Applicatieontwikkeling in cross-platform en platform-specifieke frameworks

faculteit industriËle

ingenieurswetenschappen

**Campus geel**

Een vergelijkende studie

Masterproef ingediend tot het behalen van de graad van master of Science in de industriële wetenschappen: *elektronica-ICT, afstudeerrichting ICT*

Lennert VAN LOOVEREN

Promotor: Dr. Ing. Peter Karsmakers

Co-promotor: Bram Schrijvers

Academiejaar 2017-2018

© Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, kan u zich richten tot KU Leuven Campus Geel, Kleinhoefstraat 4, B-2440 Geel, +32 14 80 22 40 of via e-mail iiw.geel@kuleuven.be.

Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma’s voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Voorwoord

Op het einde van de opleiding master industriële wetenschappen is de masterproef een essentieel onderdeel van het programma. Hiervoor heb ik adequate hulp gekregen van bepaalde mensen die ik extra moet bedanken.

Ik heb deze masterproef vooral kunnen verwezenlijken dankzij mijn (co)-promotoren, met speciale dank aan Bram Schrijvers. Zijn dagelijkse beschikbaarheid op Zappware en kennis van zaken heeft mij meermaals uit de nood geholpen. Eveneens promotor Peter Karsmakers verdiend een dankwoord, voor steeds beschikbaar te zijn voor vragen en diverse problemen te behandelen voor de KULeuven.

Ik wil ook graag mijn ouders bedanken, die via hun morele en financiële steun mij begeleid hebben gedurende mijne hele studieperiode. Eveneens wil ik de collega’s bij Zappware bedanken aangezien zij voor een leuke sfeer zorgden en mij welkom deden voelen. Tevens bedank ik mijn vrienden die deze masterproef nagelezen hebben. Ten slotte wil ik de jury bedanken voor de tijd te nemen om deze masterproef te lezen en te beoordelen.

Lennert Van Looveren, Geel, 19 mei 2018

Samenvatting

Huidige mobiele applicaties kunnen op verscheidene manieren gecreëerd worden. De meest courante methode hiervoor omvat het schrijven van enerzijds de iOS app via XCode in Swift met behulp van een MAC, en anderzijds de Androidapplicatie via Android Studio in bijvoorbeeld Java of Kotlin. Dit betekent dat er voor één applicatie twee volledige projecten dienen uitgeschreven te worden. In het kader van deze masterproef werd onderzocht of deze enigszins omslachtige procedure vermeden kan worden door het hanteren van een cross‑platform framework wat één bepaalde geschreven taal omzet naar een bruikbare versie voor op zowel Android als iOS. Hiervoor werden twee applicaties geschreven met als doel dezelfde app voor te stellen als degene die Zappware ontwikkeld heeft in de twee native platformen. De cross-platform frameworks die hiervoor onderzocht werden, omvatten: Qt, React Native, Xamarin en Apache Cordova.

Deze werden eerst theoretisch tegen elkaar afgewogen aan de hand van verscheidene parameters (open-source, programmeertaal en de manier waarop het zijn UI rendert), waaruit React Native als meest optimale uit de bus kwam, met Xamarin op de 2de plaats. Hierdoor werden deze twee frameworks verkozen om effectief de applicatie in te ontwikkelen. Vervolgens werden deze uitgerold op Android en iOS, waarna het proces beschreven werd. Uit dit proces bleek dat React Native een complexloos, goed gedocumenteerd framework waarvoor weinig setup nodig is en dat al zijn functies en features gratis aanbiedt. Xamarin daarentegen heeft een uitgebreidere IDE in Visual Studio waardoor het debuggingsproces aangenamer en duidelijker verliep. Desondanks bleek dat dit framework minder goed gedocumenteerd is, en waarbij niet alle features open-source te vinden zijn. De conclusie was dat Xamarin eerder een concept voor langetermijn-projecten is en dat React Native een ideale omgeving is voor prototypes op te bouwen. Nadien werd de gecreëerde applicaties getest op verschillende factoren: snelheid, geheugengebruik en user experience. Dit laatste zal de belangrijkste zijn aangezien de snelheid en geheugengebruik uiteindelijk de gebruikerservaring beïnvloedt.

Voor de gebruikerservaring te testen werd een enquête gedaan met tien deelnemers die verschillende vragen over de applicaties moesten invullen. De resultaten bewezen dat de native applicatie, logischerwijs, de meest complete en gebruiksvriendelijkste was. Als dan de resultaten van de React Native en Xamarin app worden vergeleken blijkt dat er maar minimale verschillen bestaan uit deze applicaties waarbij de redenen niet bij het framework lagen. Hierdoor is de impact van het framework op de gebruikerservaring niet significant verklaard. Wat wel opviel is de performantie van deze apps, ondanks dat ze basic ontworpen waren hadden ze geen perfecte resultaten. Met deze informatie werd geconcludeerd dat native applicatieontwikkeling toch de juiste keuze is om de gebruikerservaring te garanderen. Dit betekent niet dat een cross-platform framework geen plaats kan hebben in een development omgeving. Een framework zoals React Native kan perfect gebruikt worden om prototypes en demo’s op te ontwikkelen. Deze eigenschap valt te danken aan het feit dat het met Javascript een bekende taal gebruikt, er veel documentatie rond bestaat en alle mogelijkheden errond open-source aanbiedt.

Inhoud

[Voorwoord i](#_Toc514835218)

[Samenvatting ii](#_Toc514835219)

[1 Inleiding 1](#_Toc514835220)

[Zappware 1](#_Toc514835221)

[Onderzoeksvraag 1](#_Toc514835222)

[2 Cross-platform frameworks 2](#_Toc514835223)

[2.1 Qt 2](#_Toc514835224)

[2.1.1 Oorsprong 2](#_Toc514835225)

[2.1.2 Basis 2](#_Toc514835226)

[2.1.3 Grafische architectuur 3](#_Toc514835227)

[2.1.4 Main feature: Signals and slots 4](#_Toc514835228)

[2.1.5 Qt Creator 5](#_Toc514835229)

[2.1.6 Portfolio 5](#_Toc514835230)

[2.2 React Native 6](#_Toc514835231)

[2.2.1 Oorsprong 6](#_Toc514835232)

[2.2.2 Basis 6](#_Toc514835233)

[2.2.3 Features 7](#_Toc514835234)

[2.2.4 Gebruik 10](#_Toc514835235)

[2.2.5 Portfolio 10](#_Toc514835236)

[2.3 Xamarin 11](#_Toc514835237)

[2.3.1 Oorsprong 11](#_Toc514835238)

[2.3.2 Basis 11](#_Toc514835239)

[2.3.3 Features 11](#_Toc514835240)

[2.3.4 Gebruik 13](#_Toc514835241)

[2.3.5 Portfolio 13](#_Toc514835242)

[2.4 Apache Cordova/PhoneGap 14](#_Toc514835243)

[2.4.1 Oorsprong 14](#_Toc514835244)

[2.4.2 Basis 14](#_Toc514835245)

[2.4.3 Architectuur 14](#_Toc514835246)

[2.4.4 Gebruik 15](#_Toc514835247)

[2.4.5 Main feature: WebView 16](#_Toc514835248)

[2.4.6 Portfolio 16](#_Toc514835249)

[2.5 Kwaliteitscontrole van de applicaties 17](#_Toc514835250)

[2.5.1 Theoretisch 17](#_Toc514835251)

[2.5.2 Tools 18](#_Toc514835252)

[2.5.3 Praktisch 19](#_Toc514835253)

[3 Applicatie 23](#_Toc514835254)

[4 Vergelijking 24](#_Toc514835255)

[4.1 Development effort 24](#_Toc514835256)

[4.1.1 React Native 24](#_Toc514835257)

[4.1.2 Xamarin 26](#_Toc514835258)

[4.1.3 Conclusie 28](#_Toc514835259)

[4.2 Geheugen 30](#_Toc514835260)

[4.2.1 Vergelijking 30](#_Toc514835261)

[4.2.2 Conclusie 31](#_Toc514835262)

[4.3 Snelheid 32](#_Toc514835263)

[4.3.1 Vergelijking 32](#_Toc514835264)

[4.3.2 Conclusie 33](#_Toc514835265)

[4.4 User experience 33](#_Toc514835266)

[4.4.1 Resultaten enquête 34](#_Toc514835267)

[4.4.2 Bespreking resultaten 34](#_Toc514835268)

[4.4.3 Conclusie 36](#_Toc514835269)

[5 Besluit 36](#_Toc514835270)

[6 Verwijzingen 37](#_Toc514835271)

[Bijlagen 40](#_Toc514835272)

Lijst met figuren

Figuur 2‑2: Voorbeeld *‘signals and slots’* [8] 12

Figuur 2‑3: Schematische voorstelling van een Document Object Model (DOM) 15

Figuur 2‑6: Complexer voorbeeld Flux [14] 17

Figuur 2‑7 Voorbeeld Flux [14] 17

Figuur 2‑8: Opbouw Xamarin [19] 19

Figuur 2‑9: Xamarin.Forms’ oplossing architectuur [19] 20

Figuur 2‑10 De relaties tussen de 3 kerncomponenten van MVVM [21] 20

Figuur 2‑11: Architectuur van een Cordova applicatie [28] 22

Figuur 2‑12 Verschil tussen native apps, hybride apps en web apps [31] 24

Figuur 3.1: Einddoel applicatie 31

Figuur 4‑1: React Native applicatie 33

Figuur 4‑2: Xamarin applicatie 35

Lijst met tabellen

Tabel 2‑1 Theoretische eigenschappen verschillende frameworks 25

Tabel 2‑2 Enquête user experience 30

Tabel 4‑1 Vergelijking ontwikkelingseigenschappen 37

Tabel 4‑4 Vergelijking geheugengebruik Android applicaties 38

Tabel 4‑5 Vergelijking geheugengebruik iOS applicaties 39

Tabel 4‑2 Vergelijking snelheden Android applicaties 40

Tabel 4‑3 Vergelijking snelheden iOS applicaties 40

Tabel 4‑6 Resultaten enquête Android applicaties 42

Tabel 4‑7 Resultaten enquête iOS applicaties 42

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Lijst met afkortingen

|  |  |  |
| --- | --- | --- |
| API | Application programming interface |  |
| (G)UI | (Graphical) User interface |  |
| SDK | Software development kit |  |
| UWP | Universal Windows Platform |  |
| GLUT | OpenGL Utility Toolkit |  |
| GLU | OpenGL Utility Library |  |
| GLX | OpenGL Extension to the X Window System |  |
| AGL | Apple Graphics Library |  |
| GNU | GNU Compiler Collection |  |
| ICC | Intel C++ Compiler |  |
| MSVC  DOM  IDE  APK  IPA  OS  CLI | Microsoft Visual C++  Document Object Model  Integrated development environment  Android Package File  Iphone Application Archive  Operating system  Command line interface |  |

**Druk <Ctrl + Alt + Shift + S> om het taakvenster met stijlen weer te geven**

# Inleiding

## Zappware

Het onderzoek in het kader van deze masterproef is uitgevoerd bij Zappware, gestationeerd in Hasselt. Dit globaal bedrijf houdt zich bezig met zowel het ontwerpen van video UI-design als client-software development [1]. Deze masterproef kadert binnen de client-software kant, waarbij alternatieven gezocht werden om het proces van individuele app development te kunnen omzeilen bij het programmeren van mobiele applicaties. Deze ontwikkelingen zouden het immers mogelijk maken om een snel prototype van Zappware’s typische UI‑implementaties te bouwen om deze vervolgens te kunnen demonstreren. Indien hierbij zeer goede resultaten zouden verkregen worden, kunnen deze alternatieven zelfs de originele Android- en iOS-ontwikkeling vervangen. Zappware voorzag bij deze masterthesis voorbeelden van mogelijke applicaties om te ontwikkelen en de expertise van hun personeel.

## Onderzoeksvraag

In een eerste studie is Qt onderzocht. Dit is een cross-platform framework dat in staat is één code om te zetten dat deze zowel op Android als iOS uitgevoerd kan worden. Hierbij is bekeken of dit platform een effectieve vorm van programmeren opleverde en wat de mogelijkheden waren van Qt. In een vroeg stadium van deze studie werd echter geconcludeerd dat Qt niet het gewenste resultaat leverde naar de vereisten van Zappware. Dit is geconstateerd nadat de demo na een implementatie niet de beoogde ‘*look and feel’* opleverde. Deze vaststelling motiveerde de zoektocht naar alternatieven op Qt, wat uiteindelijk de basis van deze masterproef vormde.

