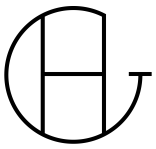


HoGent

Faculteit Bedrijf en Organisatie

Kotlin/Native, het nieuwe cross-platform framework voor de mobiele omgeving

Ilias Van Wassenhove

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

Promotor:
Johan Van Schoor
Co-promotor:
Sander Goossens

Instelling: Endare

Academiejaar: 2017-2018

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Kotlin/Native, het nieuwe cross-platform framework voor de mobiele omgeving

Ilias Van Wassenhove

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

Promotor:
Johan Van Schoor
Co-promotor:
Sander Goossens

Instelling: Endare

Academiejaar: 2017-2018

Tweede examenperiode

Woord vooraf

Graag neem ik even de tijd om mijn onderwerpkeuze te motiveren.

Ik heb dit onderwerp gekozen omdat ik een grote passie heb voor programmeren. Tijdens het eerste semester van het derde academiejaar toegepaste informatica heb ik kennis mogen maken met Android en iOS, het ontwikkelen van mobiele (native) applicaties. Hierbij heb ik ontdekt dat dit de richting is waarin ik mij wens te specialiseren, namelijk mobiele applicaties. Tijdens de lessen Android werd er af en toe eens gediscussieerd waarom we nog steeds Java gebruikten in plaats van Kotlin. Ik had persoonlijk nog niet veel van Kotlin gehoord maar toen is mijn interesse ontstaan. Ik ben dan onmiddellijk begonnen met het opzoeken van informatie over deze programmeertaal. Hierbij ontdekte ik dat reeds heel wat bekende applicaties Kotlin gebruikten, voor de ontwikkeling van hun Android-applicatie. Een aantal voorbeelden hiervan zijn Trello, Pinterest en Evernote. Hieruit kon ik constateren dat Kotlin reeds sterk gebruikt werd. De verschillende onderzoeken over Kotlin op het internet, waarin Kotlin vaak als een innovatieve taal werd voorgesteld, hebben er alleen voor gezorgd dat mijn interesse in deze programmeertaal groeide.

Toen het tijd was om een bachelorproefvoorstel in te sturen wou ik zeker en vast iets onderzoeken wat te maken heeft met mobiele applicaties. Niet veel later stuurde mijn stagebedrijf een bachelorproefonderwerp door over Kotlin en cross-platform applicaties. De keuze was snel gemaakt aangezien ik reeds een grote interesse in Kotlin had.

Ik heb door deze bachelorproef kennis gemaakt met een nieuw cross-platform framework en ik heb Kotlin leren gebruiken voor Android development. Persoonlijk was dit een zeer interessant onderwerp om te onderzoeken en heb ik met veel plezier aan dit onderzoek gewerkt.

Tenslotte wil ik nog even de tijd nemen om enkele personen te bedanken.

Ten eerste wil ik mijn promotor, Johan Van Schoor, bedanken voor de hulp, raad en opbouwende commentaar die ik gekregen heb bij het opstellen van deze bachelorproef. Mede dankzij zijn begeleiding ben ik tot een mooi resultaat gekomen.

Ten tweede zou ik graag mijn co-promotor, Sander Goossens, willen bedanken, wie ook mijn stagementor was. Hij stond altijd paraat om mijn vragen te beantwoorden en indien ik hulp nodig had was het nooit een probleem om even uitleg te geven.

Tenslotte zou ik iedereen waarmee ik heb samengewerkt willen bedanken voor de vlotte samenwerking. Er werd steeds samengewerkt op een zeer toffe en ontspannen manier waarbij iedereen gerespecteerd werd.

Hierdoor kan ik met een voldaan gevoel deze bachelorproef afgeven. Ik wens u veel plezier tijdens het lezen van mijn werkstuk.

Ilias Van Wassenhove

Samenvatting

Nog niet zo lang geleden had men bij het bouwen van mobiele applicaties (Android, iOS en Windows) enkel en alleen de mogelijkheid om drie aparte applicaties te bouwen, namelijk native applicaties. Dit vergde veel werk en was heel kostelijk voor veel bedrijven. Enerzijds is er per platform de tijd en kost om de applicatie te ontwikkelen, anderzijds is er de kost om deze applicaties uit te breiden of te onderhouden. Als reactie hierop zijn de cross-platform frameworks uitgevonden, denk maar aan: React, Xamarin en Ionic met Angular. Bij deze frameworks moet er maar éénmalig code geschreven worden, de user interface is hetzelfde voor alle platformen en het framework zorgt ervoor dat de applicatie op elk besturingssysteem kan draaien. Een kleine nuance, origineel had Xamarin niet de mogelijkheid om één user interface te ontwikkelen die hetzelfde was voor alle platformen. Er mag misschien nog een framework toegevoegd worden aan het lijstje van frameworks om cross-platform applicaties te ontwikkelen, namelijk Kotlin/Native.

Kotlin is een nieuwe programmeertaal die geïntroduceerd werd in 2011 door JetBrains. Origineel is het een programmeertaal die draait op de JVM[1], net zoals Java. Maar JetBrains is veel verder gegaan dan toestellen die een JVM kunnen draaien. Met Kotlin/Native hebben ze zich gericht tot alle platformen en besturingssystemen.

Maar waarom is het nu belangrijk om Kotlin/Native gedetailleerd te gaan bestuderen? Kotlin/Native is nog heel jong en er zijn al heel wat ontwikkelaars die reeds Kotlin gebruiken voor verschillende doeleinden. Deze ontwikkelaars staan te springen om aan de slag te gaan met Kotlin/Native. Met Kotlin spreken we over de programmeertaal, terwijl met Kotlin/Native het framework voor cross-platform applicatieontwikkeling bedoeld wordt dat gebruik maakt van de programmeertaal Kotlin, zie sectie 2.6 voor verdere uitleg. De enige beperking die zij momenteel ervaren is documentatie. Van dit onderzoek wordt

[1] Java Virtual Machine

verwacht dat door het gedetailleerd bestuderen van Kotlin/Native een mooie documentatie kan opgeleverd worden over de werking van dit framework waar iedere ontwikkelaar mee aan de slag kan gaan.

In dit onderzoek werd de werking van de LLVM compiler en Kotlin/Native onderzocht. Voor dit onderzoek is het belangrijk om de LLVM te onderzoeken aangezien deze ervoor zorgt dat Kotlin omgezet wordt naar platformonafhankelijke uitvoerbare bestanden en dus Kotlin op alle platformen kan gebruikt worden. Daarna werd er aan de hand van het onderzoek rond Kotlin/Native een kleine proof-of-concept opgesteld waarbij duidelijk moest worden wat de huidige mogelijkheden en beperkingen zijn van dit framework.

Het resultaat van dit onderzoek is enerzijds een documentatie over de werking van de LLVM compiler en Kotlin/Native, anderzijds een uitgewerkt proof-of-concept aan de hand van dit framework.

De conclusie die uit dit onderzoek kan worden getrokken is dat Kotlin/Native nog heel jong is en de mogelijkheden nog zeer beperkt zijn. Momenteel is JetBrains bezig met het verder uitwerken van versie 0.6 en er is eigenlijk nog geen release versie beschikbaar. Het is wel reeds mogelijk om aan de slag te gaan met dit framework maar het opzetten van een project is niet gemakkelijk. Dit aangezien er momenteel geen plugin bestaat om te installeren in een IDE zoals IntelliJ en er is geen officiële documentatie beschikbaar over de opzet van een Kotlin/Native project. Maar aan de hand van de proof-of-concept kan besloten worden dat Kotlin/Native momenteel wel reeds kan worden gebruikt om domeinlogica te delen over verschillende platformen. Het is dus mogelijk om domeinlogica, die geschreven is in Kotlin, te gebruiken voor iOS.

Dit framework heeft zeker en vast zijn troeven. Applicaties die een zeer uitgebreide domeinlogica hebben zoals bijvoorbeeld een webshop, hebben zeker voordeel indien deze applicaties Kotlin/Native gebruiken. De domeinlogica kan gedeeld worden over de verschillende platformen, waardoor deze maar één maal moet worden geschreven. De user interface kan per platform opgebouwd worden.

Er wordt sterk gewerkt aan Kotlin/Native en dat zal in de toekomst niet veranderen. Er is nog veel werk aan de winkel, de opzet van een project is momenteel heel omslachtig en hiervoor zal JetBrains toch een oplossing moeten vinden. De functionaliteiten zullen ook verder uitgebreid moeten worden. Zal er bijvoorbeeld in de toekomst de mogelijkheid zijn om via Kotlin de camera te openen, waardoor dit niet specifiek per platform moet worden geprogrammeerd?

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

1. Inleiding

De ontwikkeling van mobiele applicaties is een belangrijke niche in de IT. Zo kunnen we drie verschillende besturingssystemen onderscheiden bij mobiele applicaties: Android, iOS en Windows. Die laatste wordt steeds minder en minder gebruikt voor mobiele toepassingen. Volgens het artikel „Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2017" (2018), van Statista, een portaal voor statistieken en onderzoek, heeft Android een marktaandeel van 87.7% wereldwijd. Dit wil dus zeggen dat meer dan 85% van de mensen die een smartphone heeft, een toestel heeft met het Android besturingssysteem.

De mobiele applicaties die draaien op Android werden allemaal geprogrammeerd in Java. Maar sinds kort is er een nieuwe mogelijkheid, Kotlin. Kotlin is een programmeertaal die ontworpen is door JetBrains in 2011. Zes jaar later (2017) biedt Google volledige ondersteuning om Android-applicaties te programmeren in Kotlin. Kotlin zou initieel ontwikkeld zijn om de productiviteit van JetBrains te vergroten. Zie sectie 2.10 voor meer uitleg.

1.1 Begrippen

1.1.1 Wat is cross-platform?

Wat is de betekenis van cross-platform? Gaat dit over het delen van user interfaces, het delen van domeinlogica, of gaat dit over beiden? Techopedia (2017), een website voor IT professionals en geeks, heeft op hun website een verklaring voor een cross-platform product. Zij verklaren dat een cross-platform product een platformonafhankelijk computerproduct of

systeem is dat op meerdere platformen of besturingssystemen kan werken. Zij geven echter ook aan dat wanneer bepaalde zaak van de applicatie hergebruikt worden per platform, er gedaan wordt aan cross-platform development. Zowel het delen van domeinlogica, het delen van interfaces of beiden is cross-platform ontwikkeling. Cross-platform is echter ook bekend als multiplatform of platformonafhankelijk.

1.1.2 Wat is een framework?

Een framework kan in het algemeen beschreven worden als een soort van structuur die bedoeld is als ondersteuning bij het bouwen van een bepaald iets. Het is goed te vergelijken met een skelet waarop iets wordt gebouwd en het biedt functionaliteiten aan om te focussen op bepaalde specifieke taken.

Frameworks worden sterk gebruikt bij het ontwikkelen van software en vooral in de wereld van de mobiele applicaties. Momenteel bestaan er verschillende frameworks waarmee mobiele applicaties kunnen worden gebouwd. Twee voorbeelden hiervan zijn Cordova en PhoneGap (TechTarget, 2015).

1.1.3 Wat is een native applicatie?

Een native applicatie is een softwareprogramma dat ontwikkeld is om gebruik te worden op een specifiek platform of besturingssysteem. Aangezien een native applicatie gebouwd is om op een bepaalde platform te worden gebruikt, kan het gebruik maken van de hardware en software van het toestel. Native applicaties kunnen geoptimaliseerde prestaties bieden en hebben het voordeel om gebruik te kunnen maken van de laatst nieuwe technologieën zoals GPS, wat veel moeilijker is bij bijvoorbeeld een webapplicatie (TechTarget, 2018).

1.1.4 Wat is een hybride applicatie?

Een hybride applicatie is een applicatie die de features van enerzijds een native applicatie en anderzijds een webapplicatie combineert. Hybride applicaties worden dus niet ontwikkeld met de bedoeling om gebruik te worden op een bepaald platform of besturingssysteem. De bedoeling van hybride applicaties is om de mogelijk aan te bieden om de applicatie op eender welk besturingssysteem te gebruiken. Hybride applicaties maken gebruik van een ingebouwde browser om de user interface te weer te geven. Bij hybride applicatieontwikkeling wordt er gebruik gemaakt van Javascript, HTML en CSS. Enkele voorbeelden van dit type framework zijn Cordova, Ionic en Trigger.IO (TechTarget, 2011).

1.1.5 Wat is een cross-platform applicatie?

Cross-platform applicaties zijn gelijkaardig aan hybride applicaties. Het grote verschil tussen beide types is dat cross-platform applicaties geen gebruik maken van een ingebouwde browser om de user interface te tonen aan de gebruiker. De user interface wordt getoond

aan de hand van native elementen. Javascript wordt gebruikt om de functionaliteiten in de applicatie op te bouwen. HTML en CSS zorgen voor de opbouw van de user interface. Enkele voorbeelden van dit type framework zijn Xamarin, React/Native en NativeScript (Newgenapps, 2018).

1.1.6 Native versus cross-platform- en hybride development

Voor het ontwikkelen van mobiele applicaties bestaan er dus verschillende methodes. Het wel of niet kiezen voor native applicatieontwikkeling is afhankelijk van heel wat criteria. Denk maar aan de platformen dat de applicatie moet ondersteunen, de eisen waaraan een applicatie moet voldoen, de verschillende functionaliteiten en ga zo maar door. Iedere methodiek voor het ontwikkelen van applicaties heeft zo zijn eigen voor- en nadelen (Newgenapps (2018), InOutput (2017)).

Native ontwikkeling

Voordelen:

- Groot aantal functionaliteiten aangezien er gebruik kan worden gemaakt van alle mogelijkheden van het gebruikte toestel.
- Snelle software prestaties.
- Een user interface die overeenkomt met de gebruikerservaringen van het besturingssysteem.

Nadelen:

- Indien er meerdere platformen ondersteund moeten worden, moeten er twee aparte applicaties ontwikkeld worden.
- Kosten om meerdere platformen te beheren.
- Er kan geen code gedeeld worden tussen verschillende platformen.

Cross-platform- en hybride ontwikkeling

Voordelen:

- Applicatie kan sneller ontwikkeld worden.
- Verschillende frameworks die helpen bij de ontwikkeling.
- Het delen van code over de verschillende platformen.

Nadelen:

- Niet alle code kan gedeeld worden over de verschillende platformen. Soms moet er native code geschreven worden.
- Toegang tot het toestel en besturingssysteem zijn afhankelijk van de mogelijkheden van het framework.
- Snelheid van de applicatie kan trager zijn dan native applicatie aangezien er gewerkt

wordt via een tussenliggende taal, namelijk Javascript.
- Vaak afhankelijk van third parties.

1.2 Probleemstelling

Vanuit dit onderzoek zal blijken hoe goed Kotlin/Native is als cross-platform framework en wat de mogelijkheden zijn van Kotlin/Native om een applicatie te programmeren voor verschillende platformen. Deze vraag kwam vanuit het bedrijf genaamd Endare BVBA. Endare is een bedrijf dat gericht is op het ontwikkelen van mobiele- en webapplicaties. Een developer bij Endare is reeds bezig met het aanleren van Kotlin en heeft reeds enkele Android-projecten gemaakt met Kotlin. Bij Endare wordt er zeer veel aan cross-platform ontwikkeling gedaan. Zo heeft een stagiair tijdens zijn stage bij Endare, gewerkt met React/Native, het populaire framework van Facebook voor cross-platform ontwikkeling.

