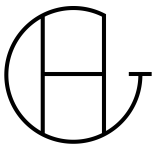


HoGent

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. 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 onmiddelijk dingen beginnen opzoeken over deze programmeertaal. Toen het tijd was om een bachelorproefvoorstel in te sturen wou ik zeker iets doen met mobiele applicaties en wanneer mijn stagebedrijf nog eens een onderwerp over Kotlin en cross-platforme applicaties doorstuurde, was de keuze snel gemaakt.

Tenslotte wil ik nog enkele personen 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, native applicaties. Dit vergde zeer veel werk en was heel kostelijk voor veel bedrijven. Als reactie hierop zijn de hybride frameworks uitgevonden, denk maar aan: React, Xamarin en Ionic met Angular. Bij deze frameworks moest er maar eenmalig code geschreven worden, de user interface was hetzelfde voor alle platformen en het framework zorgde er voor dat je applicatie op elk besturingssysteem kon draaien. We mogen misschien nog een framework toevoegen aan het lijstje van hybride frameworks, namelijk Kotlin/Native.

Kotlin is een nieuwe programmeertaal die geintroduceerd 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. Zij staan te springen om aan de slag te gaan met Kotlin/Native. De enige beperking die zij momenteel ervaren is documentatie. Van dit onderzoek wordt verwacht dat door het gedetailleerd bestuderen van Kotlin/Native een mooie documentatie kan opgeleverd worden over de werking van dit framework waar iedere ontwikkelaar met aan de slag kan gaan.

In dit onderzoek werd de werking van de LLVM compiler en Kotlin/Native onderzocht. 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 zijn van

[1] Java Virtual Machine

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 Kotlin/Native.

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. 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 gebruikt kan worden om business logica te delen over verschillende platformen. Het is reeds mogelijk om Kotlin code 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, en 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 men specifiek per platform moet programmeren?

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Inleiding

De ontwikkeling van mobiele applicaties is een zeer 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%. Dit wil dus zeggen dat meer dan 85% van de mensen die een smartphone heeft, een toestel heeft met het Android besturingssyteem.

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.

0.1 Probleemstelling

Vanuit dit onderzoek zal blijken hoe goed Kotlin is als cross-platform ontwikkelingstaal en wat de mogelijkheden zijn van Kotlin om te programmeren voor verschillende platformen. Deze vraag kwam vanuit mijn stagebedrijf (Endare BVBA). 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 mede-stagiair tijdens zijn stage bij Endare, gewerkt met React Native. Een framework waarbij je via één codebase native applicaties kan schrijven voor iOS en Android. Tegenwoordig worden worden er nog maar weinig native applicaties gebouwd.

Waarom? Voor veel bedrijven zijn native applicaties een zéér hoge kost. Je hebt enerzijds twee keer (indien we voor iOS en Android ontwikkelen) een kost om de applicaties te bouwen, anderszijds heb je twee keer een kost om de applicaties te onderhouden of eventueel uit te breiden. Maar er zijn natuurlijk wel redenen waarom bedrijven nu net wel native applicaties willen in plaats van hybride applicaties. Zo zal men bij native applicaties gebruik kunnen maken van alle features van het besturingssysteem, wat niet altijd kan bij hybride applicaties, en de performance zou ook een stuk beter zijn.

Tegenwoordig heb je meer dan genoeg keuze om hybride applicaties te maken. Zo is er:

- Ionic
- Cordova
- React/Native
- Xamarin

En ga zo maar door... 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 het stagebedrijf, Endare, heeft deze bachelorproef een grote meerwaarde aangezien zij iedere dag met cross-platform frameworks aan de slag moeten.

0.2 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?

0.3 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

0.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 1 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.

In Hoofdstuk 2 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 3 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 4 zal Kotlin/Native bestudeerd worden. Hierin zal de werking van Kotlin/Native gedocumenteerd worden.

In Hoofdstuk 5 zal er praktisch een Kotlin/Native project worden uitgewerkt.

In Hoofdstuk 6, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

1. 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, web-applicatie 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 met de laatste nieuwigheden rond Kotlin.

1.1 Wat is Kotlin

Kotlin is een open-source programmeertaal die object-georiënteerde en functionele programmatie features combineert. Kotlin is ook een statically typed programmeertaal. Dit betekent het type van de variabele is toegekend wanneer de code wordt gecompileerd. Javascript is een dynamically typed programmeertaal, waarbij je aan een variabele verschillende types kan toekennen. 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 (Tjsechië) 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, Objective-C, C++, C# en JavaScript.

[1] Integrated Development Environment

1.2 Kotlin en Android

In 2017 heeft Google bekend gemaakt dat het Kotlin volledig zou ondersteunen voor Android applicatieontwikkeling. Een Kotlin project maken in Android Studio is dan ook heel gemakkelijk. In Android Studio 3.0 heb je bij het aanmaken van een project de mogelijkheid om direct de ondersteuning voor Kotlin in te schakelen. Hierdoor zal het aangemaakte project onmiddelijk in Kotlin geschreven zijn.

Het is mogelijk 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 geinstalleerd.

1.3 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, is te beslissen door de developer.

1.4 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.

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.

Kotlin-applicaties kunnen worden geïmplementeerd op elke host die Java-webapplicaties ondersteunt, inclusief Amazon Web Services, Google Cloud Platform en veel meer.

1.5 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 moeten om éénmalige business logica 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 JVM, zie sectie 1.6 voor meer info. De bijnaam die gegeven wordt aan Kotlin/Native is *Konan*.