Het doel van deze masterproef omvat het vinden van een cross-platform framework dat aan bepaalde eisen van performantie, gebruiksvriendelijkheid en ontwikkelingstijd voldoet. Vervolgens zijn de gecreëerde applicaties zowel met elkaar als met applicaties die native gebouwd zijn vergeleken. Uiteindelijk is bepaald welke omgevingen het meest geschikt bevonden zijn voor gebruik of eventueel als een platform kunnen dienen om prototypes van hun mobiele applicaties op te bouwen.

# Cross-platform frameworks

## Qt

### Oorsprong

Qt is ontworpen nadat de co-founders van Trolltech in 1990 samenwerkten aan een database applicatie voor ultrasound-afbeeldingen, geschreven in C++, dat zowel op MAC OS, UNIX en Windows diende te opereren. Om dit te kunnen verwezenlijken, kwamen deze onderzoekers tot de vaststelling dat hiervoor een object-georiënteerd display systeem benodigd is, wat resulteert in een basis voor de object-georiënteerde cross-platform GUI‑framework dat later gebouwd werd onder de naam Qt, Q-toolkit.

Na 5 jaar schrijven aan de nodige C++ klassen is in 1995 Qt 0.90 uitgebracht, bruikbaar voor zowel Windows als Unix development en gebruikte voor beide platforms dezelfde API.   
In 2008 is TrollTech overgenomen door Nokia waarbij deze laatste streefde om van Qt het meest gebruikte ontwikkelingsplatform te maken voor hun apparaten.

Vanaf 2014 is Qt beheerd door “The QT Company”, een dochterbedrijf van Digia, die de rechten van Qt overgekocht had van Nokia in 2012. Digia verzorgde de revolutionaire Qt 5.0 waarbij met behulp van JavaScript en QML de performantie en eenvoud van het ontwerpen van UI sterk verbeterd werd. Sindsdien is Qt ook een ‘*open-source* *governance’*, waardoor ontwikkelaars buiten Digia eveneens verbeteringen konden voorstellen. [2]

In 2016 is Digia en “The Qt Company” gesplitst in 2 onafhankelijke bedrijven. [3]

### Basis

Qt is een cross-platform applicatie framework dat in C++ geschreven is en onder andere Android, iOS en WindowsPhone ondersteunt. Het doel van Qt bestaat eruit om zonder codewijzigingen op elk platform zonder performantie-verlies te opereren en native te lijken.   
Qt ondersteunt elke standaard C++ compiler, zoals GCC, ICC, MSVC en Clang. [4]   
De applicatie is geschreven in een combinatie van C++ en QML, een declaratieve scripting-taal die Javascript gebruikt voor de logica mee op te bouwen.

Qt dankt zijn cross-platform eigenschap aan het feit dat door de C-preprocessor (*cpp*) condities sommige code chuncks worden enabled of disabled afhankelijk van het platform. Het bouwproces van het framework gaat configuratie scripts runnen om preprocessor *flags* op te merken, af te zetten en aan te passen. [5]

### Grafische architectuur



Figuur 2.1 de grafische architectuur van Qt [43]

De grafische architectuur weergegeven die toegepast wordt op Qt wordt weergegeven in Figuur 2.1. Hierbij valt het op dat de manier waarop de UI gebouwd wordt bestaat uit twee systemen, het window systeem en het grafische systeem Het grafische systeem beheert de display hardware, waarbij vooral OpenGL-GL, een library die 2D- en 3D-vectoren rendert, gebruikt wordt als API. [6] Anderzijds genereert het window systeem de gebruikvensters en beheert en controleert het onder andere de events en venster aanpassingen. Het window systeem bezit een eigen API voor programmeren, dat meestal in het OS is geïntegreerd zoals Microsoft Windows en MAC OS.

GLUT is een interface die de communicatie tussen het window systeem en het graphisch systeem verzorgt. Dit gebruikt GLU en GL om de graphics te genereren en controle van operating en window systemen door voornamelijk bv. GLX en AGL te gebruiken. [7] Deze laatsten zijn voorbeelden van OpenGL extensies. Qt zal op hetzelfde niveau als GLUT zowel het window als het graphics systeem kunnen raadplegen waardoor het in staat zal zijn functies voor beide systemen te kunnen oproepen.

### Main feature: Signals and slots

Qt behandelt events, zoals *quit(), onkeydown(),…* , niet rechtstreeks. Zogenaamde *callback* functies die voorzien zijn in de code hebben een alternatief, namelijk *‘signals and slots’*. Een signaal wordt verzonden wanneer een bepaald evenement gebeurt of een verandering van status plaatsvindt (Zie Figuur 2.2). Daarnaast vormt een slot in het model een functie die opgeroepen wordt wanneer een bepaald signaal verzonden wordt. Deze signal-slot relatie zit reeds in vele Qt-widgets (interface objecten) ingebouwd. De mogelijkheid om zelf slots te definiëren voor bepaalde signalen wordt ook toegepast. Dit signal-slot gebruik is veelzijdig, zodat meerdere signalen aan meerdere slots gekoppeld kunnen worden en ook verschillende signalen aaneengeschakeld kunnen worden. [8]

Figuur 2.2: Voorbeeld *‘signals and slots’* [8]

Een voorbeeld voor deze signal-slot relatie wordt hieronder geïllustreerd aan de hand van een eenvoudige *quit()* functie.

Int main (int argc, char \*argv[])

{   
QApplication app(argc, argv);

QPushButton \*button = new QPushButton(“Quit”);

QObject::connect (button, SIGNAL(clicked()), &app, SLOT(quit());

button->show();

return app.exec();

}

Hierbij wordt eerst een applicatie-object *‘app’* en een knop *‘button’* gecreëerd. De volgende stap connecteert deze knop met een signaal *‘clicked()’*. Hierdoor zal functie *quit()* via het slot opgeroepen worden in object *‘app’* als button het signaal *‘clicked()’* verzendt.

### Qt Creator

Qt Creator is het C++, JavaScript en QML IDE dat deel is van het SDK voor het Qt GUI-applicatie development framework. Deze bevat een visuele debugger en een geïntegreerde GUI layout. Deze IDE bevat een eigen editor met typische kenmerken zoals syntax markering en het automatisch aanvullen van code die zou ontbreken.

In verband met de installatie van Qt Creator is het mogelijk om online een gratis open source versie downloaden die Qt zelf aanbiedt. Voor commerciële doeleinden beschikt Qt over een uitgebreider pakket, echter tegen betaling, die meer tools, features en support zal bieden.

De installatie zelf is rechtuit: installeren en opstarten. Om de omgeving te laten opereren moeten enkele omgevingsvariabelen aangepast worden aan Windows/MAC en development kits geïnstalleerd worden in Qt zelf om de code te testen op het geprefereerde platform. Hierbij zijn de mogelijkheden eindeloos: van verschillende versies Android tot desktopapplicatie op Windows, analoog op MAC.

### Portfolio

Noemenswaardige applicaties die Qt of QML gebruiken: [9]

* Google Earth
* Adobe Photoshop Album
* Adobe Photoshop Elements
* Spotify for Linux
* VLC media player
* Teamviewer

## React Native

### Oorsprong

React ,tevens gekend als react.js of ReactJS, is een JavaScript library die in 2013 ontwikkeld is door werknemers van Facebook en Instagram om UI’s te bouwen. Het doel van deze library bestaat eruit om grote schaalbare webapplicaties te maken, waarvan de data constant veranderd kan worden zonder de pagina te herladen. [10]

React Native is gestart als een hackathlon project in de zomer van 2013 [11]. Na een jaarlange ontwikkeling van het prototype, kreeg het framework zijn eerste taak toebedeeld: het ontwikkelen van een onafhankelijk werkende iOS app, met als doel was een volledige applicatie te maken die volgens user experience identiek was aan een app die in Objective-C geschreven is. Dit project is met succes afgerond, waardoor beslist werd om een React Native cross-platform te ontwikkelen. Dit begon met een Android team die de basic Android Runtime ,de omgeving gebruikt door de Android OS, en de componenten errond schreef. Begin 2015 is het framework vervolgens publiek tentoongesteld en tegen het einde van het jaar werd het volledig open source. [10]

### Basis

React Native is volledig gebaseerd op React. Enerzijds omvat dit cross-platform framework een open-source library die onderhouden wordt door zowel individuen, als kleine en grote bedrijven. Anderzijds is het een JavaScript framework om mobiele applicaties voor iOS, Android en UWP native te laten renderen en dit via een JavaScript library die reeds courant gebruikt werd voor webapplicaties. Dit framework zorgt ervoor dat de geschreven code geschikt is voor elk van de drie mobiele platforms. Deze applicaties zijn geschreven met een mix van JavaScript en JSX, een taal die hard lijkt op XML. React Native gebruikt het gewenste uitrolplatform zoals bijvoorbeeld Android of iOS zijn eigen standaard rendering API waardoor de mobiele applicatie dezelfde *‘look and feel’* zal bezitten als een native designed applicatie. [12]

### Features

#### Virtual DOM

In een webapplicatie is één van de meest belastende operaties het veranderen van de DOM. Dit is een API dat een HTML, XHTML of XML bestand behandelt als een boomstructuur waarin elke node een object voorstelt als deel van het document. [13] (Figuur 2.3)

Figuur 2.3: Schematische voorstelling van een Document Object Model (DOM)

React onderhoudt een virtuele representatie van deze DOM, Virtual DOM dus. Samen met een ‘diffing’ algoritme, die twee boomstructuren vergelijkt, kan React Native het verschil t.o.v. de oorspronkelijke DOM bepalen en enkel het deel updaten dat tussentijds veranderd is. Deze eigenschap is noodzakelijk voor real time applicaties die enige complexiteit bevatten.

In plaats van de geïnduceerde veranderingen op een webpagina meteen te renderen, zal React eerst de benodigde aanpassingen berekenen in zijn memory en vervolgens het minimaal mogelijke van de pagina opnieuw renderen. Zo dient niet de gehele pagina te moeten worden herladen, wat echter bij gewoonlijke webapplicaties wel het geval is. Figuur 2.4 toont aan wat er gebeurt als er iets moet veranderen in het DOM. Van links naar rechts vormen zich drie stappen: Stap één omvat een effectieve statusverandering van het DOM; Stap twee calculeert het verschil tussen beide webpagina’s; Stap drie besluit het opnieuw renderen van de UI. Bij deze laatste stap wordt dan ook het browser DOM aangepast, zodat deze niet geheel opnieuw gerenderd hoeft te worden.

Figuur 2.4: Voorbeeld calculatieproces Virtual DOM en bijhorende DOM [12]

Het gebruik van Virtual DOM heeft dus zijn performance voordelen, maar het potentieel is echter veel groter dan bovengenoemde verbeteringen doen vermoeden. Wat als React een ander doel kon renderen dan de browser zijn bijhorende DOM? Met deze filosofie werd React Native gebouwd, waarbij in plaats van de browser DOM te renderen, React Native Objective-C APIs aangewend worden om iOS componenten te renderen en analoog Java APIs aangewend worden om Android componenten te renderen. Op basis van deze eigenschap onderscheidt React Native zich van andere cross-platform development opties, die meestal een web-based view renderen. [12]

#### One-way data flow: MVC – Flux - Redux

React maakt gebruik van Flux, een architectuur om data layers te creëren in JavaScript applicaties en een alternatief op het Model-View-Controller-model (MVC) dat in vele Java applicaties wordt toegepast. Twee voorbeelden hiervan worden aangetoond in Figuur 2.4 en 2.5. MVC bezit als architectuur echter verscheidene nadelen wanneer de applicatie complexer en groter wordt:

* De relaties tussen View, Models & Controllers worden te complex.
* De code is zeer moeilijk te debuggen.
* Oneindige loops worden te gemakkelijk geactiveerd.