De vraag naar native applicaties wordt steeds minder. Waarom? Uit sectie 1.1.6 bleek dat native applicaties voor veel bedrijven een grote kost zijn. Er is enerzijds twee keer (indien er voor iOS en Android ontwikkeld wordt) een kost om de applicaties te bouwen, anderzijds is er twee keer een kost om de applicaties te onderhouden of eventueel uit te breiden. Volgens een artikel van Comentum, een toonaangevend bedrijf op het gebied van digitale oplossingen, varieert de kost voor het ontwikkelen van een native applicatie (Android of iOS) tussen de 9000 en 90000 dollar, afhankelijk van de vereisten van de applicatie (Kohan, 2015).

Project	Small MVP	Small Enterprise	Medium MVP	Medium Enterprise	Large MVP	Large Enterprise
iOS native development	$9,000	$11,000	$37,000	$45,000	$60,000	$90,000
Android native development	$9,000	$11,000	$37,000	$45,000	$60,000	$90,000

Figuur 1.1: Kostprijs ontwikkeling native applicatie (Kohan, 2015)

Uit sectie 1.1.6 valt te besluiten dat er ook redenen zijn waarom bedrijven nu net wel native applicaties willen in plaats van cross-platform applicaties.

Momenteel is er meer dan genoeg keuze om applicaties voor verschillende platformen tegelijkertijd te ontwikkelen. Met Kotlin/Native komt er een nieuw framework zich aanbieden. Dit framework is nog zeer jong en er bestaat bijna geen documentatie. Ontwikkelaars die dit framework willen gebruiken moeten momenteel zelf de werking van Kotlin/Native proberen achterhalen. Daarom is het belangrijk om te bestuderen of Kotlin een mogelijkheid biedt tot een nieuwe cross-platform programmeertaal, hoe de compiler van Kotlin ervoor kan zorgen dat het op meerdere platformen kan draaien en hoe men nu juist gebruik kan maken van Kotlin/Native.

Voor Endare heeft deze bachelorproef een grote meerwaarde aangezien zij iedere dag met cross-platform frameworks aan de slag moeten. Kotlin/Native zou de mogelijkheid geven om native applicaties te ontwikkelen in combinatie met een domeinlogica die gedeeld wordt tussen de verschillende platformen.

1.3 Onderzoeksvraag

De onderzoeksvragen voor deze bachelorproef zijn:

- Hoe zorgt de Kotlin compiler ervoor dat Kotlin op verschillende platformen kan gebruikt worden?
- Wat is de werking van Kotlin/Native?
- In hoeverre kan Kotlin gebruikt worden voor cross-platform applicatieontwikkeling?

1.4 Onderzoeksdoelstelling

Voor deze bachelorproef zijn er verschillende criteria van succes:

- De werking van de compiler, die Kotlin gebruikt, documenteren
- Goede documentatie opstellen over de werking van Kotlin/Native
- Bepalen in hoeverre Kotlin gebruikt kan worden als cross-platform programmeertaal

1.5 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4 zal onderzocht worden, via een literatuurstudie, wat de werking is van de Kotlin compiler. Hoe ervoor kan gezorgd worden dat Kotlin ook op apparaten zonder een JVM kan draaien.

In Hoofdstuk 5 zal Kotlin/Native bestudeerd worden. Hierin zal de werking van Kotlin/Native gedocumenteerd worden.

In Hoofdstuk 6 zal er praktisch een Kotlin/Native project worden uitgewerkt.

In Hoofdstuk 7, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

Dit hoofdstuk is een literatuurstudie over de huidige stand van zaken rond Kotlin. Hierin wordt bekeken wat Kotlin eigenlijk is en waarom Kotlin is uitgevonden. Daarna is Kotlin op verschillende platformen aan de beurt, meer specifieke is dit iOS, Android, webapplicatie en als back-end. Tenslotte het gebruik van Kotlin, hoeveel ontwikkelaars maken er reeds gebruik van Kotlin? Na het lezen van dit hoofdstuk bent u volledig op de hoogte van de laatste nieuwigheden rond Kotlin.

2.1 Wat is Kotlin?

Volgens de FAQ van Kotlin (JetBrains, 2017a) is Kotlin een open-source programmeertaal die object-georiënteerde en functionele programmatie features combineert. Kotlin is ook een statically typed programmeertaal. Dit betekent dat het type van de variabele is toegekend wanneer de code wordt gecompileerd. Javascript is een dynamically typed programmeertaal, waarbij aan een variabele verschillende types kan worden toegekend. Zo kan een variabele bij Javascript in het begin een getal zijn, maar wat verder in de code kan dit veranderd worden naar een tekst.

Kotlin is ontworpen door JetBrains. JetBrains is een organisatie afkomstig van Sint-Petersburg, Rusland. De naam Kotlin is afkomstig van het Kotlin eiland, 30 km ten westen van Sint-Petersburg. JetBrains is een software ontwikkelingbedrijf dat gesticht is in het jaar 2000. Hun hoofdkantoor is gevestigd in Praag (Tsjechië) en hun core-business is het ontwikkelen van tools die gebruikt kunnen worden door verschillende types van software ontwikkelaars. Zo hebben zij IDE's[1] ontwikkeld voor Java, Ruby, Python, PHP, SQL,

[1] Integrated Development Environment

Objective-C, C++, C# en JavaScript (JetBrains, 2018a).

2.2 Kotlin en Android

Volgens het artikel van JetBrains (2017b) heeft Google in 2017 bekend gemaakt dat het Kotlin volledig zou ondersteunen voor Android applicatieontwikkeling. In Android Studio zal vanaf versie 3.0 Kotlin ondersteund worden voor Android development. Een Kotlin project maken in Android Studio is dan ook heel gemakkelijk. In Android Studio is er bij het aanmaken van een project de mogelijkheid om direct de ondersteuning voor Kotlin in te schakelen. Hierdoor zal het aangemaakte project onmiddellijk in Kotlin geschreven zijn.

Het is mogelijk in Android Studio om een reeds bestaand Android Java project om te zetten naar Kotlin. Hierbij zal de actie 'Convert Java File to Kotlin File' moeten uitvoerd worden (rechtermuisknop in het Java bestand). Hierdoor zal Android Studio detecteren dat er gebruik gemaakt zal worden van Kotlin, waardoor hij zal vragen om de Kotlin plugin te installeren via Gradle indien deze nog niet is geïnstalleerd.

2.3 Automigration

Door de intergratie van Kotlin in Android Studio werd er een conversietool ter beschikking gesteld. Met behulp van deze tool kan bestaande Java-code eenvoudig worden omgezet naar Kotlin. Dit zorgt ervoor dat veel tijd kan worden bespaard en het programmeren van dubbele code zo wordt vermeden. Maar deze conversietool bevat wel een klein risico. Het kan wel eens gebeuren dat code soms fout wordt geconverteerd.

Het is ook reeds mogelijk om zowel Java en Kotlin te combineren. Zo kunnen de verschillende object classes in Java worden geschreven en kan je via Kotlin alle objecten aanmaken. Of dit nuttig en best practice is, valt te beslissen door de developer.

Indien bestaande Java code online of van een vorige project gekopieerd en geplakt wordt naar een bestaand Android Studio project, dan zal Android Studio zelf detecteren dat er Java code gebruikt zal worden in een Kotlin project. Hij zal dan zelf ook de suggestie doen om de Java code om te zetten naar Kotlin code (Avantica, 2017).

2.4 Kotlin web en back-end

Kotlin kan net zoals Java gebruikt worden om webapplicaties te bouwen. Dit in combinatie met bijvoorbeeld het Spring Framework, waarbij HttpServlets gebruikt worden om de webpagina's te tonen (JetBrains, 2018b).

Wens je echter een full-stack webapplicatie te bouwen, dan heb je de mogelijkheid om ook een Kotlin server op te zetten. Zo kan je bijvoorbeeld aan de webapplicatie een RESTfull

server hangen om verschillende API calls te doen (JetBrains, 2018c).

Kotlin-applicaties kunnen worden geïmplementeerd op elke host die Java-webapplicaties ondersteunt, inclusief Amazon Web Services, Google Cloud Platform en veel meer. Zowel een Kotlin webapplicatie als back-end maakt gebruik van de JVM.

2.5 Kotlin/Native

Waarschijnlijk momenteel één van de meest nieuwe en innovatieve projecten van JetBrains is Kotlin/Native. Momenteel is men gekomen aan versie 0.6 en er zijn al verschillende voorbeeldprojecten beschikbaar gesteld door JetBrains. Kotlin/Native zou het mogelijk maken om éénmalig domeinlogica te schrijven in een applicatie en deze te delen over verschillende platformen, bijvoorbeeld Android en iOS. De user interfaces zou men wel nog per platform moeten opbouwen, waardoor je toch het 'native applicatie'-gevoel krijgt. Kotlin/Native maakt gebruik van een totaal andere compiler dan de Javac (Java) of Kotlinc (Kotlin) compiler die bytecode genereert voor de JVM, zie sectie 2.8 voor meer info. De bijnaam die gegeven wordt aan Kotlin/Native is *Konan*. Binnen dit onderzoek zal Kotlin/Native nauw onderzocht worden (Gao, 2018b).

2.6 Verschil tussen Kotlin en Kotlin/Native

Het is belangrijk om een duidelijk verschil te zien tussen Kotlin en Kotlin/Native. Met Kotlin spreken we over de programmeertaal. Met Kotlin/Native wordt het framework bedoeld dat gebruikt kan worden voor cross-platform applicatieontwikkeling. Het framework zal gebruik maken van de programmeertaal Kotlin.

2.7 Kotlin en iOS

Verschillende ontwikkelaars, zoals Gao (2018a), zijn reeds aan de slag gegaan met deze Kotlin/Native plugin. Op zijn website is te zien hoe het mogelijk is om aan de hand van deze plugin, Kotlin te gebruiken voor iOS development.

Er zijn twee mogelijkheden. Enerzijds is er de mogelijkheid om het project in Objective-C aan te maken. De code wordt, rechtstreeks in de Objective-C bestanden, in Kotlin geprogrammeerd in combinatie met Objective-C annotations. Anderzijds kan het project in Swift aangemaakt worden. De overige Kotlin code wordt geprogrammeerd in aparte Kotlin bestanden, gecompileerd naar een iOS framework en toegevoegd aan het project.

Alhoewel het niet echt optimaal is om Kotlin direct te programmeren in Xcode, aangezien Xcode een Kotlin bestand niet zal herkennen, is het dus mogelijk. Indien de applicatie enkel en alleen ontwikkeld moet worden voor het iOS platform, dan is en blijft Swift of Objective-C de beste manier om een iOS applicatie te bouwen.

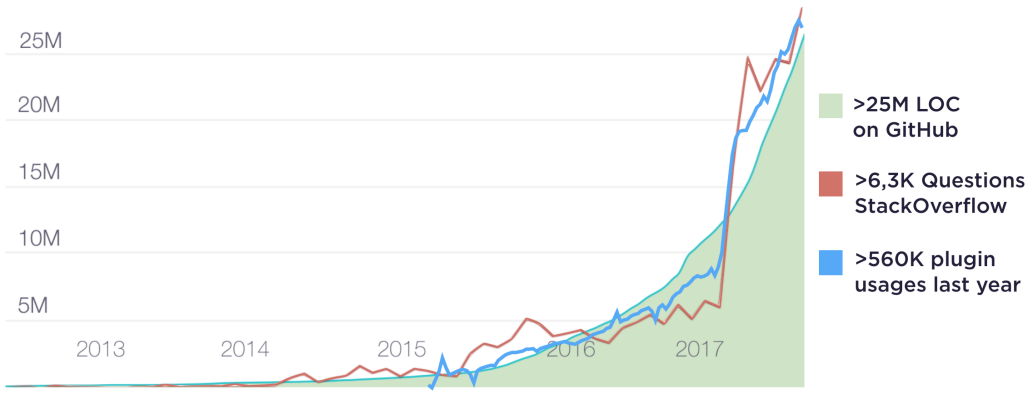
2.8 Compiler

Net zoals Java draait Kotlin op de JVM. Dit wil dus zeggen dat alle toestellen die een JVM kunnen draaien, ook Kotlin code ondersteunen. Maar sinds de dag dat JetBrains besloten heeft om zich niet enkel meer te richten op platformen die enkel en alleen de JVM ondersteunen (Andrey Breslav, 2017), hebben zij ervoor gezorgd dat ongeacht welk platform of besturingssysteem er gebruikt wordt, de Kotlin code wordt ondersteund. Dit komt door de LLVM compiler die Kotlin/Native gebruikt. Deze wordt in hoofdstuk 4 verder besproken.

2.9 Het gebruik van Kotlin

Het gebruik van Kotlin is gedurende de jaren zeer sterk gestegen. Op de blog van JetBrains (Dmitry Jemerov, 2017) zijn grafieken te vinden waarmee wordt aangetoond dat de populariteit van Kotlin enkel maar stijgt.

Figuur 2.1 toont het aantal lijnen Kotlin code beschikbaar op GitHub, het aantal vragen gesteld op Stackoverflow over Kotlin en het aantal keer dat de Kotlin plugin werd gebruikt. Hieruit kan besloten worden dat het gebruik van Kotlin sterk stijgt de laatste drie jaren.



Figuur 2.1: Hoeveelheid Kotlin code op GitHub (Dmitry Jemerov, 2017)

Volgens statistieken van een Android-software ontwikkelingsbedrijf genaamd AppBrain (AppBrain, 2018), is Kotlin het beste framework voor Android-applicaties te ontwikkelen. Verschillende veel gebruikte applicaties zoals Netflix, Twitter en Candy Crash bevatten grote delen Kotlin code.

Figuur 2.2 toont de verschillende user groups over de volledige wereld. Het toont de sterke opkomst van Kotlin over de wereld, met een sterke concentratie in Europa.

Figuur 2.2: Kotlin user groups in de wereld (Dmitry Jemerov, 2017)

2.10 Waarom Kotlin?

Maar waarom heeft JetBrains besloten om te beginnen met een nieuwe programmeertaal en deze dan later verder uit te bouwen met een cross-platform framework?

Een verklaring die op het internet te lezen is („Why JetBrains Invented and Promotes Kotlin", 2017), is dat JetBrains Kotlin heeft uitgevonden om hun eigen productiviteit te vergroten. Ze vonden dat Java niet al hun verwachtingen kon inlossen en daarom moest er een nieuwe programmeertaal op de markt komen. Momenteel hebben ze reeds een groot aantal IDE's ontwikkeld die geschreven zijn in Java. Dat is dan ook de reden dat men een programmeertaal ontwikkeld heeft die naar Java compileerbaar is. Een andere reden zou zijn dat men ontwikkelaars zou willen migreren naar een binnenshuis programmeertaal die gemakkelijker te ondersteunen is.

Anderzijds het feit dat ze kiezen voor een taal die draait op de JVM, betekent dus dat JetBrains niet alle bestaande libraries wil herschrijven, maar hergebruiken. Dit is ook te zien aan de compatibiliteit van beide programmeertalen. Beiden talen zijn volledig compatibel met elkaar. Sectie 2.3 is hiervan een mooi voorbeeld.

Op 2 augustus 2011 heeft JetBrains, in het jaar dat Kotlin bekend gemaakt werd, een artikel geschreven op hun blog „Why JetBrains needs Kotlin" (2011) waarin Dmitry Jemerov, de Kotlin tools team lead, verklaart waarom JetBrains Kotlin heeft ontworpen. In deze blogpost vindt men volgend zin terug: "We willen productiever worden door over te schakelen op een meer expressieve taal.". Ze geven dus duidelijk aan dat men productiever wil worden, maar wil men hiermee bevestigen dat Java enkele tekortkomingen heeft?