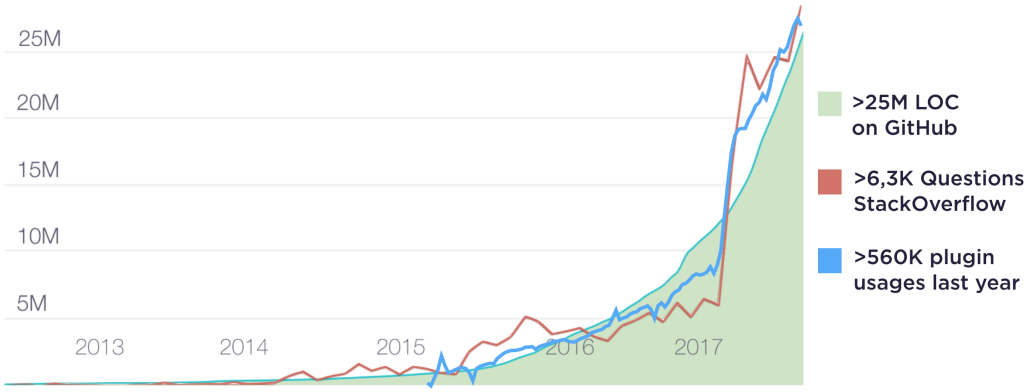
1.6 Compiler

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, 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 3 verder besproken.

1.7 Het gebruik van Kotlin

Het gebruik van Kotlin is gedurende de jaren zeer sterk gestegen. Op de blog van JetBrains zijn grafieken te vinden waarmee men aantoont dat de populariteit van Kotlin enkel maar stijgt.

Figuur 1.1 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. We kunnen hieruit besluiten dat het gebruik van Kotlin sterk stijgt de laatste drie jaren.



Figuur 1.1: Hoeveelheid Kotlin code op GitHub (Dmitry Jemerov, 2017)

Figuur 1.2: Kotlin user groups in de wereld (Dmitry Jemerov, 2017)

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

Figuur 1.2 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.

1.8 Waarom Kotlin?

Maar waarom heeft JetBrains nu besloten 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 uigevonden 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.

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?

2. Methodologie

Vooraleer er aan de slag kon worden gegaan met het uittesten van Kotlin Native, of het analyseren van voorbeeldprojecten van JetBrains, was 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 is daarom nuttig te onderzoeken hoe het mogelijk is gemaakt om Kotlin cross-platform te gebruiken. Dit is gebeurd aan de hand van een literatuurstudie. Het was niet nodig om dit praktisch te doen aangezien dit geen meerwaarde zou bieden aan deze bachelorproef. Kotlin/Native maakt gebruik van de LLVM compiler. Deze literatuurstudie is uitgevoerd aan de hand van de aosabook van LLVM, Lattner (2018), een zeer uitgebreide documentatie.

Daarna werden de voorbeeldprojecten van JetBrains, geschreven in Kotlin/Native, geanalyseerd om de werking van dit framework te achterhalen. Hierbij was het de bedoeling om te onderzoeken hoe het mogelijk was, om in een zeer vroeg stadium van Kotlin/Native, reeds aan de slag te gaan met dit framework. Dit is begonnen met het downloaden van de code van de repositories van Kotlin/Native en de Kotlin multiplatform projects. Dit was niet gemakkelijk. Vaak waren er problemen met verkeerde versies of programma's die ontbraken op de computer. Na enige tijd kon er aan de slag worden gegaan met de projecten. In eerste instantie werden er dingen gewijzigd in het originele project van Kotlin/Native zelf om te kijken hoe alle modules, klassen, gradle bestanden werkten en gelinkt waren.

Eenmaal de werking van Kotlin Native duidelijk was, was het de bedoeling om zelf aan de slag te gaan met dit framework en een kleine cross-platform applicatie te schrijven. Hierbij werd er eerst een Kotlin/Native project opgezet, wat toch enige tijd in beslag heeft genomen. Eens het project was opgezet kon de applicatie gebouwd worden. Het prototype

dat is gemaakt is een kleine shopping cart applicatie.

3. 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 literatuurstudie is uitgevoerd aan de hand van de aosabook van LLVM, Lattner (2018), een zeer uitgebreide documentatie.

3.1 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 commerciele en open-source projecten en zelfs in academisch onderzoek.

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 geintegreerd. 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.

3.1.1 Deelprojecten LLVM

Enkele voorbeelden van deelprojecten van de LLVM developers group:

- **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.

3.2 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?

3.2.1 De werking van een standaard compiler

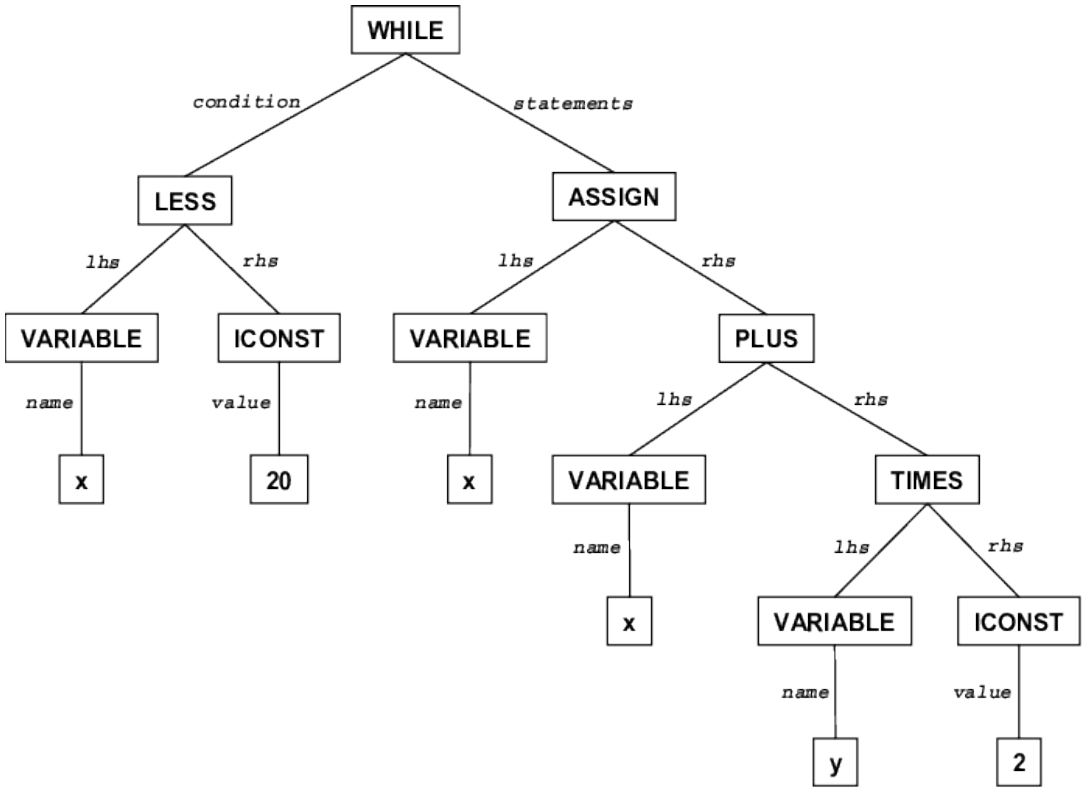
Het meest populaire ontwerp voor een statische compiler, zoals de meeste C-compilers, is het driefasenontwerp waarbij de belangrijkste componenten de **front-end**, de **optimizer** en de **back-end** zijn.