Figuur 2‑4: Voorbeeld MVC [14]

Figuur 2.5 Complexer voorbeeld MVC [14]

 De Flux architectuur zorgt ervoor dat elke interactie met de View via één enkel pad naar de Dispatcher gaat, dewelke het centrale punt vormt die alle acties naar alle Stores stuurt. De Stores zelf bezitten dan de logica om te bepalen of deze actie al dan niet iets verandert in de eigen data. Elke Store is verantwoordelijk voor een domein van de applicatie. Dit domein update enkel zichzelf als reactie op de acties die worden doorgezonden vanuit de Dispatcher. Het grootste voordeel van Flux is het feit dat de data maar in één richting gaat, waardoor complexiteit, infinite loops, debug problemen, *etc.*  vermeden worden. Daarnaast is deze architectuur en flow van data eveneens veel toegankelijker voor personen die hierover nog geen kennis verworven hebben. [14]

Figuur 2.4: Complexer voorbeeld Flux [14]

Figuur 2.5 Voorbeeld Flux [14]

Tegenwoordig wordt voor de React library veelal de Redux architectuur gehanteerd, dewelke een uitbreiding vormt op Flux. Redux heeft als core dezelfde architectuur als Flux, maar lost nog enkele complexiteit issues van Flux op:

* Redux hanteert geen Dispatcher en vertrouwt op pure functies i.p.v. event emitters: pure functies zijn gemakkelijk te maken en hebben geen aparte entiteit nodig die ze managet. [15]
* De callback registration van Flux wordt vervangen door een functionele compositie waardoor *reducers* kunnen genest worden i.p.v. een Store die ‘vlak’ is en helemaal niet flexibel is i.v.m. nesting. In Flux is het moeilijk om de data te onderscheiden voor verschillende requests op de server doordat de Stores onafhankelijke alleenstaande items zijn. Dit lost Redux op door enkel één store te bevatten die gemanaged wordt door *reducers* in een tree-vorm, waarbij de root gereduceerd wordt, dan de takken die daaruit voortkomen, enzoverder. Deze kan vervolgens zeer eenvoudig de data refreshen, verschillende states van de store op slaan, *etc.* Door dit laatste kan Redux dan ook flexibel zijn qua veiligheid, aangezien het de verschillende states kan gebruiken als fail-safe. [16]

### Gebruik

Voor React Native is het niet nodig om een IDE te downloaden. Projecten worden immers opgebouwd met behulp van ‘Create React Native App’ geïnstalleerd op Node.js. Dit is een platform gebaseerd op de V8 JavaScript engine van Google. Deze genereert machine-code uit JavaScript code en verzorgt de back-end voor het mogelijk maken van front-end development zonder rekening te moeten houden met I/O events zoals web calls, netwerk communicatie, *etc.*

Nadien wordt er via de command line *npm* gebruikt om *create-react-native-app* te installeren. Nu kan het project worden gecreëerd eveneens via de command line. Daarna kan de app gebruiksvriendelijk aangepast worden in een tekst-editor naar keuze aangezien dit enkel pure Javascript code inhoudt. Om een simulatie van die applicatie te kunnen verkrijgen zijn er enkele mogelijkheden. Ten eerste is het mogelijk om Expo client applicatie te installeren op een iOS of Android apparaat en deze te connecteren op hetzelfde netwerk als de computer waarop je de app bouwt. Met deze applicatie is het mogelijk om via een QR code, die de terminal weergeeft bij het runnen van het project, de voorbeeldapp te openen. Als deze voorbeeldapplicatie werkt, kan je de applicatie in de tekst-editor veranderen en zal deze je aanpassingen real-time opnemen en weergeven.

Ten tweede kan een emulator draaien op de computer via Xcode (MAC/iOS) of Android Studio (PC/Android). De command line voorziet een automatische verbinding naar deze emulator als React Native aan het runnen is. Het voordeel hierbij is dat de keuze van apparaten waarop de app kan worden uitgevoerd bijna oneindig is. Het is bovendien zeer gemakkelijk dat er via Android Studio bijvoorbeeld de app op hetzelfde apparaat kan gesimuleerd worden om deze dan te bekijken en vergelijken op performantie, geheugen en snelheid.

### Portfolio

Duizenden apps gebruiken React Native, van *Fortune 500* bedrijven tot nieuwe startups. De bekendste voorbeelden: [17]

* Facebook (iOS & Android)
* Instagram (iOS & Android)
* Skype (iOS & Android)
* Discord (iOS)
* Tencent QQ (Android): het grootste messaging platform van China

## Xamarin

### Oorsprong

Xamarin is een ontwikkelingsplatform ontwikkeld door het gelijknamige softwarebedrijf opgericht in 2011 door de ingenieurs die Mono, Mono for Android and MonoTouch hebben gecreëerd. Dit zijn cross-platform implementaties van het Common Language Infrastructure (CLI) en Microsoft .NET. In 2013 werd het development platform Xamarin 2.0 uitgebracht. Deze release had twee voornamelijke componenten: Xamarin Studio, een open-source IDE , en een integratie met Visual Studio. Dit is Microsoft’s IDE voor het .NET framework, waardoor Visual Studio gehanteerd kon worden voor het creëren van applicaties voor Android, iOS en Windows. In 2016 werd Xamarin overgenomen door Microsoft en er werd aangekondigd dat de Xamarin SDK open-sourced zal worden en dat deze zal gebundeld worden als een gratis deel binnen Visual Studio’s geïntegreerde development omgeving. [18]

### Basis

Het Xamarin platform is een .NET omgeving met iOS en Android C# verbonden libraries. Zoals te zien in Figuur 2.8 is onderliggend aan Xamarin.Android Mono for Android, en onder Xamarin.iOS is MonoTouch. Deze zijn de C# verbindingen tot het native Android en iOS APIs voor het ontwikkelen op mobiele devices. Dit geeft het vermogen de Android/iOS UI, graphics, enz. te gebruiken terwijl er enkel in C# geschreven wordt. Xamarin.Forms is een laag boven de andere UI-verbindingen, die zorgt voor een volledig cross-platform UI library. [19]

Figuur 2.6: Opbouw Xamarin [19]

### Features

#### Xamarin.Forms

Xamarin.Forms is een toolkit van cross-platform UI-klassen gebouwd boven de meer fundamentele platform-specifieke UI-klassen: Xamarin.Android en Xamarin.iOS. Deze zijn voorzien gemapte klassen aan hun respectievelijk native UI SDK’s: iOS UIKit en Android SDK. [19] Xamarin.Forms verbindt zich ook direct aan de native Windows Phone SDK. Dit voorziet een set van UI-componenten die allemaal in de drie native operating systemen kunnen gerendered worden en dus allen cross-platform zijn.

Deze elementen zijn gebouwd met Extensible Application Markup Language (XAML) of zijn gecodeerd in C# via de Page, Layout en View klassen. Hierdoor kunnen native mobiele apps voor verschillende platformen tegelijk worden gecreëerd. Om een goede architectuur en herbruikbaarheid te bekomen, gebruikt de Xamarin.Forms mogelijkheid dikwijls gedeelde C# applicatie code die de business logic en de data access layer bevat. Dit wordt aangeduid als de Core Library. De Xamarin.Forms UI layer is eveneens C# en de eindlayerszijn een minimum aan platform-specifieke C# UI-code die nodig is voor de applicatie op elke native OS te initialiseren en op te starten (Figuur 2.9)



Figuur 2.7: Xamarin.Forms’ oplossing architectuur [19]

De trade-off tussen de native Xamarin SDKs en de Xamarin.Forms oplossing is dus de veelzijdigheid van Xamarin.Forms tegenover de volledige features en functionaliteit van de platform-specifieke UI’s.

#### MVVM (Model-View-Viewmodel)

De Xamarin.Forms ontwikkelaar creëert zijn UI typisch in XAML, en voegt daarna zijn back-end code toe dat op de user interface werkt. Als applicaties groter en complexer worden kunnen bepaalde onderhoudsproblemen toetreden aangezien de code onoverzichtelijk wordt.

Figuur 2.8 De relaties tussen de 3 kerncomponenten van MVVM [21]

Het Model-View-Viewmodel, zoals geïllustreerd in Figuur 2.10 helpt met het scheiden van de ontwikkeling van de GUI en van de back-end logica. Deze gescheiden houden zorgt ervoor om deze problemen te voorkomen en de applicatie makkelijker testbaar en onderhoudbaar te maken. Het View Model van de MVVM is de tussenstap die de data objecten van het model verandert zodat deze gemakkelijk zijn te beheren en presenteren door het View. [20]

Het Model bevat: een deel klassen die de business logica en data voorstellen. Het definieert ook de regels waarin staat hoe deze data kan veranderd en gemanipuleerd worden.

Het View stelt de UI-componenten voor. Deze is enkel verantwoordelijk voor de data weer te geven die het verkrijgt van de controller, het View Model dus.

Het View Model is verantwoordelijk voor een aantal acties. Het onderhouden van de methodes, commands en andere eigenschappen die de state van het View moeten onderhouden, het model manipuleren als resultaat van de acties op het View en events in het View triggeren. [21]

De voordelen die het MVVM-patroon aanbiedt:

* Het model, de back-end basis, aanpassen qua logica kan riskant of moeilijk zijn. In dit scenario kan het View Model bescherming bieden aan deze Model klassen en voorkomt dat er schadelijke veranderingen zouden gebeuren in de Model-code.
* Ontwikkelaars kunnen testen zonder de View te moeten gebruiken. Deze tests kunnen exact hetzelfde functioneren alsof ze gebruikt worden door een View.
* De UI van de applicatie kan compleet opnieuw ontworpen worden, als deze helemaal in XAML geïmplementeerd is, zonder ook maar één lijn back-end code aan te passen.
* Deze separatie van View en de andere componenten zorgt er ook voor dat designers en ontwikkelaars volledig apart kunnen werken aan hun eigen componenten. [22]

### Gebruik

Xamarin gebruikt de Visual Studio IDE van Microsoft als ontwikkelingsomgeving. Xamarin kan geïnstalleerd worden als deel van een nieuwe Visual Studio installatie, of kan achteraf erbij geïnstalleerd worden. Hierna kan de code voor applicaties geschreven worden in de editor van Visual Studio. Met een installatie van Xamarin.Forms zal men enkel één code moeten schrijven, en deze wordt automatisch overgebracht naar Xamarin.iOS en Xamarin.Android. Een simulatie kan dan bekeken worden met de Xamarin.Forms Previewer die rechtstreeks in de IDE is geintegreerd. Het is eveneens evident om de build rechtstreeks op een mobiel device te testen.

### Portfolio

Lijst van applicaties developed met Xamarin voor bekende bedrijven: [23]

* EasyJet’s app (Android/iOS)
* Snap Attack door Microsoft (Android/iOS)
* World Bank’s Survey Solutions (Android)
* Pepsi’s augmented reality app (Android/iOS)
* MixRadio’s muziek app (Android/iOS)
* Pinterest (Android/iOS)

## Apache Cordova/PhoneGap

### Oorsprong

Apache Cordova is een ontwikkelingsframework voor mobiele applicaties gemaakt door het Canadese bedrijf Nitobi. Eerst is het benoemd als PhoneGap, onder die naam is het gecreëerd op een iPhoneDevCamp evenement in San Francisco in 2009 door het Canadese bedrijf Nitobi. Adobe kocht in 2011 Nitobi over. [24] Nadat beslist werd door Adobe dat de PhoneGap codebase gedoneerd ging worden aan de Apache Software Foundation kreeg het de naam Apache Cordova. Deze versie werd dan open source. Hierna ging Adobe te werk om een uitgebreide versie van PhoneGap te releasen, wat de naam Adobe PhoneGap kreeg. [25] Wat is nu het essentieel verschil tussen Adobe PhoneGap en Apache Cordova? PhoneGap is Cordova plus Adobe services en extensies die de capaciteiten verder verbeteren. Cordova kan nog steeds apart gebruikt worden, maar wordt vooral gebruikt voor zijn mogelijkheden om componenten van mobiele devices te kunnen gebruiken. [26] [27]

### Basis

Apache Cordova laat toe om standaard web technologieën -HTML5, CSS3 en JavaScript- te gebruiken voor cross-platform ontwikkeling. Applicaties worden uitgevoerd binnenin native applicatie wrappers die voor elk platform specifiek gebouwd worden door het framework. Hierop komt dan een WebView die toegang krijgt tot device-level APIs waardoor het gebruik kan maken van sensoren, data, netwerk status, … Deze WebView voorziet de hele UI voor de applicatie, voor sommige platformen kan dit ook een component zijn binnenin een grote, hybride applicatie die de WebView mixt met native applicatie componenten. [28]

