/nl-be/windows

¹⁸apple.com/benl/macos

¹⁹linux.org

²⁰microsoft.com/en-us/windows/windows-10-specifications

²¹apple.com/benl/macos/big-sur

²²ubuntu.com

²³intel.com

²⁴amd.com

²⁵intel.com/content/www/us/en/products/sku/212325/intel-core-i911900k-processor-16m-cache-up-to-

3970X²⁶.

- Combinatie van programmeertaal en/of besturingssysteem en/of hardware:
Een laatste mogelijkheid om een platform te beschrijven is door een combinatie van bovenstaande punten te gebruiken. Zo kan er specifiek op bepaalde platformen gericht worden, dit kan voor bepaalde situaties zeer handig zijn. Een voorbeeld hiervan is een computer die uitgerust met een macOS Big Sur besturingssysteem en een Intel processor. Indien bepaalde niche software enkel door zo een type toestel gebruikt zal worden kan het handig zijn om het platform zo gedetailleerd te beschrijven.

2.2.2 Ontwikkelingsvormen

Nu duidelijk is wat de term platform allemaal kan omvatten en wat hierbij de mogelijke scenario's zijn, kan er gekeken worden naar verschillende vormen van ontwikkeling en hun relatie met bepaalde platformen. De ontwikkelingsvormen die besproken zullen worden zijn native en cross-platform applicatieontwikkeling. Andere vormen van applicatieontwikkeling worden niet besproken omdat deze geen meerwaarde bieden binnen deze bachelorproef.

Native ontwikkeling

De eerste ontwikkelingsvorm die zal bekeken worden is native applicatieontwikkeling. Deze vorm van ontwikkeling spits zich toe op een specifiek platform. Deze vorm van ontwikkelen biedt aan de applicatie volledige toegang tot de platform specifieke zaken zoals hardware van het platform of specifieke features die het platform aanbiedt.(Rahul Raj & Seshu Babu Tolety, 2012) Deze applicaties zullen dus ook maar op één platform werken. De ontwikkeling van native applicaties gebeurt via bepaalde Software Development Kits, ontwikkelingstalen en software die wordt aangeboden door het platform waarvoor men wil ontwikkelen.(Lim, 2015) Een voorbeeld van dit soort ontwikkelingstalen is Swift²⁷. Deze taal zal gebruikt worden voor de platformen van Apple. Echter moet hierin wel nog onderscheid gemaakt worden tussen ontwikkeling voor iPhone²⁸, iPad²⁹... En andere native taal is Kotlin deze taal zal vooral gebruikt worden voor native Android. De software die door de platformen wordt aangeboden is meestal zeer specifiek voor die platformen en ontwikkelingstalen gemaakt, enkele voorbeelden hiervan zijn Xcode³⁰, Android Studio³¹... Het grote nadeel aan een native applicatie is dus het feit dat elke applicatie moet ontwikkeld worden per platform, wat dus meer tijd zal kosten.

²⁶5-30-ghz/specifications.html

²⁶amd.com/en/products/cpu/amd-ryzen-threadripper-3970x

²⁷developer.apple.com/swift

²⁸apple.com/benl/iphone

²⁹apple.com/benl/ipad

³⁰developer.apple.com/xcode

³¹developer.android.com/studio

Cross-platform ontwikkeling

Als tweede vorm van applicatieontwikkeling zal er gekeken worden naar cross-platform applicatieontwikkeling. Deze ontwikkelingsvorm wordt echter door veel software toolkits en omgevingen aangeboden. Voor deze besprekking zal cross-platform in het algemeen bekijken en in een verder stadium zal er toegespitst worden op Kotlin Multiplatform Mobile.

Cross-platform ontwikkeling is echter een zeer ruim gegeven en kan opgesplitst worden in een aantal categorieën. Uit de studie van Xanthopoulos en Xinogalos (2013) komen volgende categorieën:

- Web applicaties

Web applicaties zijn applicaties die gebruikers kunnen gebruiken binnen de web-browser en maken gebruik van HTML³², CSS³³ en JavaScript³⁴. Een groot voordeel aan dit soort applicaties is dat deze geen installatie vereisen wat het makkelijker maakt voor de eindgebruiker. Deze applicaties kunnen echter wel een soort van installatie³⁵ krijgen waarbij er een link voorzien word naar de online versie. Daar hangt wel een nadeel aan vast, aangezien de applicaties niet geïnstalleerd worden op het toestel kunnen ze geen gebruik maken van de hardware van het platform. Een ander belangrijk gegeven is dat deze applicaties vereisen dat de eindgebruiker een internetverbinding heeft minstens voor het eerste gebruik. Deze kunnen echter voorzien worden om een bepaalde periode te werken zonder internet. Recente technologieën proberen aan de hand van APIs de ontwikkelaar toch de mogelijkheid te geven om bepaalde platform specifieke hardware of software te gebruiken. Door deze oplossing wordt deze variant al veel interessanter voor potentiële gebruikers en ontwikkelaars. Indien een web applicatie voldoet aan volgende tien voorwaarden (Osmani, 2015) dan kunnen we spreken van een progressive web applicatie (PWA). Deze tien voorwaarden zijn:

1. Progressief
 - De PWA werkt voor elke gebruiker onafhankelijk van de keuze van browser.
2. Responsief
 - De PWA is beschikbaar voor verschillende formaten zoals desktop of mobiel.
3. Netwerk onafhankelijk
 - De PWA is voorzien van service workers en kan werken zonder internet.
4. App-like
 - De PWA voelt aan als een native applicatie
5. Recent
 - Alle processen van de PWA zijn up-to-date
6. Veilig
 - De PWA is voldoende beveiligd zodat alle data veilig en betrouwbaar is.
7. Ontdekbaar

³²html.spec.whatwg.org

³³w3.org/Style/CSS

³⁴developer.mozilla.org/en-US/docs/Web/JavaScript

³⁵support.google.com/chrome/answer/9658361?co=GENIE.Platform%3DDesktop&hl=en

- De PWA is vindbaar aan de hand van verschillende zoekmachines
 - 8. Makkelijk opnieuw inschakelbaar
 - De PWA maakt gebruik van meldingen op het toestel zodat de gebruiker snel terug kan gebruiken.
 - 9. Installeerbaar
 - De PWA is installeerbaar en terug te vinden op het beginscherf van het toestel van de gebruiker.
 - 10. Linkbaar
 - De PWA is eenvoudig te delen aan de hand van een URL
- Hybride applicaties

Deze applicaties zijn een combinatie van native applicaties en web applicaties. Voor deze applicaties wordt er meestal gebruik gemaakt van een HTML5³⁶ web applicatie in combinatie met JavaScript. Deze applicatie zal dan in een native webcontainer container getoond worden. Binnen deze container zal de web applicatie gewoon werken zoals binnen een browser. Doordat deze vorm van applicaties geïnstalleerd worden op het platform is het wel mogelijk om bepaalde platform specifieke hardware of software aan te spreken en te gebruiken.
 - Geïnterpreteerde applicaties

Dit soort applicaties maakt gebruik van een platform onafhankelijke bedrijfslogica met daar bovenop een native user interface. Deze aanpak is efficiënt op het vlak van de user interface maar kan nadelig zijn op vlak van de bedrijfslogica. Hierbij kan het probleem ontstaan dat de applicatie te afhankelijk wordt van de gekozen technologie van de bedrijfslogica. Een voorbeeld van een probleem dat dit kan creëren is dat bepaalde nieuwe features die beschikbaar komen voor een platform niet kunnen geïmplementeerd worden, dit omdat de technologie van de bedrijfslogica dit niet ondersteunt.
 - Gegenereerde applicaties

Gegenereerde applicaties zijn cross-platform applicaties die per platform een native applicatie creëren die dan op het toestel native kan verwerkt worden. Deze applicaties zijn zeer performant aangezien deze in de native programmeertaal voor het platform gegenereerd zijn. Deze versies zullen dus het dichtste aanleunen tegen de effectieve native applicaties. Deze applicaties zijn echter niet perfect aangezien het proces dat de native code genereert niet altijd zonder problemen verloopt. Alsook is het niet altijd mogelijk om alle code te delen voor alle platformen.

Uit het onderzoek van Xanthopoulos en Xinogalos (2013) kan volgende tabel gehaald worden, zie tabel 2.1, die bovenstaande informatie kort samenvat. De tabel omvat volgende factoren:

- Heeft de applicatie een marktplaats-implementatie of kan deze enkel via de browser worden gebruikt/gedownload?
- Is de technologie die gebruikt wordt een veelgebruikte technologie?
- Is de platform specifieke hardware en software bereikbaar voor de applicatie?

³⁶developer.mozilla.org/nl/docs/Web/Guide/HTML/HTML5

- Hoe zal de user interface eruitzien voor de gebruiker?
- Hoe zal de prestatie van de applicatie zijn?

Tabel 2.1: Vergelijking en samenvatting van de verschillende cross-platform categorieën

Factor	Web applicatie	Hybride applicatie
Marktplaats-implementatie	Nee	Ja, maar niet altijd
Veel gebruikte technologie	Ja	Ja
Platform hardware en software toegang	Beperkt	Beperkt
User interface	Simulatie	Simulatie
Performantie	Laag	Gemiddeld
Factor	Geïnterpreteerde applicatie	Gegenererde applicatie
Marktplaats-implementatie	Ja	Ja
Veel gebruikte technologie	Ja	Nee
Platform hardware en software toegang	Beperkt	Volledige toegang
User interface	Native	Native
Performantie	Gemiddeld	Hoog

Na dit overzicht van de verschillende versies van cross-platform applicaties kan de term cross-platform algemeen omschreven worden. Cross-platform applicaties zijn applicaties die zich richten op verschillende platformen tegelijk. Dezelfde applicatie of varianten van die applicatie werken op alle vooraf gekozen platformen tegelijk.

Voor het onderzoek zal er gekeken worden naar de SWOT-analyse voor cross-platform applicaties. Deze analyse schetst een beeld over de sterktes (Strengths) en zwaktes (Weaknesses) en over de kansen (Opportunities) en bedreigingen (Threats). Binnen deze analyse zijn de sterktes en zwaktes de interne factoren en kansen en bedreigingen de externe factoren.(Leigh, 2010) De SWOT-analyse is een veel gebruikte sterke-zwakteanalyse en is daardoor ideaal om dit onderzoek te ondersteunen.

In het onderzoek van Nivanaho (2019) kan er een SWOT-analyse gevonden worden voor cross-platform applicaties. Uit deze SWOT-analyse kunnen de factoren die specifiek voor React Native³⁷ eruit worden. Dan wordt volgende SWOT-analyse voor cross-platform applicaties in het algemeen bekomen. Dit geeft een beeld over de positie van cross-platform applicaties binnen het huidige landschap van applicatieontwikkeling.

Een overzicht van de SWOT-analyse voor cross-platform applicaties:

- Sterktes of Strengths
 - Sneller te ontwikkelen
 - Kostenbesparend
 - De applicatie ondersteunt meerdere platformen tegelijkertijd
- Zwaktes of Weaknesses
 - Nog steeds nood aan native code per platform
 - Upgrades kunnen omslachtiger zijn

³⁷reactnative.dev

- Kansen of Opportunities
 - Snelle ontwikkeling voor meerdere platformen tegelijkertijd
 - Native applicaties kunnen relatief vlot omgevormd worden naar cross-platform applicaties
- Bedreigingen of Threats
 - Updates aan het gekozen cross-platform systeem kunnen de reeds geschreven applicaties onbruikbaar maken

Na het overzicht van de SWOT-analyse zal elk element nog even besproken worden.

- Sterktes of Strengths:

De eerste factor die hier werd aangehaald was dat cross-platform applicaties sneller te ontwikkelen zijn. Zoals eerder vermeld wordt bij de ontwikkeling van deze soort applicaties direct ontwikkeld voor verschillende platformen tegelijk. Hierdoor zal het ontwikkelproces veel sneller zijn. Vanuit dit punt kan er direct doorgaan worden naar het volgende punt. Deze applicaties zijn dus sneller te ontwikkelen en daardoor zullen de ontwikkelingskosten ook lager liggen. Hierbij wordt vooral gedacht aan projecten die een prijsberekening doen op basis van het aantal gepresteerde uren van de ontwikkelaars. De laatste sterkte van dit soort applicaties is dat deze direct verschillende platformen zullen ondersteunen. Dit is een logisch gevolg aangezien deze van de eerste fase al ontwikkeld zijn voor deze verschillende platformen. Dit voordeel is vooral interessant wanneer bepaalde software direct een groot doelpubliek moet aanspreken dat men niet kan dekken met één platform.

- Zwaktes of Weaknesses:

Binnen de huidige markt van de cross-platform software toolkits of omgevingen zullen deze allemaal nog steeds nood hebben aan bepaalde stukken van native code. Aangezien dit nog nodig is zullen de ontwikkelaars van de applicaties nog steeds kennis moeten hebben van de native talen voor de platformen in kwestie. Gezien niet alle delen van de code gedeeld zijn over alle platformen kan het updaten van de code minder vlot verlopen. Hierbij is vooral het risico dat bepaalde zaken voor bepaalde platformen vergeten worden of verkeerd geïmplementeerd worden. Anderzijds kan dit, indien goed uitgevoerd, wel een sterkte zijn omdat alle platformen waarvoor de software ontwikkeld is de update gelijktijdig zullen krijgen.

- Kansen of Opportunities:

Bepaalde kansen die cross-platform applicaties hebben is dat door recente technologieën de applicaties nog sneller ontwikkeld kunnen worden voor verschillende platformen tegelijk. Hierdoor zullen deze applicaties steeds interessanter worden voor potentiële gebruikers. Een andere punt is het feit dat de meeste technologieën het zeer makkelijk maken voor de ontwikkelaars om de applicaties om te vormen naar een cross-platform applicatie. Hierdoor kunnen geïnteresseerde gebruikers met een reeds bestaande native applicatie toch de overstap maken. Native ontwikkelen is dus niet iets dat vastligt of wat de ontwikkelaar strikt moet blijven volgens eens dat gekozen is. Cross-platform is bijgevolg dus voor bijna alle applicaties een optie.

- Bedreigingen of Threats: Een van de grote bedreigingen voor cross-platform applicaties is de afhankelijkheid van de gekozen software toolkit of omgeving. Indien door updates binnen deze toolkits of omgevingen bepaalde code niet meer zou werken, zullen alle platformen daar de problemen ondervinden. Dit is een gegeven dat binnen native applicaties geen probleem aangezien elke applicatie voor elk platform een eigen afzonderlijk geheel vormt los van de andere platformen en hun applicaties.

Hierbij moet nogmaals vermeld worden dat deze factoren gelden voor cross-platform applicaties in het algemeen. Hierdoor zijn sommige factoren mogelijk niet toepasbaar op bepaalde software toolkits of omgevingen.

2.3 Kotlin Multiplatform

Kotlin Multiplatform (KM) is een cross-platform software development kit (SDK) dat verwerkt zit in Kotlin en kan geplaatst worden onder de gegenereerde applicaties. Dit wil zeggen dat Kotlin compiler de geschreven code zal kunnen omzetten naar native code voor de gekozen platformen.(Evert, 2019) Deze SDK werd voor het eerst verwerkt in Kotlin 1.2 in november 2017 als experimentele functie.(Jemerov, 2017) Naast KM werd ook een SDK uitgebracht specifiek gericht op de mobiele toestellen namelijk Kotlin Multiplatform Mobile (KMM).

Deze SDK's zullen het concept van cross-platform anders aanpakken dan andere alternatieven binnen het huidige landschap van cross-platform ontwikkeling. Hierbij zit het verschil vooral in welke code van de applicatie gedeeld wordt tussen de verschillende platformen. In de Kotlin documentatie staat beschreven hoe bepaalde delen van de code correct kunnen gedeeld worden tussen verschillende platformen. KM zal anders dan andere alternatieven de business logica delen tussen de verschillende platformen.(Kotlin & JetBrains, 2021a) Hierbij wordt ook vermeld dat men enkel de business logica moet delen die gebruikt kan worden op alle platformen. De andere alternatieven worden besproken binnen 2.4 Kotlin Multiplatform versus alternatieven.