Figuur 3.1: Driefasenontwerp (Lattner, 2018)

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 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 JVM is ook een implementatie van dit driefasenontwerp. Een voorbeeld van een syntaxboom is te zien in figuur 3.2.



Figuur 3.2: 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 uitvoeringtijd 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

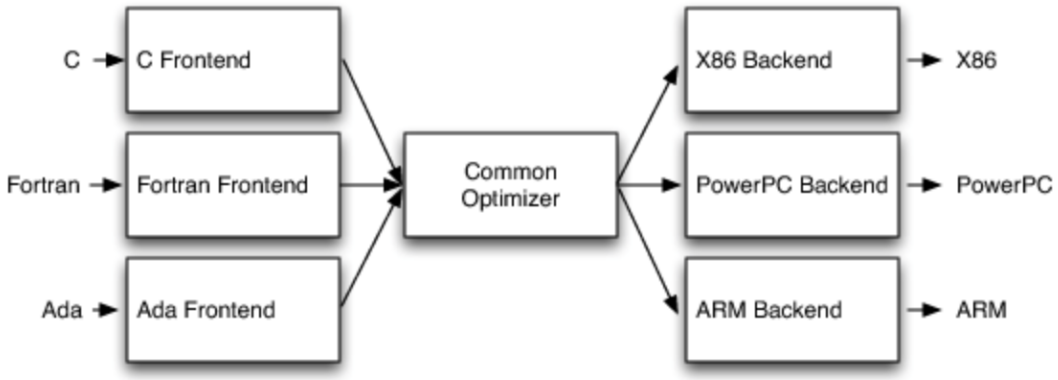
De back-end, ook bekend als de codegenerator, gaat de code gaan mappen naar een instructieset. Naast het maken van deze instructieset is het ook verantwoordelijk om ervoor te zorgen dat deze instructieset gebruik maakt van de ongewone kenmerken van de ondersteunde architectuur. Iedere architectuur is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

3.2.2 Meerdere front- en backends

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. Op figuur 3.3 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 doelarchtectuur zou ondersteunen.



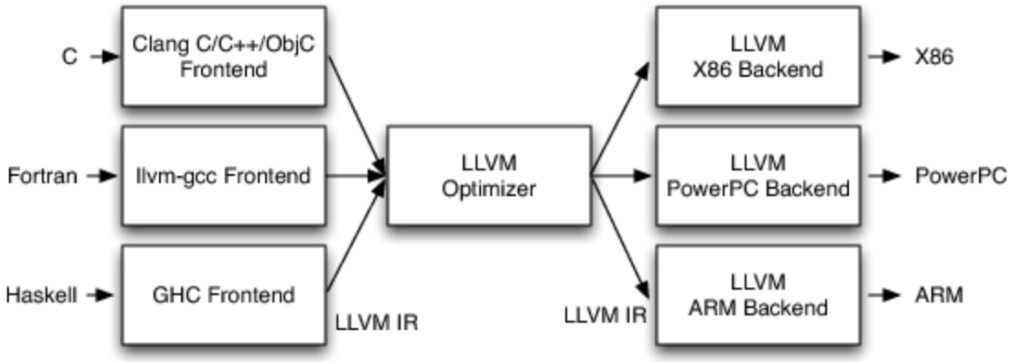
Figuur 3.3: Meerdere front- en backends (Lattner, 2018)

3.3 Verschil met LLVM

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 IR[1], 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.

[1] Intermediate Representation

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 optimalizeren en stuurt de IR code naar de back-end die dient als codegenerator met als resultaat machinecode.



Figuur 3.4: LLVM implementatie van driefasenontwerp (Lattner, 2018)

3.4 LLVM en Kotlin

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.

3.5 LLVM en JVM

De LLVM is een low-level register gebaseerde compilatie technologie. Het is ontworpen om onderliggende hardware abstraheren en een lijn te trekken tussen de front-end en de back-end van een compileerprogramma.

De JVM is een high-level stack gebaseerde virtuele machine. De JVM biedt garbage collection aan, heeft kennis van objecten en virtuele methode aanroepen. JVM biedt dus een veel hogere infrastructuur voor taalinteroperabiliteit.

Het grote verschil tussen LLVM en JVM is dus het niveau waarop beide werken.

4. 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 door het delen van domeinlogica.

4.1 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.