### https://cordova.apache.org/static/img/guide/cordovaapparchitecture.pngArchitectuur

Figuur 2.9: Architectuur van een Cordova applicatie [28]

Als de architectuur bekeken wordt, valt op dat de Web App het deel is waar de geschreven applicatie code zit. (Figuur 2.11) De applicatie zelf is geïmplementeerd als een webpagina die CSS, JavaScript en de recourses die het nodig heeft refereert. Via HTML APIs is dan een WebView gecreëerd, ofwel een HTML Rendering Engine. Deze WebView is dan geprojecteerd op het mobiel via een API van het OS zelf. De Cordova Plugins zijn eveneens belangrijk, zij voorzien een interface voor Cordova en native componenten om met elkaar te communiceren en voorzien verbindingen naar standaard device APIs zoals de camera, contacten, … [28]

### Gebruik

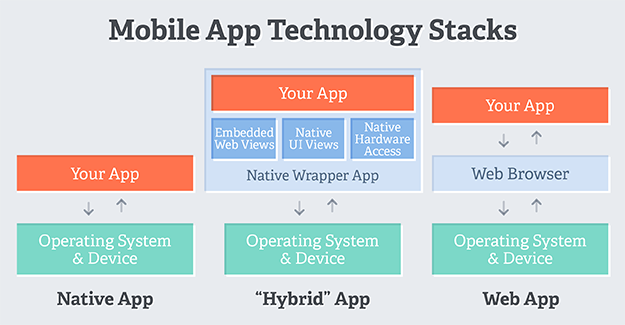
Het gebruik van Apache Cordova is vergelijkbaar met dat van React Native. Via de packager *npm,* zoals beschreven in sectie 2.2.4, kan de *cordova* CLI geïnstalleerd worden. Hierna kan via de command line dan ook een project geïnitialiseerd worden. Om het project te bewerken kan een tekst-editor naar keuze gebruikt worden.

Om de applicaties te testen kunnen er via de command line emulators gedownload worden. Deze zijn eenvoudig te gebruiken door het project te builden op een Android/iOS emulator via commands in de command line. Hierbij is de optie om de app te builden op een reëel apparaat mogelijk. Alles verloopt via de command line van het OS. Plugins zoals een camera- of accelerometercomponent installeren voor de applicatie zijn beschikbare opties. [29]

Daarnaast is het mogelijk om via bepaalde tools de architectuur van Apache Cordova te gebruiken. Adobe PhoneGap is een systeem met als fundering Apache Cordova, een ander voorbeeld is het Ionic framework. Beide frameworks breiden Cordova uit om meer features en mogelijkheden aan te bieden, hierop zijn dan ook het merendeel van de Apache Cordova applicaties op gebouwd. Maar deze zijn niet noodzakelijk om cross-platform applicaties te maken. Eindeloze mogelijkheden bestaan rond Cordova bijvoorbeeld de Visual Studio IDE gebruiken voor debugging en geavanceerde builds te lanceren, via Monaca de applicatie via een cloud online te ontwikkelen, … [30]

### Main feature: WebView

Over de details van Apache Cordova is weinig gerapporteerd en beschreven. Wat er wel duidelijk te bevinden is, is het feit dat dit framework een Webview gebaseerde aanpak heeft om de applicatie weer te geven. Dit betekent dat het in feite een browser opent om daarop een applicatie te initialiseren. In Figuur 2.12 wordt de werking van een hybride applicatie, zoals een app gebouwd met het Cordova framework, voorgesteld en vergeleken met de werking van een native en web app. Hierop valt te zien hoe de hybride app een wrapper app gebruikt die bestaat uit een WebView, een native UI view en native hardware toegang in de back-end.

Een WebView is niet meer dan een iframe of tab in een vermomde browser. Het Cordova framework zet de HTML5 code in een WebView en biedt dan een *‘foreign function interface’ (FFI)* aan. Dit is een interface die het obstakel naar het native OS overbrugt. Deze FFI zorgt er voor dat de native mogelijkheden zoals bijvoorbeeld toegang tot de camera doorgegeven worden aan het Cordova-framework. Zo kan het framework met die middelen communiceren en ze gebruiken. Een nadeel aan het gebruik van de FFI is het feit dat deze verbinding altijd trager zal gaan dan een native gebouwde applicatie. [31]

Figuur 2.10 Verschil tussen native apps, hybride apps en web apps [31]

### Portfolio

Een aantal applicaties gebouwd met als fundering Cordova zoals Ionic, PhoneGap, … [32] [33]

* Logitech SqueezeBox Controller van Logitech Inc. (PhoneGap)
* MarketWatch van Dow Jones (Ionic)
* TD trading van TD (Ionic)
* Tripcase van Saber (PhoneGap)

## Kwaliteitscontrole van de applicaties

### Theoretisch

Om de platformen van elkaar te onderscheiden wordt er bekeken in welke eigenschappen de platformen verschillen (zie Tabel 2-1). Hieruit kunnen conclusies getrokken worden om in te schatten dewelke het meest geschikt zijn om een native Android/iOS implementatie te vervangen.

Tabel 2‑1 Theoretische eigenschappen verschillende frameworks

|  |  |  |  |
| --- | --- | --- | --- |
| **Platform** | **Programmeertaal** | **UI Rendering?** | **Commercieel gebruik** |
| **React Native** | JavaScript en JSX | Native UI-controllers | Gratis, onder de MIT-licentie. [34] |
| **Xamarin** | C# en XAML | Native UI-controllers | Gratis, onder de MIT-licentie. [35] Heeft een betaalde Visual Studio Enterprise/Professional versie waarbij meer features en tools worden aangeboden vanaf 250/45$ per maand per gebruiker. [36] |
| **Qt** | C++ en QML | Native en OpenGL | Gratis. Onder de LGPL v3-licentie of GPLV2/GPLV3-licentie [37] Heeft een commerciële licentie waarbij bepaalde restricties wegvallen vanaf 460$ per maand per gebruiker. [38] |
| **Apache Cordova** | HTML5, CSS3 en JavaScript | HTML.CSS  (WebView gebaseerd) | Gratis. Onder de Apache licentie versie 2.0. [39] Heeft platformen (Adobe PhoneGap, Ionic, …) gebaseerd op Apache Cordova die wel bepaalde kosten zullen vragen voor bepaalde features te kunnen gebruiken. |

Uit de programmeertalen wordt niet veel informatie gehaald. Hierbij is het belangrijk met welke talen iemand het meest ervaren is en is het eerder een keuze uit comfort en kennis. Boeiend te bespreken is dat JSX, XAML en QML allen afgeleid zijn van een bekend alternatief:

* JSX uit HTML
* XAML uit XML
* QML uit CSS

Toch zijn deze talen vrij specifiek gebouwd voor hun respectievelijke platformen. Hierdoor wordt geconcludeerd dat de programmeertalen enkel vanuit een persoonlijke voorkeur een rol zal spelen, maar dat het enige platform zonder leercurve voor ervaren programmeurs Apache Cordova zal zijn.

Als bekeken wordt hoe de platformen hun UI rendert, dus hoe het zijn componenten gebruikt om een UI op het scherm te krijgen dan wordt opgemerkt dat zowel Xamarin als ReactNative echt native UI-controllers hanteren. Dit betekent dat bij deze platformen de applicatie niet enkel Android/iOS aanvoelt, maar effectief is. Voor deze reden is er de voorkeur gegeven om deze twee platformen uit te diepen.

Apache Cordova zal zijn applicatie starten door een WebView op te zetten die daarop dan zijn UI toont. Voor alle andere platformen zullen er effectief Android/iOS componenten aan te pas komen. Dit is een voordeel bij de *look and feel* van de applicatie aangezien de applicaties ook eerder Android & iOS zullen aanvoelen. Deze eigenschap wordt theoretisch dan ook minder interessant verklaard.

Alles is open source te vinden, dit is een voordeel aan al deze cross-platform frameworks. Eén van de redenen dat deze het hoogste zal gerangschikt staan op de finale theoretische lijst is het feit dat React Native al zijn features onder een courante licentie gratis aanbiedt, is. Dit zal zeer interessant zijn voor een bedrijf als Zappware, die ook aan de financiële kant van development frameworks moet denken.

Bij het opsommen van deze feiten wordt bevestigd dat theoretisch gezien **React Native** het boeiendste concept is van alle platformen. Het meeste potentieel als de platformen vanuit een theoretische ooghoek bekeken worden zit in een combinatie van free-to-use, native UI-controllers en een veelgebruikte bekende taal. Eén van de belangrijkste onderdelen van de criteria is de *look and feel,* aangezien **Xamarin** eveneens de native UI-controllers gebruikt komt deze als 2de meest geschikte crossplatform framework qua potentieel.

Tot slot blijft Qt en Apache Cordova over. Aangezien **Qt** geen WebView gebruikt om zijn UI op te renderen, wordt hieraan toch de voorkeur geven ten nadele van **Apache Cordova**. Door tijdsgebrek zal niet alles geïmplementeerd kunnen worden, hierdoor zullen deze laatste twee applicaties niet effectief gebouwd worden. Dit komt tevens omdat al ervaring is opgedaan met Qt in het vorige jaar. Daaruit is gebleken dat dit platform geen ideaal cross-platform bleek waarbij de *look and feel* van geïmplementeerde applicaties slechter presteerde als verwacht.

### Tools

Om de praktische controles uit te voeren is er gebruik gemaakt van een aantal tools. Deze zullen helpen bij het vergelijken in termen van performantie. Ten eerste worden deze besproken in het algemeen. Ten tweede zal er naar deze hulpmiddelen verwezen worden wanneer ze in het proces zijn toegepast.

#### Testobject

Www.testobject.com is een online test-API die via een cloud verzekert dat de Android of iOS applicaties die je ontwikkeld hebt, vlot werken op een bereik van verschillende devices. Dit werd ontwikkeld door een Amerikaans bedrijf genaamd Sauce Labs, die eerder al implementaties van cloud-hosted, web en mobiele applicaties tests automatiseerden. De feature is tegen betaling, maar heeft een open-source project genaamd Open Sauce en een gratis trial voor nieuwe gebruikers. [40]

Deze gecreëerde .apk of .ipa files, respectievelijk voor Android en iOS, is geüpload op de cloud. Deze bestanden zullen door een geautomatiseerd testomgeving beoordeeld worden. De resultaten die Testobject aanbiedt bestaan uit onder andere crash rapporten, kwaliteitsrapporten en manuele tests.

Deze rapporten worden in de vorm van logs weergegeven, waarbij de installatietijd en opstartsnelheid af te leiden zijn door de timestamps van het begin van het proces met het einde te vergelijken.

#### Activity monitor

Voor het geheugengebruik te monitoren van een Android applicatie kan simpelweg het real device gebruikt worden. De Android software heeft namelijk vanuit de eigen OS mogelijkheden om het RAM-geheugen van apps afzonderlijk te bekijken.

Indien het geheugengebruik voor een iOS applicatie gemonitord moet worden, is een tool gebruikt die vanuit de MAC de applicatie kan analyseren. Xcode heeft instrumenten ter beschikking om allerlei parameters op te meten. Dit vertrekkend van CPU-gebruik, batterijgebruik tot geheugengebruik. Dit laatste gebeurt met het Activity monitor instrument. Hierbij wordt de applicatie normaal gebruikt terwijl de app “opgenomen” wordt. Daaruit kunnen verschillende parameters afgelezen worden. Onder andere het *leaked*, het *abandoned* geheugen en het algemene geheugengebruik waaronder het cache- en RAM-geheugengebruik. Slechts de laatste twee gaan effectief besproken worden. Dit omdat de applicaties niet de complexiteit omvatten om een hoeveelheid geheugen te lekken, i.e. het geheugen wordt statisch gealloceerd en kan dus niet in een *loop* geheugen blijven alloceren.

### Praktisch

Ten eerste wordt besproken hoe de controles in het algemeen worden aangepakt. Vervolgens wordt behandeld hoe de applicaties het stuk voor stuk deden.

#### Development effort

Om de ontwikkelingstijd van de applicatie te meten, worden de stappen besproken die nodig zijn om:

* De omgeving in te stellen.
* De code, emulator en/of real device op punt te stellen.
* De applicatie uit te rollen naar iOS en Android.