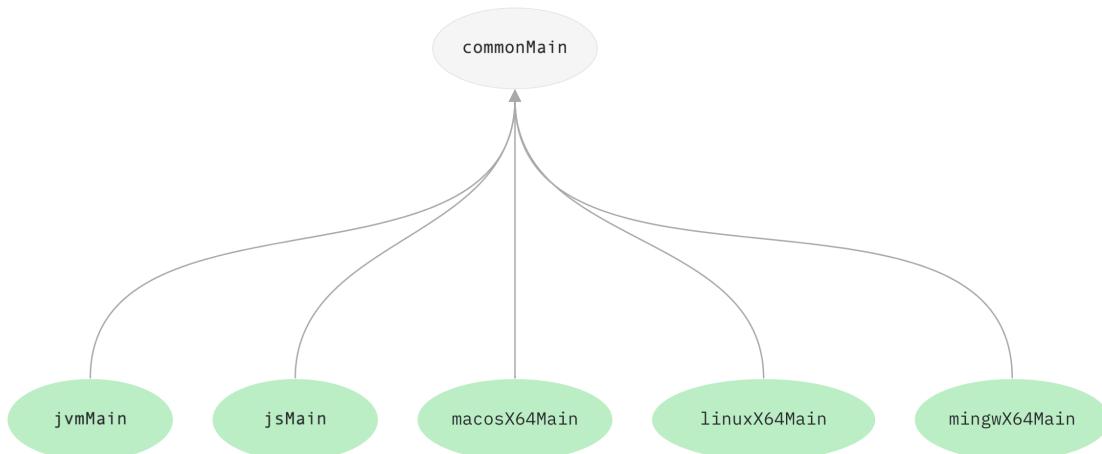
Ondertussen heeft de ontwikkeling van KMM niet stil gestaan en is in augustus 2020 de alpha versie van deze SDK uitgebracht voor het grote publiek.(Petrova, 2020) In november 2020 werd de 0.2.0 versie uitgebracht, deze is verwerkt in Kotlin 1.4.20.(JetBrains, 2020) Enkele delen van deze SDK en bijhorende componenten zijn nog steeds in de experimentele fase van ontwikkeling, maar JetBrains geeft aan dat het nu al een ideaal moment is om deze software te testen.(Petrova, 2020) De community achter KMM en het open source verhaal spelen hier een grote rol en zullen ook zorgen voor een snellere ontwikkeling van deze software. Bedrijven kunnen dus momenteel al experimenteel aan de slag met deze nieuwe software en voorbeelden van enkele bedrijven die de stap al hebben gezet naar KMM zijn onder andere Netflix³⁸, VMware³⁹ en Autodesk⁴⁰.(Kotlin, 2020)

³⁸[netflix.com](https://www.netflix.com)

³⁹[vmware.com](https://www.vmware.com)

⁴⁰[autodesk.com](https://www.autodesk.com)

Momenteel kan Kotlin Multiplatform voor volgende platformen code generen: Android NDK⁴¹, iOS⁴², Linux⁴³, macOS, Windows en WebAssembly⁴⁴ en daarboven op kan het ook omzetten naar native JavaScript.(Kotlin & JetBrains, 2021b) Dit wordt grafisch voorgesteld in figuur 2.1.



Figuur 2.1: Grafische voorstelling Kotlin Multiplatform en de verschillende platformen (Kotlin & JetBrains, 2021b)

Na de algemene uitleg rond KM en KMM kan er nu in meer detail gekeken worden naar KMM. Om deze uitleg te ondersteunen zal verwezen worden naar figuur 2.2. Deze figuur geeft een goed algemeen beeld van de structuur van KMM. De shared code in de figuur verwijst hier naar Common Kotlin of CommonMain, dit is het gedeelde binnendoor KM dat de gedeelde logica zal bevatten, dit was ook al zichtbaar op figuur 2.1. Daarnaast zal er nog per platform een aparte main zijn die de code zal implementeren. Hier in de figuur zal het project ook nog een iosMain en een kotlinMain, deze kunnen later ook nog uitgebreid worden met bijvoorbeeld een macosX64Main van KM. De situatie geïllustreerd in de figuur is een applicatie waar Kotlin Multiplatform Mobile gebruikt wordt. Deze zal zich speciaal richten op applicaties voor iOS en Android. Voor de communicatie tussen het common deel en de platform specifieke delen zal Kotlin gebruik maken van het expected/actual systeem. Hierbij worden binnenin de CommonMain elementen gedeclareerd met expect en de platform specifieke delen zullen dezelfde elementen declareren met actual. Deze structuur kan gebruikt worden voor functies, klassen, interfaces, enumeraties, properties en annotaties.

In figuur 2.3 kan bekijken worden hoe KMM geïmplementeerd zal worden binnendoor een project dat geschreven is voor de platformen iOS en Android. Bij de figuur kan ook volgende code teruggevonden worden in de documentatie van KMM.(Kotlin & JetBrains, 2021b)

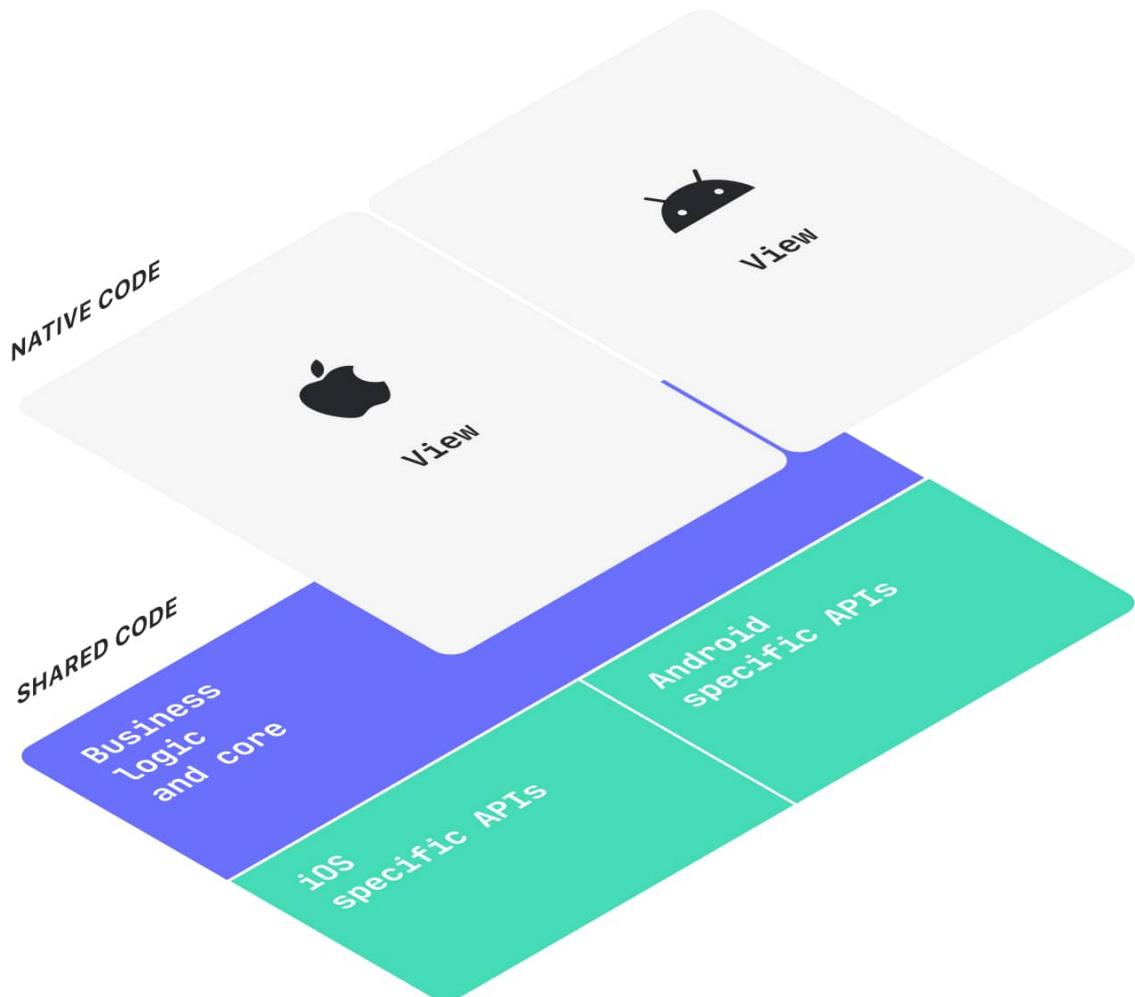
- Code die in het common deel staat van KMM:

⁴¹developer.android.com/ndk

⁴²apple.com/benl/ios/ios-14

⁴³linux.org

⁴⁴webassembly.org



Figuur 2.2: Grafische voorstelling Kotlin Multiplatform Mobile (Kotlin & JetBrains, g.d.)

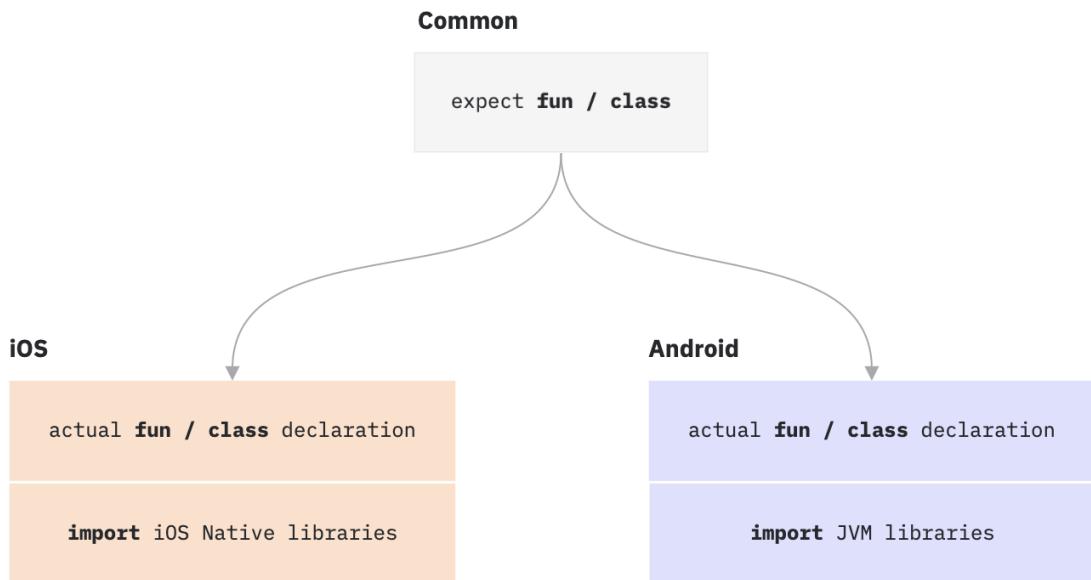
```
//Common  
expect fun randomUUID(): String
```

- Code die in het Android deel zal staan van de applicatie

```
//Android  
import java.util.*  
actual fun randomUUID() = UUID.randomUUID().toString()
```

- Code die in het iOS deel zal staan van de applicatie

```
//iOS  
import platform.Foundation.NSUUID  
actual fun randomUUID(): String = NSUUID().UUIDString()
```



Figuur 2.3: Grafische voorstelling van de code binnen KMM met iOS en Android als voorbeeld (Kotlin & JetBrains, 2021b)

2.4 Kotlin Multiplatform versus alternatieven

Nu duidelijk is wat Kotlin Multiplatform (KM) en Kotlin Multiplatform Mobile (KMM) omvat kan er bekeken welke andere alternatieven er zijn binnen het huidige landschap van cross-platform ontwikkeling. Een van de alternatieven die besproken zal worden is Flutter⁴⁵.

Flutter is een user interface toolkit ontwikkeld door Google en zit aan versie 1.22 sinds oktober 2020.(Sells, 2020) De toolkit richt zich op mobile, web en desktop applicaties en zal de code native compileren. Zoals reeds beschreven is Flutter een user interface toolkit en zal dus de user interface delen over de verschillende platformen. Hiervoor zal Flutter widgets gebruiken die geïnspireerd zijn door React.(Flutter, g.d.) Dit impliceert dus dat hierbij de business logica zal moeten verwerkt worden per platform.

De ontwikkelaar van de cross-platform applicaties kan dus verschillende kanten uitgaan voor de ontwikkeling van de applicatie. Een eerste mogelijk is de reeds besproken technologie van KM en KMM hierbij zal de business logica gedeeld worden tussen de verschillende platformen. Het grootste voordeel aan deze manier van werken is dat de gebruiker een native ervaring zal hebben bij het gebruik van de applicatie. Daartegenover staat wel dat de user interface niet voor alle platformen hetzelfde is. Een andere mogelijke manier is het delen van de user interface, een besproken voorbeeld hiervan is Flutter. Deze manier zal als voordeel hebben dat alle applicaties op de verschillende platformen dezelfde look en feel zullen hebben, een nadeel is echter dat de business logica per platform verwerkt zal moeten worden.

⁴⁵flutter.dev

2.5 Testcriteria

Om een goede vergelijkende studie te maken is het belangrijk om op voorhand bepaalde testcriteria vast te leggen die gebruikt kunnen worden als maatstaf voor de geschreven applicaties. Hierbij is het belangrijk rekening te houden met het feit dat deze testcriteria van toepassing moeten zijn op zowel de native als cross-platform applicaties. Voor deze vergelijkende studie zijn volgende testcriteria gekozen en aan de hand van deze testcriteria zal gedocumenteerd worden hoe goed de cross-platform applicaties presteren ten opzichte van de native varianten. De testcriteria zijn: het aantal lijnen code, de compileersnelheid, de voetafdruk, de ontwikkeltijd, de kostprijs. Deze testcriteria worden hieronder kort besproken.

- Aantal lijnen code
 - Het totaal aantal lijnen code van een applicatie zal geëvalueerd worden over het gehele project. In het geval van native applicaties worden deze opgeteld bij elkaar.
- Compileersnelheid
 - Dit is de snelheid waarmee de specifieke applicatie zal kunnen compileren. Dit kan gemeten worden in de ontwikkelingssoftware voor de desbetreffende programmeertaal van de applicatie.
- Voetafdruk
 - Dit impliceert de omvang die de applicatie zal innemen op het platform waarvoor deze ontwikkeld is. Hiervoor kan de applicatie gebruikt worden die de ontwikkelingssoftware genereert.
- Ontwikkeltijd
 - Deze tijd beschrijft het aantal werkuren dat een ontwikkelaar nodig heeft om een specifieke applicatie te schrijven. Hierbij kan onder andere gebruik gemaakt worden van platformen zoals GitHub om deze tijd te meten of inschat-ten.
- Kostprijs
 - Hierbij wordt de geschatte kostprijs om een applicatie te laten ontwikkelen door een IT-bedrijf in kaart gebracht. Dit wordt berekend aan de hand van geschatte werkuren en een gemiddelde kostprijs per uur. Daarnaast wordt nagegaan of cross-platform een effect zal hebben op het systeem van vooraf bepaalde totaalprijzen indien bedrijven daarmee werken.

3. Methodologie

3.1 Installatie

De eerste stap om dit onderzoek uit te voeren is het installeren van de correcte software. Hieronder wordt een overzicht van de delen binnen de installatie getoond:

- Hardware
- Android Studio
- Xcode
- JDK

Binnen dit deel van de methodologie zullen per deel de vereiste stappen besproken worden om de software te installeren.

3.1.1 Hardware

Vooraleer deze software te installeren is het belangrijk te weten dat een computer of laptop die draait op macOS¹ noodzakelijk is. Hierbij is het ook aangeraden te controleren dat de geïnstalleerde versie minstens macOS Big Sur² 11.3.1 is. Dit zal later van belang zijn gezien de testen gedraaid worden op emulators met iOS 14.3³ en hiervoor macOS 11.3.1 vereist is. De software kan echter ook geïnstalleerd worden op een computer of laptop die draait op Windows⁴ maar dan kunnen geen applicaties, noch native noch cross-platform,

¹apple.com/benl/macos

²apple.com/benl/macos/big-sur

³apple.com/benl/ios/ios-14

⁴microsoft.com/nl-be/windows

gemaakt worden voor iOS⁵.

Een ander gegeven is dat nog niet alle software, voornamelijk Android Studio⁶, de laatste Apple Silicon⁷ processors ondersteunen. Een eventuele omweg kan gebeuren via Apple Rosetta 2⁸ maar dit lost niet alle problemen op. Indien er toch op een Apple Silicon toestel dient gewerkt te worden, is het aanbevolen de websites van de desbetreffende software te controleren voor compatibiliteit.

Voor dit onderzoek zal bijgevolg een laptop gebruikt worden die draait op macOS. Het toestel in kwestie is een MacBook Pro 16 inch uit 2019. Meer technische informatie over dit toestel kan op volgende link teruggevonden worden:

<https://support.apple.com/kb/SP809>

Meer informatie over de recentste versies van **macOS Big Sur** kan teruggevonden worden op volgende link:

<https://support.apple.com/en-us/HT211896>

3.1.2 Android Studio

Android Studio is het eerste deel van de vereiste software, en wordt aangeboden door JetBrains⁹ en door Google¹⁰ als software voor de ontwikkeling van Android¹¹ applicaties. Android Studio is gebaseerd op IntelliJ IDEA¹² van JetBrains. De laatste stabiele versie van Android Studio voor macOS is 4.2. Afhankelijk van het project dat gebruikt wordt, kunnen er echter nog enkele problemen optreden met de versie van Android Jetpack¹³. Dit kan opgelost worden door de Canary build van Android studio te downloaden. De laatste versie van Android Studio Canary build¹⁴ is Artic Fox (2020.3.1) Canary 15.

De laatste versie van **Android Studio** kan hier teruggevonden worden:

<https://developer.android.com/studio>

De laatste versie van **Android Studio Canary build** kan hier teruggevonden worden:

<https://developer.android.com/studio/preview>

Oudere versies van Android Studio ondersteunen de recentste plug-ins voor Kotlin Multi-platform Mobile (KMM) niet. Hierbij is het dus belangrijk dat de versie die geïnstalleerd wordt 4.2 of recenter is.