3. Methodologie

3.1 Opzoeken informatie over Kotlin

Nadat het bachelorproefvoorstel werd goedgekeurd werd er onmiddellijk aan de slag gegaan. De eerste stap voor deze bachelorproef was het opzoeken van informatie rond Kotlin. Het was belangrijk om alle mogelijkheden van Kotlin goed in kaart te brengen. Tijdens deze studie werd ontdekt dat Kotlin reeds veel mogelijkheden heeft. Het kan momenteel gebruikt worden voor server-side, javascript en Android applicatieontwikkeling. Ondertussen werd er geëxperimenteerd met enkele Kotlin projecten om gewoon te worden aan de Kotlin syntax.

Het doel van deze bachelorproef was het onderzoeken of Kotlin reeds enkele mogelijkheden had om gebruikt te worden voor cross-platform applicatieontwikkeling. Hierbij werd duidelijk dat JetBrains bezig was met het ontwikkelen van een framework dat gebruikt kon werden voor cross-platform applicaties te ontwikkelen. Echter was er op de website van JetBrains weinig informatie te vinden over dit framework. Meer onderzoek was dus nodig.

3.2 Opzoeken informatie over Kotlin/Native

Nadat er kennis werd gemaakt met Kotlin/Native, was het duidelijk dat er meer informatie nodig was om zomaar met dit framework aan de slag te gaan. Eerst en vooral werd de beknopte informatie over Kotlin/Native op de website van Kotlin doorgenomen. Op de website werd duidelijk dat Kotlin/Native gebruik maakte van een speciale compiler, namelijk LLVM. Hierover moest informatie opgezocht worden. Verder was er op het internet, behalve de algemene uitleg over Kotlin/Native, weinig nieuwe informatie te vinden

en werd het duidelijk dat de echte werking aan de hand van enkele voorbeeldprojecten van JetBrains duidelijk moest worden.

3.3 Bekijken voorbeeldprojecten JetBrains

De volgende stap was het bekijken, onderzoeken en analyseren van de voorbeeldprojecten van JetBrains. Er waren verschillende voorbeelden aanwezig op de GitHub respository van JetBrains. Een voorbeeld hiervan was de Calculator applicatie. Een cross-platform applicatie waarbij de domeinlogica een rekenmachine was. Deze werd gedeeld tussen de verschillende platformen, zijde Android en iOS. De opzet van een Kotlin/Native project was helemaal niet duidelijk. De enige mogelijkheid om de werking van een project duidelijk te begrijpen was het kopiëren van een project en hierin aanpassingen te maken. Zo werd duidelijk wat de effecten waren van deze aanpassingen en kon de werking van bepaalde elementen achterhaald worden. Naast de voorbeeldprojecten van JetBrains werd er een repository gevonden op GitHub, van een developer genaamd Gao (2018b). Deze persoon heeft zelf ook een Kotlin/Native project opgezet, gericht op Android en iOS.

3.4 Praktische uitwerking Kotlin/Native

Daarna was het tijd om de grote stap te zetten en volledig zelfstandig een Kotlin/Native project op te zetten. De volledige mappenstructuur van een voorbeeldproject werd overgenomen van de repository van Gao (2018b). Na genoeg projecten onderzocht te hebben was het al iets gemakkelijker om een project vanaf nul op te zetten. Bij het opzetten werd duidelijk dat Gradle voortdurend gebruikt zou worden aangezien de volledige opzet van een project en het gebruik van de Kotlin/Native plugin momenteel moet gebeuren via Gradle. Naarmate het project vorderde werd de werking van Gradle duidelijker en was het makkelijker om dit build systeem te gebruiken. Door het zelf schrijven van Gradle scripts werd de werking van Kotlin/Native ook duidelijker. Indien er iets ontbrak in de scripts, werden er nuttige foutboodschappen getoond in de console door Kotlin/Native zelf.

3.5 Literatuurstudie LLVM compiler

Parallel met het onderzoeken van Kotlin/Native werd de compiler van dit framework bestudeerd. In sectie 3.2 werd geconstateerd dat Kotlin/Native de LLVM compiler gebruikt. Na opzoekwerk op het internet bleek dat er niet bijzonder veel specifieke informatie te vinden was over deze compiler. Echter bleek dat er op het internet een aosabook (Lattner, 2018), 'The Architecture of Open Source Applications', bestond. Hierin werd de volledige werking van een eenvoudige compiler (voor één programmeertaal), een meer geavanceerde compiler (voor meerdere programmeertalen) en de LLVM compiler volledig uitgelegd en het verschil tussen LLVM en een standaard compiler werd aangetoond. Voor dit onderzoek was deze documentatie een zeer grote hulp.

4. De compilatie van Kotlin/Native

Kotlin is een programmeertaal die gebruik maakt van de JVM. Sinds de beslissing van JetBrains om zich ook te focussen op cross-platform development, moesten ze met een oplossing komen voor de JVM. De JVM wordt namelijk niet ondersteund op alle besturingssystemen. Zo ondersteunen bijvoorbeeld MacOS en iOS (mobile) geen JVM. JetBrains moest dus op zoek gaan naar een oplossing... de LLVM compiler. Deze compiler is een reeds bestaande compiler en is dus niet ontwikkeld in functie van Kotlin. Deze literatuurstudie is uitgevoerd aan de hand van de aosabook van LLVM, Lattner (2018), een zeer uitgebreide documentatie.

4.1 Wat is LLVM?

Het LLVM-project is een verzameling modulaire en herbruikbare compiler- en toolchain-technologieën. Het project is ontwikkeld door de LLVM developer group, waarvan Vikram Adve en Chris Lattner de originele ontwikkelaars zijn. De naam 'LLVM' zelf is geen acroniem, het is de volledige naam van het project. Ondanks de naam heeft LLVM weinig te maken met de traditionele virtuele machines.

Het LLVM-project begon als een onderzoeksproject aan de universiteit van Illinois, met als doel een compilatiestrategie aan te bieden die in staat is om zowel statische als dynamische compilatie van programmeertalen aan te bieden. Een voorbeeld van een statische taal is Java, een voorbeeld van een dynamische taal is Javascript. Zowel beide types van programmeertalen kunnen dus gecompileerd worden door LLVM.

Sinds het ontstaan van LLVM is het project uitgegroeid tot een overkoepelend project dat bestaat uit een aantal deelprojecten. Veel van deze deelprojecten worden momenteel sterk

gebruikt bij commerciële en open-source projecten en zelfs in academisch onderzoek.

Enkele voorbeelden van commerciële en open-source projecten zijn:

- **Vuo**, een moderne visuele programmeertaal voor multimedia ontwerpers.
- **Pony**, een object-georiënteerde programmeertaal.
- **Crack**, een programmeertaal die het gemak van de ontwikkeling van een scripttaal biedt met de uitvoering van een gecompileerde taal.

Het grote voordeel van het gebruik van LLVM is de veelzijdigheid, flexibiliteit en herbruikbaarheid. Dit wil zeggen dat het zo goed als in ieder soort project kan worden geïntegreerd. Het wordt daarom tegenwoordig gebruikt voor een groot aantal verschillende taken, dit gaande van het compileren van enkele kleine code-projecten tot het compileren van code voor massieve computers (LLVM developer group, 2018).

4.1.1 Deelprojecten LLVM

Enkele voorbeelden van deelprojecten van de LLVM developers group (LLVM developer group, 2018):

- **Clang** is een LLVM native C/C++/Objective-C compiler dat als doel heeft om verbazingwekkend snelle compilaties aan te bieden. Het zorgt overigens voor nuttige fout- en waarschuwingsberichten.
- Het **LLDB** project bouwt verder op libraries die aangeboden worden door LLVM en Clang om te zorgen voor een native debugger.
- Het **libc++** en **libc++ ABI** project voorziet een krachtige implementatie van de standaard C++ bibliotheek, met ondersteuning voor C++11.

4.2 De werking van LLVM

LLVM is dus een bibliotheek die gebruikt wordt om tussenliggende en/of machine-code te genereren en optimaliseren. Maar wat is nu de exacte werking van deze compiler? LLVM is totaal verschillend van de JVM. Zie sectie 4.5 voor meer uitleg.

4.2.1 De werking van een standaard compiler

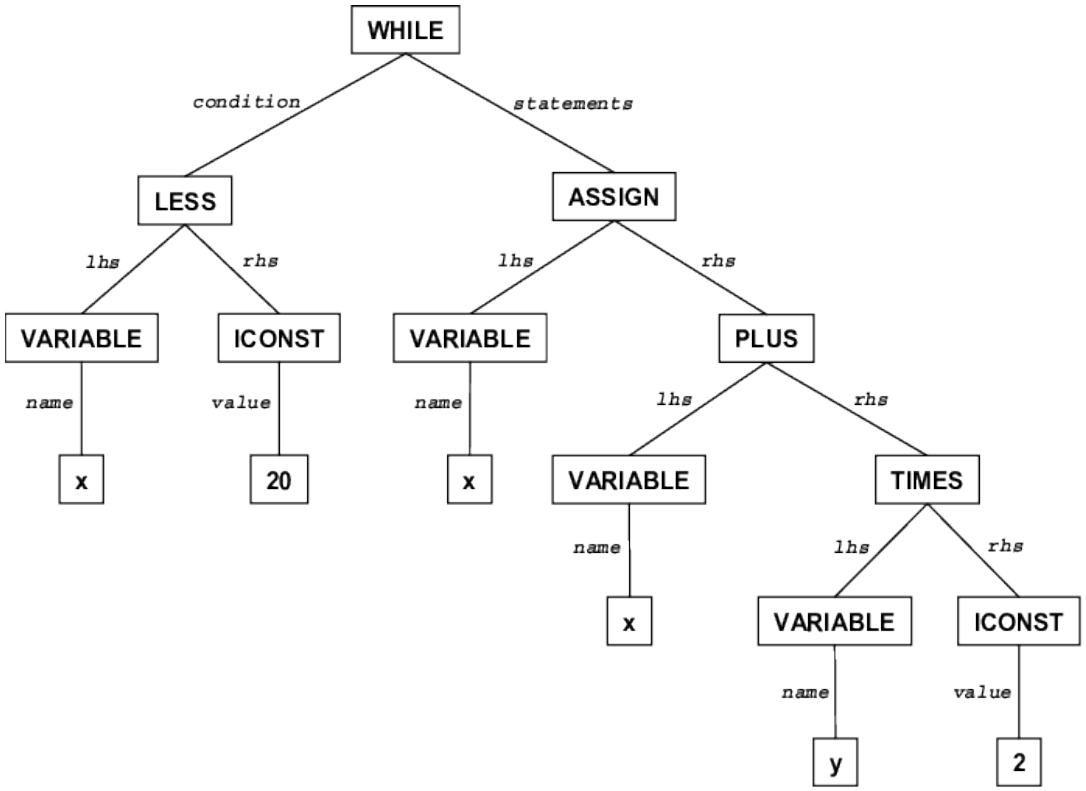
Het meest populaire ontwerp voor een statische compiler, zoals de meeste C-compilers, is het driefasenontwerp waarbij de belangrijkste componenten de **front-end**, de **optimizer** en de **back-end** zijn.

De front-end

De front-end zorgt voor de analyse van de broncode. Deze heeft dus als taak om ervoor te zorgen dat geen code gecompileerd kan worden waarbij fouten aanwezig zijn. Dit

Figuur 4.1: Driefasenontwerp (Lattner, 2018)

gebeurt door het opstellen van een taalspecifieke abstracte syntaxboom om de syntaxcode voor te stellen. Deze syntaxboom wordt optioneel geconverteerd naar een nieuwe boom voor optimalisatie en de optimizer en de back-end worden uitgevoerd aan de hand van deze boom. De Javac compiler is ook een implementatie van dit driefasenontwerp. Een voorbeeld van een syntaxboom is te zien in figuur 4.2.



Figuur 4.2: Taalspecifieke abstracte syntaxboom (ResearchGate, 2009)

De optimizer

De optimizer is verantwoordelijk, zoals de naam het zelfs zegt, voor het uitvoeren van een breed gamma van transformaties om de uitvoeringstijd van de code te optimaliseren. Dit wordt gedaan door het elimineren van overtollige berekeningen, maar dit is meestal afhankelijk van de taal die gebruikt werd en de doelarchitectuur.

De back-end

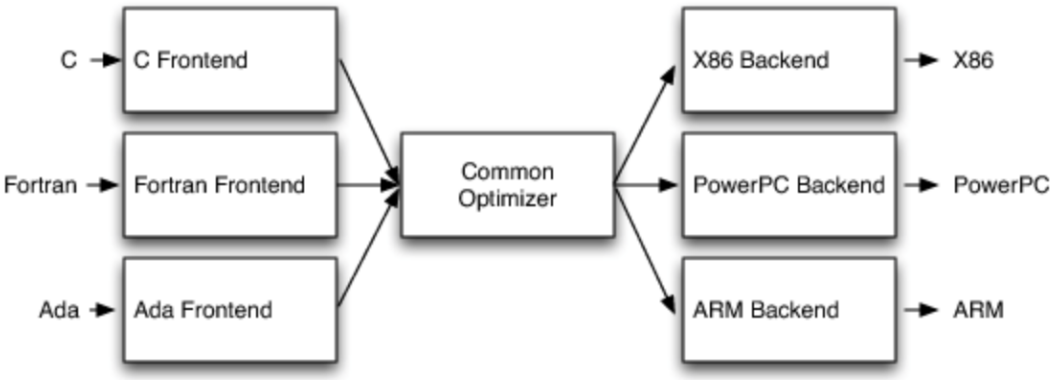
De back-end, ook bekend als de codegenerator, gaat de code gaan mappen naar een instructieset. Naast het maken van deze instructieset is het ook verantwoordelijk om ervoor te zorgen dat deze instructieset gebruik maakt van de ongewone kenmerken van de ondersteunde architectuur. Iedere architectuur is namelijk anders en de gegenereerde instructieset moet dus afgestemd worden op de doelarchitectuur.

4.2.2 Meerdere front- en back-ends

Het grote voordeel van dit driefasenontwerp treedt op wanneer een compiler besluit om meerdere brontalen en architecturen te ondersteunen. Indien de optimizer een gemeenschappelijke code representatie gebruikt, dan kan een front-end geschreven worden voor elke taal die gecompileerd kan worden naar die gemeenschappelijke representatie, en een back-end kan worden geschreven voor elke doelarchitectuur dat daaruit kan compileren. Door het gebruik van deze gemeenschappelijke code representatie heeft de optimizer geen kennis nodig van de doelarchitectuur. Op figuur 4.3 is te zien dat door de common optimizer te gebruiken, er meerdere front-ends kunnen worden geschreven en meerdere doelarchitecturen kunnen worden ondersteund.

Met dit ontwerp, indien men wenst een nieuwe brontaal te ondersteunen, moet men enkel een nieuwe front-end schrijven maar de bestaande optimizer en back-end kunnen worden hergebruikt. Indien deze delen (front-end, optimizer en back-end) niet waren gescheiden, zou het implementeren van een nieuwe brontaal vereisen om helemaal opnieuw te beginnen, dus een volledig nieuwe compiler te schrijven. Indien men N doelarchitecturen heeft en M brontalen, dan zou men N*M compilers hebben.

Nog een voordeel van dit ontwerp is dat deze soorten van compilers een veel bredere set van programmeurs kan tevreden stellen. Dit is minder het geval indien men slechts één brontaal en doelarchitectuur zou ondersteunen.