Hieruit moeten de volgende vragen kunnen beantwoord worden:

* Hoe groot is de overgangstap naar het benoemde platform qua setup?
* Hoe moeilijk is de taal, vanuit een beginnerstandpunt?
* Hoe lang duurt het implementeren van een prototype applicatie?
* Hoe gedocumenteerd is de taal?
* Wat zijn de mogelijkheden van het platform qua schaalbaarheid, bij grotere complexere applicaties?
* Is de mogelijkheid om de applicatie te publiceren reëel?
* Is het platform eerder geschikt om prototypes mee te bouwen of kan het als effectieve vervanger van de native Android en iOS projecten dienen?

In dit hoofdstuk wordt eerst onze persoonlijke ervaring met het proces van de ontwikkeling op een chronologische wijze besproken. Naderhand wordt dit veralgemeend, waaruit conclusies worden getrokken.

#### Geheugengebruik

Android

Het geheugengebruik wordt getest op een eigen echt apparaat, geen simulatie dus. Dit gebeurt op een Motorola Moto C Plus met als belangrijkste specificaties:

**OS**: Android 7.0

**API level**: 23

**CPU**: Quad-core 1.3 GHz Cortex-A53

**RAM**: 2048 MB

iOS

Het geheugengebruik is getest op een eigen echt apparaat, de iPhone SE met dezelfde specificaties als degene waarop de snelheid gemeten wordt:

**OS**: iOS 11.3

**CPU**: Dual-core 1.84 GHz Twister

**RAM**: 2048 MB

Hier is het cache-geheugen, het RAM-geheugen en het interne geheugen met elkaar vergeleken. Hoe minder geheugen er gebruikt wordt, hoe beter.

Maar aangezien iOS geen optie heeft om het RAM-geheugen te aanschouwen, moet dit zich voordoen via *Monitor Memory Usage*, zoals beschreven in sectie 2.5.2.2.

#### Snelheid

Android

Om de snelheid van deze applicaties te testen is *testobject.com* benut, zoals besproken in sectie 2.5.2.1. Als resultaten worden er logs vanuit het device verkregen. Er is gezocht naar volgende lijn output op die logs (vb. Xamarin, Android):

13:15:52.347912935/?I/ActivityManager: Displayed XamarinProject.XamarinProject.Android/md5d751a0ebcd50a7cee6c5dbde2620e260.MainActivity: +6s601ms

Hieruit kan de opstartsnelheid gehaald worden, namelijk 6.601s voor dit voorbeeld. De installatiesnelheid bezorgt Testobject achteraf in zijn kwaliteitsrapport.

Eén van de devices die op de test-API beschikbaar is: de Motorola Moto E (2nd generation), gratis te gebruiken en leunt dicht aan bij het eigen apparaat. De belangrijkste specificaties van dit device zijn:

**OS**: Android 6.0

**API level**: 23

**CPU**: ARM | quad-core | 1200 MHz

**RAM**: 1024 MB

iOS

Om de snelheid van de iOS applicatie testen worden console commands van Xcode ingeschakeld. Dit is een onderdeel dat Testobject niet in zijn resulterende logs weergeeft. Daardoor moet dit manueel uitgevoerd worden in Xcode. Door een omgevingsvariabele mee te geven in Xcode kan de pre-main time, oftewel de tijd die nodig is om de applicatie te initialiseren, geanalyseerd worden en via de console in XCode vrijgegeven worden. (vb. React Native, iOS):

*Total pre-main time: 46.39 milliseconds (100.0%)*

*dylib loading time: 13.92 milliseconds (30.0%)*

*rebase/binding time: 6.25 milliseconds (13.4%)*

*ObjC setup time: 6.01 milliseconds (12.9%)*

*initializer time: 20.15 milliseconds (43.4%)*

*slowest intializers :*

*libSystem.B.dylib : 4.50 milliseconds (9.7%)*

*libBacktraceRecording.dylib : 3.40 milliseconds (7.3%)*

*MediaServices : 3.30 milliseconds (7.1%)*

*ProjectReactNative : 12.54 milliseconds (27.0%)*

Hierbij kan de opstartsnelheid gehaald worden,namelijk 46.39 ms. Bij dezen is geweten hoeveel tijd bepaalde delen van de initialisatie nodig had. De installatiesnelheid bezorgt Testobject achteraf in zijn kwaliteitsrapport.

Het device dat hiervoor gehanteerd wordt is de Apple iPhone SE. Dit is eveneens hetzelfde dat ter ons beschikking is gesteld. De belangrijkste specificaties van dit device zijn:

**OS**: iOS 11.3

**CPU**: Dual-core 1.84 GHz Twister

**RAM**: 2048 MB

Aangezien deze applicaties worden vergeleken met hetzelfde apparaat zal dit in deze vergelijkende studie niet uitmaken voor zowel Android als iOS.

Onder de snelheid gaat begrepen worden: de installatiesnelheid, de tijd die dit apparaat nodig heeft om de applicatie te installeren, en de opstarttijd of/en de tijd die het nodig heeft om de geïnstalleerde app op te starten zonder dat er iets in het cachegeheugen gestoken is.

Opvallend is dat bij het latere onderdeel *user experience*, (sectie 4.4) eveneens de vraag voorkomt hoe snel de applicatie-startsnelheid aanvoelt. Bij dezen kan geconcludeerd worden of deze snelheid minder/meer voelbaar is in vergelijking met de concurrentie.

#### User experience

Voor het user experience-onderdeel is het moeilijk harde cijfers te vinden over bepaalde parameters omdat deze subjectief zijn, bijvoorbeeld vlotheid, uitstraling of functionaliteit. Dit zijn allemaal eigenschappen waar personen anders op reageren. Vandaar dat de gebruikerservaring getest wordt met behulp van een enquête. Tien personen zullen de applicaties beoordelen op bepaalde factoren. De factoren worden getest aan de hand van een aantal vragen. Deelnemers duiden een score op een schaal van van één tot vijf aan. De steekproefomvang is tien, telkens personen met een verschillende achtergrond en kennis van zaken. De vragen worden gesteld voor alle platformen waarop een applicatie uitgerold is en de native applicatie die collega’s van Zappware ontworpen hebben. Dit betekent: drie voor Android en drie voor iOS. In totaal zullen dus zes applicaties getest worden.

De tabel met deze te beoordelen onderdelen voor beide operating systems is te vinden op Tabel 2-2.

*Geef een score van één tot vijf op de volgend vragen.*

**Android/iOS**

Tabel 2‑2 Enquête user experience

|  |  |  |  |
| --- | --- | --- | --- |
|  | App 1 | App 2 | App 3 |
| Hoe snel start de applicatie op? |  |  |  |
| Scroll door de lijst, van begin tot einde en terug, hoe vlot verloopt dit? |  |  |  |
| Zet de video op pauze, hoe responsief is de videoplayer? |  |  |  |
| Druk op de datum, hoe responsief is deze actie? |  |  |  |
| Scroll nu nog eens door de lijst, hoe vlot verloopt dit? |  |  |  |
| Geef een score op de algemene look & feel van de applicatie. Zonder rekening te houden met de gegenereerde visuele aspecten zoals de afbeeldingen en tekstobjecten. |  |  |  |

# Applicatie

De applicatie die gebouwd wordt via de cross-platform frameworks zal een basic tv-gids applicatie zijn. Deze zijn basic opdat de mogelijkheid bestaat om ze te kunnen vergelijken op basis van verschillende criteria zoals development effort, snelheid, geheugen, enzovoorts. De layout zal op de platformen verschillen van elkaar afhankelijk van welke componenten beschikbaar zijn. De reden hiervoor is dat er in andere talen gewerkt wordt. Het algemene beeld van de applicatie gaat er als volgt uitzien:

Figuur 3.1: Einddoel applicatie

Boven zijn er drie componenten: één logo zonder functies, één zone die het uur/moment/dag aanduidt en één zoek-optie met een zoekfunctie waarbij er bepaalde shows kunnen gezocht worden.

Hieronder is de video-component die een stream van het huidige onderdeel op het aangeduide kanaal moet kunnen weergeven.

Onderaan wordt per kanaal getoond wat op het aangeduide moment actief is, hoelang dit al actief is, of dit opgenomen wordt, … En hierbij moeten er categorieën, onder andere favorieten en sport, als keuzes kunnen aangeduid worden. Deze optie is nadien eruit gelaten vermits dit meer codewerk is en niet essentieel is. Dit om de tijd beter te beheren.

Om een voorbeeld applicatie te bouwen is een JSON-bestand geüpload op een webserver met behulp van www.myjson.com zodat deze kan aangesproken worden op elk apparaat via het internet. Dit JSON-bestand bevat alle gegevens over de programmatie van de kanalen, alle metadata van de kanalen, het feit dat het programma momenteel aan het opnemen is, …. Dit is gemakkelijk aan te spreken in alle programmeertalen opdat dit consequent kan gebeuren bij de verschillende platformen. Eveneens is dit een kleine database met 23 kanalen zodat eventuele performantie-problemen niet onnodig groot zijn en nog op dezelfde manier kunnen beoordeeld worden. Het doel is om op elk platform deze voorbeeldapp zo goed mogelijk na te bouwen en de limieten en mogelijkheden van het platform te vinden.

# Vergelijking

## Development effort

### React Native

#### Het proces

Om te starten met een React Native applicatie, heeft men een Javascript-library en een React-framework nodig. Via Node.js, een softwareplatform voor Javascript-toepassingen die gewoonweg te gebruiken is met behulp van een command prompt van de PC, installeren we “Create React Native App”. In dit framework zit alles dat nodig is. Eens dit geïnstalleerd is kaneen React Native project worden opgezet. Dit project kan via een tekst-editor worden aanpassen en uitbreiden. Expo is een applicatie die we dan op het mobiel device kunnen installeren om een voorbeeld van onze applicatie te projecteren. Dit framework opzetten in zijn geheel duurde 1,5u in totaal. Er kan ook een emulator van Android Studio of Xcode worden gebruiken voor een voorbeeld hoe de applicatie er zal uit zien.

Documentatie voor React Native is heel uitgebreid te vinden op het internet. Voor een complexvrije applicatie zoals deze is genoeg informatie te vinden op de officiële pagina van React Native. Wat opvalt bij het gebruiken van het React Native-framework is dat bij errors het debuggen niet vlot verliep. Errors geven problemen aan op plaatsen die niet het probleem waren, problemen met de emulator, problemen bij het installeren van bepaalde componenten voor je applicatie, … Op het internet staan wel oplossingen maar dit resulteerde in een zoektocht. Hierdoor is er veel tijd kwijtgeraakt aan kleine problemen. De algemene indruk is dat het een nieuw framework is, die nog volop in ontwikkeling is. Om die reden zijn er nog heel wat complicaties mogelijk en die niet steeds gemakkelijk op te lossen zijn.

Om bepaalde componenten, in ons geval bijvoorbeeld videoplayers , te hanteren kon men Expo niet meer inschakelen omdat hiervoor (nog) geen ondersteuning geboden werd. Daarom moest je het project losmaken van Expo. Dit betekent dan dat dit onderdeel van de infrastructuur wegvalt. Daarom moest alles via Android Studio moest opgezet worden en alles niet meer vlot verliep zoals bij het Expo-build proces.

Nadat de app af is, moest deze nog in Android en in iOS gebouwd worden. Bij Android is onze Windows command line aangewend om een *release* versie van onze applicatie te bouwen. Om een applicatie te runnen in een reële Android omgeving moeten er eerst een aantal stappen gebeuren. Eerst moet deze gesigneerd worden met een certificaat om deze te kunnen installeren. Hiervoor moeten we een key genereren waarmee er een APK kan gegeneerd worden die gesigneerd is. Deze key is te creëren via een tool die Java aanbiedt: *keytool.* Met deze tool krijgen we een file*: my-release-key.keystore*. Deze moet op zijn beurt in de map van onze applicatie bijgevoegd worden en via onze *gradle* code van Android aanspreken met het juiste wachtwoord. Dat paswoord hebben we ingesteld waardoor onze applicatie in Androidomgevingen kan geïnstalleerd worden. Dit instellen, bouwen en uitzoeken duurde één uur in totaal.