⁵apple.com/benl/ios

⁶developer.android.com/studio

⁷developer.apple.com/documentation/apple-silicon

⁸developer.apple.com/documentation/apple-silicon/about-the-rosetta-translation-environment

⁹jetbrains.com

¹⁰about.google

¹¹android.com

¹²jetbrains.com/idea

¹³developer.android.com/jetpack

¹⁴developer.android.com/studio/preview



Figuur 3.1: De ADV Manager in Android Studio met de Pixel 4a emulator geïnstalleerd

Tijdens de installatie van Android Studio kan er best geopteerd worden om de standaard instellingen te gebruiken. Daarnaast zullen verschillende zaken geïnstalleerd worden namelijk:

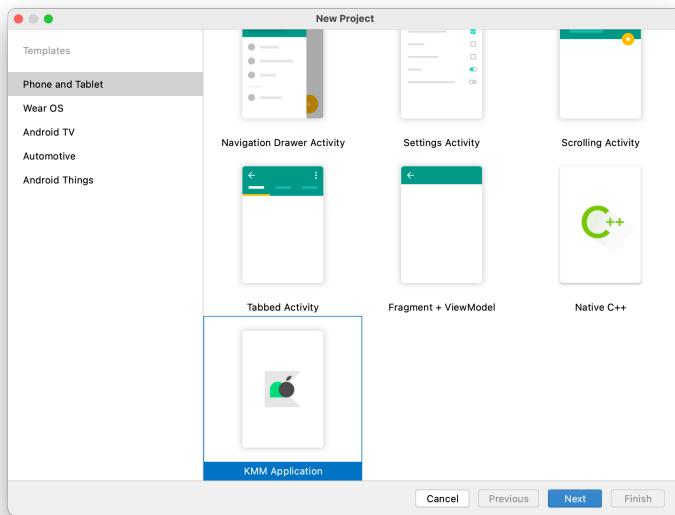
- Android Emulator
- Android SDK Build Tools 30.0.3
- Android SDK Platform 30
- Android SDK Platform-Tools
- Android SDK Tools
- Google APIs Intel x86 Atom System Image
- Intel x86 Emulator Accelerator (HAXM installer)
- SDK Patch Applier v4
- Sources for Android 30

Deze zijn allemaal van belang voor zowel de native Android ontwikkeling als de KMM cross-platform ontwikkeling. Eens de installatie voltooid is kan via de AVD manager een emulator geïnstalleerd worden. Voor deze studie zal er een Pixel 4a¹⁵ geïnstalleerd worden met Android 11¹⁶ (API Level 30). Deze emulator zal gebruikt worden om de native Android en de KMM cross-platform applicatie te testen. Eens deze emulator correct is geïnstalleerd kan deze teruggevonden worden in de ADV Manager zoals op figuur 3.1

Zoals net vermeld, is het ook nodig om de KMM plug-in te installeren voor Android Studio. De plug-in maakt het mogelijk om KMM projecten aan te maken. Hierdoor kan de business logica geschreven worden voor beide platformen, gedeeld of per platform. Een andere feature van de plug-in is dat de iOS applicatie kan gestart en gedebugd worden vanuit Android Studio. Daarnaast zullen de gebruikers door de plug-in ook de mogelijkheid krijgen om nieuwe KMM applicaties aan te maken.

¹⁵store.google.com/us/product/pixel_4a?hl=en-US

¹⁶android.com/android-11



Figuur 3.2: De opties voor een nieuw project binnen Android Studio, hierbij is een KMM applicatie ook beschikbaar

De laatste versie van de **KMM plug-in** kan hier teruggevonden worden:

<https://plugins.jetbrains.com/plugin/14936-kotlin-multiplatform-mobile>

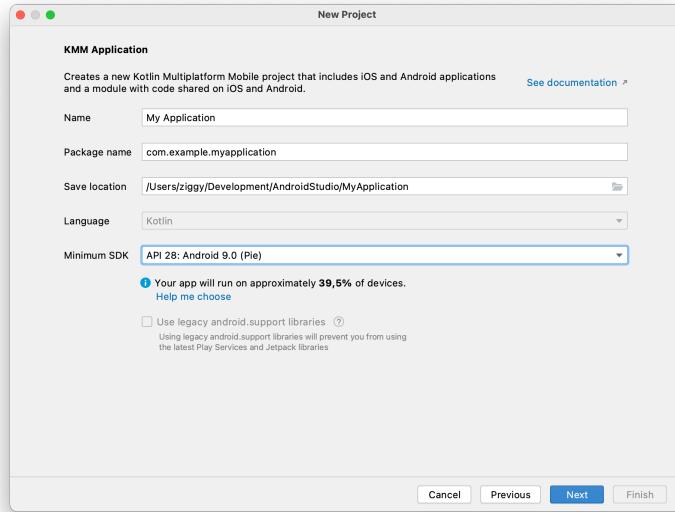
Onderstaande figuren tonen het proces om een nieuwe KMM applicatie te maken aan de hand van de KMM plug-in.

- Figuur 3.2 toont de mogelijke opties binnen Android Studio voor het aanmaken van een nieuw project, eens de plugin correct is geïnstalleerd, zal daar ook de optie 'KMM Application' beschikbaar zijn.
- Figuur 3.3 toont de standaard instellingen voor een KMM project, deze instellingen lopen gelijk met de standaard instellingen voor een standaard Android project.
- Figuur 3.4 toont de mogelijkheden om de verschillende delen van de KMM te benoemen, standaard zullen hier de waarden androidApp, iosApp en shared staan. Daarnaast kunnen deze delen nog een beschrijving krijgen. Als laatste is er de optie om templates voor testen toe te voegen aan de shared module.

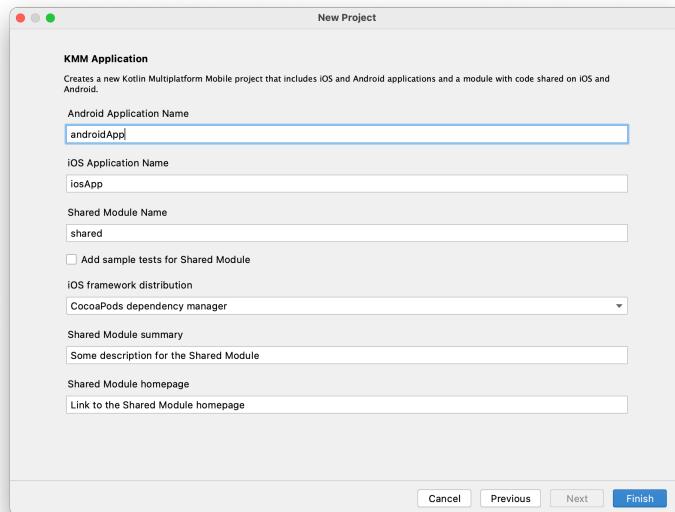
Indien er problemen zouden zijn met de KMM plug-in kan er via de instellingen van Android Studio gecontroleerd worden of de plug-in goed geïnstalleerd is en actief staat.

[Android Studio -> Voorkeuren -> Plugins -> Kotlin Multiplatform Mobile](#)

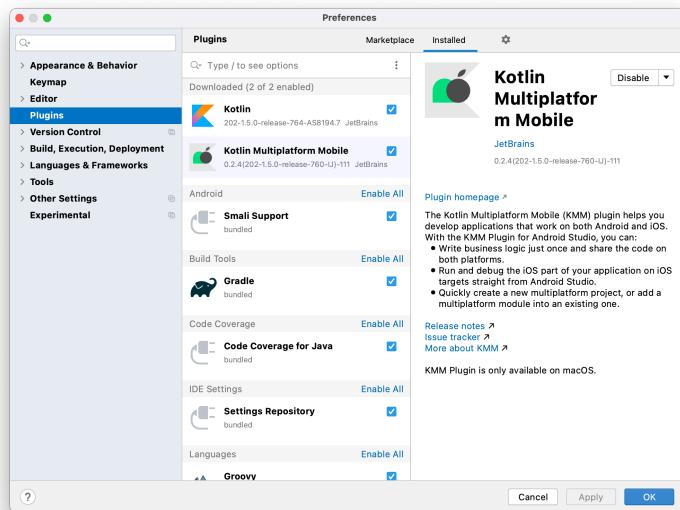
De plug-in zou bij de geïnstalleerde plug-ins moeten staan en moet actief staan.



Figuur 3.3: De standaard instellingen voor een KMM project



Figuur 3.4: De instellingen van een KMM project voor de gedeelde logica en de native iOS en Android delen



Figuur 3.5: De plug-in geïnstalleerd binnen Android Studio

3.1.3 Xcode

Het tweede deel van software dat dient geïnstalleerd te worden is Xcode¹⁷, de softwareontwikkeling omgeving van Apple¹⁸ voor toestellen die draaien op iOS, iPadOS¹⁹, macOS... Deze software is uitsluitend beschikbaar voor toestellen die draaien op macOS. De laatste beschikbare versie is Xcode 12.5, deze ondersteunt het bouwen applicaties tot iOS 14.3, iPadOS 14.3²⁰, watchOS 7.4²¹...

De laatste versie van **Xcode** kan geïnstalleerd worden via de Apple Developer website of via de App Store²² op macOS:

<https://developer.apple.com/xcode/>

<https://apps.apple.com/nl/app/xcode/id497799835?mt=12>

Naast Xcode zal voor deze studie ook gebruik gemaakt worden van Xcode Command Line Tools, deze komen normaal standaard mee met de Xcode IDE. Indien de Xcode Command Line Tools niet geïnstalleerd zouden zijn kan dit via volgend terminal commando gedaan worden.

```
//Terminal.app
xcode-select --install
```

¹⁷developer.apple.com/xcode

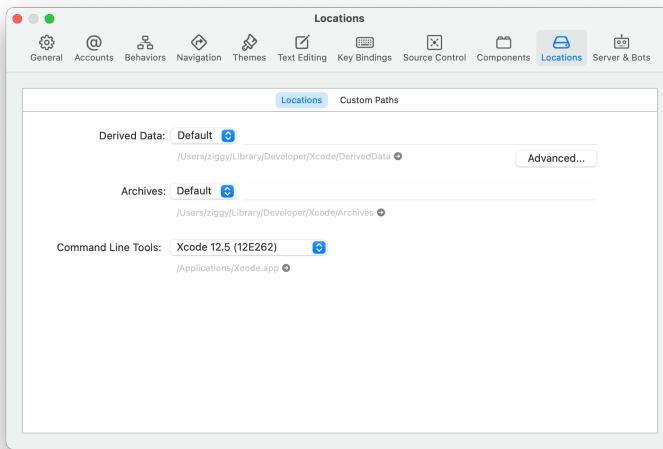
¹⁸apple.com

¹⁹apple.com/benl/ipados

²⁰apple.com/benl/ipados/ios-14

²¹apple.com/benl/watchos/watchos-7

²²apple.com/benl/app-store



Figuur 3.6: De Xcode Command Line Tools correct geconfigureerd binnen Xcode

Eens deze installatie voltooid is, dient er nog binnen Xcode een aanpassing gemaakt te worden zodat de Command Line Tools actief zijn. Via Xcode -> Voorkeuren -> Locaties, moeten de Command Line Tools ingesteld worden op ‘Xcode 12.5 (12E262)’ deze optie zou beschikbaar moeten zijn in de selectielijst. Figuur 3.6 toont het instellingenmenu voor Locaties met de correcte waarden.

Tijdens de installatie van Xcode worden alle emulators al voorzien waardoor er hier geen extra instellingen moeten aan gebeuren. Voor deze studie zal gebruik gemaakt worden van een iPhone 12²³ emulator met iOS 14.3.

Naast de installatie van Xcode zal ook CocoaPods geïnstalleerd moeten worden om de KMM plugin correct te laten werken. CocoaPods kan geïnstalleerd worden aan de hand van onderstaand commando.

```
//Terminal.app  
sudo gem install cocoapods
```

3.1.4 JDK

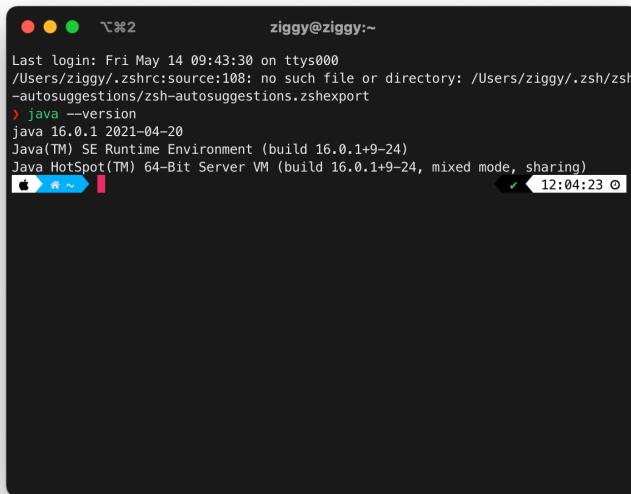
De laatste stap van het installatieproces is het installeren van een JDK of Java Development Kit. Voor deze studie zal Java SE 16²⁴ geïnstalleerd worden. Dit is de laatst beschikbare versie van Java²⁵. Alle andere versies van Java SE zijn hier terug te vinden <https://www.oracle.com/java/technologies/oracle-java-archive-downloads.html>

De laatste versie van de **Java JDK** kan geïnstalleerd worden via volgende website:

²³apple.com/benl/iphone-12

²⁴oracle.com/java/technologies/javase-downloads.html

²⁵java.com



Figuur 3.7: De terminal bevestigt dat Java SE 16 correct is geïnstalleerd op de betreffende computer

<https://www.oracle.com/java/technologies/javase-jdk16-downloads.html>

Eens de installatie voltooid is, kan de gebruiker controleren of de installatie geslaagd is door onderstaand commando uit te voeren.

```
//Terminal.app
java --version
```

Indien de installatie geslaagd is en Java SE 16 correct is geïnstalleerd, zal de gebruiker volgende output krijgen zoals in figuur 3.7

3.2 Basis Kotlin Multiplatform Mobile applicatie

Nadat alle software is geïnstalleerd kan er overgegaan worden naar de eerste Kotlin Multiplatform Mobile applicatie (KMM). De eerste applicatie zal een kleine applicatie zijn die aan de gebruiker toont welke software versie op het toestel geïnstalleerd staat.

De geschreven applicatie kan ook op GitHub²⁶ teruggevonden worden. Dit kan via volgende link:

https://github.com/ziggymoens/KMM_Application_Basic

Bij het opstarten van Android Studio geeft deze de optie om een nieuw project te star-

²⁶github.com

ten, eens dat is aangeklikt wordt er gekozen voor de optie ‘KMM Application’. Als naam wordt ‘Basic KMM Application’ gekozen en hierbij wordt de minimum SDK op level 28 (Android 9.0²⁷) gezet.

Eens het project correct is ingeladen en geïndexeerd, kan gezien worden dat het project al standaard de code bevat om een gebruiker zijn software versie te tonen. Daarnaast zal de KMM plugin binnen Android Studio de juiste omgevingen aanmaken om de code te draaien/testen binnen de juiste emulators.

Eens de emulators ingesteld zijn, kan het project voor iOS en Android worden opgestart. Na 8s 8374ms was de Android applicatie voor de eerste keer opgestart en na 6s 202ms was de iOS applicatie opgestart. Hieronder een beeld van beide applicaties in actie.

Binnen het project kan de expect/actual structuur duidelijk teruggevonden worden. Binnen de shared-map, die te zien is op figuur 3.11, bevindt zich de commonMain-map. Hierin zitten volgende twee bestanden.

```
//Greeting.kt

class Greeting {
    fun greeting(): String {
        return "Hello, ${Platform().platform}!"
    }
}

//Platform.kt

expect class Platform() {
    val platform: String
}
```

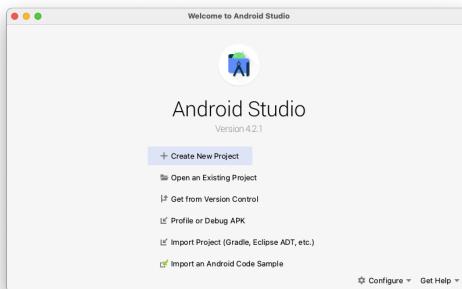
Ook in de shared-map kan de iosMain-map gevonden worden. Hierin komt het Platform.kt bestand nogmaals terug. Hierbij wordt echter de ‘expect’ geïmplementeerd en verwerkt in de ‘actual’.

```
//Platform.kt

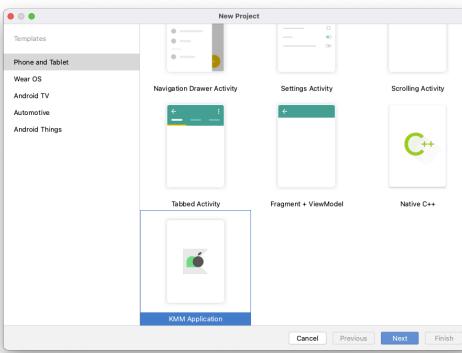
actual class Platform actual constructor() {
    actual val platform: String =
        UIDevice.currentDevice.systemName() +
        " " +
        UIDevice.currentDevice.systemVersion
}
```

De laatste map binnen de shared-map die interessant is voor deze studie is de androidMain-map en hierin wordt ook de Platform.kt verwerkt.