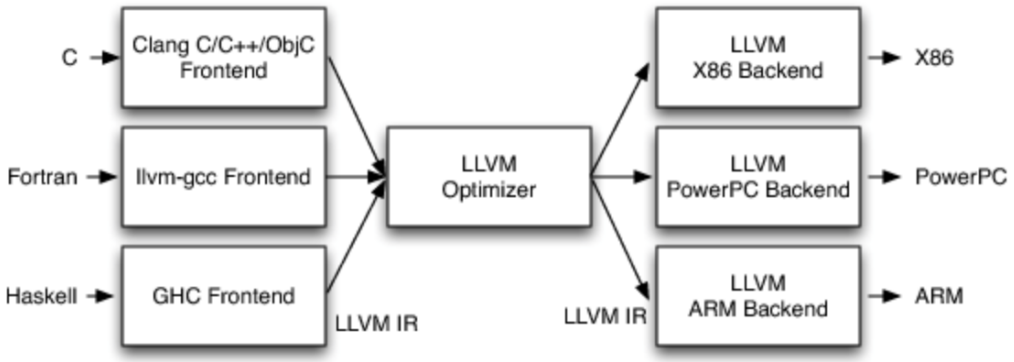
Figuur 4.3: Meerdere front- en back-ends (Lattner, 2018)

4.3 LLVM en het driefasenontwerp

In een LLVM-gebaseerde compiler heeft de front-end dezelfde taak als de front-end in een normale compiler. De code wordt daarna door de front-end vertaald naar LLVM intermediate representation (IR), dit meestal door het opbouwen van een syntaxboom en deze later te converteren naar IR. De IR is eigenlijk het hart van de LLVM. Het is een low-level programmeertaal die zeer dicht aansluit bij assembly code. Deze IR heeft als taak om de code te verbeteren, waarna deze code in een soort van codegenerator wordt gestoken waarbij deze wordt omgezet naar native machinecode.

In een korte notendop: de front-end krijgt een brontaal binnen en gaat deze taal parsen, valideren en analyseren. Hij stelt de syntaxboom op en zet deze om naar LLVM IR. De optimizer gaat de code optimaliseren en stuurt de IR code naar de back-end die dient als codegenerator met als resultaat machinecode.

LLVM is dus een implementatie van het driefasenontwerp. Het grote verschil tussen een gewone implementatie van dit ontwerp en LLVM is het gebruik van de IR. Deze IR is de enige interface voor de optimizer. Dit betekent dat men enkel moet weten wat IR en wat de werking van deze IR is om een front-end te schrijven voor deze LLVM, wat het ondersteunen van nieuwe talen veel makkelijker maakt.



Figuur 4.4: LLVM implementatie van driefasenontwerp (Lattner, 2018)

4.4 LLVM en Kotlin/Native

Kotlin/Native is een technologie dat Kotlin via LLVM direct compileert naar machine code. De Kotlin/Native compiler produceert zelfstandige uitvoerbare bestanden die zonder virtuele machine kunnen worden uitgevoerd. Hierdoor is het mogelijk om Kotlin te gebruiken op ieder platform of besturingssysteem.

4.5 Java Virtual Machine (JVM)

De Java Virtual Machine, ook wel JVM genoemd, is een virtuele machine voor het uitvoeren van Java bytecode (Techopedia, 2018a). De Javac is de compiler die verantwoordelijk is voor het omzetten van Java-code naar bytecode. Deze bytecode is een object-georiënteerde code die gecompileerd is om op een virtuele machine te worden uitgevoerd. De code kan op elk besturingssysteem of platform, waarvoor er een JVM beschikbaar is, worden gebruikt. De virtuele machine transformeert de code, geschreven in een bepaalde programmeertaal, in machinetaal die door de CPU[1] van een systeem kan worden uitgevoerd. Deze machinetaal moet geoptimaliseerd zijn voor de CPU aangezien andere platformen verschillende interpretatietechnieken kunnen gebruiken. Dit alles gebeurt at-runtime. Bij de JVM zal de bytecode meestal het resultaat zijn van de compilatie van Java-code. Echter kunnen talen zoals Scala, Clojure, Groovy en natuurlijk Kotlin (maar via de Kotlinc compiler) ook uitgevoerd worden op deze virtuele machine. De JVM heeft kennis van geheugenbeheer, optimalisatie van de uitvoering van de applicatie en garbage collection. Daarnaast begrijpt de JVM het concept van objecten en virtuele methode aanroepen. De gebruiker hoeft hierdoor geen rekening te houden met deze drie zaken. Deze virtuele machine kan ook gebruikt worden voor Kotlin. Echter zal de Javac geen bytecode genereren, maar zal deze vervangen worden door de Kotlinc compiler die geoptimaliseerd is voor Kotlin (Techopedia, 2018b).

4.6 LLVM en JVM

LLVM en JVM zijn twee aparte dingen. De JVM is, zoals te lezen in sectie 4.5, een vertaler die broncode omzet naar machinetaal die door de processor van een computer onmiddelijk kan worden uitgevoerd. Het is een high-level virtuele machine aangezien deze kennis heeft van zaken zoals de garbage collection, geheugenbeheer en objecten. Bij de JVM gebeurt alles at-runtime.

De LLVM is een low-level register gebaseerde compilatie technologie. Het neemt de LLVM IR, zie sectie 4.3, en gaat deze overbrengen naar native uitvoerbare bestanden voor een specifiek platform. LLVM kan tijdens het vertaalproces naar IR veel optimalisaties uitvoeren. Echter gebeurt bij de LLVM alles at-compiletime.

[1] Central Processing Unit

5. De werking van Kotlin/Native

JetBrains heeft met Kotlin/Native de interesse van elke cross-platform ontwikkelaar getrokken. Niemand had verwacht dat Kotlin uitgebreid ging worden met een cross-platform framework. Zo zal JetBrains met Kotlin/Native een nieuwe markt betreden, het maken van native applicaties in combinatie met het delen van domeinlogica tussen de verschillende ondersteunde platformen.

5.1 Wat is Kotlin/Native?

Kotlin/Native is een technologie die zorgt voor de compilatie van Kotlin naar native binaire bestanden die zonder VM draaien. Kotlin/Native maakt gebruikt van een LLVM gebasseerde backend voor de Kotlin-compiler en een native implementatie van de Kotlin bibliotheek. Origineel werd Kotlin/Native uitgevonden om compilatie van Kotlin mogelijk te maken op platformen die geen virtuele machines, zoals de JVM, ondersteunen. Een voorbeeld hiervan is iOS dat geen ondersteuning biedt voor de JVM.

Kotlin Native is momenteel heel jong en zit nog in volledige ontwikkeling. De eerste versie van Kotlin/Native werd vrijgegeven op 31 maart 2017. Er zijn reeds enkele versies aanwezig op de GitHub van JetBrains waar ontwikkelaars met aan de slag kunnen. De ondersteuning voor de IDE's is beschikbaar als een plugin voor CLion.

5.2 Het delen van code in Kotlin/Native

Het doel van Kotlin/Native is om een framework aan te bieden dat gebruikt kan worden voor cross-platform ontwikkeling. De focus ligt niet op het maken van user interfaces, met alle functionaliteiten inbegrepen, via één codebase, maar wel op het delen van de domeinlogica. JetBrains wil dus via Kotlin/Native de mogelijkheid aanreiken om alle domeinlogica in een applicatie te hergebruiken op verschillende platformen en per platform de user interface op te bouwen.

Voordelen:

- Hergebruik van de domeinlogica.
- Mogelijkheid om per platform, in de user interface, andere accenten naar voor te brengen.

Nadelen:

- Verplichting om meerdere user interfaces op te bouwen (indien men meerdere platformen wil ondersteunen).
- Grotere kosten en meer tijd nodig om beide applicaties te ontwikkelen (Android en iOS).
- Opzet van dit soort projecten is (momenteel) niet gemakkelijk voor de onervaren gebruiker.

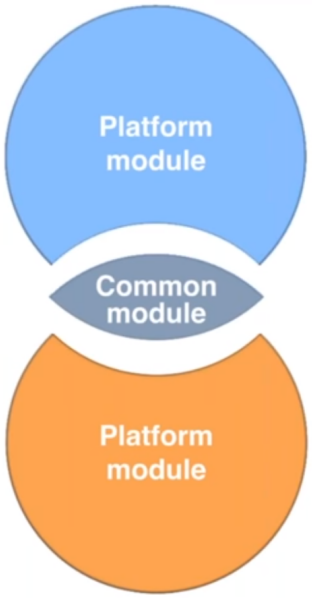
Figuur 5.1 toont hoe Kotlin/Native domeinlogica deelt over de verschillende platformen heen. Er wordt gebruik gemaakt van een **common** module. In deze module bevindt zich de gemeenschappelijke code, die gedeeld wordt met alle platformen en dus voor elk platform hetzelfde zal zijn. Daarnaast is er per platform dat ondersteund moet worden een aparte platformmodule. Deze platformmodules maken gebruik van de common module en er kan in deze platformmodules platformspecifieke code geschreven worden. De mogelijkheid is dus aanwezig om voor een bepaalde klasse, per platform een andere implementatie te voorzien. Zie sectie 5.4.1 voor een meer technische uitleg.

5.3 De structuur van een Kotlin/Native project

Alvorens te beginnen met het ontwikkelen van een Kotlin/Native project is het belangrijk om na te gaan wat de structuur is waar een Kotlin/Native project moet aan voldoen. Er is namelijk geen IDE of plugin die een Kotlin/Native project volledig automatisch genereert, daarom is het essentieel dat er vanaf het begin een goede mappenstructuur wordt opgebouwd. Figuur 5.2 toont een goede structuur van een Kotlin/Native project. Deze mappenstructuur is verplicht om aan te houden, de namen van de mappen kunnen veranderd worden mits ook kleine verandering van de build scripts in 6.

Op deze figuur zijn dus een aantal mappen te vinden:

- De **.gradle** map, automatisch gegenereerd, houdt alle instellingen en andere be-



Figuur 5.1: Het delen van code in Kotlin Native (Developine, 2017)

standen bij die gebruikt worden door Gradle om het project te bouwen. Dit wordt automatisch gegenereerd wanneer er een Gradle run wordt uitgevoerd.

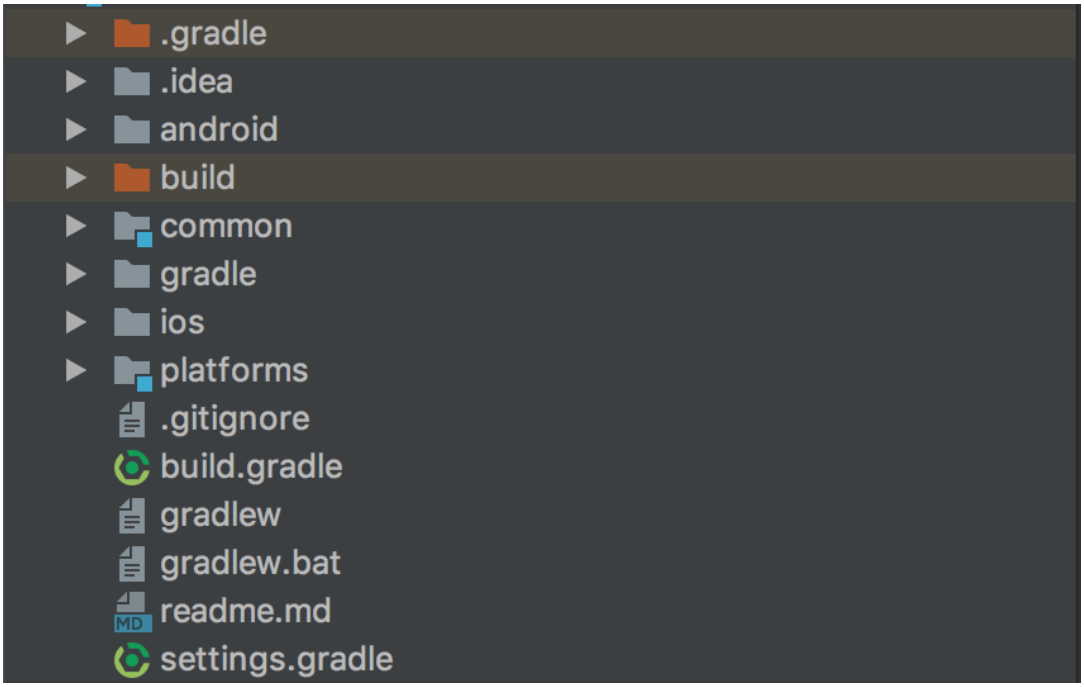
- Voor dit voorbeeld werd IntelliJ gebruikt als IDE. IntelliJ zal de map, **.idea**, automatisch genereren om verschillende instellingen van het project bij te houden. Alle projectspecifieke instellingen zijn aanwezig in deze map.
- **android** bevat het volledige Android project.
- De **build** map bevat informatie over de laatste uitgevoerd build.
- De **common** map bevat gemeenschappelijke code dat over de verschillende platformen wordt gedeeld.
- De **gradle** map, automatisch gegenereerd, bevat een properties bestand. In dit bestand worden enkele belangrijke properties onthouden, zoals de locatie van de gradle installatie.
- **ios** bevat het volledige iOS project.
- De **platforms** map bevat voor ieder platform, dat men wenst te ondersteunen, een map. In dit voorbeeld zijn er dus twee mappen, een iOS en Android-map. De bedoeling is om hierin specifieke code te schrijven die varieert per platform. Zie sectie 5.2 voor meer informatie.

5.4 Kotlin/Native code

5.4.1 Expect en actual klassen

Stel dat men wenst om een ABCMethods Kotlin klasse te hebben die de methodes callA(), callB() en callC() heeft, maar waarvan de implementatie verschillend is per platform.

In de common module wordt de ABCMethods klasse aangemaakt. Hierbij wordt het



Figuur 5.2: Kotlin/Native structuur

keyword *expect* gebruikt. Het keyword zegt bijna zelf waarvoor het gebruikt moet worden: er wordt een ABCMethods klasse 'verwacht' voor ieder platform. Het expect keyword kan ook vergeleken worden met een standaard Java interface.

```
expect class ABCMethods() {
   fun callA():String
   fun callB():String
   fun callC():String
}
```

Met het **actual** keyword geven we aan dat deze klasse een concrete implementatie is van ABCMethods. De compiler zal in eerste instantie zoeken naar de ABCMethods klasse in de common module. Hij zal daar merken dat ABCMethods een expect klasse is waardoor hij naar de platformspecifieke folder zal gaan en daar de actual klasse van ABCMethods zal gebruiken. Voor ieder platform wordt dus een concrete implementatie voorzien van de ABCMethods klasse.

Een mogelijke implementatie voor iOS:

```
actual class ABCMethods actual constructor() {
   actual fun callA():String {
      return "Calling A from iOS"
   }

   actual fun callB():String {
      return "Calling B from iOS"
   }
```

```
    actual fun callC():String {
        return "Calling C from iOS"
    }
}
```

Een mogelijke implementatie voor Android:

```
actual class ABCMethods actual constructor() {
    actual fun callA():String {
        return "Calling A from Android"
    }

    actual fun callB():String {
        return "Calling B from Android"
    }

    actual fun callC():String {
        return "Calling C from Android"
    }
}
```

Door gebruik te maken van Kotlin/Native is het niet nodig om zelf aan te geven welk bestand de compiler moet gebruiken voor welk platform. De compiler zal detecteren welk platform er gebruikt wordt en het juiste bestand gebruiken.