Weglaten? Kotlin Native ondersteunt volledig de interoperabiliteit met native code. Wat betreft het gebruik van Kotlin libraries, deze zijn volledig ter beschikking van de ontwikkelaar. Indien een bepaalde bibliotheek niet ondersteund wordt, bestaat er een tool om een tussenliggende bibliotheek te genereren van een C-header bestand, waardoor deze bibliotheek toch gebruikt kan worden. Op macOS en iOS wordt samenwerking met Objective/C-code ook ondersteund.

Kotlin Native is momenteel heel jong en zit nog in volledige ontwikkeling. Er zijn echter enkele preview-releases aanwezig om te proberen. De ondersteuning voor de IDE's is beschikbaar als een plugin voor CLion.

4.2 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 interfaces, 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
- Opzet van dit soort projecten is (momenteel) niet gemakkelijk voor de onervaren gebruiker

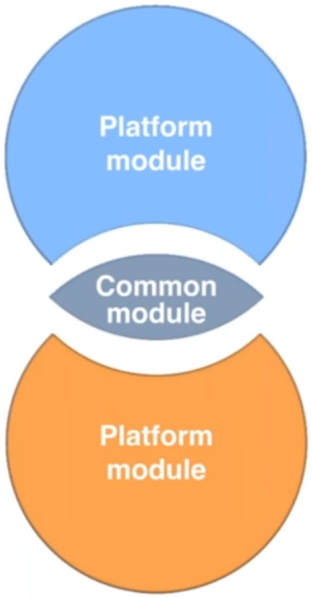
Figuur 4.1 toont hoe Kotlin/Native domeinlogica deelt over de verschillende platformen heen. Er wordt gebruik gemaakt van een **common** module. In deze module bevindt zich de gemeenschappelijke code, die gedeeld wordt met alle platformen. Daarnaast is er per platform dat je wenst te ondersteunen een aparte platformmodule. Deze platformmodules maken gebruik van de common module en men kan in deze platformmodules platformspecifieke code schrijven. Zie sectie 4.4 voor een meer technische uitleg.

4.3 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 van een Kotlin Native project. Er is namelijk geen IDE die een volledig Kotlin Native project voor genereert, daarom is het essentieel dat er vanaf het begin een goede mappenstructuur wordt gebouwd. Figuur 4.2 toont een goede structuur van een Kotlin/Native project.

Op deze figuur zijn dus een aantal mappen te vinden:

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



Figuur 4.1: Het delen van code in Kotlin Native (Developine, 2017)

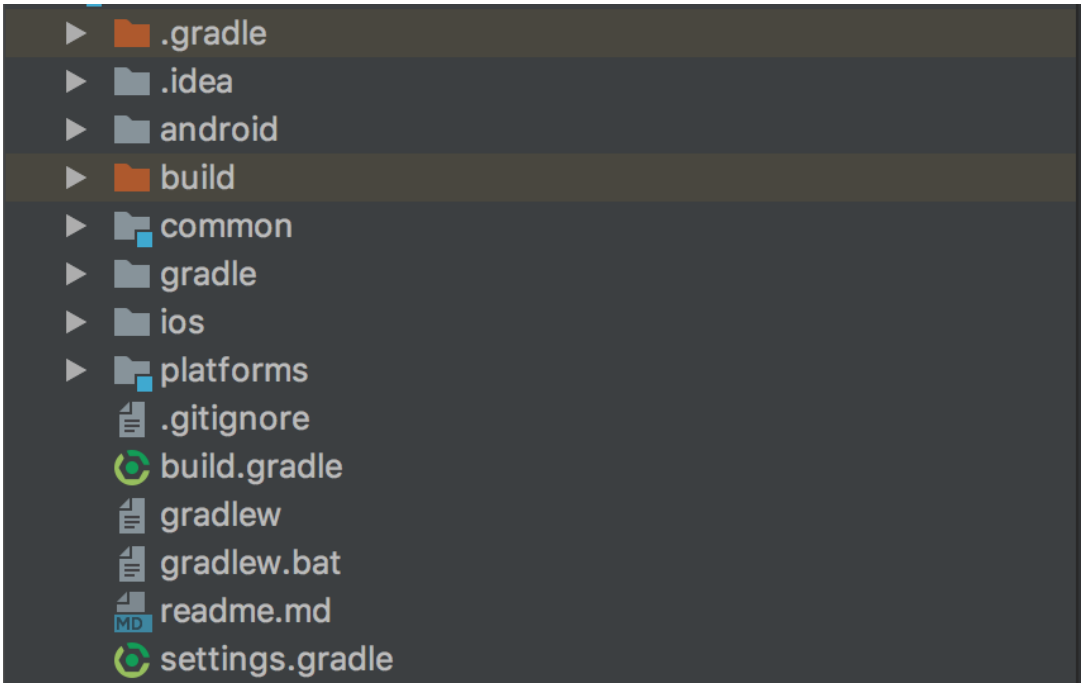
- **android** bevat het volledige Android project.
- De **build** map bevat informatie over de laatste uitgevoerd build.
- De **common** map bevat gemeenschappelijke code dat over de verschillende platformen wordt gedeeld.
- De **gradle** map, automatisch gegenereerd, bevat een properties bestand. In 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 4.2 voor meer informatie.

4.4 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 keyword **expect** gebruikt. Het keyword zegt bijna zelf waarvoor het gebruikt moet worden: we 'verwachten' een klasse ABCMethods voor ieder platform. Het expect keyword kan ook vergeleken worden met een standaard interface.

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



Figuur 4.2: Kotlin/Native structuur

```
}
```

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.