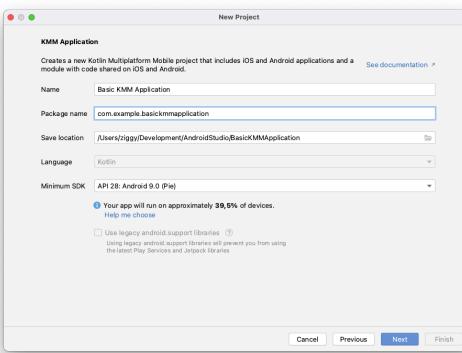
²⁷android.com/versions/pie-9-0



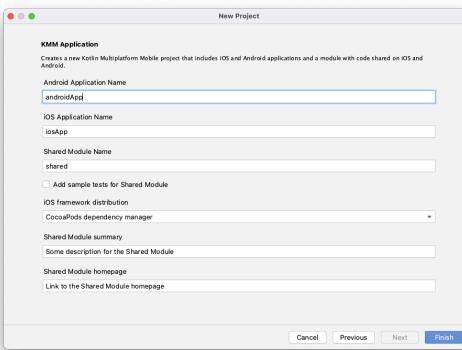
3.8.1: Startscherm Android Studio



3.8.2: Nieuw project met de KMM application optie

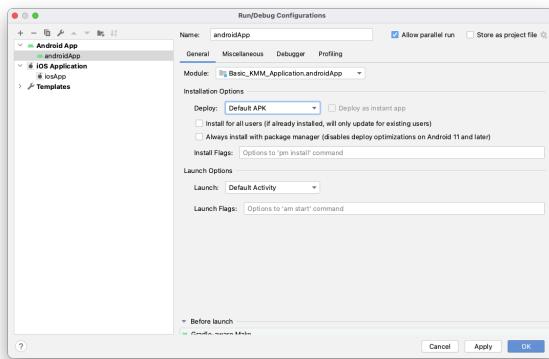


3.8.3: Standaard instellingen Android Project

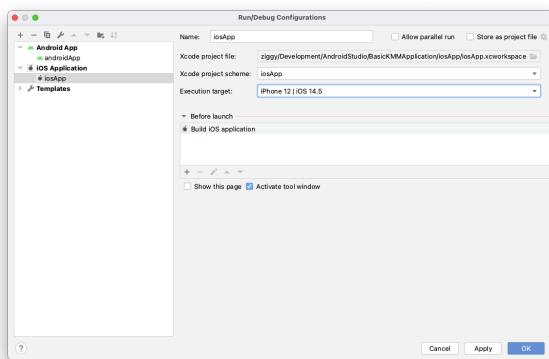


3.8.4: Specifieke KMM project instellingen

Figuur 3.8: Proces voor het opzetten van een KMM project

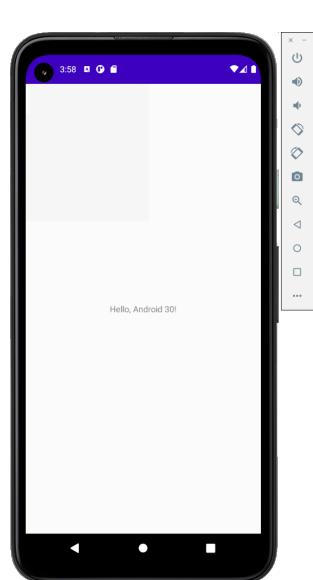


3.9.1: Instellingen Android emulator



3.9.2: Instellingen iOS emulator

Figuur 3.9: Android en iOS emulators binnen Android Studio

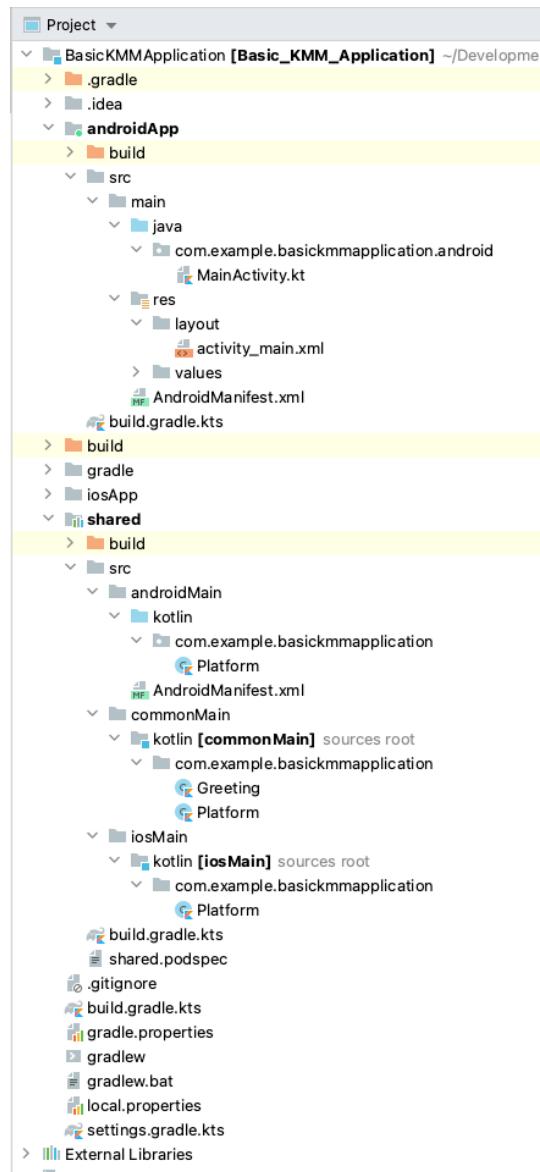


3.10.1: Android applicatie



3.10.2: iOS applicatie

Figuur 3.10: KMM basic applicatie op Android en iOS



Figuur 3.11: Projectstructuur van het KMM project

```
//Platform.kt

actual class Platform actual constructor() {
    actual val platform: String =
        "Android ${android.os.Build.VERSION.SDK_INT}"
}
```

Verdere uitleg omtrek de code zal gebeuren in de delen van de native applicaties gezien de code daar hetzelfde is.

Zoals duidelijk zichtbaar zal in de shared-map de globale structuur worden vastgelegd en deze zal dan verwerkt worden binnen de platform specifieke delen.

Uiteindelijk zal de Greetings klasse worden opgeroepen binnen de Android en iOS applicaties zoals hieronder geïllustreerd.

```
//Android - MainActivity.kt

fun greet(): String {
    return Greeting().greeting()
}

//iOS - ContentView.swift

struct ContentView: View {
    let greet = Greeting().greeting()

    var body: some View {
        Text(greet)
    }
}
```

3.3 Basis native applicaties

Nu de KMM applicatie gemaakt is, kan de native variant voor Android en iOS gemaakt worden. Hierbij zal geprobeerd worden deze zo goed mogelijk op elkaar af te stemmen.

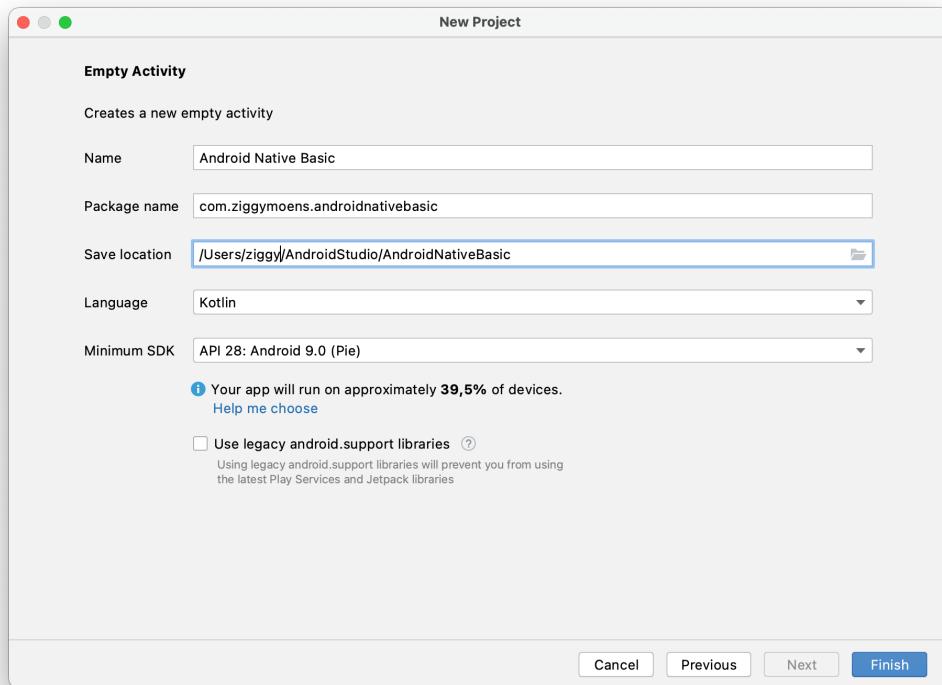
De geschreven applicaties kunnen ook op GitHub²⁸ teruggevonden worden. Dit kan via volgende links:

Android:

https://github.com/ziggymoens/Android_Native_Basic

iOS:

²⁸github.com



Figuur 3.12: Instelling voor het native Android project

https://github.com/ziggymoens/iOS_Native_Basic

3.3.1 Android

Het eerste native project dat gemaakt zal worden is het Android project. Dit project zal ook geschreven worden in Kotlin binnen Android Studio. Bij het aanmaken van een nieuw project kan de optie 'Empty Activity' gekozen worden. Daarna worden volgende gegevens ingevuld, te zien of figuur 3.12. Voor de layout worden de AndroidX²⁹ bibliotheken gebruikt.

Eens het project is geïndexeerd en Gradle³⁰ voor de eerste keer alles heeft gebuild, kan gestart worden met de ontwikkeling van de applicatie. Hierbij zijn er twee zaken die moeten gebeuren:

- Het invullen van de MainActivity zodat de juiste versie van Android getoond wordt aan de gebruiker
- Het aanmaken van een layout file zodat ook de user interface overeenstemt met de KMM applicatie

²⁹developer.android.com/jetpack/androidx

³⁰gradle.org

Binnen de Main Activity kan ‘android.os.Bundle’ geïmporteerd worden, via deze weg kan achterhaald worden wat de huidige Android versie van het betreffende toestel is. Deze waarde zal een getal teruggeven en binnen de gecontroleerde omgeving van de emulator wordt hier versie 30 verwacht.

Voor de Main Activity is de code als volgt:

```
//MainActivity.kt
class MainActivity : AppCompatActivity() {

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

        val tv: TextView = findViewById(R.id.text_view)
        tv.text = "Hello, Android ${android.os.Build.VERSION.SDK_INT}!"
    }
}
```

Binnen de layout file wordt enkel een tekstveld geplaatst met het id ‘text_view’, zodat deze binnen de Main Activity kan worden opgeroepen. Daarnaast wordt het tekstveld horizontaal en verticaal gecentreerd binnen het scherm, dit gebeurt aan de hand van een ConstraintLayout³¹.

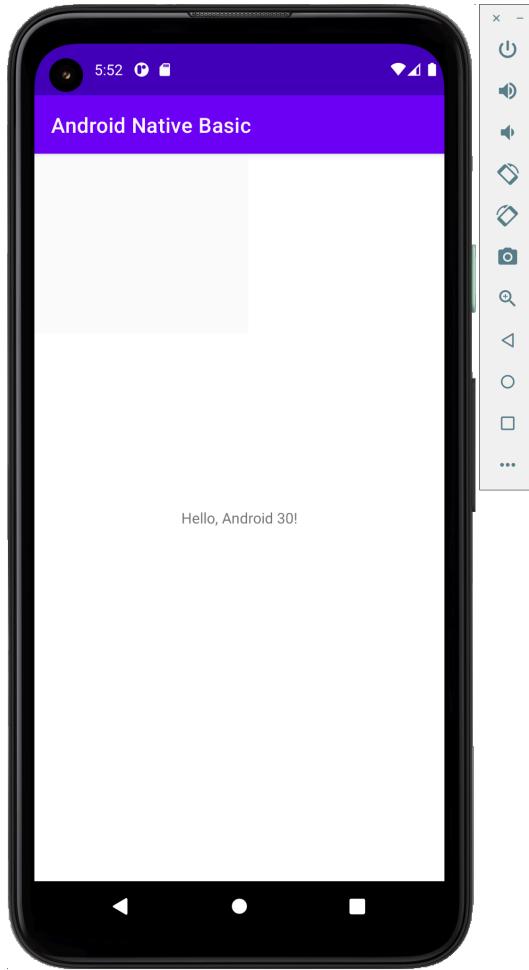
Voor de layout file wordt volgende code bekomen:

```
//activity_main.xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:text="Hello World!" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

³¹developer.android.com/training/constraint-layout



Figuur 3.13: Interface van de basic native Android applicatie

Eens de code geschreven is kan deze voor een eerste maal uitgevoerd worden op de emulator. Figuur 3.13 toont de user interface van de Android applicatie.

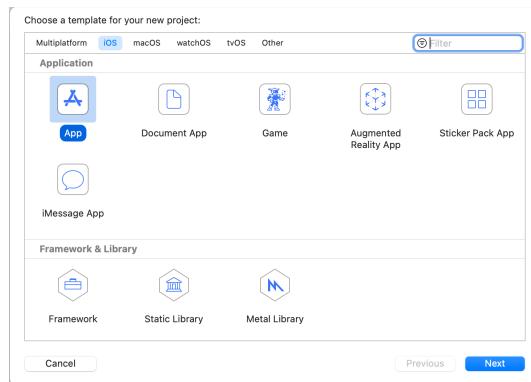
3.3.2 iOS

De andere native applicatie die dient geschreven te worden is een iOS applicatie. Deze applicatie zal geschreven worden in Swift³² binnen Xcode. Voor deze applicatie zal gebruik gemaakt worden van UIKit³³ voor de layout van de applicatie. Binnen Xcode zal gekozen worden voor een nieuw project namelijk een iOS App.

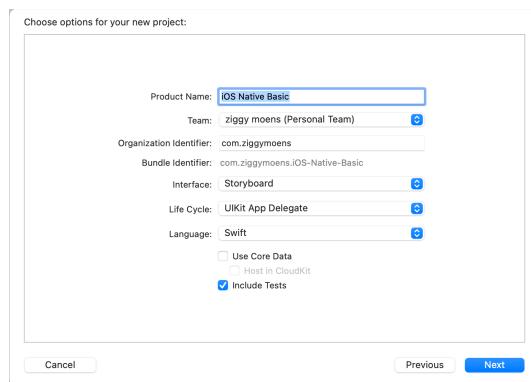
De eerste stap voor het iOS project is het aanmaken van het label in de user interface, dit gebeurt via het Storyboard. Daarna kan de link gelegd worden tussen de user interface en de view controller. Binnen de view controller kan daarna de juiste waarde gegeven worden aan het tekstveld. De iOS applicatie kunnen we aan de hand van 'UIDevice' de gegevens ophalen van het toestel dat de applicatie aan het draaien is. UIDevice is een deel

³²apple.com/benl/swift

³³developer.apple.com/documentation/uikit



Figuur 3.14: De optie voor een nieuwe iOS App binnen Xcode



Figuur 3.15: Instelling voor het native iOS project

van het UIKit framework en zit standaard geïmplementeerd in de view controllers.

Uiteindelijk wordt volgende code voor de view controller bekomen:

```
// ViewController.swift

class ViewController: UIViewController {

    @IBOutlet weak var label: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()