5.4.2 Gemeenschappelijke klassen

Naast de expect en actual klassen, die ervoor zorgen dat we platformspecifieke code kunnen schrijven en dus een onderscheid kunnen maken tussen de verschillende ondersteunde platformen, kunnen we ook klassen voorzien die voor alle platformen hetzelfde zijn. De implementatie van deze klasse is voor ieder platform hetzelfde.

```
class Example {
    private var name: String

    constructor() {
        this.name = "Ilias.vw"
    }

    fun helloKotlin(): String {
        return "Hello Kotlin Native, from $name"
    }
}
```

5.5 Het gebruiken van de Kotlin code

Maar hoe kan de gemeenschappelijke en platformspecifieke code gebruikt worden? Simpel. Bij een Android-project kunnen we de Kotlin code gebruiken door simpelweg de bibliotheek te importeren. Dit is niks meer dan een import bovenaan in de Android activity. iOS gerelateerde code zal gedeeld worden via een iOS framework dat gegenereerd wordt door Kotlin/Native. Dit framework zal toegevoegd worden in de build phases van het Xcode project. Zie sectie 6.10 voor meer informatie.

Kotlin/Native zal de code uit de common map en de code uit de platformspecifieke map bundelen tot één geheel. Zoals reeds gezegd zal dit bij iOS een framework zijn, bij Android echter een package die kan worden geïmporteerd. Dit gebeurt aan de hand van verschillende plugins, zie hoofdstuk 6 voor meer informatie over deze build scripts.

6. Kotlin/Native in de praktijk

In dit hoofdstuk zal er praktisch een Kotlin/Native project worden opgebouwd. De bedoeling is om aan de hand van een voorbeeld de volledige werking uit te leggen. In dit voorbeeld zal er een simpele applicatie gebouwd worden, gericht op Android en iOS, waarbij gebruikers een shopping cart kunnen aanmaken, producten kunnen bekijken en toevoegen aan de shopping cart en kunnen bekijken welke producten reeds in hun winkelmandje aanwezig zijn. Het volledige project is te vinden op https://github.com/Iliasvw/kotlin-native-example.

6.1 Domeinmodel

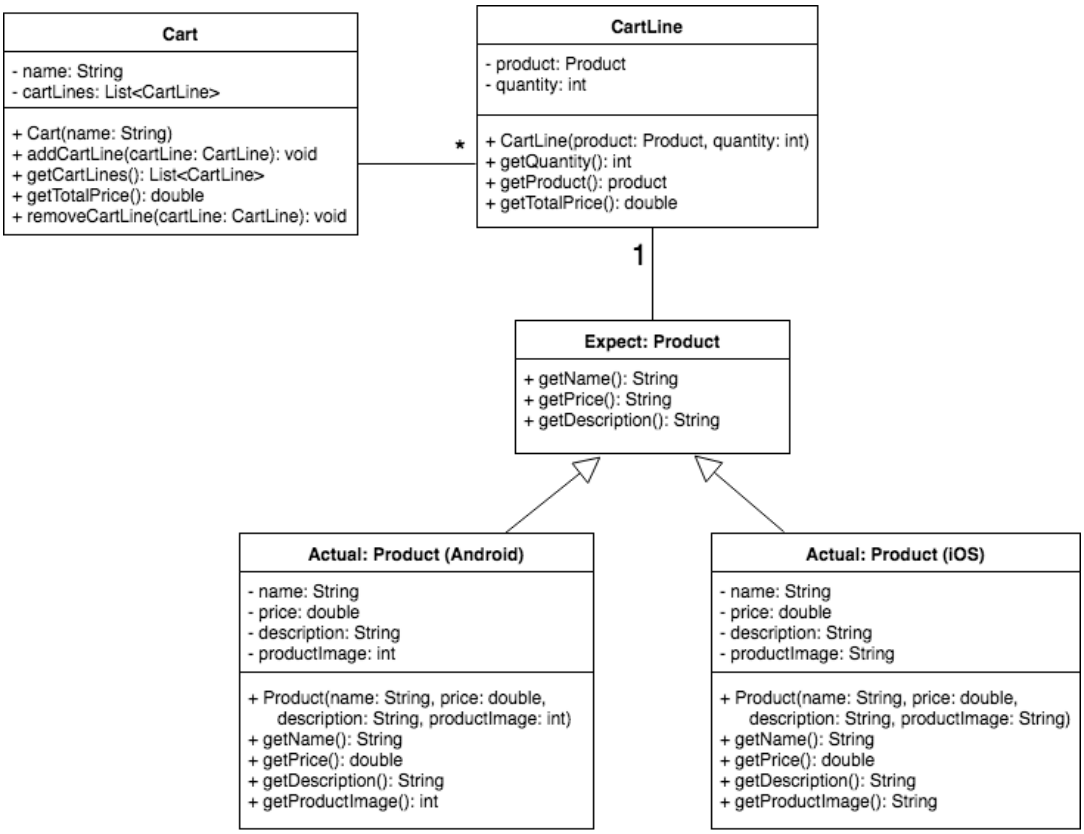
Aangezien het doel van Kotlin/Native het delen van domeinlogica is, is het handig om op voorhand na te denken over het domeinmodel. Welke klassen zijn er nodig, welke attributen en methodes moeten deze klassen hebben, is er eventueel in implementatie van klassen tussen Android en iOS? Een domeinmodel geeft ook een duidelijk overzicht over de logica van de applicatie. Zie figuur 6.1 om het domeinmodel te bekijken van deze voorbeeldapplicatie.

Het domeinmodel beschikt over drie klassen:

- De klasse Cart stelt het winkelmandje voor. Deze beschikt over een naam en kan nul, één of meerdere CartLines bevatten. De klasse heeft een constructor die een naam verwacht en methodes om een CartLine toe te voegen, alle CartLines op te vragen, de totale prijs van het winkelmandje te berekenen en een CartLine te verwijderen van het winkelmandje.
- Een CartLine object heeft een product en quantity (hoeveelheid) attribuut. Deze

klasse beschikt over een constructor die twee parameters verwacht, die tevens onze attributen zijn. Deze is ook voorzien van twee getters, om de attributen op te vragen (aangezien deze private zijn) en een methode om de kostprijs van deze CartLine te berekenen.

- De product klasse is de klasse die per platform wordt geïmplementeerd, dit is echter illustratief voor deze proof-of-concept. Deze klasse wordt dus als expect gedeclareerd en beschikt over drie expect methodes om de attributen van de klasse op te vragen. Per platform, in dit voorbeeld zal dit Android en iOS zijn, wordt er een implementatie voorzien van deze klasse. Deze actual klassen hebben elk een name, price, description en productImage en een implementatie van de methodes van de expect Product klasse. Het type van de productImage verschilt per platform.



Figuur 6.1: Klassendiagram domeinlogica

6.2 Requirements

Om gebruik te kunnen maken van Kotlin/Native voor het ontwikkelen van cross-platform applicaties (Android en iOS), zijn er enkele vereisten:

- **Android studio** voor het ontwikkelen van de Android applicatie.
- **Xcode** voor het ontwikkelen van de iOS applicatie (toestel met MacOS is vereist).
- **IntelliJ IDEA** (optioneel) voor het opzetten van het project. Dit kan eventueel met een andere IDEA, die gradle projecten ondersteunt, worden opgezet.

- **Gradle** om gebruik te kunnen maken van de Kotlin/Native compiler en plugin. Deze wordt automatisch geïnstalleerd bij het instellen van IntelliJ IDEA of Android Studio.

6.3 Wat is gradle?

Gradle is een build systeem. Het combineert de beste features van vele andere build systemen en combineert deze tot één systeem. Het is een op JVM gebaseerd build systeem wat betekent dat er eigen geschreven Java scripts kunnen worden gebruikt.

Android Studio maakt gebruikt van Gradle. Waarom maakt Google gebruikt van Gradle? Google zag Gradle als een zeer geavanceerd build systeem en realiseerde dat er eigen geschreven build scripts gebruikt kunnen worden met zo goed als geen leercurve. Het was ook niet nodig om een andere programmeertaal aan te leren aangezien er scripts kunnen worden geschreven in Java. Hierdoor is de Gradle plugin toegevoegd aan Android Studio (Krill, 2013).

6.4 Stap 1: project initiatie

De eerste stap in het ontwikkelen van een Kotlin/Native project is het maken van een nieuwe map op een gewenste locatie op de harde schijf van de computer. Hierin zal er een eerste bestand worden aangemaakt: **build.gradle**.

```
subprojects {
 buildscript {
  ext.kotlin_version = '1.2.31'
  ext.kotlin_native_version = '0.6.2'

  repositories {
   jcenter()
   google()
   maven { url "http://kotlin.bintray.com/kotlinx" }
   maven { url "https://plugins.gradle.org/m2/" }
   maven { url "https://dl.bintray.com/jetbrains/
       kotlin-native-dependencies" }
  }

  dependencies {
   classpath "org.jetbrains.kotlin:kotlin-gradle-plugin: $kotlin_version"
   classpath 'com.android.tools.build:gradle:3.0.1'
   classpath "org.jetbrains.kotlin:kotlin-native-gradle-plugin:
       $kotlin_native_version"
  }
 }

 group 'ilias.vw'
```

```
 version '1.0-SNAPSHOT'

 repositories {
  jcenter()
  maven { url "http://kotlin.bintray.com/kotlinx" }
 }

 tasks.withType(Test) {
  testLogging {
   showStandardStreams = true
   events "passed", "failed"
  }
 }
}
```

Hieronder wordt de inhoud van het build.gradle bestand besproken.

6.4.1 Versies

Bovenaan wordt er aangegeven welke versie van Kotlin en Kotlin/Native er gebruikt zal worden voor de opzet van dit project. Dit zijn beide de nieuwste versies. Opgelet, het is niet altijd vanzelfsprekend om de laatste versie van Kotlin/Native te gebruiken. Er wordt voortdurend gewerkt aan Kotlin en Kotlin/Native en nieuwe dingen worden continu gepusht op de GitHub repository. Het kan al eens gebeuren dat sommige features niet altijd even goed werken, wat voor problemen kan zorgen bij de ontwikkeling van de applicatie. Bij twijfels, neem een minder recente versie van de Kotlin/Native plugin.

6.4.2 Repositories

In de repositories tag worden de juiste repositories gelinkt:

- **Kotlinx** bevat alle coroutines[1] die Kotlin kan gebruiken.
- **Gradle**
- **Kotlin-native-dependencies** stelt alle dependencies, die Kotlin/Native nodig heeft, ter beschikking.

6.4.3 Dependencies

In de dependencies tag worden de juiste dependencies gelinkt:

- **Kotlin-gradle** laadt de Kotlin-gradle plugin met de versie van Kotlin die wordt aangegeven bovenaan het build script. Deze plugin zorgt voor het compileren van

[1] Programmaonderdelen die asynchroon programmeren vergemakkelijken door gecompliceerde logica in bibliotheken te steken

Kotlin bronnen en modules.
- **Android build tools** is nodig voor het bouwen van de Android-applicatie.
- **Kotlin-native-gradle-plugin** is de Kotlin/Native plugin die gebruikt zal worden om te zorgen voor een cross-platform applicatie.

6.4.4 Overige informatie

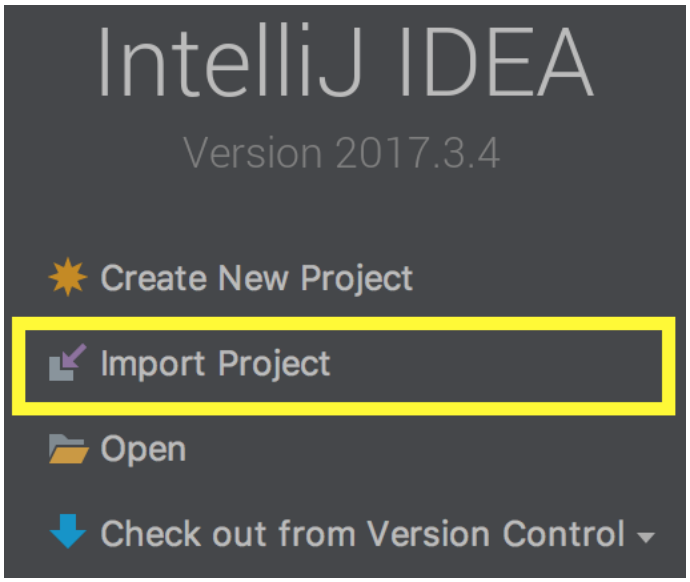
- **Group** stelt het groupId van het project in.
- **Version** geeft de versie van het project weer.
- **testLogging** wordt gebruikt voor het uitvoeren van de testen.

6.4.5 Build map

Voor elke map die een build.gradle bestand bevat zal er automatisch een build folder worden gegenereerd. De build map bevat alle gecompileerde bestanden, van een bepaalde map, die gegenereerd worden door Kotlin/Native en deze worden bij iedere gradle run en/of build aangemaakt.

6.5 Stap 2: project structuur

Nadat het build.gradle bestand aangemaakt is, is het de bedoeling om via IntelliJ IDEA het gradle bestand te importeren. Zie figuur 6.2. IntelliJ zal een dialoogvenster openen, waarbij het build.gradle bestand moet worden geopend. De IDE zal hierna het build.gradle bestand uitvoeren en het genereert enkele bestanden.



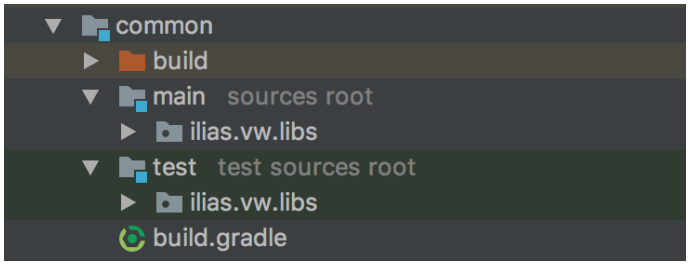
Figuur 6.2: Importeren van een project

In de huidige versie van Kotlin/Native worden de overige mappen nog niet automatisch

aangemaakt. Alle andere mappen, die te zien zijn in figuur 5.2 moeten handmatig aangemaakt worden. Deze mappenstructuur zal geleidelijk aan opgebouwd worden naargelang de stappen vorderen.

6.6 Stap 3: Common map

In de common map wordt alle gemeenschappelijke code geschreven. De inhoud van deze map moet aan een bepaalde structuur voldoen, zie figuur 6.3. In sectie 6.6.2 wordt uitgelegd waarom deze structuur zo belangrijk is.



Figuur 6.3: Common module structuur

6.6.1 Main en test map

Zoals te zien is op figuur 6.3 heeft zowel de main als test map enkele subpackages. Deze beginnen met het groupId dat ingesteld is in sectie 6.4.4 met daarin nog eens een 'libs' package. De naam van deze 'libs' package is vrij te kiezen, maar in dit voorbeeldproject wordt altijd 'libs' gebruikt.

De main map zal alle gemeenschappelijke klassen bevatten. Dit zijn zowel de klassen die expect gedeclareerd zijn als de klassen die een concrete implementatie hebben. Zie sectie 5.4.1 voor uitleg over het expect keyword.

De test map zal alle gemeenschappelijke test klassen bevatten, gebruikmakend van de klassen in de main map.

6.6.2 Build.gradle

Natuurlijk moet de common module ook gecompileerd worden door Kotlin/Native. Hiervoor is er een build.gradle nodig.

```
apply plugin: 'kotlin-platform-common'

sourceSets {
 main.kotlin.srcDirs += 'main/'
 test.kotlin.srcDirs += 'test/'
}
```

```
dependencies {
 compile "org.jetbrains.kotlin:kotlin-stdlib-common:$kotlin_version"
 testCompile "org.jetbrains.kotlin:kotlin-test-annotations-common:
    $kotlin_version"
 testCompile "org.jetbrains.kotlin:kotlin-test-common:$kotlin_version"
}
```

De bovenste regel in de build.gradle geeft aan dat de kotlin-platform-common plugin gebruikt zal worden. Deze zal verantwoordelijk zijn voor het compileren en delen van alle gemeenschappelijke code over de verschillende platformen.