4.5 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"
        }
}
```

4.6 Het gebruik van Kotlin code

Maar hoe kunnen we nu gebruik maken van de gemeenschappelijke en platformspecifieke code? 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 je Android acitivity. 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 5.9 voor meer informatie.

5. Praktische uitwerking Kotlin/Native

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.

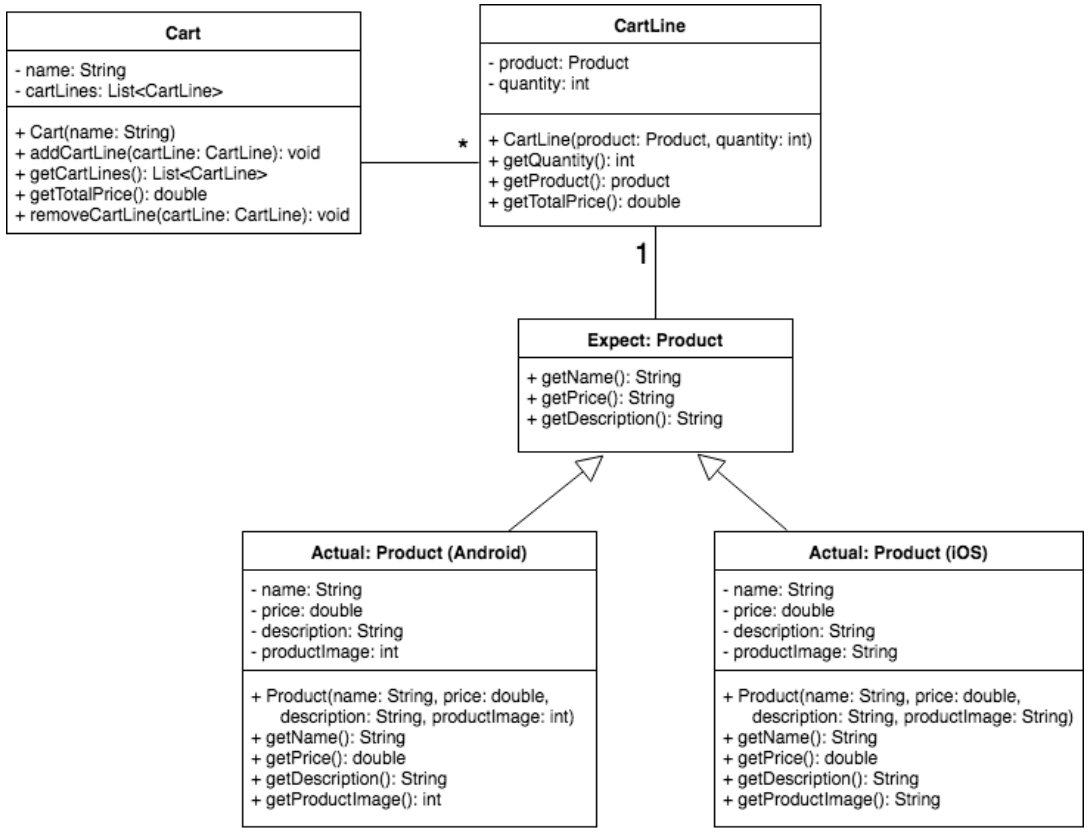
5.1 Domeinmodel

Aangezien het doel van Kotlin/Native het delen van business logica is, is het handig om op voorhand na te denken over het domeinmodel. Welke klassen heb ik nodig, welke attributen en methodes moeten deze klassen hebben, is er eventueel een verschil in klassen tussen Android en iOS? Een domeinmodel geeft je ook een duidelijk overzicht over je applicatie. Zie figuur 5.1 om het domeinmodel te zien van deze voorbeeldapplicatie.

5.2 Requirements

Om gebruik te kunnen maken van Kotlin/Native voor het ontwikkelen van cross-platform applicaties (iOS & Android), 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).



Figuur 5.1: Klassendiagram domeinlogica

- **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 geïnstalleerd bij het instellen van IntelliJ IDEA of Android Studio.

5.3 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 je 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"
  }
}
}
```

5.3.1 Versies

Bovenaan geven we aan welke versie van Kotlin en Kotlin/Native we willen gebruiken. 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/Native en nieuwe dingen worden continu gepushed 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.

5.3.2 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.

5.3.3 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 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.

5.3.4 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.

5.4 Stap 2: project structuur

Nadat het build.gradle bestand gemaakt is, is het de bedoeling om via IntelliJ IDEA het gradle bestand te importeren. Zie figuur 5.2. 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.

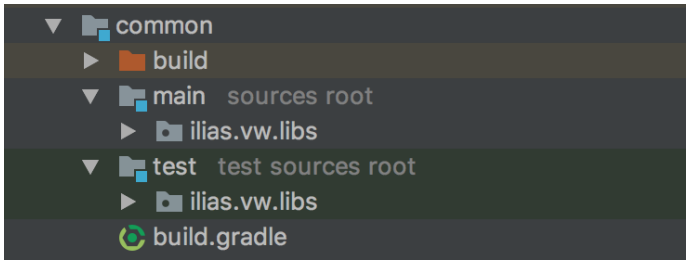
In de huidige versie van Kotlin/Native worden de overige mappen nog niet automatisch aangemaakt. Alle andere mappen, die te zien zijn in figuur 4.2 moeten handmatig aangemaakt worden. Deze mappenstructuur zal geleidelijk aan opgebouwd worden naargelang de stappen vorderen.

5.5 Stap 3: Common map

In de common map wordt alle gemeenschappelijke code geschreven. In deze map moet er aan een bepaalde structuur worden voldaan, zie figuur 5.3.

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

Figuur 5.2: Importeren van een project



Figuur 5.3: Common module structuur

5.5.1 Build map

De build map bevat alle gecompileerde bestanden die genereerd worden door Kotlin/Native en worden bij iedere gradle run en/of build aangemaakt.

5.5.2 Main en test map

Zoals te zien is op figuur 5.3 heeft zowel de main als test map enkele subpackages. Deze beginnen met het groupId dat ingesteld is in sectie 5.3.4 met daarin nog eens een 'libs' map. De naam van deze 'libs' map is vrij te kiezen, maar in dit voorbeeld wordt altijd 'libs' gebruikt.