        label.text = "Hello, " + UIDevice.current.systemName + " " +
            UIDevice.current.systemVersion
    }
}
```

Eens al de code geschreven is kan de code getest worden op de iPhone 12 emulator binnen Xcode. Figuur 3.16 toont de interface van de basic native iOS applicatie.



Figuur 3.16: Interface van de basic native iOS applicatie

3.4 Testen van de basis applicaties

Gezien de basis applicaties reeds geschreven werden, kan nu overgegaan worden op de testen ervan. Hierbij een overzicht van de uit te voeren testen op de applicaties:

- Aantal lijnen code
- Compileersnelheid
- Voetafdruk
- Ontwikkeltijd
- Kostprijs

3.4.1 Aantal lijnen code

De eerste test gaat het aantal lijnen code na. Voor deze test is de Statistics³⁴ plug-in geïnstalleerd binnen Android studio

De **Statistics plug-in** kan via volgende link teruggevonden worden:

<https://plugins.jetbrains.com/plugin/4509-statistic>

De instellingen kunnen op volgende plek teruggevonden worden

Android Studio -> Voorkeuren -> Tools -> Statistics

Voor deze testen worden volgende specifieke instellingen geselecteerd:

Excluded file types:

‘class;svn-base;svn-work;Extra;gif;png;jpg;jpeg;bmp;tga;tiff;ear;war;zip;jar;iml;iws;ipr;bz2;gz;pyc;’

Separate TABs file types:

‘java;xml;css;html;js;properties;jsp;txt;php;php4;php5;phtml;inc;py;vue;kt’

Daarnaast dienen volgende opties aangevinkt te worden:

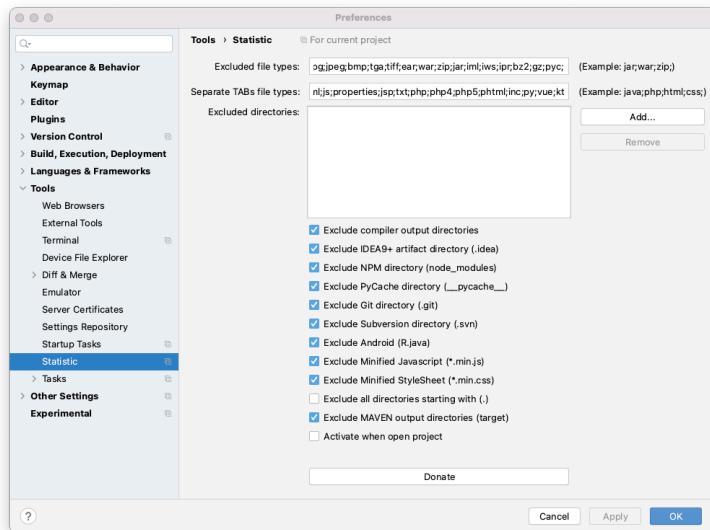
- Exclude compiler output directory
- Exclude IDEA9+ artifact directory (.idea)
- Exclude NPM directory (node_modules)
- Exclude PyCache directory (__pycache__)
- Exclude Git directory (.git)
- Exclude Subversion directory (.svn)
- Exclude Android (R.java)
- Exclude Minified Javascript (*.min.js)
- Exclude Minified StyleSheet (*.min.css)
- Exclude all directories starting with (.)
- Exclude MAVEN output directories (target)

Figuur 3.17 toont de instellingen binnen Android Studio.

Voor de projecten zal dezelfde plug-in en IDE gebruikt worden. Hierdoor kunnen fouten in de meting tot een minimum beperkt worden. Voor de gedetailleerde resultaten wordt verwezen naar bijlage C ‘Resultaten van aantal lijnen code’.

In tabel 3.1 staat de samenvatting van resultaten de drie projecten. Het iOS werd, net zoals Android en KMM, ingeladen in Android Studio zodat de Statistics plug-in gebruikt kon worden.

³⁴<https://plugins.jetbrains.com/plugin/4509-statistic>



Figuur 3.17: Instellingen voor de Statics plug-in

Tabel 3.1: Statistiek resultaten voor KMM en de native projecten

Factor	iOS	Android	KMM
Aantal bestanden	18	20	41
Totale grote	90649 B	17906 B	77725 B
Kleinste bestand	75691 B	3745 B	42370 B
Grootste bestand	82640 B	10466 B	54864 B
Gemiddelde grootte	78547 B	5322 B	48330 B
Aantal lijnen	1857	495	1779
Lijnen code	1634	389	1605

Conclusie

Uit deze resultaten kunnen enkele conclusies getrokken worden.

- Het KMM platform bevat niet meer lijnen code dan native Android en iOS samen
KMM: 1605 lijnen
Native Android + native iOS: 2023 lijnen
 - Het iOS project is groter, zowel lijnen als omvang, dan het KMM project
KMM: 1605 lijnen en 77725 B
iOS: 1634 lijnen en 90649 B
 - De Native Android applicatie uitzonderlijk veel kleiner is dan het iOS project
Android: 389 lijnen en 17906 B
iOS: 1634 lijnen en 90649 B

Dit zijn enkele van de meest opvallende en daarnaast ook de meest relevante zaken voor

dit onderzoek.

3.4.2 Compileersnelheid

Deze volgende test die werd uitgevoerd is de compileersnelheid test. Hiervoor zal elke applicatie meermaals gecompileerd worden. Hierbij werd geëvalueerd welke applicaties het snelste compileren en dus het meest efficiënt zijn.

Tijdens deze test zullen de basic KMM, Android en iOS applicaties 100 maal gebouwd worden. De KMM applicatie werd gebouwd in Android Studio (AS) en Xcode. De gedetailleerde resultaten kunnen in deel B. De resultaten worden opgesplitst in twee delen namelijk de eerste build en de andere builds. Hierdoor wordt er rekening gehouden met de cache van de IDE.

Eerste compilatie

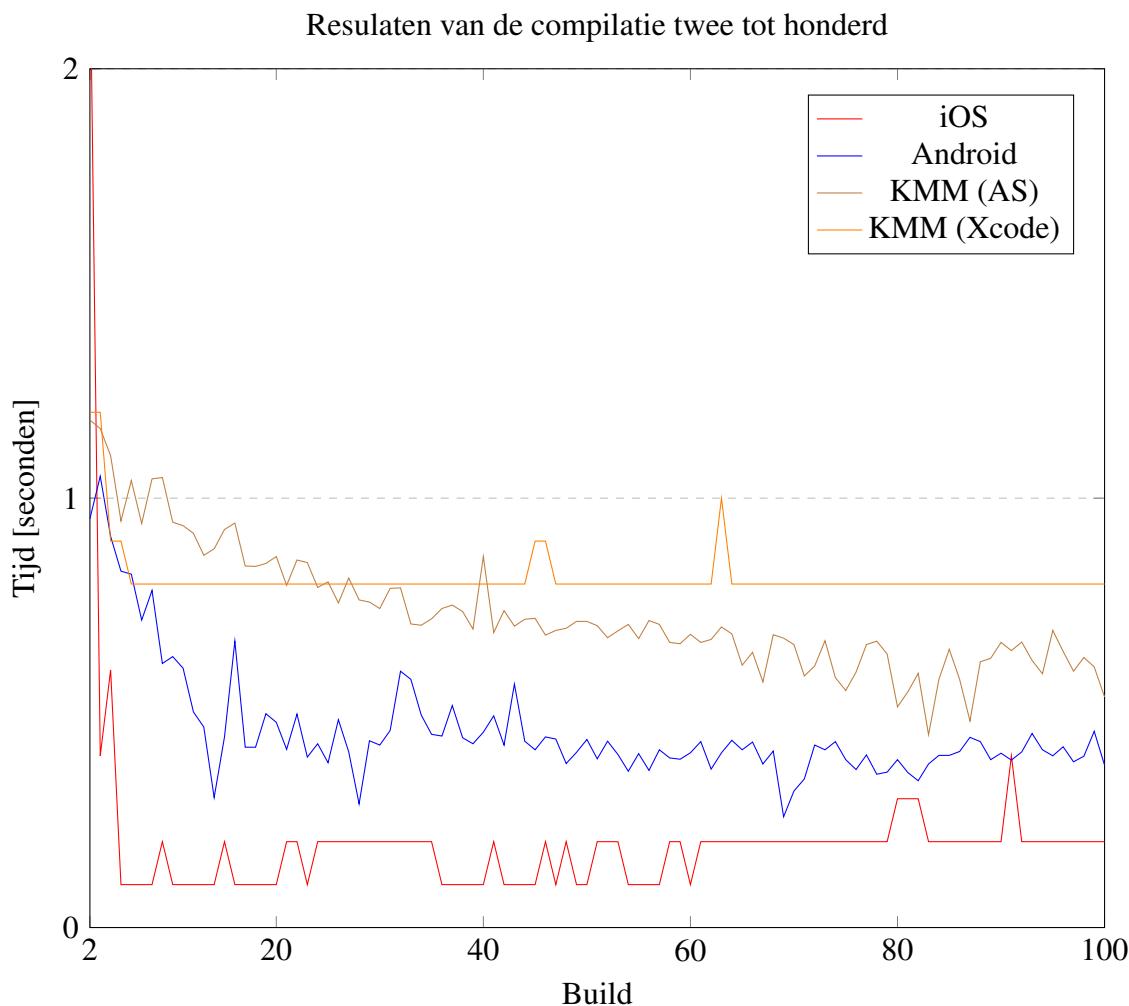
Tabel 3.2 toont de resultaten van de eerste compilatie van de applicaties. Hierbij kan worden opgemerkt dat KMM significant sneller is dan native Android en iOS, zeker wetende dat KMM automatisch gebouwd is voor beide varianten. De eerste builds bevatten nog geen cache in de IDE dus de performantie is puur afhankelijk van het framework en hoe efficiënt dit werkt. Uit deze resultaten kan dus geconcludeerd worden dat voor deze studie KMM sneller was dan de native varianten.

Tabel 3.2: Resultaten van de statistiek voor het KMM project

Platform	Eerste compileertijd
Native Android	10,900
Native iOS	21,531
KMM - AS	5,873
KMM - Xcode	10,900

Andere compilaties

Grafiek 3.4.2 toont de resultaten voor de drie applicaties voor compilatie twee tot en met 100:



Tabel 3.3 is gebaseerd op dezelfde gegevens als grafiek 3.4.2.

Tabel 3.3: Resultaten van de compilatie twee tot honderd

Factor	Android	iOS	KMM - AS	KMM - Xcode
Gemiddelde	0,458	0,200	0,730	0,814
Mediaan	0,420	0,200	0,697	0,800
Standaarddeviatie	0,134	0,226	0,142	0,062
Variantie	0,018	0,051	0,020	0,004
Maximum	1,051	2,300	1,182	1,200
Minimum	0,258	0,100	0,449	0,800

Uit voorgaande gegevens kunnen we afleiden dat Android Studio de meest performante IDE is voor KMM en dat er dus geen tijdsvermindering door Xcode te gebruiken. Daarnaast valt ook op dat de IDE na de tweede build niet veel meer versnellen en de waarden stagneren.

Conclusie

Tabellen 3.4 en 3.5 tonen een overzicht van de resultaten in verband met de compileersnelheid bij vergelijking tussen native en KMM. De waarden voor native zijn bekomen door voor elke build iteratie de waarden op te tellen en nadien deze waarden te gebruiken voor de statistiek. Voor de waarden van KMM worden de waarden uit tabel 3.3 van ‘KMM - AS’ gebruikt.

Tabel 3.4: Resultaten van de eerste compilatie

Platform	Eerste compilatie
Native	32,431
KMM	5,873

Tabel 3.5: Resultaten van de compilatie twee tot honderd

Factor	Native	KMM - AS
Gemiddelde	0,658	0,730
Mediaan	0,607	0,697
Standaarddeviatie	0,305	0,142
Variantie	0,093	0,020

Hieronder wordt een overzicht getoond van de conclusies die getrokken kunnen worden uit de resultaten van de testen.

- Android Studio is de snelste IDE voor KMM, dit is ook de aanbevolen IDE
- KMM scoort significant beter op de eerste compilatie
- Native heeft een sneller gemiddelde na de eerste build.
- De waarden van Xcode zijn veel stabieler dan die van Android Studio

Enkele zaken rond de conclusies, KMM staat nog in alfa dit kan verklaren waarom de cache van de native applicaties beter is. De stabiliteit van Xcode kan te verklaren zijn door een afronding die Xcode intern doet op de compileertijden.

De conclusie van deze test is dat KMM sneller is dan de native varianten en dus de betere keuze is op vlak van compileertijden.

3.4.3 Voetafdruk

Een ander gegeven dat bekeken wordt is de voetafdruk van de verschillende applicaties. Hierbij is een lagere waarde te verkiezen, gezien de applicaties dezelfde functionaliteit hebben. Hogere waarden zouden wijzen op nutteloze gegevens/bestanden die mogelijk overbodig zijn in het project.

Android

Op figuur 3.18 wordt de voetafdruk voor de Android applicatie getoond. Deze werd via Android Studio werd geïnstalleerd op de emulator gekozen voor het project. Op een Android toestel is het mogelijk de persoonlijke gegevens en de cache van de applicatie apart te zien en te verwijderen. Het verwijderen van deze zaken had echter geen invloed op de grootte van de applicatie zelf. Op de figuren van figuur 3.18 kunnen de waarden teruggevonden worden.

iOS

De voetafdruk van een iOS applicatie meten is echter niet mogelijk via de emulators binnen Xcode. Om onderstaande resultaten te verzamelen werd gewerkt met een fysiek toestel namelijk de iPhone 11 Pro Max³⁵. De applicatie werd via Xcode op het toestel geïnstalleerd. Daarna kan via de instellingen van de iPhone de grootte van de applicatie bekijken worden.

KMM

Zoals reeds vermeld, zal de iOS applicatie geïnstalleerd worden op de fysieke iPhone 11 Pro Max en de Android applicatie op de emulator van binnen Android Studio.

Conclusie

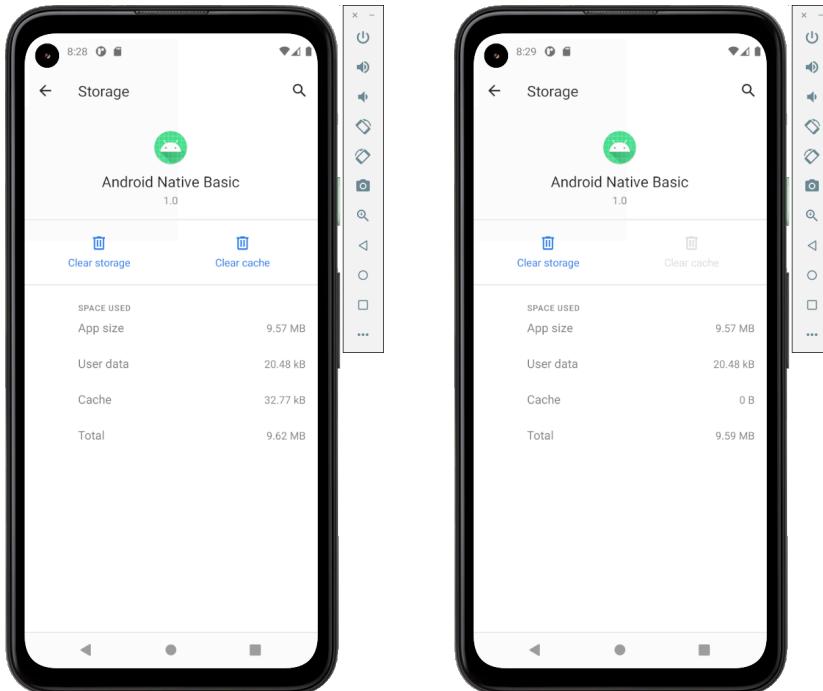
Bij evaluatie van de resultaten op vlak van voetafdruk, wordt gezien dat KMM Android iets beter scoort dan native, maar dat KMM iOS veel slechter scoort dan native iOS. Tabel 3.6 geeft een overzicht van de voetafdrukken. Om een ander beeld te krijgen, kunnen de waarden van native en KMM opgeteld worden. Hieruit kan dan wel geconcludeerd worden dat native net iets meer performant is op het vlak van de voetafdrukken op de toestellen.

Tabel 3.6: Overzicht van de voetafdruk van de applicaties

Platform	Native	KMM
Android	9,57 MB	7,92 MB
iOS	0,233 MB	2,1 MB
Totaal	9,803 MB	10,020 MB

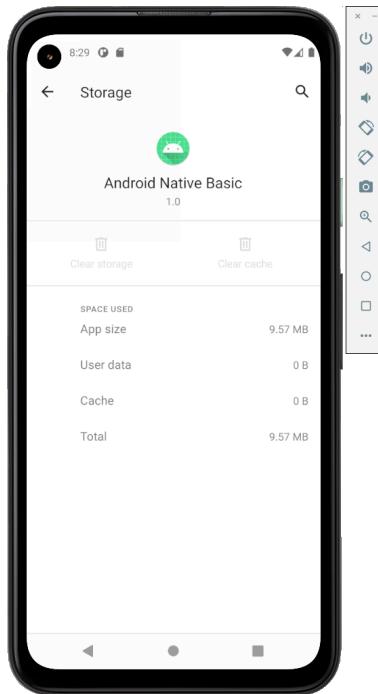
Deze resultaten en conclusie geven een indicatie over wat de verhouding is tussen native en KMM op vlak van de voetafdruk. De waarden zijn echter irrelevant gezien deze aan de hand van meerdere applicaties en over een langere periode dient getest te worden vooraleer hieruit een conclusie kan getrokken worden.

³⁵support.apple.com/kb/SP806



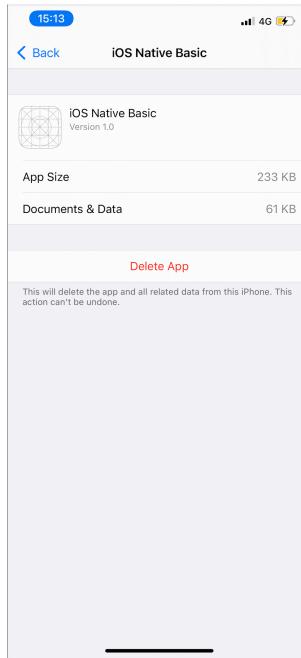
3.18.1: Voetafdruk van de native Android met cache en lokale opslag

3.18.2: Voetafdruk van de native Android zonder cache en met lokale opslag



3.18.3: Voetafdruk van de native Android zonder cache en lokale opslag

Figuur 3.18: Voetafdruk van de native Android applicatie



Figuur 3.19: Voetafdruk van de native iOS applicatie

3.4.4 Ontwikkeltijd

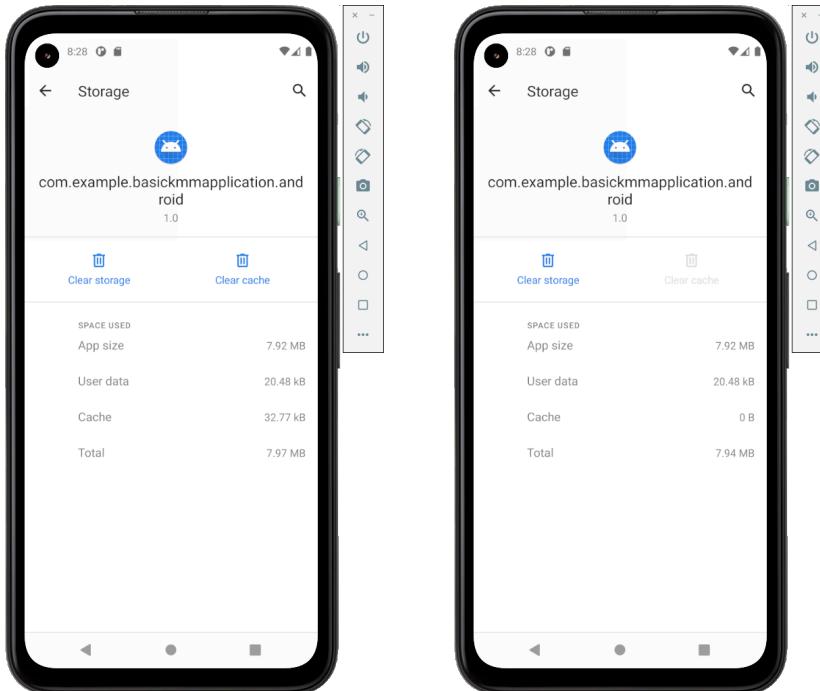
De ontwikkeltijd is een eerder subjectief gegeven, gezien dit zeer afhankelijk is van de persoon die deze test afneemt. Voor deze studie had de ontwikkelaar geen ervaring met KMM maar wel met Kotlin en Swift. Door de ontwikkelaar waren in het verleden reeds applicaties geschreven voor Android en iOS. De ontwikkeltijden zijn een schatting en omvatten het hele traject van opstart tot het builden van de laatste correcte versie. Deze waarden zijn gebaseerd op commit history en local history binnen de IDE's alsook time-tracking van de ontwikkelaar zelf.

Tabel 3.7: Overzicht van de ontwikkeltijd van de applicaties

Platform	Ontwikkeltijd
Android native	01u 32min
iOS native	01u 56min
KMM	3u 26min

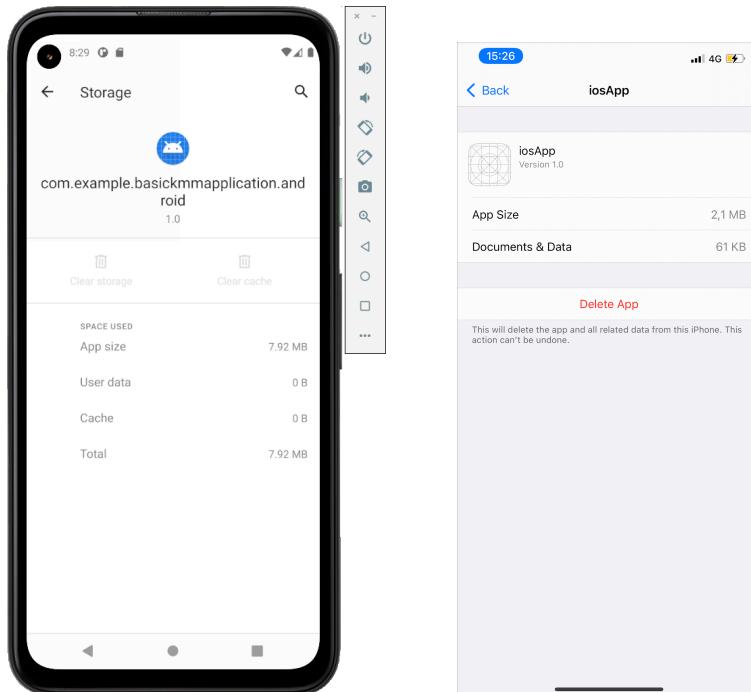
Aan de hand van de resultaten kan geconcludeerd worden dat het voor de ontwikkelaar in de studie net sneller was om een KMM applicatie te maken dan twee native varianten. Een van de belangrijkste zaken hierbij is het feit dat voor native twee projecten moeten opgestart worden. Daarnaast bevatte de KMM applicatie al wat logica op voorhand die kon hergebruikt worden.

Indien deze studie zou uitgevoerd worden door een meer ervaren ontwikkelaar die ervaring heeft met Kotlin, Swift en KMM kan verwacht worden dat de KMM applicatie



3.20.1: Voetafdruk van KMM
Android met cache en lokale opslag

3.20.2: Voetafdruk van KMM
Android zonder cache en met lokale
opslag



3.20.3: Voetafdruk van KMM
Android zonder cache en lokale
opslag

3.20.4: Voetafdruk van
KMM iOS kmm

Figuur 3.20: Voetafdruk van de KMM applicaties

nog sneller ontwikkeld kan worden. De gedeelde domeinlogica kan een heel stuk tijd besparen dit gegeven kwam in dit project ook naar boven kwam.

3.4.5 Kostprijs

Voor de kostprijs zal gebaseerd worden op gegevens die aangereikt werden door Endare, het ondersteunende bedrijf voor deze bachelorproef. Tabel 3.8 toont een overzicht van de geschatte kosten. De ontwikkelingstijd wordt gebruikt als basis voor de andere zaken als design, analyse... in te schatten. Deze inschatting gebeurd aan de hand van de ontwikkelingstijd te vermenigvuldigen met een factor van 1,7. Voor de prijs te berekenen werkt Endare met een uurtarief van € 85 excl. btw.

Tabel 3.8: Overzicht van de ontwikkeltijd van de applicaties

Platform	Ontwikkeltijd	Totale tijd	Prijs (excl. btw)
Android native	01u 32min	2u 36min	€ 221,57
iOS native	01u 56min	2u 17min	€ 279,37
KMM	3u 26min	5u 50min	€ 496,12

De totale kostprijs voor de twee native applicaties komt dus op € 500,93 en de KMM applicatie komt op € 496,12. Het kleine verschil tussen de resultaten is te verklaren door het feit dat gebruik gemaakt is van de resultaten van deel 3.4.4: ‘Ontwikkeltijd’. Daar werd op het einde ook gewezen op het feit dat bij meer ervaren developers een KMM applicatie interessanter zou zijn. Dit gegeven heeft ook rechtstreeks effect op de prijs, gezien een snellere ontwikkeltijd zorgt voor een lagere prijs.