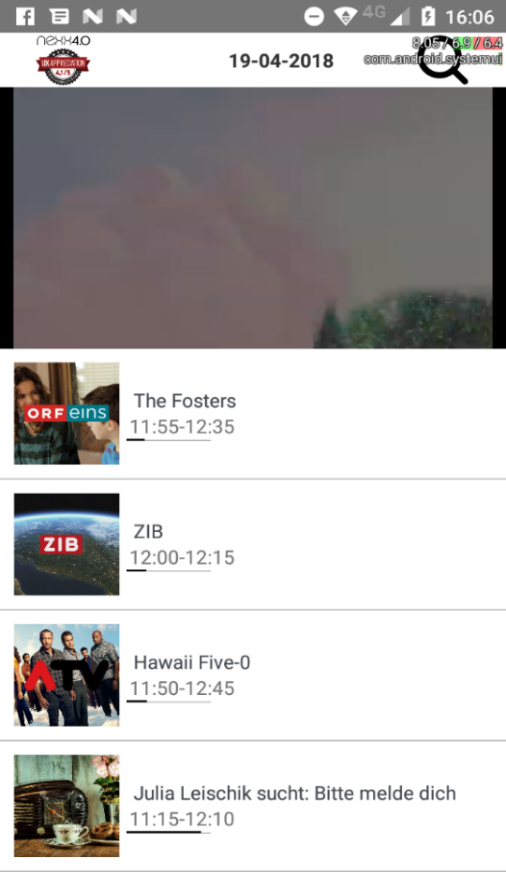
De installatie op iOS volgt een analoog concept. Aangezien de Android applicatie als eerste ontwikkeld was, moest enkel de omgeving errond geregeld worden en de benodigde pakketten voor onze applicatie installeren. De geschreven code voor de Android app is? hierna direct bruikbaar in Xcode.

Het deployment van deze iOS applicatie loopt eveneens analoog. In XCode wordt de React Packager ook gebruikt om de applicatie te bouwen naar een .ipa vorm. Deze moet gesigneerd worden door een geverifieerd Apple ID. Dit was echter een uitdaging, enkele aanpassingen – die allemaal online te vinden waren – werden toegepast. Na zwoegen werd een werkende .ipa file bekomen.

Door problemen werd een simpel project tijdrovend om te creëren en heeft het uiteindelijk naar schatting 80 uren geduurd om het te maken. Hierbij moet rekening gehouden worden met het gebrek aan ervaring en *knowhow* waarmee dit project begon, aangezien het de eerste applicatie was die gemaakt werd. Hierdoor is dit getal niet helemaal representatief.

#### Details

Beginnend aan een dergelijke applicatie, wordt eerst de structuur van de app nagemaakt en in componenten opgedeeld: (<> stelt Javascript componenten voor)

* Vanboven: één <View> opgedeeld in 3 stukken rij, daarin kunnen we 3 componenten steken. In dit geval:
  + Links: <Image>
  + Midden: <DatePicker>: Een component die ervoor zal zorgen dat je een datum kunt kiezen, zowel in Android als iOS
  + Rechts: <Image> met functionaliteit onPress: een zoekactie ondernemen.
* Midden: een <VideoPlayer> component die enkel een url als bron nodig heeft om te werken in zowel Android als iOS.
* Vanonder: een <Flatlist> waarmee het JSON-bestand kan uitlezen worden en via <ListItem> deze één voor één getoond wordt. In component ListItem wordt dan bepaald welke data gekozen wordt via verschillende props die worden ingesteld.

Figuur 4.1: React Native applicatie

### Xamarin

#### Het proces

Voor de Xamarin applicatie te creëren is de Visual Studio IDE nodig. Deze installatie duurt lang en neemt veel geheugen in beslag. Na drie uur is de installatie compleet en kunnen we beginnen aan een lege applicatie. Bij deze lege applicatie werden direct problemen gevonden: vele instellingen stonden standaard verkeerd en het debuggen via een emulator had allerlei kinderziektes, van niet opstarten tot errors aanduiden zonder een bruikbare reden. Hiervoor waren op het internet wel oplossingen te vinden maar doordat de problemen bleven opstapelen, duurde het nog drie uur voordat er effectief begonnen kon worden aan de applicatie met de juiste omgeving. Voor een snelle preview bestaat er in VS het handig venster Xamarin.forms previewer, hier kan men zien hoe de applicatie eruit ziet op verschillende platformen.

Om de applicatie effectief te testen gebruiken we een *real* device, de Moto C Plus, via USB. Dit kan ook met Xamarin Live, een applicatie in de Play Store of App Store waarbij de gsm verbonden wordt met Visual Studio en in real-time een voorbeeldapplicatie op je apparaat kan uitgerold worden. Maar dit wordt niet gedaan aangezien er geen debugging kan gebeuren van de code, wat cruciaal is. Er zijn mogelijkheden in VS verwerkt via emulators waarvan de installatie omslachtig is en veel geheugen in gebruik neemt, daarom werd er een real device.

Tijdens het schrijven van de code viel direct op hoe sommige componenten niet beschikbaar zijn. Een videoplayer-component vinden die native videospelers aanspreekt was -in het begin- enkel te vinden tegen betaling. Terwijl hiervoor bij React Native wel direct meerdere gratis versies te vinden waren. Uiteindelijk werd een werkende, gratis Xamarin video player verkregen door een uitgebreide gamma aan open source pakketten te doorzoeken. Het installeren van deze uitbreidingen ging onbelemmerd echter het implementeren niet. Mede door errors met de oorzaak bij het .NET framework waarop deze gebaseerd was terwijl VS een heel ander .NET framework had getarget. Nadat de geïnstalleerde componenten geevalueerd waren kon er geconcludeerd worden dat er nog een paar Visual Studio installatie onderdelen ontbreken. Er moet rekening gehouden worden met het feit dat de standaardinstallatie van VS niet voldoende is om alle componenten te gebruiken.

Na 40 uren werken aan de Xamarin implementatie was de applicatie nog niet klaar. Ten gevolge van tijdsgebrek werd het project gepauzeerd. Ondanks was de structuur bruikbaar en moesten enkel details afgewerkt worden. Maar deze konden potentieel geheel tijdrovend zijn.

Het uitrollen van de applicatie op Android was evident dankzij de interne documentatie van Xamarin. Desondanks was er bij het uitrollen nog wat kinderziektes die VS maar niet kan vermijden. Deze waren direct opgelost, de APK-file was direct bruikbaar.

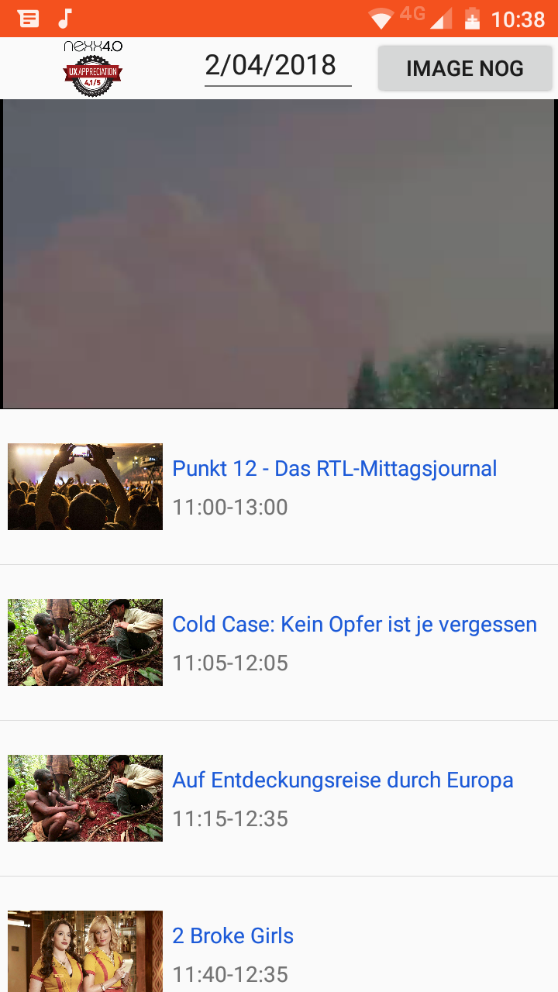
Voor de app op iOS uit te rollen was de connectie scheppen met een MAC eenvoudig en logisch. Visual Studio kon zelf connecteren met de MAC die is gebruikt en zo Xcode zelf gebruiken om de applicatie te renderen en uitrollen op een iPhone. De IPA file instellen was eveneens een analoge zaak als de APK-file. De documentatie hierrond was duidelijk en gestructureerd.

#### Details

Wegens tijdsgebrek is de Xamarin implementatie niet beëindigd . Hiervolgt de structuur: (<> stelt XAML componenten voor)

* Eén <Grid> opgedeeld in drie kolommen en één rij, daarin kunnen er drie componenten in geplaatst worden. In dit geval:
  + Links: <Image>
  + Midden: <DatePicker>:
  + Rechts: <Button> met een standaard eigenschap onClicked die via een C# functie te verbinden is met de gegeven functionaliteit + een image component hierop instellen (niet geimplementeerd).
* Midden: een <VideoPlayer> component die enkel een url als bron nodig heeft om te werken in zowel Android als iOs.
* Vanonder: een <Listview> die statisch geregistreerde data-onderdelen weergeeft. Bij de <ListView> zou een JSON uitgelezen moeten worden. Om de structuur van onze applicatie te behouden hebben we onze JSON onderdelen dan omgezet naar statische eigenschappen van een klasse <List>, de data manueel toegevoegd en deze klasse gebruikt als databron voor onze <ListView>.

Meer niet-geïmplementeerde onderdelen:

* Een tijdsbalk die aangeeft hoever het programma gevorderd is.

Figuur 4.2: Xamarin applicatie

### Conclusie

Het gebruik van Xamarin en React Native is weinig verschillend. Het feit dat React Native geen IDE hanteert is een groot voordeel voor de setuptijd van het framework. Dit bedraagt maar één uur, terwijl de installatie van Visual Studio drie uren inneemt. Daarbij komt kijken dat veel tools voor het debuggen ontbreken, die Visual Studio wel uitgebreid aanbiedt. Er kan enkel uitgegaan worden van de error-berichten die de applicatie aangeeft. Maar daarvoor zijn alternatieven, zoals React Native Devtools, die open-source te vinden zijn en die meer info bieden over details van de applicatie.

De talen waarin geschreven werd voelen vertrouwd aan. In zowel C# als Javascript is er inmiddels al ervaring opgedaan. Het feit dat React Native voornamelijk JS is voelde echter als een gemakkelijkere omgeving dan de C# van Xamarin Hier is de structuur toch onduidelijker en structurele fouten niet makkelijk te herkennen. Javascript voelt directer en eenvoudiger aan. Zeker bij een straight-forward applicatie als deze.

Dit wordt vereenvoudigd door de documentatie rond Xamarin vrij onduidelijk en moeilijk te vinden is. Goede voorbeelden zijn te vinden na lang zoeken, maar hun eigen documentatie brengt meestal niet meer op dan weten wat een bepaalde functie doet, zonder context. De gemeenschap rond React Native hebben een veel actiever en behulpzamer voorkomen. Voor elk soort probleem is reeds een antwoord. Daarentegen is bij Xamarin nog geen pasklare oplossing. Er werd een probleem aangekaart waarop een oplossing werd bedacht, dit kost tijd en inzicht. Vooral voor een onervaren persoon zal dit de doorslag geven. Bij beiden was het proces van de setup, allerlei installaties en de essentiële basis goed gedocumenteerd.

De structuur van beide frameworks is bruikbaar voor grotere complexe applicaties. Dit dankzij hun respectievelijke MVVM- en Flux-architectuur. Bij Xamarin zorgt MVVM ervoor dat de back-end en de front-end mooi gescheiden blijft, en werkt met een tussenstation voor het versturen van de data. Terwijl voor React Native Flux de data slechts naar één kant stuurt waarbij één dispatcher alle *callbacks* verzendt naar de entiteiten waarvoor deze bestemd is.

Het deployment in Android was bij React Native en Xamarin compleet identiek proces, die beiden goed gedocumenteerd waren. Enkel bij Xamarin werkte de .apk file niet van de eerste keer, maar dit vormde op lange termijn geen probleem. Bij iOS waren wel verschillen, het project moest voor React Native volledig op de MAC gezet worden. Hierna moest deze in Xcode geopend worden. Om dit dante laten werken waren verschillende aanpassingen aan de instellingen in Xcode nodig. In Visual Studio moest slechts één connectie gemaakt worden met de MAC en dit betekende dat Xcode enkel als tussenpoort moest dienen om de .ipa file te creëren en de applicatie op het device te krijgen.

