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

Onderzoeksvoorstel Bachelorproef 2020-2021

Ziggy Moens[1]

Samenvatting

In deze bachelorproef zal de huidige stand van zaken omtrent de cross platform SDK Kotlin Multiplatform Mobile en de mogelijkheid om dit als sneller en efficiënter alternatief te gebruiken voor native applicatieontwikkeling toegelicht worden. Cross platform applicatieontwikkeling wordt de laatste tijd steeds populairder, dit kan onder andere verklaard worden door de kleinere impact op het ontwikkelingsbudget van bedrijven. Dit vooral omdat deze manier van werken minder lijnen code vereist en deze dus sneller kan geschreven worden. In dit onderzoek zullen twee native applicaties geschreven worden, één voor Android en één voor iOS. Er zal eveneens een Kotlin Multiplatform Mobile applicatie geschreven worden die op deze twee besturingssystemen werkt. Hierbij zal gebruik gemaakt worden van Android Studio en Xcode, deze software werkt voor zowel native als Kotlin Multiplatform Mobile applicaties mits dat de Kotlin Multiplatform Mobile plugin geïnstallerd is. Een belangrijke factor hierbij is dat deze applicaties dezelfde functionaliteiten zullen aanbieden aan de gebruiker. Voor dit onderzoek zullen de applicaties aan enkele testcriteria onderworpen worden namelijk: het aantal lijnen code, de kostprijs, de ontwikkeltijd, de compileersnelheid, de voetafdruk en de uitbreidingsmogelijkheden van de applicaties. Verondersteld wordt dat dit onderzoek vooral op vlak van het aantal lijnen code, kostprijs, ontwikkeltijd en de uitbreidingsmogelijkheid in het voordeel zal zijn van Kotlin Multiplatform Mobile. De resultaten van de overige testcriteria zullen wellicht in het voordeel van de native applicaties zijn. Daarnaast zal ook bekeken worden of het huidig beeld op de SWOT-analyse van cross-platform applicaties nog steeds klopt. De verwachte resultaten in het voordeel van Kotlin Multiplatform Mobile zijn onder andere te verklaren doordat de gedeelde business logica een groot voordeel biedt in de omvang van de code. De criteria waar de native applicaties de bovenhand nemen is vooral omdat deze beter geoptimaliseerd zijn voor de specifieke toestellen. Verder kan dit ook verklaard worden doordat Kotlin Multiplatform Mobile momenteel nog in de kinderschoenen staat. Op basis van dit onderzoek zal een bedrijf kunnen evalueren wat de meest interessante optie is voor soortgelijke casussen. Naar de toekomst toe wordt verwacht dat de effectieve productieversie van Kotlin Multiplatform Mobile veel dichter zal liggen bij of zelfs efficiënter zal doen dan de native applicaties.

Sleutelwoorden

Mobiele applicatieontwikkeling. Kotlin Multiplatform Mobile — Cross platform — Native

Co-promotor

Kenneth Saey[2] (Endare)

Contact: [1] ziggy.moens@student.hogent.be; [2] kenneth.saey@endare.com;

Inhoudsopgave

1. Introductie

De keuze voor cross platform applicaties wordt de dag van vandaag steeds aantrekkelijker. Dit kan onder andere verklaard worden door de financiële voordelen ten opzichte van native applicaties, meer specifiek op vlak van de impact op het ontwikkelingsbudget van bedrijven. Cross platform applicaties worden al heel lang gepromoot als zijnde 'beter' dan native applicaties. Deze cross platform applicaties zijn sneller te ontwikkelen en vereisen minder lijnen code. De slogan "Write once, run anywhere" van Sun Microsystems (Oracle, g.d.) wordt vaak gebruikt om het grote voordeel van cross platform applicaties aan te duiden, dit duidt op het idee dat grote delen van de code kunnen gebruikt worden voor verschillende besturingssys-