Uiteindelijk zou ook met het kleine verschil nu de KMM variant goedkoper uitkomen gezien de factor 1,7 geen projectmanagement kosten bevat. Deze kosten zouden in het native verhaal dus twee keer dienen aangerekend te worden en bij de KMM variant maar één keer. Dit maakt dus dat de KMM applicaties goedkoper zal zijn dan de native applicaties.

4. Conclusie

Na hoofdstuk 3: ‘Methodologie’ kan nu overgegaan worden tot de interpretatie van de resultaten van de testen. Daarna worden de onderzoeksvragen geëvalueerd aan de hand van de resultaten.

4.1 Resultaten van de testen

Aan de hand van de resultaten van de testen uit hoofdstuk 3: ‘Methodologie’ worden volgende conclusies bekomen.

De eerste test binnen dit onderzoek was omtrent het aantal lijnen van de applicaties, hieruit kwamen volgende conclusies.

- Kotlin Multiplatform Mobile (KMM) scoort ongeveer 25% beter op vlak van het aantal lijnen code. Binnen dit onderzoek had het KMM project 1605 lijnen code en de native applicaties samen 2023.
- Het iOS project was groter dan het KMM project zowel in aantal lijnen code als in de totale omvang van het project.
- Het iOS project bevatte 420% meer lijnen code dan het Android project en was ook bijna 500% groter in omvang.

Hieruit kan besloten worden dat binnen deze studie KMM interessanter is als er gekeken wordt op basis van lijnen code en omvang. Daarnaast is hoogstwaarschijnlijk de grootte van het iOS deel de belangrijkste factor die de omvang en het aantal lijnen code zal bepalen.

De test in verband met de compileersnelheid is gebleken dat de eerste compilatie van KMM 550% sneller is dan de twee native applicaties samen. Op vlak van caching tussen de compilaties door hebben de native varianten schnellere compileertijden. Anderzijds is er ook geen tijdswinst te halen uit het compileren van KMM in Xcode, Android Studio blijft de beste IDE voor KMM.

Uit laatste technische test omtrent de voetafdruk kunnen volgende zaken geconcludeerd worden.

- De KMM applicatie had een kleinere voetafdruk op het Android device dan de native variant, KMM was 20% kleiner dan de native applicatie.
- de KMM applicatie had een grotere voetafdruk op het iOS device dan de native variant, de KMM applicatie was 900% groter dan KMM.

Indien er echter gekeken werd naar de totale voetafdruk van de twee native applicaties ten opzichte van de KMM applicaties kwam native hier als beste naar boven binnen deze studie. De twee native applicaties samen hadden een voetafdruk die 2% kleiner is dan de KMM applicaties. Dit geeft een inschatting en is geen relevantie informatie om een keuze op te baseren, hiervoor dienen meerdere testen gedaan te worden

De testen in verband met de ontwikkeltijd en de kostprijs waren echter zeer gerelateerd aan elkaar gezien de kostprijs berekend werd aan de hand van de ontwikkeltijd. Binnen deze studie duurde het maken van de KMM applicatie 3u 26min en het maken van de twee native applicaties duurde samen 3u 28min. Hierbij is echter een zeer klein verschil op te merken tussen KMM en native. Een interessant gegeven hierbij is het feit dat de developer geen ervaring had met KMM. Hieruit kan geconcludeerd worden dat een ervaren developer een groter verschil zal verkrijgen dat in het voordeel van KMM zal zijn.

Uit het feit dat native en KMM zeer dicht bij elkaar lagen van ontwikkelingstijd was er dus ook een klein verschil in kostprijs. Voor deze studie zou de kostprijs voor een KMM applicatie uitkomen op € 496,12 en voor de twee native applicaties zou het op 500,93 komen. Ook hier dient opgemerkt te worden dat met een meer ervaren developer KMM interessanter zou zijn op vlak van kostprijs. Daarnaast dient ook rekening gehouden te worden met het feit dat bij een KMM project maar eenmaal een projectmanagement kost dient aangerekend te worden en bij de native applicaties tweemaal.

4.2 Verder onderzoek

Na deze studie zijn er enkele zaken die verder zouden kunnen onderzocht worden, hieronder enkele van de mogelijke vervolgonderzoeken.

- Voor de test in verband met de voetafdruk zou een uitgebreidere studie kunnen gedaan die meerdere applicaties gaat maken over een langere periode. Op deze manier zou de voetafdruk van native ten opzichte van KMM beter in kaart kunnen gebracht worden.

- De huidige applicatie verder uitbreiden, aangezien dit een zeer ruim gegeven is worden al enkele mogelijkheden gegeven die als uitbreiding kunnen gezien worden.

- Toevoegen van tekstvelden en user interface elementen
 - Off-line data en de verwerking ervan
 - On-line data aan de hand van API-calls
 - Toevoegen van wiskundige berekeningen

Daarnaast kan uitbreiding ook gezien worden als het uitbreiden naar meerdere devices zoals Apple Watch¹, iPad², Android Auto³...

- Deze studie laten uitvoeren door een aantal developers met verschillende niveaus binnen het ontwikkelen met Kotlin, Swift en KMM. Aan de hand van deze test zou de impact van de developer op de testen kunnen in kaart gebracht worden.
- Kotlin Multiplatform Mobile vergelijken met andere cross-platform frameworks zoals Flutter⁴, React Native⁵... Aan de hand van deze test zal bepaald kunnen worden of KMM de meest interessante variant is tussen alle cross-platform oplossingen.

4.3 Conclusie

Binnen deze conclusie zullen antwoorden geformuleerd worden voor de onderzoeks vragen opgesteld voor deze studie. Hierbij een overzicht van de onderzoeks vragen voor deze studie. De antwoorden voor deze vragen houden rekening met het feit dat de native applicaties geschreven zijn met het oogpunt dat er twee varianten komen, één voor iOS en één voor Android.

- **Is native applicatieontwikkeling nog steeds de beste optie voor Android?**

Voor Android is de overstap naar KMM zeer interessant gezien er uit de testen is gebleken dat KMM applicaties een kleinere voetafdruk hebben, een snellere eerste compilatie en minder lijnen code hebben op Android devices dan de native variant.

- **Is native applicatieontwikkeling nog steeds de beste optie voor iOS?**

Voor iOS valt op dat de native applicatie veel kleiner is dan de native variant, daarnaast is de snelheid van de eerste compilatie en het aantal lijnen code in het voordeel van KMM.

- **Is KMM een sneller alternatief voor native applicaties?**

De eerste compilatie van KMM is sneller dan de 2 native applicaties samen, het is hier ook belangrijk om te weten dat bij het builden van de KMM direct iOS en Android gebuild worden. De build die na de eerste build komen en dus gebruik maken van de caching binnen de IDE zijn sneller voor de native variant.

- **Is KMM een efficiënter alternatief voor native applicaties?**

¹apple.com/benl/watch

²apple.com/benl/ipad

³android.com/auto

⁴flutter.dev

⁵reactnative.dev

De applicaties van KMM bevatten minder lijnen code dan de twee native varianten samen, daarnaast is de KMM applicatie zelfs efficienter dan enkel de iOS applicatie. Als de efficiëntie bekeken word op basis van de voetafdruk nemen de native applicaties hier de bovenhand.

- **Is het huidige beeld op de SWOT-analyse van cross-platform applicaties toepasbaar op KMM?**

Het huidige beeld van de SWOT-analyse voor cross-platform applicaties staat uitgeschreven in deel 2.2.2: ‘Cross-platform ontwikkeling’. De SWOT-analyse wordt besproken per peiler.

- Sterktes of Strengths

Deze studie heeft getoond dat de gegeven sterktes van cross-platform allemaal geldig zijn voor KMM. De applicaties kwamen sneller en goedkoper uit de test. Daarnaast is het logisch dat een cross-platform applicatie meer platformen ondersteunt dan een native applicatie. Text

- Zwaktes of Weaknesses

Binnen KMM is er nog steeds nood aan native code, dit werd gezien als een zwakte op maar dit kan ook gezien worden als een sterkte. De native code zorgt voor een meer native ervaring voor de eindgebruiker. Daarnaast kan door de native code beter gebruik gemaakt worden van de core librabies van het systeem. Een andere zwakte was het updaten van code of toestellen, tijdens het proces van de studie zijn er geen problemen opgedoken als er devices of code diende geüpdateerd te worden. Vanuit dat standpunt zou dus gezegd kunnen worden dat deze studie dit niet als een zwakte ziet.

- Kansen of Opportunities

De kansen die werden aangegeven blijken nog steeds toepasbaar voor KMM, de applicaties werden sneller ontwikkeld met KMM dan native en dit wetende dat de developer in kwestie geen ervaring had met KMM. In verband met het omvormen van een native applicatie naar KMM staat in de documentatie duidelijk beschreven hoe dit kan gedaan worden. Omvormen naar een KMM applicatie is voor zowel Android als iOS een mogelijkheid.

- Bedreigingen of Threats

Tijdens het proces van de studie heeft Apple software updates uitgebracht voor de iPhone, namelijk iOS 14.5 en 14.51. Deze hadden geen effect op de applicaties en deze konden nog steeds gebruikt worden. Ook de Android emulator heeft updates gekregen binnen het proces en ook hier waren geen problemen te ondervinden.

Nu de onderzoeksvragen beantwoord zijn kan overgegaan worden tot de finale conclusie van deze studie, namelijk is Kotlin Multiplatform Mobile een alternatief voor native applicaties. Binnen het opzicht van de studie en rekening houdend met de resultaten die verkregen zijn kan hierop een positief antwoord gegeven worden. KMM toont veel po-

tentieel om als cross-platform alternatief gebruikt te worden. Doordat enkel de business logica gedeeld wordt, behouden de applicaties hun native look en feel. Daarnaast is ook uit de testen gebleken dat KMM zeker niet moet onderdoen op vlak van prestaties. KMM zit op het einde van deze studie nog steeds in alfa waardoor deze resultaten niet meer relevant kunnen zijn naar de toekomst. Eens KMM uit alfa gaat zullen ook meer developers ervaring kunnen opdoen waardoor ze er sneller mee kunnen werken.

Indien bedrijven na het lezen van deze studie de overstap naar KMM zouden overwegen dienen zij vooral rekening te houden met de overschakeling periode en de leercurve voor de developers. Het grote voordeel hierbij is dat KMM geschreven word in Kotlin en dat daarnaast ook de native taal Swift gebruikt word. Deze talen zijn door de meeste developers die native ontwikkelen wel gekend. Eens bedrijven de stap zetten naar KMM kunnen zij zeker voordeel halen uit KMM. De testen hebben aangetoond dat KMM efficienter is op vlak van lijnen code, een snellere eerste compilatie heeft en geen significant grotere voetafdruk dan de native varianten. Daarnaast is het ontwikkelen van een KMM goedkoper gezien de snellere ontwikkeling en de lagere projectmanagement kosten.

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.

A.1 Introductie

De keuze voor cross platform applicaties wordt de dag van vandaag steeds aantrekkelijker. Dit kan onder andere verklaard worden door de financiële voordelen ten opzichte van native applicaties, meer specifiek op vlak van de impact op het ontwikkelingsbudget van bedrijven. Cross platform applicaties worden al heel lang gepromoot als zijnde 'beter' dan native applicaties. Deze cross platform applicaties zijn sneller te ontwikkelen en vereisen minder lijnen code. De slogan Write once, run anywhere van Sun Microsystems (Oracle, g.d.) wordt vaak gebruikt om het grote voordeel van cross platform applicaties aan te duiden, dit duidt op het idee dat grote delen van de code kunnen gebruikt worden voor verschillende besturingssystemen. De duurdere native applicaties hebben echter nog steeds een aantal voordelen ten opzichte van de cross platform applicaties, aangezien deze specifiek geschreven zijn voor het gekozen besturingssysteem zoals bijvoorbeeld voor Android of iOS. De meeste reeds bestaande cross-platform frameworks, zoals Flutter, delen de user interface van de applicatie over de verschillende besturingssystemen. Dit zorgt voor een applicatie die voor geen enkel van de gekozen besturingssysteem perfect zal zijn, aangezien hierbij dan geen gebruik kan gemaakt worden van de specifieke frameworks voor dat specifieke besturingssysteem. Kotlin Multiplatform Mobile (KMM) gooit het concept van cross platform over een andere boeg en werkt met een gedeelde businesslogica in plaats van een gedeelde user interface. De applicaties worden dus elk afgewerkt met een native user interface. De opzet van deze bachelorproef is na te gaan of native applicatieontwik-

keling voor Android en iOS apparaten nog steeds de beste keuze is of KMM een sneller en efficiënter alternatief kan vormen binnen het huidige beeld van applicatieontwikkeling. Om dit te onderzoeken zijn er bepaalde testcriteria opgesteld. Deze testcriteria zijn: het aantal lijnen code, de kostprijs, de ontwikkeltijd, de compileersnelheid, de voetafdruk en de uitbreidingsmogelijkheden van de applicaties. Aan de hand hiervan kan besloten worden of de KMM-applicaties een beter alternatief kunnen vormen voor native applicaties. Daarnaast zal ook de vraag gesteld worden of het huidige beeld van de SWOT-analyse voor cross-platform applicaties ook toepasbaar is voor KMM-applicaties, indien niet zal gekeken worden hoe deze dient aangepast te worden.