Uit deze ervaring kan er een opsomming gemaakt worden die kort weergeeft wat de voor- en nadelen zijn van de twee cross-platform frameworks vergeleken met elkaar, zoals aangetoond in Tabel 4-1.

Tabel 4‑1 Vergelijking ontwikkelingseigenschappen

|  |  |  |
| --- | --- | --- |
|  | **React Native** | **Xamarin** |
| **Setup** | Kort (1u) en palmt  weinig geheugen in. | Relatief lang (3u) en palmt veel geheugen in. |
| **Debugging** | Mogelijkheden beschikbaar. | Uitgebreide geïntegreerde mogelijkheden. |
| **Programmeertalen** | Vertrouwd, weinig complexiteit | Vertrouwd, matige complexiteit. |
| **Documentatie** | Uitgebreid qua eigen documentatie. Zeer veel vragen/antwoorden op het internet. | Matig uitgebreid qua eigen documentatie. Matig veel vragen/antwoorden op het internet. |
| **Scalability** | Zeer schaalbaar | Zeer schaalbaar |
| **Deployment**  **Android** | Straight-forward | Straight-forward |
| **Deployment**  **iOS** | Na instelling-wijzigingen:  straight-forward | Straight-forward |

Bij deze vergelijking kan er worden geconcludeerd dat React Native en Xamarin aan elkaar gewaagd zijn als het over de ontwikkeling op het framework gaat. Dit is gebaseerd op het feit dat ze beiden voor- en nadelen hebben, bv. het deployment op iOS is eenvoudig op Xamarin, maar dit is weinig relevant aangezien een one-time effort voldoende is om het React Native project werkende te krijgen, en de meer uitgebreide debug-mogelijkheden van Visual Studio. Dit tegenover de ready-to-use eenvoudigheid die React Native aanbiedt door geen IDE te behandelen, een programmeertaal die minder ervaring en inzicht zal vergen en de documentatie die uitgebreider te vinden valt.

Door deze bevindingen kan geconcludeerd worden dat om Xamarin efficiënt te gebruiken eerder aan een langetermijn-project gedacht moet worden. Dit terwijl React Native een ideale prototype-omgeving aanbiedt om kortetermijn-projecten op te bouwen. Als we hierbij de theoretische vergelijking aanhalen is het opvallend dat deze frameworks dan ook ideaal zijn om op deze manier te gebruiken. React Native heeft een eenvoudige open-source licentie waarbij alle features en tools gratis te vinden zijn. Xamarin heeft eveneens die open-source mogelijkheden, maar een groot deel features, tools en mogelijkheden zitten in Visual Studio Enterprise/Professional die wel enkel tegen een maandelijkse betaling beschikbaar zijn. Dit versterkt de theorie dat React Native het betere prototype-framework zou zijn, aangezien een bedrijf geen financiële middelen willen inzetten om een framework te kunnen hanteren die het niet altijd gebruikt.

Het gebruik zelf is uiteindelijk aan de ontwikkelaar, hierdoor is een zekere persoonlijke voorkeur van toepassing. Daarom is een eindige conclusie nooit van de orde.

## Geheugen

Het minimaliseren van alle soorten geheugens dat een applicatie inneemt is belangrijk zowel om het interne opslagruimte van de gebruiker te besparen als het RAM-geheugen niet te verzadigen.   
Om het geheugenverbruik van de applicaties te vergelijken, moet eerst duidelijkheid geschept worden over het feit dat de apps niet hetzelfde zijn. De native applicatie heeft meer inhoud, bijvoorbeeld: meerdere pagina’s, meer achtergrondprocessen, een authenticatiescherm, enz. dan de cross-platform gebouwde apps. De React Native en Xamarin resultaten zijn ongeveer dezelfde indien dit bekeken wordt vanuit een geheugen standpunt. Zij verschillen enkel in hoe het zijn data haalt. React Native haalt dit uit een online JSON terwijl Xamarin dit statisch doet. Goed om te weten is dat de libraries die de Xamarin applicatie nodig heeft om een JSON te kunnen uitlezen bij de APK zit hiervan. Dit betekent dat de grootte van Xamarin wel degelijk kan vergeleken worden met die van React Native. De grootte van de native versie is echter weinig representatief tegenover de andere, maar ook hier kunnen nog conclusies uit getrokken worden.

### Vergelijking

Android

Op de Motorola Moto C wordt de interne opslag en het cachegeheugen rechtstreeks uitgelezen. Voor het RAM-geheugen wordt een maximale waarde gebruikt, die het device zelf registreert, nadat de applicatie uitgebreid gebruikt is. Dit gebeurt omdat in het werkgeheugen een fluctuerend aantal bytes wordt gebruikt. De resultaten werden opgenomen in Tabel 4-4.

Tabel 4‑4 Vergelijking geheugengebruik Android applicaties

|  |  |  |  |
| --- | --- | --- | --- |
|  | Native (Android) | React Native | Xamarin |
| Interne opslagruimte |  |  |  |
| RAM-geheugen | 343 |  |  |
| Cachegeheugen |  |  |  |

De opvallendste resultaten zijn het verschil in gebruik van RAM-geheugen. De native applicatie is logischerwijs het zwaarste. De reden hiervoor is dat deze veel uitgebreider is als de cross-platform alternatieven. Hierbij is opvallend dat de React Native versie een serieus zwaardere klant is dan de Xamarin versie wat actief geheugengebruik betreft. Zowel het RAM-geheugen als het cachegeheugen wordt meer op de proef gesteld bij de React Native applicatie. Dit kan te wijten zijn aan het feit dat de React Native applicatie meer functionaliteiten bevat, zoals bv. de tijdlijn-aanduider. Bij sectie user experience zal besproken worden of dat een effect zal hebben op de performantie. Ook opvallend is dat voor de Xamarin app geen significant cachegeheugen wordt opgeslagen in tegenstelling tot de React Native app.

iOS

Op de iPhone SE kan eveneens de applicatie-grootte en het cache-gebruik rechtstreeks worden afgelezen. Voor het RAM-geheugen op te meten moet een instrument genaamd Activity Monitor, zoals besproken in sectie 2.5.2.2, in Xcode gebruikt worden. Hierbij werd de hoogste waarde genomen voor dezelfde reden als bij Android. De resultaten voor de iOS applicaties kunnen gevonden worden in Tabel 4-5.

Tabel 4‑5 Vergelijking geheugengebruik iOS applicaties

|  |  |  |  |
| --- | --- | --- | --- |
|  | Native (iOS) | React Native | Xamarin |
| Interne opslagruimte |  |  |  |
| RAM-geheugen |  |  |  |
| Cachegeheugen |  |  |  |

Dito voor de iOS versies gelden dezelfde regels als bij Android. Het RAM-geheugen heeft dezelfde resultaten als men de volgorde bekijkt, dit heeft dan ook dezelfde redenen. Opvallend is dat het minder RAM-geheugen nodig heeft in het algemeen. Dit is een iOS eigenschap waarbij de werking van het OS ervoor zorgt dat er minder RAM moet gealloceerd worden, voor Android moet er namelijk aan *garbage collection* gedaan worden terwijl dit principe niet bestaat voor een iOS systeem. [41]

Opvallend is hoe bij iOS de React Native versie geen cachegeheugen alloceert terwijl nu dit bij Xamarin wel gebeurt. Eveneens zijn de .ipa files exact dezelfde grootte terwijl voor Android de React Native applicatie toch significant kleiner was.

### Conclusie

Als we het geheugengebruik analyseren van de applicaties, kan het geheugengebruik van de native app t.o.v. de cross-platform applicaties niet vergeleken worden. Aangezien de native app veel uitgebreider is, zal deze meer geheugen in beslag nemen in alle regionen. Dit is dan ook te zien in de resultaten waarbij deze steeds het grootst aantal bytes is. Wel is te zien dat deze applicatie amper cachegeheugen zal innemen doordat deze werkt met een GraphQL fetch-systeem waarbij query’s worden aangemaakt afhankelijk van wat de gebruiker aangeduid heeft.

Indien de resultaten van de React Native en Xamarin applicaties naast elkaar worden gelegd, is het duidelijk dat de React Native applicatie zwaarder is voor het systeem dan zijn Xamarin concurrent. Dit kan gedeeltelijk verklaard worden door de extra features. Het is dan ook aangewezen om de komende resultaten over de snelheid en de enquête af te wachten om een oordeel te vellen over het significantie hiervan. Wat wel gaat meespelen is het feit dat Android en iOS z’n cachegeheugens anders alloceert. Bij een statische toewijzing wordt dit wel in de cache van iOS opgeslagen maar niet in die van Android. Bij een JSON-verwijzing gebeurt exact het omgekeerde. Dit zal een impact hebben op de gebruikerservaring van de applicaties.

## Snelheid

### Vergelijking

Bij deze studie is de installatiesnelheid en de opstartsnelheid van de applicaties in React Native en Xamarin vergeleken met degene die in Android en iOS native gebouwd zijn. Vooral de opstartsnelheid is uitermate belangrijk aangezien deze actie een cruciaal onderdeel is van een mobiele applicatie.

Android

Voor Android worden de snelheden van de applicatie op de Motorola Moto E (2nd generation) getest en vergelijken we deze met elkaar. Hiervoor werd de tool *Testobject* gebruikt zoals beschreven in sectie 2.5.3.2. Deze snelheden van de verschillende applicaties is te vinden in Tabel 4-2.

Tabel 4‑2 Vergelijking snelheden Android applicaties

|  |  |  |  |
| --- | --- | --- | --- |
|  | Native (Android) | React Native | Xamarin |
| Installatietijd [s] |  |  |  |
| Opstarttijd [s] |  |  |  |

Als de installatietijden van de applicaties bekeken worden is de correlatie duidelijk. Dit is makkelijk te verklaren aan het feit dat de grootte van de .apk file exact correleren aan de installatietijd. Zoals besproken in sectie 4.2.1, is React Native de kleinste applicatie en de native app de grootste. Dit is een logische uitkomst aangezien het OS hetzelfde soort bestand moet installeren.

De opstarttijden verschillen duidelijk. Hierbij valt het op hoeveel tijd meer de Xamarin app nodig heeft om te initialiseren. Dit verschil kan verklaard worden doordat deze zijn database statisch opbouwt en dit iedere keer moet doen aangezien dit niet naar het cachegeheugen wordt gealloceerd. De verhouding React Native – Native kan verklaard worden door de complexiteit van de applicaties.

iOS

Voor iOS werd eveneens Testobject gebruikt met als device de iPhone SE om de installatietijd te bepalen. Voor de opstarttijd werd een omgevingsvariabele in XCode gebruikt zoals beschreven in sectie 2.5.3.2. De snelheden in de verschillende iOS apps kan bekeken worden in Tabel 4-3.

Tabel 4‑3 Vergelijking snelheden iOS applicaties

|  |  |  |  |
| --- | --- | --- | --- |
|  | Native (iOS) | React Native | Xamarin |
| Installatietijd [s] |  |  |  |
| Opstarttijd [s] |  |  |  |

In het oog springend is dat in het algemeen ze veel lager liggen dan bij de tests bij Android. Dit is simpelweg wegens de hardware die verwerkt zit in de Iphone SE, de Dual-core 1.84 GHz Twister is een serieuze upgrade in termen van kloksnelheid t.o.v. de Quad-core 1.3 GHz Cortex-A53 van de Motorola Moto C Plus.

De opstarttijden zijn zeer vergelijkbaar en geen enkele is aan de problematische kant. Dit is een serieus verschil t.o.v. de Android applicaties. Vooral de snelheid van de Xamarin applicatie is merkbaar, die ondanks de statische opbouw nu toch geen problemen ondervindt hierdoor. Een mogelijke verklaring is de betere CPU die de Iphone bezit.