temen. De duurdere native applicaties hebben echter nog steeds een aantal voordelen ten opzichte van de cross platform applicaties, aangezien deze specifiek geschreven zijn voor het gekozen besturingssysteem zoals bijvoorbeeld voor Android of iOS. De meeste reeds bestaande cross-platform frameworks, zoals Flutter, delen de user interface van de applicatie over de verschillende besturingssystemen. Dit zorgt voor een applicatie die voor geen enkel van de gekozen besturingssysteem perfect zal zijn, aangezien hierbij dan geen gebruik kan gemaakt worden van de specifieke frameworks voor dat specifieke besturingssysteem. Kotlin Multiplatform Mobile (KMM) gooit het concept van cross platform over een andere boeg en werkt met een gedeelde businesslogica in plaats van een gedeelde user interface. De applicaties worden dus elk afgewerkt met een native user interface. De opzet van deze bachelorproef is na te gaan of native applicatieontwikkeling voor Android en iOS apparaten nog steeds de beste keuze is of KMM een sneller en efficiënter alternatief kan vormen binnen het huidige beeld van applicatieontwikkeling. Om dit te onderzoeken zijn er bepaalde testcriteria opgesteld. Deze testcriteria zijn: het aantal lijnen code, de kostprijs, de ontwikkeltijd, de compileersnelheid, de voetafdruk en de uitbreidingsmogelijkheden van de applicaties. Aan de hand hiervan kan besloten worden of de KMM-applicaties een beter alternatief kunnen vormen voor native applicaties. Daarnaast zal ook de vraag gesteld worden of het huidige beeld van de SWOT-analyse voor cross-platform applicaties ook toepasbaar is voor KMM-applicaties, indien niet zal gekeken worden hoe deze dient aangepast te worden.

2. State-of-the-art

2.1 Kotlin

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

2.2 Cross-platform en native

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

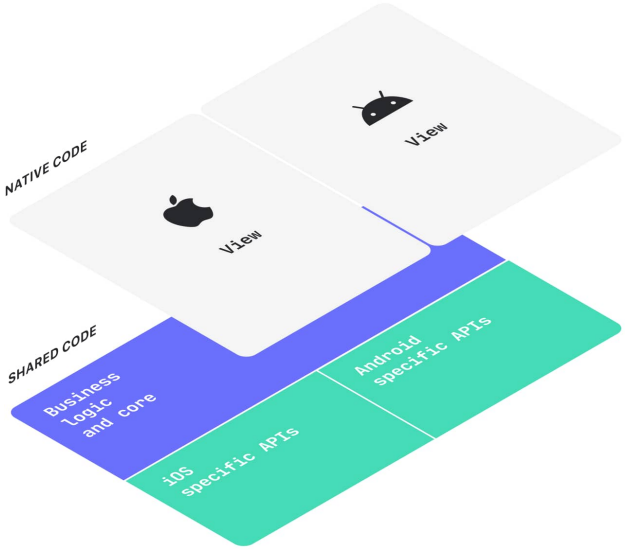
Eens de term platform duidelijker is, kan er makkelijker omschreven worden wat precies een cross-platform applicatie is en wat een native applicatie is. Een cross-platform applicatie is zoals de naam het al zegt een applicatie die op meerdere platformen zal werken. De software zal dus licht verschillende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

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

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

2.3 Kotlin Multiplatform

Kotlin Multiplatform is de nieuwe cross platform software development kit (SDK) van JetBrains. Ook werd een SDK uitgebracht specifiek gericht op de mobiele