A.2 State-of-the-art

A.2.1 Kotlin

In juli 2011 kwam JetBrains, de maker van onder andere IntelliJ IDEA, naar buiten met een nieuw project genaamd Kotlin.(Jemerov, 2011) Kotlin is een algemene, cross-platform en statische taal met type-inferentie. (Oliveira e.a., 2020) Hierbij verwijst de term algemene naar het feit dat Kotlin is ontwikkeld is voor allerlei verschillende types van software en in verschillende situaties kan gebruikt worden. Cross-platform gaat over het gegeven dat software kan bestaan in verschillende versies op meer dan één platform. Ook is er vermeld dat Kotlin statische typering gebruikt als taal, dit komt neer op het feit dat Kotlin de types van de objecten zal controleren tijdens het compileren van de code en niet tijdens het uitvoeren. Andere talen die dezelfde strategie volgen zijn Java en C. Alsook zal Kotlin gebruik maken van type-inferentie, hiermee zal de taal zelf onderscheid kunnen maken tussen datatypes van bepaalde expressies.

A.2.2 Cross-platform en native

Allereerst moet de omschrijving van een platform besproken worden, een platform zal meestal een bepaalde programmeertaal, besturingssysteem of bepaalde hardware beschrijven. Dit kan ook een combinatie van meerdere zijn.(Bishop & Horspool, 2006) Hiervan zijn enkele voorbeelden Java SE 15, hierbij wordt de taal bedoeld als platform. Een voorbeeld van een platform als besturingssysteem is bijvoorbeeld Windows 10 of MacOS Big Sur. Een voorbeeld hardware als platform is bijvoorbeeld een Intel of AMD processor. Uiteindelijk kan er ook gezegd worden dat een computer met als besturingssysteem Windows 10 en een laatste generatie Intel processor ook een specifiek platform is.

Eens de term platform duidelijker is, kan er makkelijker omschreven worden wat precies een cross-platform applicatie is en wat een native applicatie is. Een cross-platform applicatie is zoals de naam het al zegt een applicatie die op meerdere platformen zal werken. De software zal dus licht verschillende versies hebben die op hun beurt op allerlei verschillende platformen zullen werken. Native applicaties daarentegen zullen zich toespitsen op één bepaald platform.

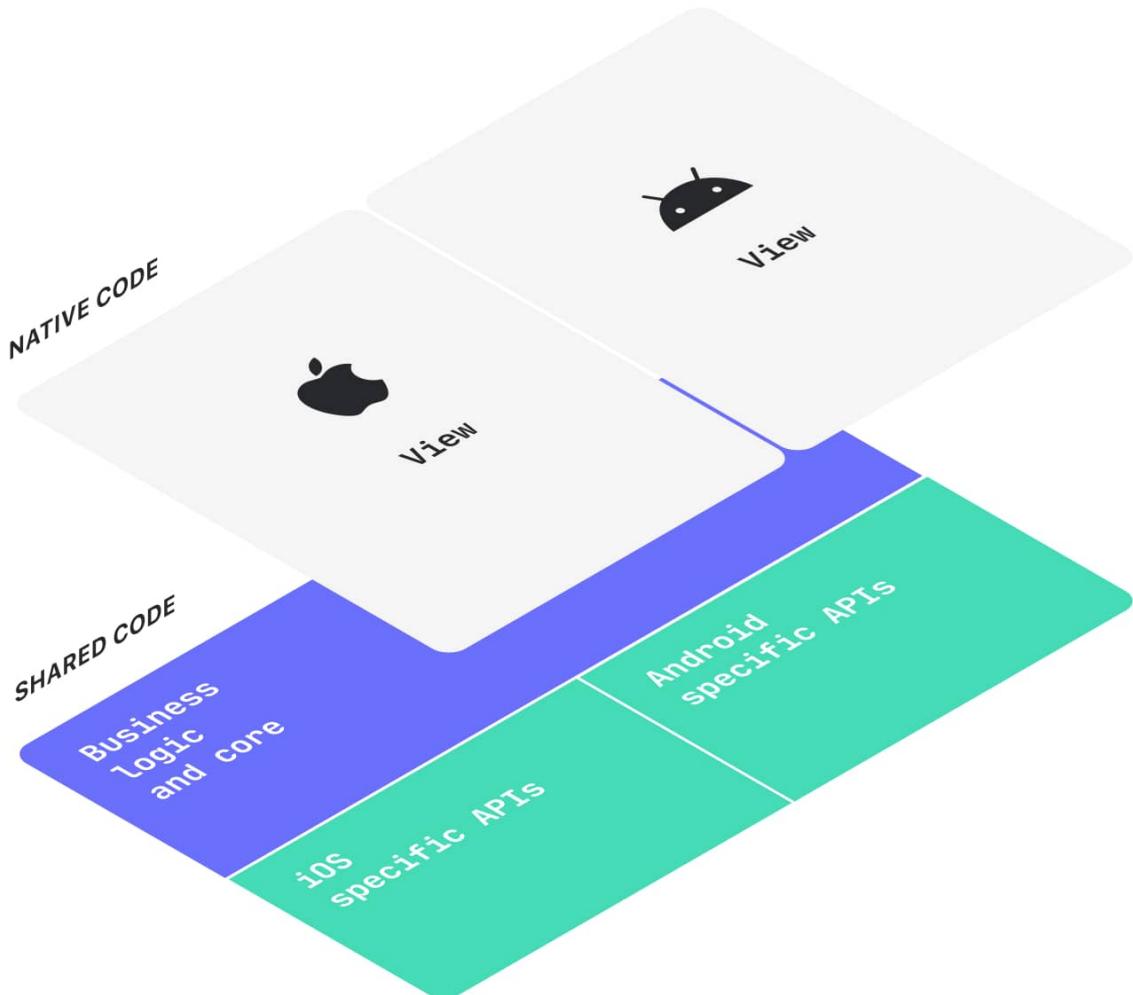
ā

Verder wordt er SWOT-analyse bekeken voor cross-platform applicaties. Deze sterkte-zwakteanalyse zal bekijken wat de sterkes en zwaktes zijn maar ook een beeld schetsen van de kansen en bedreigingen. Hiermee kunnen we een beeld schetsen wat de huidige positie is van cross-platform applicaties binnen de huidige markt. Uit onderzoek van Tommi Nivanaho naar cross-platform applicaties met React Native zijn een aantal factoren naar voren gekomen.(Nivanaho, 2019) Hier moet wel rekening gehouden worden met het feit dat sommige factoren niet toepasbaar zijn op alle cross-platform SDKs en toolkits. Hieronder een overzicht van enkele punten uit de studie toepasbaar zijn op cross-platform applicaties in het algemeen.

- Sterktes of Strengths
 - Sneller te ontwikkelen
 - Kostenbesparend
 - De applicatie ondersteunt meerdere platformen tegelijkertijd
- Zwaktes of Weaknesses
 - Nog steeds nood aan native code per platform
 - Upgrades kunnen omslachtiger zijn
- Kansen of Opportunities
 - Snelle ontwikkeling voor meerdere platformen tegelijkertijd
 - Native applicaties kunnen relatief vlot omgevormd worden naar cross-platform applicaties
- Bedreigingen of Threats
 - Updates aan het gekozen cross-platform systeem kunnen de reeds geschreven applicaties onbruikbaar maken

A.2.3 Kotlin Multiplatform

Kotlin Multiplatform is de nieuwe cross platform software development kit (SDK) van JetBrains. Ook werd een SDK uitgebracht specifiek gericht op de mobiele toestellen namelijk Kotlin Multiplatform Mobile (KMM). Kotlin Multiplatform werd voor het eerst verwerkt in Kotlin 1.2 in november 2017 als experimentele functie.(Jemerov, 2017) Ongetussen heeft de ontwikkeling van KMM niet stil gestaan en is in augustus 2020 de alpha versie van deze SDK uitgebracht voor het grote publiek.(Petrova, 2020) In november 2020 werd de 0.2.0 versie uitgebracht, deze is verwerkt in Kotlin 1.4.20.(JetBrains, 2020) Enkele delen van deze SDK en bijhorende componenten zijn nog steeds in de experimentele fase van ontwikkeling, maar JetBrains geeft aan dat het nu al een ideaal moment is om deze software te testen.(Petrova, 2020) De community achter KMM en het open source verhaal spelen hier een grote rol en zullen ook zorgen voor een snellere ontwikkeling van deze software. Bedrijven kunnen dus momenteel al experimenteel aan de slag met deze nieuwe software en voorbeelden van enkele bedrijven die de stap al hebben gezet naar KMM zijn onder andere Netflix, VMware en Autodesk.(Kotlin, 2020)



Figuur A.1: Grafische voorstelling Kotlin Multiplatform Mobile (Kotlin & JetBrains, g.d.)

A.2.4 Kotlin Multiplatform versus alternatieven

Kotlin Multiplatform (KM) zal het concept van cross-platform anders aanpakken dan andere alternatieven op de markt vandaag. Hierbij zit het verschil vooral in welke code van de applicatie zal gedeeld worden tussen de verschillende platformen. In de Kotlin documentatie staat beschreven hoe bepaalde delen van de code correct kunnen gedeeld worden tussen verschillende platformen.(Kotlin & JetBrains, 2021a) Hierbij wordt ook vermeld dat men enkel de business logica moet delen die gebruikt kan worden op alle platformen.

Figuur A.1 geeft een goed algemeen beeld van de structuur van Kotlin Multiplatform Mobile. De shared code in de figuur A.1 verwijst hier naar Common Kotlin of CommonMain, dit is het gedeelde binnenin KM dat de gedeelde logica zal bevatten. Daarnaast zal er nog per platform een aparte main zijn die de code zal implementeren. Hier in de figuur A.1 zal het project ook nog een iosMain en een kotlinMain, deze kunnen later ook nog uitgebreid worden met bijvoorbeeld een macosX64Main. De specifieke situatie in figuur A.1 is

een applicatie waar Kotlin Multiplatform Mobile gebruikt wordt. Deze zal zich speciaal richten op applicaties voor iOS en Android. Voor de communicatie tussen het common deel en de platform specifieke delen zal Kotlin gebruik maken van het expected/actual systeem. Hierbij worden binnendoor de CommonMain elementen gedeclareerd met expect en de platform specifieke delen zullen dezelfde elementen declareren met actual. Deze structuur kan gebruikt worden voor functies, klassen, interfaces, enumeraties, properties en annotaties.

Nu er een beter beeld is geschetst van hoe KM, werkt zal er gekeken worden naar een mogelijk alternatief met een andere aanpak voor de gedeelde code, bijvoorbeeld Flutter. Flutter is een user interface toolkit ontwikkeld door Google en zit momenteel aan versie 1.22 sinds oktober 2020.(Sells, 2020) De toolkit richt zich op mobile, web en desktop applicaties en zal de code native compileren. Zoals reeds beschreven is Flutter een user interface toolkit en zal dus de user interface delen over de verschillende platformen. Hier voor zal Flutter widgets gebruiken die geïnspireerd zijn door React.(Flutter, g.d.) Dit impliqueert dus dat hierbij de business logica zal moeten verwerkt worden per platform.

De developer kan dus kiezen om de user interface te delen onder de platformen en een toolkit te gebruiken zoals Flutter. Anderzijds kan er gekozen worden voor het delen van de business logica onder de platformen en dan kan KM gebruikt worden. Het delen van de user interface zal als voordeel hebben dat alle applicaties op de verschillende platformen dezelfde look en feel zullen hebben, echter is een nadeel dat de business logica per platform verwerkt zal moeten worden. Aan de kant van de gedeelde business logica is er het voordeel dat deze gedeeld is dus dat alle applicaties dezelfde logica hebben en implementeren, alsook kan er voor de user interface gebruik gemaakt worden van de platform specifieke frameworks. Er is echter ook een nadeel, hierbij kan gedacht worden aan het feit dat de user interface niet overal exact dezelfde zal zijn.

A.2.5 Testcriteria

Tijdens deze bachelorproef zullen van verschillende applicaties testcriteria geëvalueerd worden. Deze zullen ons vertellen in welke mate de cross-platform applicaties beter zijn dan de native. Volgende testcriteria werden gekozen voor deze vergelijkende studie.

- Aantal lijnen code
 - Het totaal aantal lijnen code van een applicatie zal geëvalueerd worden over het gehele project. In het geval van native applicaties zullen deze opgeteld worden met elkaar.
- Kostprijs
 - Hierbij wordt de geschatte kostprijs om een applicatie te laten ontwikkelen door een IT-bedrijf in kaart gebracht. Dit wordt berekend aan de hand van geschatte werkuren en een gemiddelde kostprijs per uur. Daarnaast wordt nagegaan of cross-platform een effect zal hebben op het systeem van vooraf bepaalde totaalprijzen indien bedrijven daarmee werken.
- Ontwikkeltijd
 - Deze tijd beschrijft het aantal werkuren dat een ontwikkelaar nodig heeft om

een specifieke applicatie te schrijven. Hierbij kan onder andere gebruik gemaakt worden van platformen zoals GitHub om deze tijd te gaan meten of inschatten.

- Compileersnelheid
 - Dit is de snelheid waarmee de specifieke applicatie zal kunnen compileren en opstarten. Dit kan gemeten worden in de ontwikkelingssoftware voor de desbetreffende taal van de applicatie.
- Voetafdruk
 - Dit impliceert de omvang die de applicatie zal innemen op het platform waarvoor deze ontwikkeld is. Hiervoor kan de applicatie gebruikt worden die de ontwikkelingssoftware aanmaakt.
- Uitbreidning van de applicatie
 - Dit criterium kan geëvalueerd worden door vooraf bepaalde features van de applicatie weg te laten. Eens de applicatie klaar is voor productie kunnen deze features terug toegevoegd worden. Om de uitbreidbaarheid van de applicatie te gaan staven kan gebruik gemaakt worden van voorgaande testcriteria.

A.3 Methodologie

Voor deze bachelorproef zullen enkele applicaties geschreven worden voor zowel Android als iOS. Eerst zullen er 2 native applicaties geschreven worden. De native iOS applicatie zal geschreven worden in Swift 5.3 of recent met iOS 14 in gedachten en voor de user interface zal er gebruik gemaakt worden van UIKit. Aan de Android kant van het native verhaal zal er een applicatie geschreven worden met behulp van Kotlin 1.4.20 of een recentere versie. Voor de native Android user interface zal gebruik gemaakt worden van de standaard Android en AndroidX bibliotheek. Naast de native applicaties zal er nog een derde applicatie geschreven worden, namelijk een Kotlin Multiplatform Mobile (KMM) applicatie. Deze applicatie zal werken op zowel Android als iOS toestellen. Hierbij zal voor het iOS-deel gebruik gemaakt worden van Swift en SwiftUI voor de user interface en voor het Android deel van Kotlin en Jetpack Compose.

Bij deze manier van werken is het belangrijk dat zowel de native applicaties als KMM dezelfde functionaliteiten hebben. Hierbij geldt ook dat Android en iOS dezelfde functionaliteiten zullen aanbieden aan de gebruikers. De geschreven applicaties zouden dus als het ware niet van elkaar te mogen onderscheiden zijn. Hierbij dient echter rekening gehouden te worden met het feit dat de user interface wel enigszins kan verschillen.

Voor de applicaties zelf zal er gekeken worden naar de sample applicaties die JetBrains aanbiedt en voorstelt. Dit zijn onder andere de PeopleInSpace applicatie (O'Reilly, 2021) en de SpaceX launches applicatie (Kotlin & JetBrains, 2020). Deze kunnen gebruikt worden om benchmarks op te testen en als inspiratiebron voor de native applicaties en andere KMM-applicaties.

Om de uiteindelijke vergelijking te maken tussen de native applicaties en de KMM-applicatie zal er gekeken worden naar verschillende factoren. Deze staan reeds beschreven in A.2.

State-of-the-art.

Voor dit onderzoek zullen volgende hardware en software gebruikt worden:

Hardware:

- MacBook Pro 16 inch uit 2019
- iPhone 11 Pro Max uit 2019
- Huawei P9 lite uit 2016

Deze laatste twee toestellen zullen echter minder van belang zijn en zullen enkel ter sprake komen indien er testen gebeuren op fysieke toestellen. Voor het grotere deel van de testen zullen emulators gebruikt worden die ingebouwd zijn in de gekozen ontwikkelingssoftware.

Software:

- Xcode versie 12.3 of recenter
- Android studio versie 4.1.1 of recenter