De main map zal alle gemeenschappelijke klassen bevatten (expect en gewone klassen, zie sectie 4.4).

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

5.5.3 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 we gebruik maken van de kotlin-platform-common plugin. 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.

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, ... Er worden nog twee test dependencies toegevoegd, één verantwoordelijk voor de annotations, de andere voor het opstellen en uitvoeren van de testen, zie sectie 5.12 voor meer informatie over testen.

5.6 Common code

Uitwerking van de common code van de voorbeeldapplicatie.

5.6.1 Cart

```
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
 }
}
```

5.6.2 CartLine

```
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
 }
}
```

5.6.3 Product

De product klasse is een expect klasse. Dit wil zeggen dat we voor iedere platform een andere implementatie hebben van 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
}
```

5.7 Stap 4: platforms folder

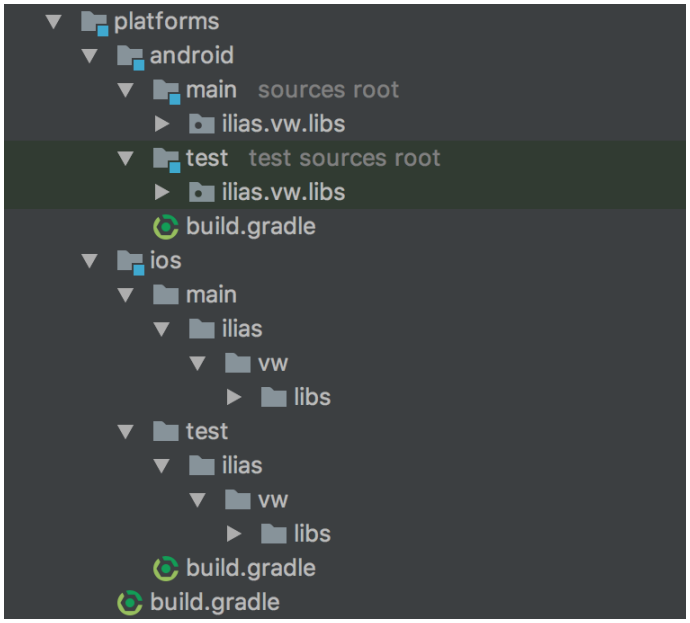
De platform folder bevat enkele subfolders, één subfolder per platform waarvoor men wenst te ontwikkelen. Bij dit voorbeeld zijn er dus twee subfolders aanwezig, namelijk Android en iOS. Zie figuur 5.4. Per submap, hebben we opnieuw twee submappen, namelijk main en test. Net zoals bij de common map.

Zowel de main en test bevatten elk nog een map. Bij Android is dit een package die begint met het groupId, zie sectie 5.3.4, en eindigt met een willekeurige naam. In dit voorbeeld is dit opnieuw 'libs'. Dit dient identiek te zijn met de naam van de package gekozen in de common map, zie sectie 5.5.2. In het geval van iOS zijn dit geen packages maar eerder een mappenstructuur.

5.7.1 android map

Build.gradle

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

repositories {
```



Figuur 5.4: Platforms module structuur

```
 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 sourceSets worden ingesteld. 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 alle dependencies geladen en wordt er aangegeven dat de common module verwacht wordt die code bevat.

Platformspecifieke code

De platformspecifieke code voor Android. Graag willen we het product ook van een afbeelding voorzien 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
 }
}
```

5.7.2 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 4.6 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 voor alle testen, de test map wordt ingesteld als bron waar alle testen zijn gelokaliseerd.

Tenslotte wordt er opnieuw aangegeven dat we een common module verwachten die code bevat.

Platformspecifieke code

De platformspecifieke code voor iOS. In iOS heb je de mogelijkheid om afbeeldingen in te laten 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
 }
}
```

5.8 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 je de locatie van het project 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.

5.8.1 settings.gradle

Voor we gebruik kunnen maken van de Kotlin klassen, moeten er eerst nog enkele wijzigingen gebeuren in de settings.gradle. 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.

5.8.2 build.gradle

Tenslotte moet er nog een kleine wijziging doorgevoerd worden in de build.gradle van de app map. In de dependencies tag moet er enkele en alleen volgende lijn toegevoegd worden in de dependencies tag:

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

Hiermee wordt er aangegeven dat de android-specifieke implementatie te vinden is in platforms-android, waarvan de locatie is opgegeven in de settings.gradle.

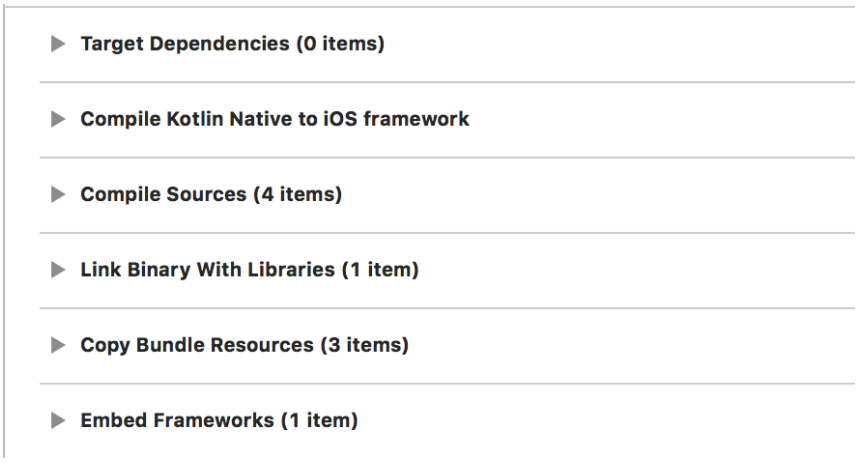
5.9 Stap 6: iOS map

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.

5.9.1 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 5.5 kan worden afgeleid dat de build phases zijn aangepast. Volgende build phase moet worden toegevoegd: Compile Kotlin Native to iOS framework.

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



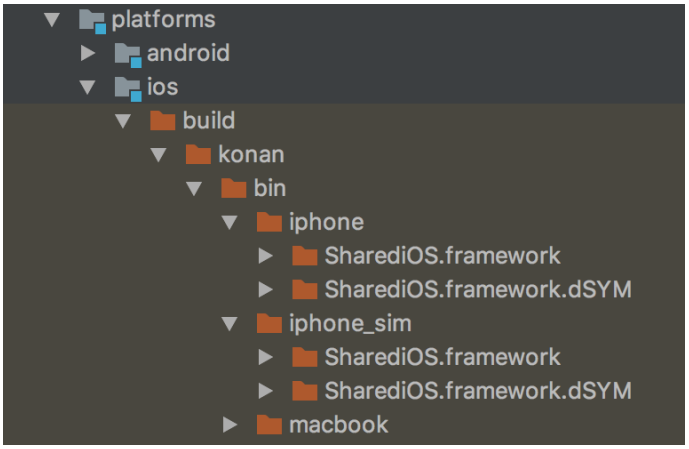
Figuur 5.5: iOS build phases