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

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

2.4 Kotlin Multiplatform versus alternatieven

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

Figuur 1 geeft een goed algemeen beeld van de structuur van Kotlin Multiplatform Mobile. De 'shared code' in de figuur 1 verwijst hier naar Common Kotlin of CommonMain, dit is het gedeelde binnenin KM dat de gedeelde logica zal bevatten. Daarnaast zal er nog per platform een aparte main zijn die de code zal implementeren. Hier in de figuur 1 zal het project ook nog een iosMain en een kotlinMain, deze kunnen later ook nog uitgebreid worden met bijvoorbeeld een macosX64Main. De specifieke situatie in figuur 1 is een applicatie waar Kotlin Multiplatform Mobile gebruikt wordt. Deze zal zich speciaal richten op applicaties voor iOS en Android. Voor de communicatie tussen het common deel en de platform specifieke delen zal Kotlin gebruik maken van het expected/actual systeem. Hierbij worden binnenin de CommonMain elementen gedeclareerd met expect en de platform specifieke delen zullen dezelfde elementen declareren met actual. Deze structuur kan gebruikt worden voor functies, klassen, interfaces, enumeraties, properties en annotaties.

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

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

2.5 Testcriteria

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

- Aantal lijnen code
  - Het totaal aantal lijnen code van een applicatie zal geëvalueerd worden over het gehele project. In het geval van native applicaties zullen deze opgeteld worden met elkaar.

- Kostprijs
  - Hierbij wordt de geschatte kostprijs om een applicatie te laten ontwikkelen door een IT-bedrijf in kaart gebracht. Dit wordt berekend aan de hand van geschatte werkuren en een gemiddelde kostprijs per uur. Daarnaast wordt nagegaan of cross-platform een effect zal hebben op het systeem van vooraf bepaalde totaalprijzen indien bedrijven daarmee werken.
- Ontwikkeltijd
  - Deze tijd beschrijft het aantal werkuren dat een ontwikkelaar nodig heeft om een specifieke applicatie te schrijven. Hierbij kan onder andere gebruik gemaakt worden van platformen zoals GitHub om deze tijd te gaan meten of inschatten.
- Compileersnelheid
  - Dit is de snelheid waarmee de specifieke applicatie zal kunnen compileren en opstarten. Dit kan gemeten worden in de ontwikkelingssoftware voor de desbetreffende taal van de applicatie.
- Voetafdruk
  - Dit impliceert de omvang die de applicatie zal innemen op het platform waarvoor deze ontwikkeld is. Hiervoor kan de applicatie gebruikt worden die de ontwikkelingssoftware aanmaakt.
- Uitbreiding van de applicatie
  - Dit criterium kan geëvalueerd worden door vooraf bepaalde features van de applicatie weg te laten. Eens de applicatie klaar is voor productie kunnen deze features terug toegevoegd worden. Om de uitbreidbaarheid van de applicatie te gaan staven kan gebruik gemaakt worden van voorgaande testcriteria.

3. Methodologie

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

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

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

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

Voor dit onderzoek zullen volgende hardware en software gebruikt worden:
Hardware:

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

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

Software:

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

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

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

4. Verwachte resultaten

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

- Aantal lijnen code
  - Kotlin Multiplatform Mobile (KMM) zal hier een voordeel hebben in het aantal lijnen code gezien de business logica maar eenmaal moet geschreven worden. Dit is niet

het geval bij de native applicaties, daar zal de business logica uitgewerkt worden voor zowel Android als iOS.

- Ontwikkeltijd
  - Aangezien de KMM-applicatie de gehele domein laag zal kunnen delen, wordt er verwacht dat deze ook sneller te ontwikkelen zal zijn dan de native varianten.
- Kostprijs
  - Verdergaand op de ontwikkeltijd kan hier ook gesteld worden dat KMM goedkoper zal zijn om te ontwikkelen. Aangezien er minder werkuren nodig zullen zijn kan ook het systeem van de totaalprijzen herzien worden. Hierdoor zullen applicaties met KMM goedkoper zijn voor de klant.
- Compileersnelheid
  - Gezien de complexe hiërarchie wordt verwacht dat de KMM-applicatie trager zal zijn dan de native applicaties. De native applicaties zijn echter ook beter geoptimaliseerd voor deze compilers en apparaten.
- Voetafdruk
  - Zoals reeds besproken zal ook de complexere hiërarchie hier ook een rol spelen. Er wordt verwacht dat de KMM-applicaties een grotere voetafdruk zullen hebben op de toestellen in kwestie, het verschil zal echter verwaarloosbaar klein zijn.
- Uitbreiding van de applicatie
  - De KMM-applicaties zullen hier ook een groter voordeel hebben omdat de nieuw geïmplementeerde features direct voor alle platformen zullen geïmplementeerd zijn. Aan de kant van de native applicaties kunnen bepaalde features vergeten of anders geïmplementeerd worden. Dit zorgt voor een verschil tussen de twee native applicaties.