Voor de KMM-applicaties zal er echter nog de Kotlin Multiplatform Mobile plugin geïnstalleerd moeten worden.

Deze software zal gebruikt worden op de MacBook Pro die hierboven reeds werd beschreven. Voor dit onderzoek geldt de beperking dat er enkel gebruik gemaakt kan worden van toestellen die MacOS gebruiken als besturingssysteem, aangezien dit een vereiste is voor de ontwikkeling van iOS-applicaties.

A.4 Verwachte resultaten

Indien het onderzoek wordt uitgevoerd zoals beschreven in A.3.Methodologie, kunnen volgende resultaten voor de testcriteria uit A.2. State-of-the-art worden verwacht.

- Aantal lijnen code
 - Kotlin Multiplatform Mobile (KMM) zal hier een voordeel hebben in het aantal lijnen code gezien de business logica maar eenmaal moet geschreven worden. Dit is niet het geval bij de native applicaties, daar zal de business logica uitgewerkt worden voor zowel Android als iOS.
- Ontwikkeltijd
 - Aangezien de KMM-applicatie de gehele domein laag zal kunnen delen, wordt er verwacht dat deze ook sneller te ontwikkelen zal zijn dan de native varianten.
- Kostprijs
 - Verdergaand op de ontwikkeltijd kan hier ook gesteld worden dat KMM goedkoper zal zijn om te ontwikkelen. Aangezien er minder werkuren nodig zullen zijn kan ook het systeem van de totaalprijzen herzien worden. Hierdoor zullen applicaties met KMM goedkoper zijn voor de klant.

- Compileersnelheid
 - Gezien de complexe hiërarchie wordt verwacht dat de KMM-applicatie trager zal zijn dan de native applicaties. De native applicaties zijn echter ook beter geoptimaliseerd voor deze compilers en apparaten.
- Voetafdruk
 - Zoals reeds besproken zal ook de complexere hiërarchie hier ook een rol spelen. Er wordt verwacht dat de KMM-applicaties een grotere voetafdruk zullen hebben op de toestellen in kwestie, het verschil zal echter verwaarloosbaar klein zijn.
- Uitbreidning van de applicatie
 - De KMM-applicaties zullen hier ook een groter voordeel hebben omdat de nieuw geïmplementeerde features direct voor alle platformen zullen geïmplementeerd zijn. Aan de kant van de native applicaties kunnen bepaalde features vergeten of anders geïmplementeerd worden. Dit zorgt voor een verschil tussen de twee native applicaties.

A.5 Verwachte conclusies

Kotlin Multiplatform Mobile (KMM) is een zeer recente technologie die volop in ontwikkeling is. Daardoor wordt verwacht dat de resultaten op dit moment nog niet de volle kracht van deze techniek zullen aantonen. Dit wil echter niet zeggen dat de technologie geen grote meerwaarde kan bieden voor bedrijven die zich momenteel bezighouden met native applicatieontwikkeling voor zowel Android als iOS. Enkele componenten van de SDK staan nog niet op punt, dit zal waarschijnlijk nog verbeteren naar de toekomst toe. Het onderzoek kan, gezien de prematuriteit van KMM, een tijdelijke referentie bieden omtrent KMM en de mogelijkheid dit als een sneller en efficiënter alternatief te gebruiken voor native applicaties. Dit is vooral interessant voor bedrijven die zich momenteel vooral toespitsen op native ontwikkeling en eventueel naar de toekomst toe de overstap willen maken naar KMM.

B. Resultaten van compileersnelheid

B.1 Tabel

Tabel B.1: Compileersnelheid resultaten van native en KMM projecten

Build	Native iOS	Native Android	KMM in AS	KMM in Xcode
1	10,900	21,531	5,873	10,900
2	2,300	0,952	1,182	1,200
3	0,400	1,051	1,162	0,900
4	0,600	0,911	1,099	0,900
5	0,100	0,830	0,947	0,800
6	0,100	0,823	1,041	0,800
7	0,100	0,716	0,941	0,800
8	0,100	0,785	1,045	0,800
9	0,200	0,615	1,048	0,800
10	0,100	0,631	0,944	0,800
11	0,100	0,604	0,936	0,800
12	0,100	0,502	0,918	0,800
13	0,100	0,467	0,867	0,800
14	0,100	0,303	0,882	0,800
15	0,200	0,444	0,927	0,800
16	0,100	0,667	0,942	0,800
17	0,100	0,420	0,842	0,800
18	0,100	0,420	0,841	0,800
19	0,100	0,498	0,848	0,800
20	0,100	0,478	0,864	0,800

Vervolg op volgende pagina

Tabel B.1 – continued from previous page

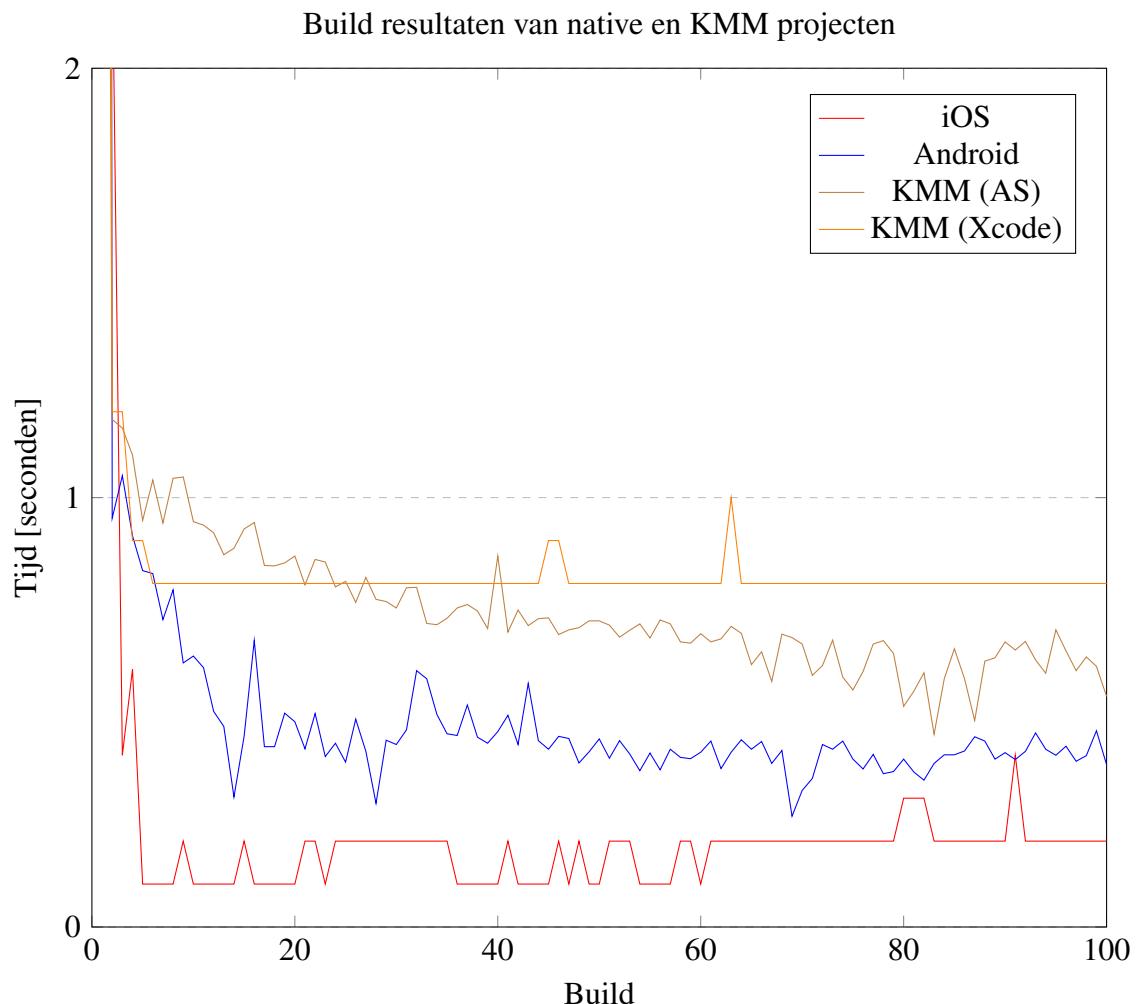
Build	Native iOS	Native Android	KMM in AS	KMM in Xcode
21	0,200	0,415	0,797	0,800
22	0,200	0,497	0,856	0,800
23	0,100	0,397	0,850	0,800
24	0,200	0,428	0,792	0,800
25	0,200	0,384	0,805	0,800
26	0,200	0,484	0,756	0,800
27	0,200	0,409	0,814	0,800
28	0,200	0,288	0,763	0,800
29	0,200	0,435	0,758	0,800
30	0,200	0,425	0,743	0,800
31	0,200	0,459	0,790	0,800
32	0,200	0,597	0,791	0,800
33	0,200	0,578	0,707	0,800
34	0,200	0,495	0,704	0,800
35	0,200	0,450	0,719	0,800
36	0,100	0,446	0,743	0,800
37	0,100	0,517	0,751	0,800
38	0,100	0,442	0,736	0,800
39	0,100	0,428	0,695	0,800
40	0,100	0,455	0,864	0,800
41	0,200	0,493	0,687	0,800
42	0,100	0,425	0,738	0,800
43	0,100	0,567	0,702	0,800
44	0,100	0,434	0,718	0,800
45	0,100	0,414	0,720	0,900
46	0,200	0,444	0,681	0,900
47	0,100	0,439	0,692	0,800
48	0,200	0,382	0,697	0,800
49	0,100	0,408	0,713	0,800
50	0,100	0,438	0,713	0,800
51	0,200	0,393	0,703	0,800
52	0,200	0,434	0,675	0,800
53	0,200	0,403	0,691	0,800
54	0,100	0,364	0,706	0,800
55	0,100	0,405	0,673	0,800
56	0,100	0,366	0,715	0,800
57	0,100	0,414	0,706	0,800
58	0,200	0,395	0,664	0,800
59	0,200	0,392	0,661	0,800
60	0,100	0,407	0,683	0,800
61	0,200	0,433	0,664	0,800
62	0,200	0,369	0,671	0,800
63	0,200	0,407	0,700	1,000

Vervolg op volgende pagina

Tabel B.1 – continued from previous page

Build	Native iOS	Native Android	KMM in AS	KMM in Xcode
64	0,200	0,436	0,684	0,800
65	0,200	0,414	0,611	0,800
66	0,200	0,432	0,641	0,800
67	0,200	0,381	0,572	0,800
68	0,200	0,411	0,682	0,800
69	0,200	0,258	0,674	0,800
70	0,200	0,318	0,659	0,800
71	0,200	0,346	0,586	0,800
72	0,200	0,425	0,609	0,800
73	0,200	0,414	0,668	0,800
74	0,200	0,433	0,582	0,800
75	0,200	0,391	0,552	0,800
76	0,200	0,368	0,595	0,800
77	0,200	0,402	0,659	0,800
78	0,200	0,357	0,667	0,800
79	0,200	0,362	0,637	0,800
80	0,300	0,391	0,514	0,800
81	0,300	0,361	0,549	0,800
82	0,300	0,342	0,592	0,800
83	0,200	0,381	0,449	0,800
84	0,200	0,401	0,578	0,800
85	0,200	0,401	0,648	0,800
86	0,200	0,410	0,578	0,800
87	0,200	0,443	0,481	0,800
88	0,200	0,433	0,619	0,800
89	0,200	0,391	0,627	0,800
90	0,200	0,406	0,664	0,800
91	0,400	0,390	0,645	0,800
92	0,200	0,409	0,665	0,800
93	0,200	0,452	0,622	0,800
94	0,200	0,414	0,591	0,800
95	0,200	0,400	0,692	0,800
96	0,200	0,421	0,643	0,800
97	0,200	0,386	0,597	0,800
98	0,200	0,399	0,629	0,800
99	0,200	0,457	0,607	0,800
100	0,200	0,378	0,537	0,800

B.2 grafisch



C. Resultaten van aantal lijnen code

C.1 Aantal lijnen code - Android

C.2 Aantal lijnen code - iOS

Extentie	Aantal bestanden			Totale grote	Kleinste bestand	Grootste bestand	Gemiddelde grootte	Aantal lijnen	Kleinste aantal lijnen	Grootste aantal lijnen	Gemiddeld aantal lijnen	Lijnen code
json	3	1777	63	1591	592	115	6	98	38	115		
pbxproj	1	22972	22972	22972	22972	594	594	594	594	594	554	
plist	5	4110	238	2067	822	132	8	66	26	132		
storyboard	2	4994	1665	3329	2497	72	25	47	36	70		
swift	5	6415	372	2300	1283	185	22	52	37	70		
xcusersstate	1	50246	50246	50246	50246	752	752	752	752	752	686	
xcworkspacestate	1	135	135	135	135	7	7	7	7	7	7	
Totaal	18	90649	75691	82640	78547	1857	1414	1616	1490	1634		

C.3 Aantal lijnen code - KMM

Bibliografie

- Bishop, J. & Horspool, N. (2006). Cross-Platform Development: Software that Lasts. *Computer*, 39(10), 26–35. <https://doi.org/10.1109/MC.2006.337>
- Evert, A.-K. (2019). *Cross-Platform Smartphone Application Development with Kotlin Multiplatform : Possible Impacts on Development Productivity, Application Size and Startup Time* (masterscriptie). KTH, School of Electrical Engineering en Computer Science (EECS). <https://www.diva-portal.org/smash/get/diva2:1368323/FULLTEXT01.pdf>
- Flutter. (g.d.). *Introduction to widgets* (Flutter, Red.). Verkregen 11 februari 2021, van <https://flutter.dev/docs/development/ui/widgets-intro>
- GitHub. (2018). *The State of the Octoverse* (GitHub, Red.). Verkregen 11 februari 2021, van <https://octoverse.github.com/2018>
- Jemerov, D. (2011, juli 19). *Hello World* (D. Jemerov, Red.). Verkregen 11 februari 2021, van <https://blog.jetbrains.com/kotlin/2011/07/hello-world-2/>
- Jemerov, D. (2017, november 28). *Kotlin 1.2 Released: Sharing Code between Platforms* (D. Jemerov, Red.). Verkregen 11 februari 2021, van <https://blog.jetbrains.com/kotlin/2017/11/kotlin-1-2-released/>
- JetBrains. (2020, november 23). *KMM plugin releases* (JetBrains, Red.). Verkregen 11 februari 2021, van <https://kotlinlang.org/docs/mobile/kmm-plugin-releases.html#update-to-the-new-release>
- JetBrains. (2021). *Corporate overview* (JetBrains, Red.). Verkregen 27 maart 2021, van https://resources.jetbrains.com/storage/products/jetbrains/docs/corporate-overview/en-us/jetbrains_corporate_overview.pdf
- Kotlin. (2020). *Case Studies* (Kotlin, Red.). Verkregen 11 februari 2021, van <https://kotlinlang.org/lp/mobile/case-studies/>
- Kotlin & JetBrains. (g.d.). *Kotlin Multiplatform Mobile* (Kotlin & JetBrains, Red.). Verkregen 11 februari 2021, van <https://kotlinlang.org/lp/mobile/>

- Kotlin & JetBrains. (2020, november 18). *Kotlin Multiplatform Hands-on: Networking and Data Storage* (Kotlin & JetBrains, Red.). Verkregen 11 februari 2021, van <https://github.com/kotlin-hands-on/kmm-networking-and-data-storage/tree/final>
- Kotlin & JetBrains. (2021a, februari 3). *Kotlin Multiplatform: Share code on platforms* (Kotlin & JetBrains, Red.). Verkregen 11 februari 2021, van <https://kotlinlang.org/docs/reference/mpp-share-on-platforms.html#share-code-on-similar-platforms>
- Kotlin & JetBrains. (2021b, februari 11). *Multiplatform programming* (Kotlin & JetBrains, Red.). Verkregen 2 april 2021, van <https://kotlinlang.org/docs/multiplatform.html>
- Leigh, D. (2010, februari). *SWOT Analysis*. John Wiley & Sons, Inc. <https://doi.org/10.1002/9780470592663.ch24>
- Lim, S.-H. (2015). Experimental Comparison of Hybrid and Native Applications for Mobile Systems. *International Journal of Multimedia and Ubiquitous Engineering*, 10(3), 1–12. <https://doi.org/http://dx.doi.org/10.14257/ijmue.2015.10.3.01>
- Meijer, E. & Drayton, P. (2004). Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages, In *Intended for submission to the Revival of Dynamic Languages*. Citeseer. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.394.3818&rep=rep1&type=pdf>
- Nivanaho, T. (2019, mei 27). *Developing a cross-platform mobile application with React Native* (masterscriptie). Lappeenranta-Lahti University of Technology LUT. <http://lutpub.lut.fi/handle/10024/159507>
- Oliveira, V., Teixeira, L. & Ebert, F. (2020, februari 18). On the Adoption of Kotlin on Android Development: A Triangulation Study, In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, London, ON, Canada. <https://doi.org/10.1109/SANER48275.2020.9054859>
- Oracle. (g.d.). *Moved by Java Timeline* (Oracle, Red.). Verkregen 11 februari 2021, van <https://www.oracle.com/java/moved-by-java/timeline/>
- O'Reilly, J. (2021, februari 5). *PeopleInSpace* (J. O'Reilly, Red.). Verkregen 11 februari 2021, van <https://github.com/joreilly/PeopleInSpace>
- Osmani, A. (2015, december). *Getting Started with Progressive Web Apps* (A. Osmani, Red.). Google. Verkregen 1 mei 2021, van <https://developers.google.com/web/updates/2015/12/getting-started-pwa>
- Petrova, E. (2020, augustus 31). *Kotlin Multiplatform Mobile Goes Alpha* (E. Petrova, Red.). Verkregen 11 februari 2021, van <https://blog.jetbrains.com/kotlin/2020/08/kotlin-multiplatform-mobile-goes-alpha/>
- Rahul Raj, C. P. & Seshu Babu Tolety. (2012, december). A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach, In *2012 Annual IEEE India Conference (INDICON)*, IEEE. <https://doi.org/10.1109/INDCON.2012.6420693>
- Sells, C. (2020, oktober 1). *Announcing Flutter 1.22*. Verkregen 11 februari 2021, van <https://medium.com/flutter/announcing-flutter-1-22-44f146009e5f>
- Shafirov, M. (2017, mei 17). *Kotlin on Android. Now official* (M. Shafirov, Red.). Verkregen 11 februari 2021, van <https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/>
- Skeen, J. & Greenhalgh, D. (2018). *Kotlin Programming: The Big Nerd Ranch Guide*. Pearson Technology Group. <https://books.google.be/books?hl=nl&lr=&id=Ol9qDwAAQBAJ&oi=fnd&pg=PT14&dq=General-purpose%20programming%>

- 20language % 20kotlin & ots = iivTPTKZ50 & sig = Der5JeIYFyHVC2JD6xCIx - gR0Mc & redir_esc = y # v = onepage & q = General - purpose % 20programming % 20language%20kotlin&f=false
- Xanthopoulos, S. & Xinogalos, S. (2013, september). A Comparative Analysis of Cross-Platform Development Approaches for Mobile Applications, In *Proceedings of the 6th Balkan Conference in Informatics*, Thessaloniki, Greece, Association for Computing Machinery. <https://doi.org/10.1145/2490257.2490292>