```
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"
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 5.6. 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.

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 je Xcode project. Bij iedere build en/of run van het Xcode project zal bovenstaand script worden uitgevoerd. Dit zal er voor zorgen dat steeds het laatst gegenereerde framework gebruikt zal worden door je 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 je Xcode project.



Figuur 5.6: iOS build map

5.10 Aanspreken van code

5.10.1 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.*
```

5.10.2 iOS

Om gebruik te kunnen maken van het SharediOS framework in de ViewControllers moet het framework worden geimporteerd 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 folder, zie sectie 5.7.2. Stel dat het framework KotlinNativeFramework heet, dan zal de prefix KNF zijn en zal de product klasse KNFProduct heten.

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

5.11 User interfaces

De user interface moet dus per platform opgebouwd worden. 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.

5.12 Optioneel: testen

Een essentieel deel van object-oriented programming is het testen van je code. In het Kotlin/Native framework is het mogelijk om je 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:

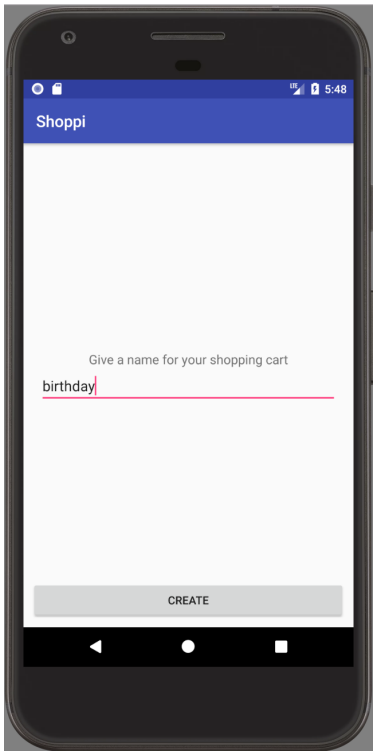
```
package ilias.vw.libs

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'.

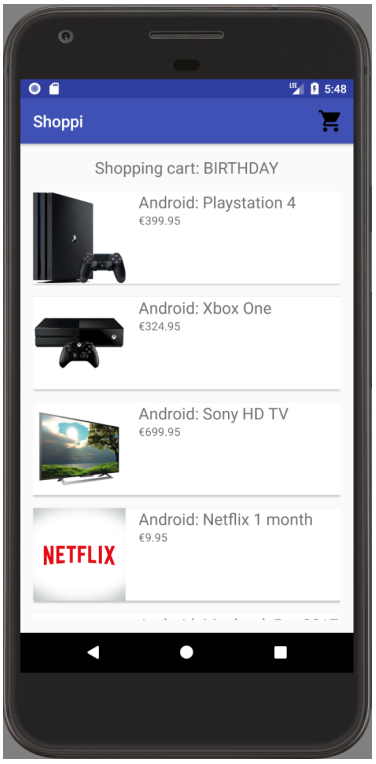
5.13 Proof-of-concept

Screenshots van de schermen van de proof-of-concept.