5. Verwachte conclusies

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

Referenties

Bishop, J. & Horspool, N. (2006). Cross-Platform Development: Software that Lasts. Computer, 39(10), 26–35. https://doi.org/10.1109/MC.2006.337

Flutter. (g.d.). Introduction to widgets (Flutter, Red.). Verkregen 11 februari 2021, van https://flutter.dev/docs/development/ui/widgets-intro

Jemerov, D. (2011, juli 19). Hello World (D. Jemerov, Red.). Verkregen 11 februari 2021, van https://blog.jetbrains.com/kotlin/2011/07/hello-world-2/

Jemerov, D. (2017, november 28). Kotlin 1.2 Released: Sharing Code between Platforms (D. Jemerov, Red.). Verkregen 11 februari 2021, van https://blog.jetbrains.com/kotlin/2017/11/kotlin-1-2-released/

JetBrains. (2020, november 23). KMM plugin releases (JetBrains, Red.). Verkregen 11 februari 2021, van https://kotlinlang.org/docs/mobile/kmm-plugin-releases.html#update-to-the-new-release

Kotlin. (2020). Case Studies (Kotlin, Red.). Verkregen 11 februari 2021, van https://kotlinlang.org/lp/mobile/case-studies/

Kotlin & JetBrains. (g.d.). Kotlin Multiplatform Mobile (Kotlin & JetBrains, Red.). Verkregen 11 februari 2021, van https://kotlinlang.org/lp/mobile/

Kotlin & JetBrains. (2020, november 18). Kotlin Multiplatform Hands-on: Networking and Data Storage (Kotlin & JetBrains, Red.). Verkregen 11 februari 2021, van https://github.com/kotlin-hands-on/kmm-networking-and-data-storage/tree/final

Kotlin & JetBrains. (2021, februari 3). Kotlin Multiplatform: Share code on platforms (Kotlin & JetBrains, Red.). Verkregen 11 februari 2021, van https://kotlinlang.org/docs/reference/mpp-share-on-platforms.html#share-code-on-similar-platforms

Nivanaho, T. (2019, mei 27). Developing a cross-platform mobile application with React Native (masterscriptie). Lappeenranta-Lahti University of Technology LUT. http://lutpub.lut.fi/handle/10024/159507

Oliveira, V., Teixeira, L. & Ebert, F. (2020, februari 18). On the Adoption of Kotlin on Android Development: A Triangulation Study, In 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), London, ON, Canada. https://doi.org/10.1109/SANER48275.2020.9054859

Oracle. (g.d.). Moved by Java Timeline (Oracle, Red.). Verkregen 11 februari 2021, van https://www.oracle.com/java/moved-by-java/timeline/

O'Reilly, J. (2021, februari 5). PeopleInSpace (J. O'Reilly, Red.). Verkregen 11 februari 2021, van https://github.com/joreilly/PeopleInSpace

Petrova, E. (2020, augustus 31). Kotlin Multiplatform Mobile Goes Alpha (E. Petrova, Red.). Verkregen 11 februari 2021, van https://blog.jetbrains.com/kotlin/2020/08/kotlin-multiplatform-mobile-goes-alpha/

Sells, C. (2020, oktober 1). Announcing Flutter 1.22. Verkregen 11 februari 2021, van https://medium.com/flutter/announcing-flutter-1-22-44f146009e5f