Er worden ook twee sourceSets toegevoegd. SourceSets vertegenwoordigen een logische groep van Java/Kotlin bronnen. Kotlin/Native zal steeds zoeken naar code in de map src/kotlin/main of src/kotlin/test, maar hier geven we aan dat code te vinden is in de main en test map en dus niet onder een src/kotlin/ map.

Tenslotte worden er drie dependencies toegevoegd. De stdlib is verantwoordelijk verlenen van toegang tot de standaard bibliotheek van Kotlin. Hierdoor hebben we toegang tot collections, streams, annotations en nog veel meer. Er worden nog twee test dependencies toegevoegd, één verantwoordelijk voor de annotations, de andere voor het opstellen en uitvoeren van de testen, zie sectie 6.13 voor meer informatie over het opstellen van testen.

6.7 Common code

Uitwerking van de common code van de voorbeeldapplicatie.

6.7.1 Cart

Implementatie van de Cart klasse in de common module. Dit is een doodnormale Kotlin klasse en heeft weinig Kotlin/Native specifiek.

```
package ilias.vw.libs

class Cart constructor(name: String) {
 var name: String = name
 private var cartLines: List<CartLine> = mutableListOf()

 fun getCartLines(): List<CartLine> {
  return this.cartLines
 }

 fun addCartLine(cartLine: CartLine) {
  for (item in cartLines) {
   if (item.getProduct().getName() == cartLine.getProduct().getName()) {
```

```
    item.add(cartLine.getQuantity())
    return
   }
  }
  this.cartLines += cartLine
 }

 fun getTotalPrice(): Double {
  var totalPrice = 0.0

  for (line in cartLines) {
   totalPrice += line.getTotalPrice()
  }

  return totalPrice
 }

 fun removeCartLine(cartLine: CartLine) {
  this.cartLines -= cartLine
 }
}
```

6.7.2 CartLine

Implementatie van de CartLine klasse in de common module. Ook deze klasse heeft niks Kotlin/Native specifiek.

```
package ilias.vw.libs

class CartLine {
 private val product: Product
 private val quantity: Int

 constructor(product: Product, quantity: Int) {
  this.product = product
  this.quantity = quantity
 }

 fun getProduct(): Product {
  return this.product
 }

 fun getQuantity(): Int {
  return this.quantity
 }

 fun getTotalPrice(): Double {
```

```
  return product.getPrice() * quantity
 }
}
```

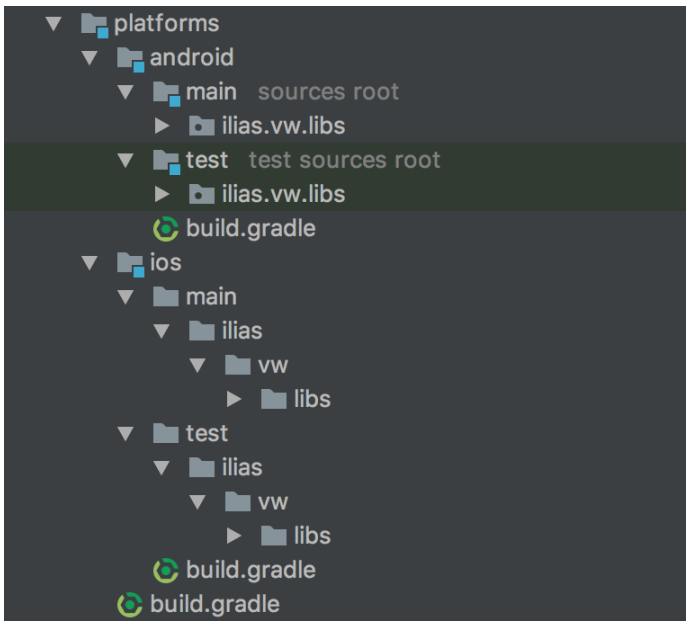
6.7.3 Product

De product klasse is een expect klasse. Dit wil zeggen dat we voor iedere platform een andere implementatie hebben voor deze klasse. Maar voor iedere platform verwachten we wel dat deze de getName, getPrice en getDescription methodes heeft.

```
package ilias.vw.libs

expect class {
 fun getName(): String
 fun getPrice(): Double
 fun getDescription(): String
}
```

6.8 Stap 4: platforms folder

De platform folder bevat enkele subfolders, één subfolder per platform waarvoor men wenst te ontwikkelen. Bij dit voorbeeld zijn er dus twee subfolders aanwezig, namelijk Android en iOS. Zie figuur 6.4. Per submap, hebben we opnieuw twee submappen, namelijk main en test. Net zoals bij de common map.



Figuur 6.4: Platforms module structuur

Zowel de main en test map bevatten elk nog een map. Bij Android is dit een package die begint met het groupId, zie sectie 6.4.4, en eindigt met een willekeurige naam. In dit voorbeeld is dit opnieuw 'libs'. Dit dient identiek te zijn aan de naam van de package gekozen in de common map, zie sectie 6.6.1. In het geval van iOS zijn dit geen packages maar eerder een mappenstructuur.

6.8.1 Platforms Android-map

Build.gradle

```
apply plugin: 'kotlin-platform-jvm'

repositories {
 jcenter()
 maven { url "http://kotlin.bintray.com/kotlinx" }
}

sourceSets {
 main.kotlin.srcDirs += 'main/'
 test.kotlin.srcDirs += 'test/'
}

dependencies {
 compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
 expectedBy project(":common")

 testCompile "junit:junit:4.12"
 testCompile "org.jetbrains.kotlin:kotlin-test-junit: $kotlin_version"
 testCompile "org.jetbrains.kotlin:kotlin-test:$kotlin_version"
}
```

In deze build.gradle wordt er gebruik gemaakt van de kotlin-platform-jvm plugin. Android-applicaties maken nog steeds gebruik van de JVM en dus ook deze Android-applicatie. De JVM voert een aantal noodzakelijk taken uit die ook Kotlin nodig heeft in Android-applicaties, zie sectie 4.5 voor meer uitleg over de JVM. De sourceSets worden ingesteld. Tenslotte worden alle dependencies geladen en wordt er aangegeven dat de common module verwacht wordt die code bevat.

Platformspecifieke code

De platformspecifieke code voor Android. De klasse product wordt voorzien van een afbeelding en aangezien drawables in Android gemapt worden naar een integerwaarde, wordt er een getal ingevuld in de constructor. De naam van het product wordt teruggeven met prefix 'Android'.

```
package ilias.vw.libs
```

```
import java.io.Serializable

actual class Product: Serializable {
 private val name: String
 private val price: Double
 private val description: String
 private val productImage: Int

 actual constructor(name: String, price: Double,
     description: String, productImage: Int) {
  this.name = name;
  this.price = price
  this.description = description
  this.productImage = productImage
 }

 actual fun getName(): String {
  return "Android: $name"
 }

 actual fun getPrice(): Double {
  return this.price
 }

 actual fun getDescription(): String {
  return this.description
 }

 actual fun getProductImage(): Int {
  return this.productImage
 }
}
```

6.8.2 Platforms iOS-map

Build.gradle

```
apply plugin: 'konan'

konanArtifacts {
 framework('SharediOS', targets: ['iphone', 'iphone_sim']){
  enableDebug true
  enableMultiplatform true

  srcDir 'main'
 }

 library('test-library') {
```

```
  enableMultiplatform true
  srcDir 'main'
 }

 program('shared-ios-test') {
  srcDir 'test'
  commonSourceSet 'test'
  extraOpts '-tr'
  libraries {
   artifact 'test-library'
  }
 }
}

dependencies {
 expectedBy project(':common')
}
```

Zoals reeds te lezen is in sectie 5.5 zal Kotlin/Native alle Kotlin code compileren naar een iOS framework. In de build.gradle wordt de naam van het framework en de toestellen (iPhone en iPhone simulators) dat men wenst te ondersteunen ingesteld.

De main map wordt als source directory ingesteld. Om ervoor te zorgen dat op iOS ook de testen kunnen worden uitgevoerd, wordt de main map ingesteld als bron van klassen die gebruikt kunnen worden voor alle testen en de test map wordt ingesteld als bron waar alle testen zijn gelokaliseerd.

Tenslotte wordt er opnieuw aangegeven dat er een common module wordt verwacht die alle gemeenschappelijke code bevat.

Platformspecifieke code

De platformspecifieke code voor iOS. In iOS is er de mogelijkheid om afbeeldingen in te laden aan de hand van de naam van de imageset. Daardoor heeft de constructor een productImage die van het type String is. De naam van het product wordt teruggegeven met prefix 'iOS'.

```
package ilias.vw.libs

actual class Product {
 private val name: String
 private val price: Double
 private val description: String
 private val productImage: String

 actual constructor(name: String, price: Double,
     description: String, productImage: String) {
  this.name = name;
```

```
  this.price = price
  this.description = description
  this.productImage = productImage
 }

 actual fun getName(): String {
  return "iOS: $name"
 }

 actual fun getPrice(): Double {
  return this.price
 }

 actual fun getDescription(): String {
  return this.description
 }

 fun getProductImage(): String {
  return this.productImage
 }
}
```

6.9 Stap 5: Android-map

In de Android-map is het de bedoeling om via Android Studio een nieuw project aan te maken. Bij het aanmaken van het project in Android Studio kan de locatie van het project worden opgeven. Deze locatie moet zodanig ingesteld zijn dat het Android-project een submap is van het Kotlin/Native project en waardoor het geïntegreerd zal worden in dit project.

6.9.1 settings.gradle

Voor we gebruik kunnen maken van de Kotlin klassen, moeten er eerst nog enkele wijzigingen gebeuren in de settings.gradle van het Android project. De settings.gradle wordt vervangen door onderstaande code:

```
include ':app'

include ':common'
project(":common").projectDir = new File("../common")

include ':platforms-android'
project(":platforms-android").projectDir = new File("../platforms/android")
```

Via de settings.gradle wordt er aangegeven om de common en platforms-android modules te includen in het project. Hierdoor worden de common en platformspecifieke Android modules geïntegreerd in het Android project.

6.9.2 build.gradle

Tenslotte moet er nog een kleine wijziging doorgevoerd worden in de build.gradle van het Android-project. In de dependencies tag moet er enkel en alleen volgende lijn toegevoegd worden in de dependencies tag:

```
implementation project(':platforms-android')
```

Hiermee wordt er aangegeven dat de specifieke Android implementatie te vinden is in platforms-android, waarvan de locatie is opgegeven in de settings.gradle. Vanaf nu kan er gebruik gemaakt worden van de klassen in de common module en de platformspecifieke Android-module.

6.10 Stap 6: iOS-map

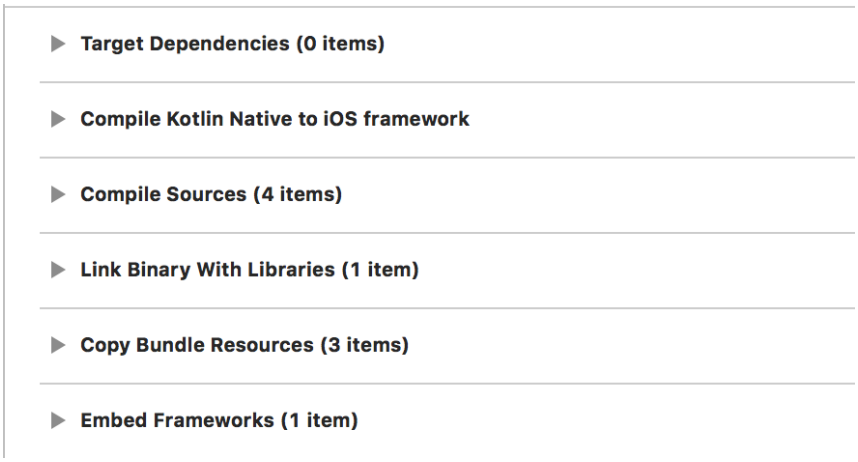
Net zoals bij de Android-map, is het de bedoeling om via Xcode een nieuw Xcode project aan te maken dat ook een submap is van het Kotlin/Native project. Men kan zowel kiezen voor een Objective-C als Swift project aangezien het gecompileerde iOS framework zowel in beide projecten kan gebruikt worden.

6.10.1 Gebruiken van het SharediOS framework

Vooraleer men in Xcode kan gebruik maken van het SharediOS framework, moeten er een aantal dingen aangepast worden in de build phases van het project. Uit figuur 6.5 kan worden afgeleid dat de build phases zijn aangepast. Volgende build phase moet worden toegevoegd in Xcode: Compile Kotlin Native to iOS framework.

In deze nieuwe build phase moet er een script, afkomstig van Gao (2018b), worden toegevoegd:

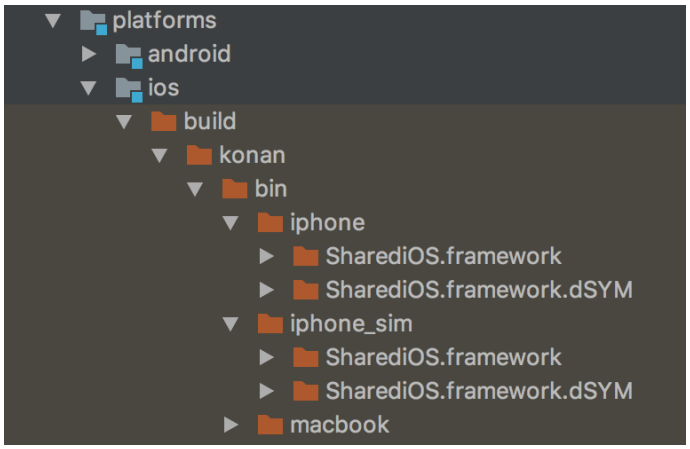
```
case "$PLATFORM_NAME" in iphoneos)
NAME=iphone;;
iphonesimulator)
NAME=iphone_sim;;
*)
echo "Unknown platform: $PLATFORN_NAME"
exit 1;;
esac

"$SRCROOT/../gradlew" -p "$SRCROOT/../platforms/ios" "build"
```



Figuur 6.5: iOS build phases

```
rm -rf "$SRCROOT/build/"
mkdir "$SRCROOT/build/"
cp -a "$SRCROOT/../platforms/ios/build/konan/bin/$NAME/" "$SRCROOT/build/"
```

Bij iedere gradle build van het Kotlin/Native project zal de build.gradle in de platforms/ios map ervoor zorgen dat de Kotlin code omgezet wordt naar een iOS framework. Dit framework vindt men terug in een submap van de build map (platforms/ios/build/). Zie figuur 6.6. Er wordt dus zowel voor een echt toestel (iphone map) als voor een simulator (iphone_sim map) een framework gegenereerd. Dit wordt gedaan omdat beiden op verschillende architecturen worden uitgevoerd.



Figuur 6.6: iOS build map

In bovenstaand build phase script wordt gekeken welk type toestel er wordt gebruikt, een echte iPhone of een simulator. Daarna zal het juiste framework gekopieerd worden naar de build folder in de map van het Xcode project. Bij iedere build en/of run van het Xcode project zal bovenstaand script worden uitgevoerd. Dit zal ervoor zorgen dat steeds het laatst gegenereerde framework gebruikt zal worden door het Xcode project.

Tenslotte zal het framework nog in de build phases 'Link Binary With Libraries' en 'Embed

Frameworks' moeten worden toegevoegd. Hierbij wordt er gelinkt naar het gekopieerde framework in de build folder van het Xcode project.