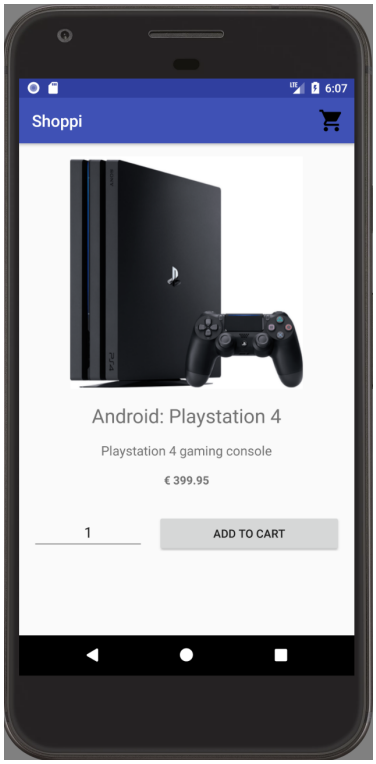
5.13.1 Android



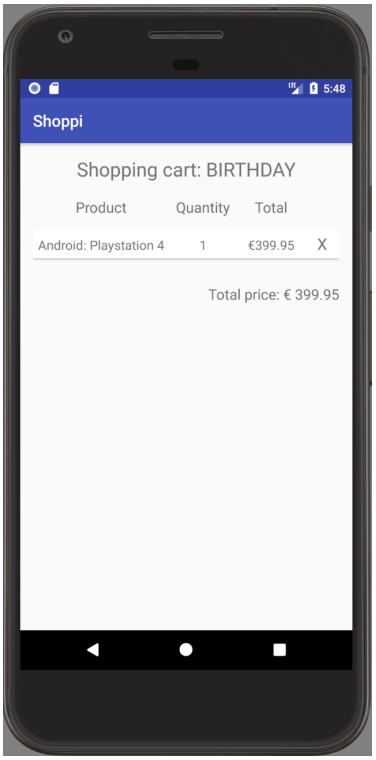
(a) Aanmaken van een shopping cart



(b) Overzicht van producten



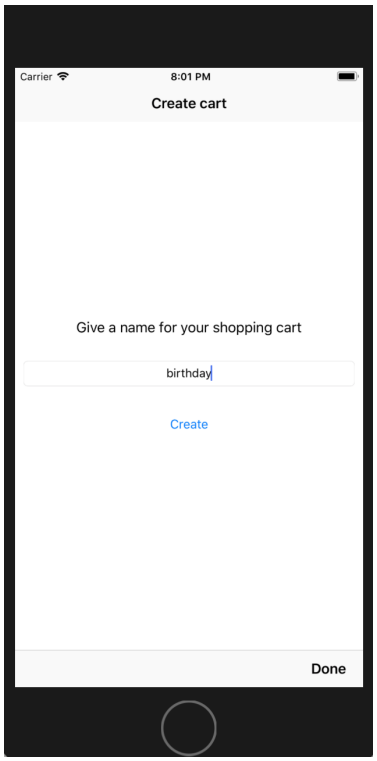
(c) Details van een product



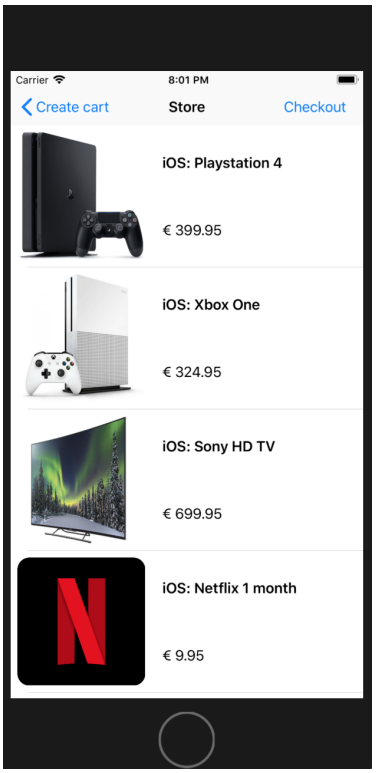
(d) Overzicht winkelmand

Figuur 5.7: Screenshots Android

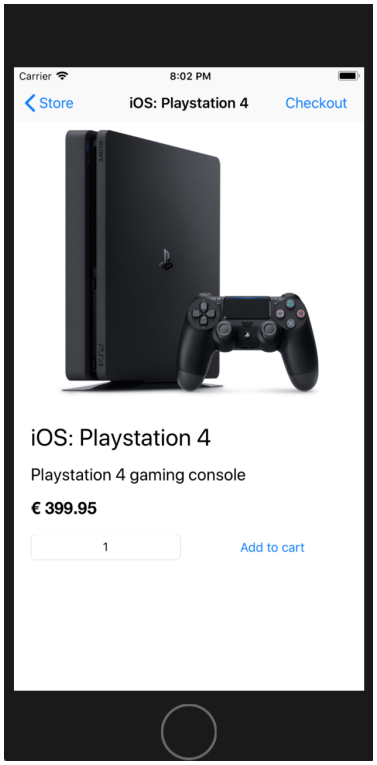
5.13.2 iOS



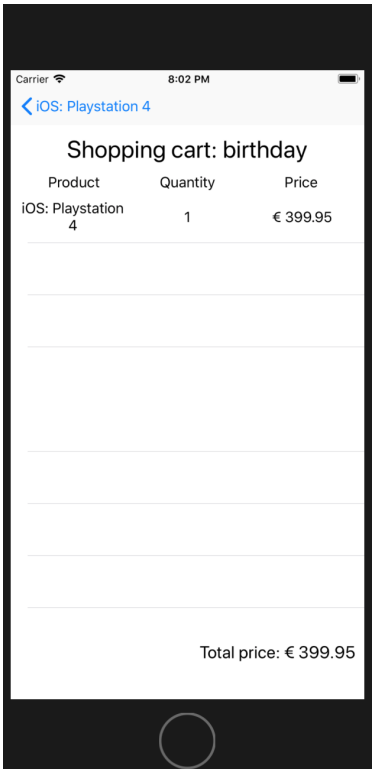
(a) Aanmaken van een shopping cart



(b) Overzicht van producten



(c) Details van een product



(d) Overzicht winkelmand

Figuur 5.8: Screenshots iOS

6. Conclusie

Uit de proof-of-concept, in sectie 5.13, 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 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 enkele 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 heb je dubbele kosten om een applicatie te maken en 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.
- 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 geimplementeerd, zoals Serializable, 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 intergreren 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 converte-

ren 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.

Ali Muzaffar. (2017). Building a full-stack web-app in Kotlin. Verkregen van https://medium.com/bcgdv-engineering/building-a-full-stack-web-app-in-kotlin-af8e8fe1f5dc

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

Chisnall, D. (2008). How the LLVM Compiler Infrastructure Works. Verkregen van http://www.informit.com/articles/article.aspx?p=1215438

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

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. (2018). 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/

JetBrains. (2017a). Kotlin for server-side development. https://kotlinlang.org/docs/reference/server-overview.html.

JetBrains. (2017b). Kotlin official website FAQ. https://kotlinlang.org/docs/reference/faq.html.

JetBrains. (2017c). Kotlin official website, Kotlin Native. https://kotlinlang.org/docs/reference/native-overview.html.

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

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

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

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

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

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/