6.11 Aanspreken van code

6.11.1 Android

Om gebruik te kunnen maken van de Kotlin code in het Android-project moet men simpelweg volgende import toevoegen om alle klassen te kunnen gebruiken:

```
import ilias.vw.libs.*
```

6.11.2 iOS

Om gebruik te kunnen maken van het SharediOS framework in de ViewControllers moet het framework worden geïmporteerd door bovenaan in de ViewController volgende lijn toe te voegen:

```
import SharediOS
```

Het aanmaken van een object in iOS is nu heel simpel geworden. Zoals te zien is in onderstaande code heeft Kotlin/Native een prefix gegeven aan de klassen. Kotlin/Native zal steeds de hoofdletters uit de naam van het framework filteren, dat gekozen wordt in de build.gradle van de iOS-map, zie sectie 6.8.2. Stel dat het framework KotlinNativeFramework heet, dan zal de prefix KNF zijn en zal de product klasse KNFProduct heten. In dit voorbeeld heet het gecompileerde iOS framework SharediOS, waardoor de prefix SOS is.

```
var product: SOSProduct = SOSProduct(name: "Playstation 4", price: 399.95,
    description: "Playstation 4 gaming console", productImage: "ps4")
```

6.12 User interfaces

De user interface moet dus per platform opgebouwd worden. Dit is echter native Android en iOS en hier wordt niet dieper op ingegaan. Er is geen manier om één user interface te ontwikkelen voor beide platformen zoals bijvoorbeeld bij React/Native of Ionic. Dit kan zowel positief als negatief zijn. Het vraagt dubbel zoveel werk maar het geeft wel de mogelijkheid om verschillende accenten te leggen in de user interface per platform.

6.13 Testen

Een essentieel deel van object-oriented programming is het testen van de code. In het Kotlin/Native framework is het mogelijk om de aangemaakte klassen te testen. Zo is er de mogelijkheid om in de common map en per platform een test map toe te voegen. Een voorbeeld van een testklasse:

```
package ilias.vw.libs

import kotlin.test.*

class TestSampleCommon {
 @Test
 fun testCheckChartName() {
  val sample = Cart("test")
  val name = sample.name
  assertEquals(name, "test")
 }
}
```

Hierbij wordt er een cart aangemaakt met een naam 'test'. Daarna wordt de naam terug opgevraagd en wordt er gekeken of de naam die we initieel hebben meegegeven effectief gelijk is aan 'test'.

Aangezien er per platform, platformspecifieke code geschreven kan worden is het belangrijk om deze code ook te kunnen testen. Er kan naargelang dat de code per platform verschilt, verschillende testcode geschreven worden in de test map van het platform. Dit zorgt ervoor dat er geen ongeteste code in het project geraakt wat eventueel bugs kan bevatten.

Een voorbeeld voor deze proof-of-concept: er kan voor de iOS test gecheckt worden of de naam van het product begint met de prefix 'iOS' en voor Android kan er gecheckt worden of deze naam start met prefix 'Android'.

6.14 Huidige mogelijkheden

Zoals uit de proof-of-concept af te leiden is, is het reeds mogelijk om Kotlin/Native te gebruiken om een cross-platform applicatie te ontwikkelen. Aan de hand van de Kotlin/Native Gradle plugin is het mogelijk om niet alleen voor Android te ontwikkelen maar ook voor iOS. Cross-platform heeft bij Kotlin/Native niet dezelfde betekenis als bij een framework zoals React/Native. Het doel van Kotlin/Native is dus het delen van domeinlogica en deze hergebruiken over de verschillende te ondersteunen platformen. In deze vroege versie (0.6) van Kotlin/Native is het reeds mogelijk om dit te doen, maar de mogelijkheden blijven beperkt. Indien de applicatie geen speciale iOS bibliotheken nodig heeft, is er geen probleem.

Een ander belangrijk luik dat Kotlin/Native reeds ondersteunt is het aanmaken van testen voor de geschreven domeinlogica. Testen zijn en blijven een belangrijke onderdeel van object-oriented programming.

6.15 Beperkingen

Aangezien er nog geen officiële versie van Kotlin/Native is vrijgegeven is het logisch dat de mogelijkheden nog zeer beperkt zijn.

Er kan voor de platformspecifieke iOS code geen iOS specifieke bibliotheken gebruikt worden. In deze proof-of-concept was het de bedoeling om de shopping cart in zowel Android als iOS te cachen. In iOS moet het object dat gecached moet worden voldoen aan een bepaalde protocol, namelijk Codable. Kotlin/Native heeft geen kennis van deze protocollen waardoor het dus niet mogelijk is om het object te cachen in iOS. Voor Android kon dit wel, indien een bepaald object gecached moet worden volstaat het om deze klasse de interface Serializable te laten implementeren. Dit is niet mogelijk in de common module aangezien iOS het protocol Serializable niet kent. Dit is wel mogelijk in de platformspecifieke code van Android. Voor Android is het wel mogelijk om interfaces te implementeren maar dan moet de klasse in de common module expect worden gedeclareerd en moet de actual klasse, in de Android platforms map, deze interface implementeren. Dit is het geval bij het domeinmodel van deze proof-of-concept, zie figuur 6.1.

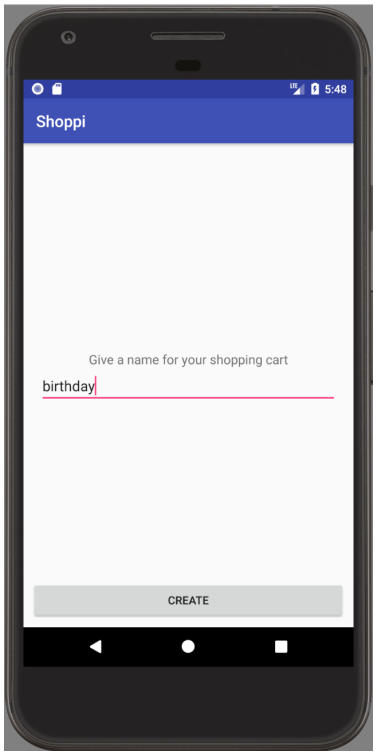
Het aanspreken van hardware, zoals de Camera, via Kotlin/Native is (nog) niet mogelijk. Momenteel moet het openen van bijvoorbeeld de camera geprogrammeerd worden in het Android en Xcode project, er is (nog) geen manier om dit te doen in bijvoorbeeld de common module.

Het gebruik van expect en actual klassen is momenteel ook nog zeer beperkt. Indien er een bepaalde klasse expect wordt gedeclareerd is het verder niet mogelijk om methodes of attributen toe te voegen aan deze klasse die niet expect zijn. Momenteel kan dus een expect klasse enkel en alleen expect methodes bevatten.

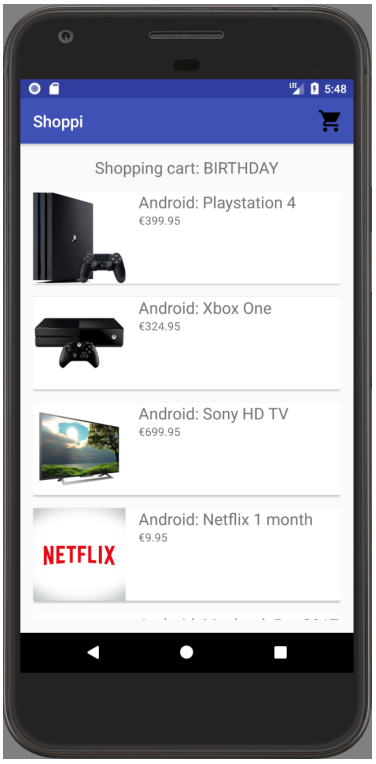
6.16 Proof-of-concept

Hieronder zijn de screenshots van de schermen van de proof-of-concept te vinden, opgesplitst per platform (Android en iOS).

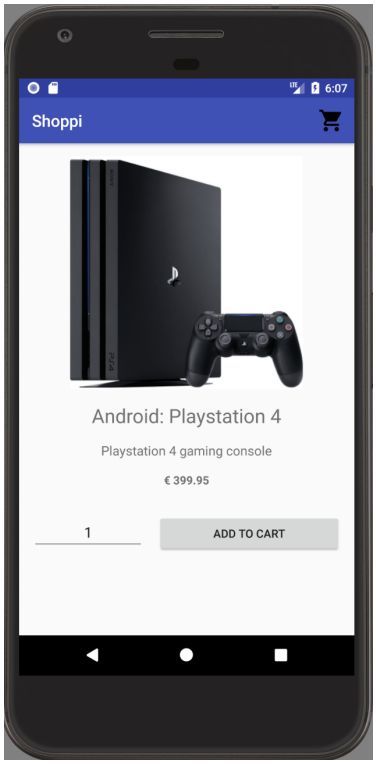
6.16.1 Android



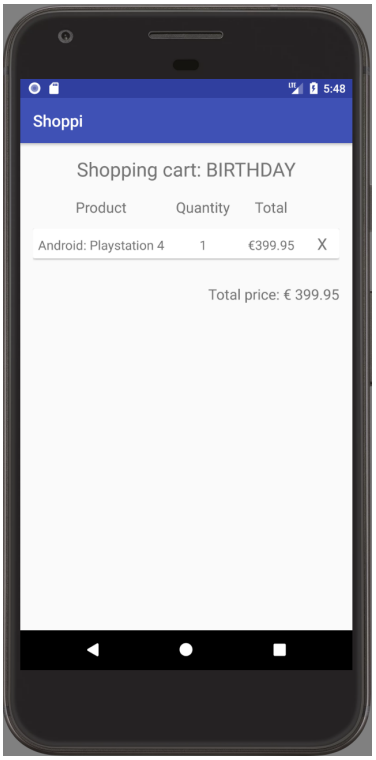
(a) Aanmaken van een shopping cart



(b) Overzicht van producten



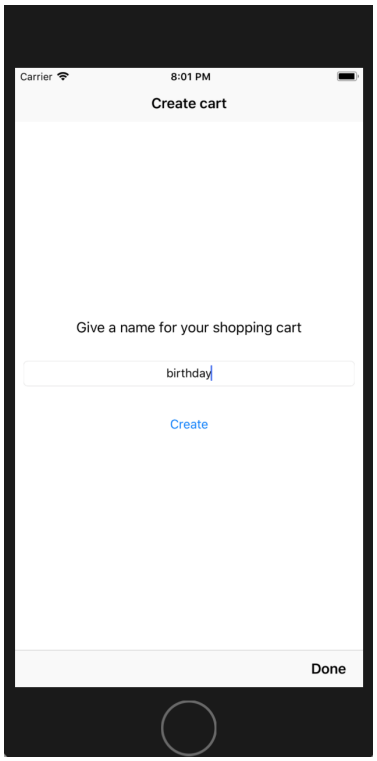
(c) Details van een product



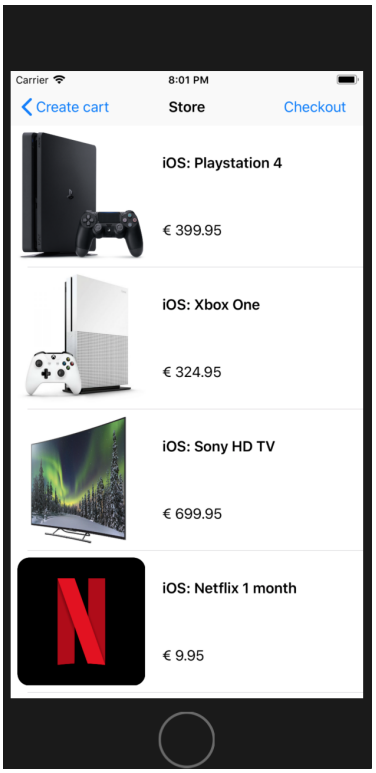
(d) Overzicht winkelmand

Figuur 6.7: Screenshots Android

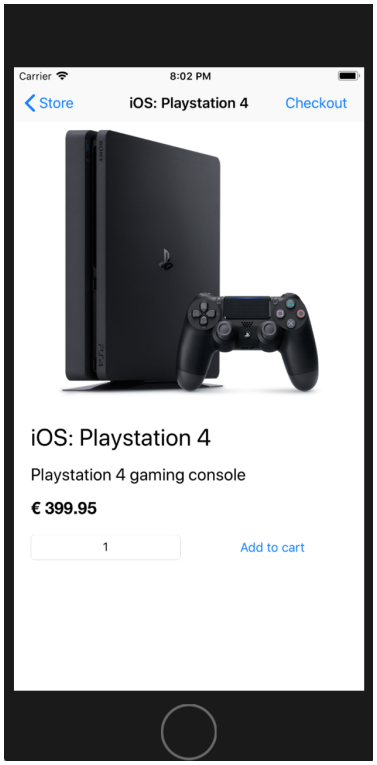
6.16.2 iOS



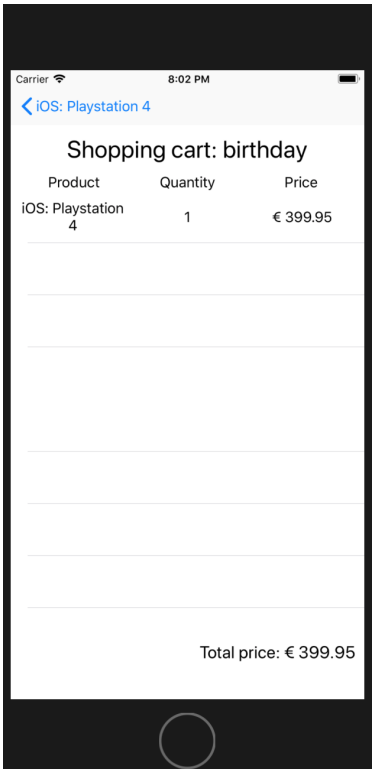
(a) Aanmaken van een shopping cart



(b) Overzicht van producten



(c) Details van een product



(d) Overzicht winkelmand

Figuur 6.8: Screenshots iOS

7. Conclusie

Uit de proof-of-concept, in sectie 6.16, valt te besluiten dat het maken van een applicatie met Kotlin/Native reeds mogelijk is. Het framework biedt de kans om domeinlogica te delen over de verschillende platformen die ondersteund moeten worden. In vergelijking met een framework zoals React/Native is Kotlin/Native een totaal ander type framework. De focus ligt op het delen van domeinlogica (cross-platform domeinlogica) en niet op het maken van cross-platform applicaties via één codebase. Het gebruiken van Kotlin/Native heeft enkel en alleen maar voordeel indien de applicatie over een grote domeinlogica beschikt. Het framework zal ervoor zorgen dat deze domeinlogica maar één maal moet worden opgebouwd, hiermee kan tijd bespaard worden en er is de mogelijkheid om platformspecifieke verschillen aan te brengen in de code. De user interfaces moeten per platform worden opgebouwd, dit kan zowel positief als negatief zijn. Er is de mogelijkheid om native applicaties te bouwen met een native uiterlijk, maar natuurlijk zijn er de dubbele kosten om een applicatie te bouwen en er is dubbel zoveel tijd nodig.

Daarnaast is de opzet van een Kotlin/Native project nog zeer omslachtig. Momenteel bestaat er geen plugin voor een IDE zoals IntelliJ om een volledig Kotlin/Native project automatisch te laten genereren. Er is kennis nodig van Gradle en de mappenstructuur moet overgenomen worden van de voorbeeldprojecten van JetBrains. Dit geeft aan dat een project niet zomaar wordt opgezet. Er is zo goed als geen officiële documentatie over de opzet en gebruik van Kotlin/Native. De enige bron van informatie zijn enkele ontwikkelaars die Kotlin/Native onderzoeken en gebruiken en de GitHub repository van JetBrains is zeker en vast aan te raden om te raadplegen.

Het framework is nog zeer jong. De huidige versie is 0.6 en de mogelijkheden zijn nog beperkt. Een aantal voorbeelden hiervan zijn:

- Er kan in de platformspecifieke iOS Kotlin code geen gebruik gemaakt worden van de iOS protcollen en libraries, wat bij Android wel mogelijk is.
- Het aanspreken van hardware via een uniforme manier is nog niet mogelijk.
- De mogelijkheden van de gemeenschappelijke code is nog zeer beperkt. Aangezien deze code zowel op iOS en Android moet kunnen draaien. Er kan dus bijvoorbeeld geen interface worden geïmplementeerd, zoals Serializable (een Android en Kotlin interface), aangezien dit niet gekend is in iOS.

Maar dit betekent niet dat dit framework nog niet gebruikt kan worden waarvoor het ontwikkeld is. Indien je domeinlogica simpel blijft en er is geen nood aan speciale interfaces (Android) of bepaalde protocollen (iOS), dan volstaan de huidige mogelijkheden van dit framework. Dit framework heeft zeker en vast potentieel maar er moet natuurlijk nog veel veranderd worden. Eerst en vooral is er nood aan een stabiele versie met alle basismogelijkheden. Daarna zal duidelijk worden of dit framework zal aanslaan bij de ontwikkelaars.

De bijdrage van dit onderzoek is een volledige handleiding over het gebruik van Kotlin/-Native. De basis over de LLVM compiler en Kotlin/Native komen aan bod en er is een documentatie over de volledige opzet van een project. Dit was toch het grote gebrek in de Kotlin/Native wereld.

De uitkomst van dit onderzoek was gedeeltelijk verwacht. Er werd verwacht dat de mogelijkheden van dit framework nog meer beperkt waren. Ik had niet verwacht dat er reeds de mogelijkheid was om Kotlin te gebruiken voor iOS development, aangezien Kotlin gebruik maakt van de JVM. Dit heeft het onderzoek zeer interessant gemaakt, iets waarvan je verwacht had dat de mogelijkheden nog zeer beperkt waren maar door ermee te werken constateer je dat er veel meer mogelijk is en je toch al mooie dingen kan maken.

Een verder verloop van dit onderzoek is zeker en vast nodig. Er kan onderzocht worden of er de mogelijkheid is om een plugin te integreren in een IDE, het aanspreken van hardware mogelijk is, zoals de camera, het gebruik van iOS protocollen en libraries in Kotlin en het importeren van het iOS framework in Xcode zonder een buildscript. Dit zijn momenteel vier belangrijke gebreken die in de toekomst verder onderzocht moeten worden.

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

Maar wat betekent nu cross-platform? Met cross-platform bedoelt men een systeem/software dat op verschillende platformen en/of besturingssystemen kan draaien. Er zijn al heel wat cross-platform frameworks op de markt. Denk maar aan Xamarin, React Native, Angular in combinatie met het Ionic framework. Momenteel is JetBrains bezig met het ontwikkelen van Kotlin Native waarmee het zou mogelijk zijn om zowel voor web, dekstop als voor mobile (Android, iOS) te ontwikkelen.

Kotlin is voor vele programmeurs nieuw. Sinds dit jaar biedt Google volledige ondersteuning voor het gebruik van Kotlin bij Android applicatieontwikkeling. Maar het gaat nog veel verder. JetBrains, de ontwikkelaars van Kotlin, werkt momenteel aan Kotlin Native, het framework dat moet zorgen voor cross-platform ontwikkeling. Maar Kotlin Native is nog heel jong. De huige versie is 0.6 en er bestaat nog maar weinig documentatie rond Kotlin Native. Het probleem echter bij dit framework is dus momenteel dat er heel weinig documentatie en programmeurs dus zelf moeten ontdekken hoe alles momenteel werkt. Dit kan via conferences, voorbeeldprojecten of youtube (kanaal van JetBrains). Maar wat zijn nu de mogelijkheden van Kotlin Native? Zijn er beperkingen? Hoe wordt de GUI opgebouwd? Hoe werkt LLVM, de compiler die alle code omzet naar machinetaal die op elk platform en/of besturingssysteem kan draaien en dus zonder het gebruik van de Java Virtual Machine?

A.2 State-of-the-art

Kotlin, de nieuwe programmeertaal. In 2011 voor het eerst aangekondigd door JetBrains. Zes jaar later biedt Google volledige ondersteuning voor Android applicatieontwikkeling. De voorbije jaren was er nog maar weinig bekend rond Kotlin. Er waren zo goed als geen boeken of andere literatuur op de markt. Hier is het laatste jaar verandering in gekomen. Er worden steeds meer en meer boeken geplubliceerd.

Op het internet zijn er reeds enkele reviews, onderzoeken en boeken beschikbaar. Hierin wordt er verteld waarom men juist wel of niet voor Kotlin moet kiezen. Zo vindt je her en der ook wel eens een vergelijkende studie tussen Java en Kotlin, maar meestal zijn deze niet zo sterk uitgewerkt.

Echter is er op het internet een studie beschikbaar (Magnus, 2017), die aangeeft waarom programmeurs Kotlin écht moeten gebruiken. Hieruit blijkt dat Kotlin volledig compatibel zou zijn met Java. Bestaande projecten die geschreven zijn in Java, kunnen dus zonder enig probleem verdergezet worden in Kotlin, dit noemt men de automigration. Bestaande Java frameworks kan men verder gebruiken indien men wenst te programmeren in Kotlin. De taal zou makkelijk te begrijpen zijn voor iedereen die ervaring heeft met OOP (Object-Oriented Programming). Volgens ontwikkelaars die ervaring hebben met Swift 4 zou Kotlin bijna een kopie zijn van Swift 4. Voorbeelden hiervan zijn: geen puntkomma's om de regels af te sluiten en declaratie van variabelen gebeurd op identieke manier.

```
var number: Integer = 5
var text: String = "text"
```

Bovenstaande code toont beide voorbeelden aan.

Een andere studie (AJ, 2016), beschrijft dan de nadelen van Kotlin. Kotlin zou geen gebruik maken van namespaces. De static modifier zou verdwenen zijn, maar een alternatieve manier is dan weer mogelijk via @JvmField. Het compileren van projecten in Android Studio zou veel tijd in beslag nemen en het gebruik van annotations zou nog niet volledig geoptimaliseerd zijn. Zo blijkt dat er toch nog wat werk is aan Kotlin. We zitten dan ook nog maar aan versie 1.2.

Wat betreft het gebruik van Kotlin als cross-platform programmeertaal (DZone, 2015), wordt er sterk gewerkt aan Kotlin Native. Momenteel is dit nog in development, JetBrains zelf biedt reeds enkele projecten aan waar ontwikkelaars met de slag kunnen. Zo geven ze de programmeurs de mogelijkheid om reeds kennis te maken met de werking van Kotlin Native ook al is er nog geen documentatie over dit framework. Er is echter ook een kleine tutorial op de website van JetBrains aanwezig om een simpele applicatie te bouwen die 'Hello World' op het scherm print.

Op het internet zijn er voldoende bronnen te vinden om applicaties te bouwen met Kotlin voor één platform. Dit kan gaan over een webapplicatie die gebruik maakt van het Spring framework, waarbij Spring gebruik maakt van de taalfeatures van Kotlin. Wat Android

betreft kan men gewoon in Android Studio een applicatie geschreven in Android converteren naar Kotlin. Tenslotte is er de mogelijkheid om voor een desktopapplicatie, KotlinFX te gebruiken. KotlinFX is een library die dezelfde functionaliteit biedt als JavaFX, een library om interfaces te bouwen.

Zoals te lezen is, is er op het internet voldoende informatie te vinden om applicaties te bouwen voor één platform. Maar is het nu ook mogelijk om via één codebase verschillende platformen aan te spreken? Kan je door het eenmalig schrijven van code, een applicatie ontwikkelen die zowel op Android als op iOS kan draaien? Dit is het grote verschil met mijn onderzoek.

A.3 Methodologie

Vooraleer er aan de slag wordt gegaan met het uittesten van Kotlin Native, of het analyseren van voorbeeldprojecten van JetBrains, is het belangrijk om te onderzoeken hoe het nu komt dat Kotlin Native op verschillende platformen kan draaien. Kotlin is oorspronkelijk, net zoals Java, een programmeertaal die gebruik maakt van de Java Virtual Machine. Het zal daarom dus belangrijk zijn om te onderzoeken waarop Kotlin Native draait, wat de taak is van de LLVM compiler. Dit zal gebeuren aan de hand van een literatuurstudie.

Daarna worden de voorbeeldprojecten van JetBrains, geschreven in Kotlin Native, geanalyseerd om de werking van Kotlin Native te achterhalen. De bedoeling is om te kijken naar bepaalde aspecten. Hoe wordt de user interface opgebouwd, wordt er per platform een UI opgebouwd of is er slechts één UI? Hoe kunnen specifieke native modules gebruikt worden?

Eenmaal de werking van Kotlin Native duidelijk is en voldoende gedocumenteerd is, is het het doel om zelf aan de slag te gaan met Kotlin Native en een kleine cross-platform applicatie te schrijven.

A.4 Verwachte resultaten

Ten eerste wordt verwacht om uit de literatuurstudie te kunnen besluiten wat de werking is van de LLVM compiler. Hierdoor is het duidelijk hoe Kotlin Native er voor zorgt dat Kotlin voor verschillende platformen kan gebruikt worden.

Anderzijds wordt een zeer duidelijke documentatie verwacht over de werking van Kotlin Native zodat mensen die geïnteresseerd zijn om Kotlin Native in de toekomst te gebruiken deze documentatie kunnen gebruiken.

A.5 Verwachte conclusies

Uit dit onderzoek verwacht ik te kunnen concluderen dat Kotlin meer dan geschikt is als cross-platform programmeertaal. Dit aangezien Kotlin volledig ondersteund wordt door Google wat betreft de ontwikkeling van Android applicaties. Er wordt nog steeds verder gebouwd aan Kotlin Native. Maar natuurlijk, Kotlin is een nieuwe taal, waar nog veel aan gesleuteld zal moeten worden. De toekomst ziet er enkel maar veel belovend uit. De kans is groot dat een groot aantal programmeurs zal overschakelen naar deze taal. Kotlin heeft zeker en vast de kracht om programmeurs, die al jaren in dit vak zitten en zich hebben vastgehecht aan een bepaalde programmeertaal, mee te slepen in het Kotlin-avontuur. Ook voor Swift programmeurs zal de aanpassing naar Kotlin niet groot zijn wegens de grote gelijkenissen tussen de twee programmeertalen.

Bibliografie

AJ, A. (2016). Why you shouldn't switch to Kotlin. https://medium.com/keepsafe-engineering/kotlin-the-good-the-bad-and-the-ugly-bf5f09b87e6f.

Andrey Breslav. (2017). Kotlin/Native Tech Preview: Kotlin without a VM. Verkregen van https://blog.jetbrains.com/kotlin/2017/04/kotlinnative-tech-preview-kotlin-without-a-vm/

AppBrain. (2018). Details of Kotlin. Verkregen van https://www.appbrain.com/stats/libraries/details/kotlin/kotlin

Avantica. (2017). What is Kotlin and how did we get here. Verkregen van https://www.avantica.net/blog/what-is-kotlin-and-how-did-we-get-here

Developine. (2017). Kotlin Native iOS Development using Multiplatform Project. Verkregen van http://developine.com/kotlin-native-ios-development-multiplatform-project/

Dmitry Jemerov. (2017). Kotlin 1.2 Released: Sharing Code between Platforms. Verkregen van https://blog.jetbrains.com/kotlin/2017/11/kotlin-1-2-released/

DZone. (2015). Kotlin and cross-platform. https://dzone.com/articles/kotlin-for-cross-platform-mobile-app-development.

Gao, A. (2018a). How to create Kotlin Native iOS project. Verkregen van https://www.infoworld.com/article/2614665/development-tools/google-positions-gradle-as-the-build-system-of-choice-for-android.html

Gao, A. (2018b). Use Kotlin to share native code among iOS, Android. Verkregen van https://github.com/Albert-Gao/kotlin-multuplatform-including-mobile

Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2017. (2018). *Statista*. Verkregen van https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

InOutput. (2017). Choosing the right mobile app for your project: Native vs cross-platform vs hybrid. Verkregen van http://inoutput.io/articles/development/choosing-the-right-mobile-app-for-your-project-native-vs-cross-platform-vs-hybrid

JetBrains. (2017a). Kotlin FAQ. Verkregen van https://kotlinlang.org/docs/reference/faq.html

JetBrains. (2017b). Kotlin on Android. Verkregen van https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/

JetBrains. (2018a). JetBrains Corporate Overview. Verkregen van https://resources.jetbrains.com/storage/products/jetbrains/docs/jetbrains_corporate_overview.pdf

JetBrains. (2018b). Kotlin - Creating Web Applications with Http Servlets. Verkregen van https://kotlinlang.org/docs/tutorials/httpservlets.html

JetBrains. (2018c). Using Kotlin for Server-side Development. Verkregen van https://kotlinlang.org/docs/reference/server-overview.html

Kohan, B. (2015). Mobile App Development Cost and Process. *Comentum*. Verkregen van http://www.comentum.com/mobile-app-development-cost.html

Krill, P. (2013). Google positions Gradle as the build system of choice for Android. Verkregen van https://www.infoworld.com/article/2614665/development-tools/google-positions-gradle-as-the-build-system-of-choice-for-android.html

Lattner, C. (2018). The architecture of Open Source Applications. *Aosabook*. Verkregen van http://www.aosabook.org/en/llvm.html

LLVM developer group. (2018). LLVM. Verkregen van https://llvm.org/

Magnus, V. (2017). Switch to Kotlin. https://medium.com/@magnus.chatt/why-you-should-totally-switch-to-kotlin-c7bbde9e10d5.

Newgenapps. (2018). Native vs Hybrid vs Cross-Platform Approach to Mobile App Development. Verkregen van https://www.newgenapps.com/blog/native-vs-hybrid-vs-cross-platform-approach-to-mobile-app-development

ResearchGate. (2009). Syntaxboom. Verkregen van https://www.researchgate.net/figure/Abstract-syntax-tree-of-the-while-loop_fig1_228792639

Techopedia. (2017). What does Cross Platform mean. Verkregen van https://www.techopedia.com/definition/17056/cross-platform

Techopedia. (2018a). What does Bytecode mean? Verkregen van https://www.techopedia.com/definition/3760/bytecode

Techopedia. (2018b). What does Java Virtual Machine (JVM) mean? Verkregen van https://www.techopedia.com/definition/3376/java-virtual-machine-jvm

TechTarget. (2011). What is a hyrbid app? Verkregen van https://searchsoftwarequality.techtarget.com/definition/hybrid-application-hybrid-app

TechTarget. (2015). What is a framework? Verkregen van https://whatis.techtarget.com/definition/framework

TechTarget. (2018). What is a native app? Verkregen van https://searchsoftwarequality.techtarget.com/definition/native-application-native-app

Why JetBrains Invented and Promotes Kotlin. (2017). *TechYourChance*. Verkregen van https://www.techyourchance.com/jetbrains-invented-promotes-kotlin/

Why JetBrains needs Kotlin. (2011). *JetBrains*. Verkregen van https://blog.jetbrains.com/kotlin/2011/08/why-jetbrains-needs-kotlin/