### Conclusie

Een applicatie beoordelen aan de installatietijd die het OS nodig heeft geen meerwaarde. Deze staat namelijk evenredig met de grootte van die applicatie zelf. De opstarttijd daarentegen is wel een nuttige parameter. Hieruit kon er al geconcludeerd worden dat bij Android de Xamarin app merkbaar slechter presteert als zijn concurrenten. Bij iOS heeft deze wel een goed resultaat. Dit wordt dan ook aan het verschil in hardware en opbouw van de applicatie gewijd, en niet aan het framework zelf. Als dit resultaat aan het framework lag, zal deze consequent slecht bij iOS en Android geweest zijn. Door deze nuances wordt er beoordeeld dat er geen significant verschil gaat zijn in de opstarttijd tussen de cross-platform frameworks, met als bewijs de iOS-tijden die slechts een klein verschil aangaven die ook verklaard kon worden aan het verschil in aantal features. De voelbaarheid en significantie van deze specifieke opstarttijden zullen bij het onderdeel user experience nog geëvalueerd worden.

## User experience

De gebruikerservaring is uitermate belangrijk voor een mobiele applicatie. Dit onderdeel is van hoogste prioriteit voor softwarebedrijven zoals Zappware aangezien dit voor alle gebruikers een cruciale vereiste is van een app. Snelheid en geheugen zijn een van de weinige meetbare onderdelen die uiteindelijk de gebruikerservaring zal beïnvloeden, dit is dan ook uiteindelijk het belangrijkste onderdeel van deze masterproef. Namelijk het bepalen of het gebruik van bepaalde cross-platform frameworks de user experience zal hinderen. Zoals besproken in sectie 2.5.3.4 is het bepalen van de gebruikerservaring voor mobiele applicaties een subjectieve zaak. Bepaalde argumenten opmeten is nuttig maar schetst niet het volledige beeld.

### Resultaten enquête

De enquête is uitgevoerd op tien personen. De resultaten werden verwerkt in Microsoft Excel 2016. Als parameters om de resultaten te analysen werden het steekproefgemiddelde en mediaan genomen. Deze worden besproken in Tabel 4-6 voor Android en Tabel 4-7 voor de iOS versie van de applicaties. Evenzeer is het eerste getal het steekproefgemiddelde en het tweede de mediaan van alle resultaten voor het bepaalde onderdeel. Deze onderdelen zijn afgekort tot de essentie en volgen dezelfde volgorde als de vragen in de enquête zoals besproken in sectie 2.5.3.4.

**Android**

Tabel 4‑6 Resultaten enquête Android applicaties

|  |  |  |  |
| --- | --- | --- | --- |
|  | React Native | Native Android | Xamarin |
| Snelheid | 4,1 / 4 | 3,4 / 3 | 1,9 / 2 |
| Scroll 1 | 3,3 / 3 | 5 / 5 | 2,7 / 2,5 |
| Video | 5 / 5 | 5 / 5 | 5 / 5 |
| Datum | 4,3 / 4,5 | 5 / 5 | 4,5 / 4,5 |
| Scroll 2 | 4,1 / 4 | 5 / 5 | 3 / 3 |
| Score | 3,7 / 4 | 5 / 5 | 3,5 / 3,5 |

**iOS**

Tabel 4‑7 Resultaten enquête iOS applicaties

|  |  |  |  |
| --- | --- | --- | --- |
|  | React Native | Native iOS | Xamarin |
| Snelheid | 2,8 / 3 | 4,7 / 5 | 5 / 5 |
| Scroll 1 | 5 / 5 | 5 / 5 | 4,7 / 5 |
| Video | 5 / 5 | 5 / 5 | 5 / 5 |
| Datum | 3,6 / 3,5 | 4,6 / 5 | 3,2 / 3,5 |
| Scroll 2 | 5 / 5 | 5 / 5 | 4,3 / 4,5 |
| Score | 3,5 / 4 | 5 / 5 | 3,9 / 4 |

### Bespreking resultaten

Eerst zullen de resultaten besproken worden die een logische uitkomst hebben en die een eenvoudige verklaring hebben. Vervolgens bekijken we de interessante uitkomsten die meer conclusies bevatten over het platform waarin de applicaties gebouwd zijn.

Indien de resultaten van de native apps bekeken worden, is het duidelijk dat deze (bijna) perfecte scores behaald over alle vragen. Aan deze applicatie zijn dan ook vele uren werk voorafgegaan. Dit om het essentieel onderdeel, de gebruikerservaring zo goed mogelijk te optimaliseren. De beoordelingen van de native app is een goede maatstaaf om de andere applicaties te kunnen beoordelen. Het enige dat opvalt is het feit dat de opstartsnelheid, zoals besproken in sectie 4.3, minder goed scoort op Android. De verklaring hiervoor lag aan het verschil in hardware, waarbij de Android hardware niet optimaal bevonden werd. Deze opstartsnelheid kan wel niet vergeleken worden met de andere opstartsnelheden aangezien de native app uitgebreider en complexer is dan de cross-platform alternatieven.

De componenten van alle applicaties zoals de videoplayer- en datepickercomponent werken allemaal naar behoren. Enkel de datumcomponent van de iOS versies bij de hybride apps werd niet als optimaal beoordeeld. Dit resultaat kan te wijten zijn aan het feit dat iOS zijn eigen componenten zoals de klok niet als een apart onderdeel beschouwt, hierdoor stond deze in de weg van de datepickercomponent. Bij Android gaf dit nochtans geen problemen. Hiervoor moeten stappen ondernomen worden in het proces voor de hybride apps om met dit gebrek in iOS rekening te houden. Dit gebrek zal resulteren in de conclusie dat een hybride app altijd deels code zal moeten bevatten voor platform-specifieke eigenschappen, waardoor een volledig hybride applicatie die meteen perfect werkt in iOS en Android eerder een uitzondering zal zijn.

De resultaten achter de opstartsnelheid zijn consistent met de resultaten in sectie 4.3. Waarbij React Native op Android beter scoort terwijl Xamarin op iOS een beter resultaat behaalt. Bovendien zijn de resultaten in die sectie volop besproken en gelinkt aan diverse redenen. Als in de enquête bekeken wordt hoeveel deze verschillen merkbaar waren, valt het op dat het gigantische verschil in opstarttijd bij Android even mild werd beoordeeld als het kleine verschil op iOS. Dit kan verklaard worden aan een perceptie waarbij het verschil tussen onmiddellijk openen en niet onmiddellijk openen op de Iphone zwaar beoordeeld wordt. Terwijl de gemiddelde opstarttijden op de Android hoger liggen worden deze even mild t.o.v. elkaar vergeleken. Als we de absolute cijfers van de resultaten hierop bekijken is het oordeel over de snelheid van de Xamarin app op Android toch minder goed dan de React Native app op iOS.

De vragen die met het scrollen van de lijst te maken hebben zijn opgesplitst in twee onderdelen. Deze opsplitsing was vereist aangezien het onderscheid bestaat tussen een eerste *load* van een lijst en een tweede, nl. dat bij de eerste laadbeurt alle data nog opgehaald moet worden, terwijl bij de volgende deze data reeds opgehaald is. De discrepantie hiertussen valt op te merken aan de beoordelingen, waarbij de eerste laadbeurt bij Android consequent slechter scoort. Opvallend is het feit dat dit verschil bij iOS niet voorkomt, dit is te wijten aan het verschil in de kracht van de CPU die ervoor zorgt dat er geen *jitters*, dit zijn haperingen in een applicatie, zullen voorkomen. De algemene beoordeling van het scrollen in de applicaties scoort beter bij de React Native applicatie dan het Xamarin alternatief.

Als de algemene scores op de *‘look and feel’* van de applicaties overlopen worden, valt te zien dat de Xamarin en React Native applicaties aan elkaar gewaagd zijn. Een beoordeling vellen over deze twee is tevens niet gemakkelijk. Ze zien er ongeveer hetzelfde uit en verschillen enkel in achtergrondstructuur en bepaalde componenten zoals bv. de videoplayer. Bij Android haalt de React Native app een gemiddelde 0,2 hoger dan de Xamarin app. Bij iOS is het omgekeerde waar met een verschil van 0,4. Hieruit kan geconcludeerd worden dat een React Native applicatie licht wordt geprefereerd in het Android OS, terwijl aan de Xamarin applicatie de voorkeur gegeven wordt in het OS van Apple.

### Conclusie

De enquête duidt een aantal bevindingen aan die kunnen helpen om een algemene conclusie te trekken. De native applicatie is beduidend de betere applicatie, in bijna alle opzichten presteert het beter dan de hybride alternatieven. Dit is een logische uitkomst aangezien veel meer middelen werden ingezet om deze op punt te krijgen. Wat betekent dat die een goede maatstaaf zal zijn om de andere applicaties tegen op te wegen.

De Xamarin applicatie behaalt in het algemeen hogere scores als de iOS applicatie besproken wordt. Dit fenomeen is omgekeerd bij de React Native app. Als de *jitters* besproken worden die de Android applicaties ondervinden, gebeuren deze voornamelijk bij de Xamarin applicatie. Hier worden een aantal elementen uit gehaald. Ten eerste dat de React Native versie nog veel werk vergt om de iOS variant te optimaliseren. Ten tweede dat de Xamarin versie voor Android zwaar aangepast moet worden om aan bepaalde condities te voldoen. De statische toewijzing van de lijst zou dit kunnen verklaren, zowel de opstarttijd als het scrollen door diezelfde lijst. Door dit gegeven is een onbetwiste conclusie trekken dat React Native een betere user experience zal opleveren niet mogelijk, en vice versa. De mogelijkheden om in beide frameworks een goed werkende applicatie te verkrijgen bestaan en worden reeds volop gebruikt, maar om een perfect uitziende volledig werkende app te verkrijgen is het nog steeds aangeraden om twee native versies te bouwen. Deze conclusie is getrokken uit het feit dat niet alles geoptimaliseerd kan worden tot in het kleinste detail bij de hybride apps, hier hang je vast aan bepaalde componenten die beperkte variabelen zullen beschikken t.o.v. componenten in een native applicatie. Daarom is een native applicatie ook zo populair, werkelijk alles is aanpasbaar en de mogelijkheden zijn enorm. Zeker een bedrijf als Zappware die de user experience hoog in het vaandel moet dragen zal het beste af zijn met de native applicaties te bouwen.

# Besluit

Een cross-platform framework is een mooi idee in de theorie. In de realiteit zullen deze niet alle oplossingen bieden om de native ontwikkeling van applicaties te vervangen. Te veel factoren zorgen ervoor dat de algemene prestatie en mogelijkheden van de hybride alternatieven ondermaats is t.o.v. die van de klassieke applicaties. Dit betekent niet dat er geen mogelijkheden zijn om die alternatieven nuttig in te zetten. Zo kunnen deze perfect ingeschakeld worden als prototype-framework om demo’s op te bouwen voor bepaalde toepassingen. De besproken frameworks zijn aan elkaar gewaagd als we de werking van de twee gecreëerde apps bespreken. De algemene voorkeur voor prototype-applicaties op te lanceren gaat alsnog naar React Native. Dit volledig open-source framework is eenvoudiger en beter gedocumenteerd dan zijn concurrent Xamarin. Deze is een mogelijkheid waarbij de focus gelegd wordt op producten met een lange houdbaarheid, maar een dure optie om als parttime framework te gebruiken. Ondanks deze bevindingen is de keuze uit deze alternatieven een persoonlijke keuze, afhankelijk van een groot aantal parameters die verschillen qua belangrijkheid van ontwikkelaar op ontwikkelaar.

# Verwijzingen

|  |  |
| --- | --- |
| [1] | Zappware NV, „Info over Zappware NV,” [Online]. Available: https://zappware.com/. |
| [2] | „Qt (software),” [Online]. Available: https://en.wikipedia.org/wiki/Qt\_(software). [Geopend Maart 2018]. |
| [3] | Digia , „Digia and Qt have demerged into two companies – Digia’s new strategy’s main themes revealed,” [Online]. Available: http://digia.com/en/actual/news/2016/digia-and-qt-have-demerged-into-two-companies--digias-new-strategys-main-themes-revealed/. [Geopend Maart 2018]. |
| [4] | N. Mehrotra, „Qt: What’s best about the cross-platform development toolkit,” [Online]. Available: http://opensourceforu.com/2017/06/qt-cross-platform-development-toolkit/. [Geopend Maart 2018]. |
| [5] | B. Starynkevitch, „How is QT cross-platform,” [Online]. Available: https://www.quora.com/How-is-QT-cross-platform. [Geopend Maart 2018]. |
| [6] | „OpenGL,” [Online]. Available: https://en.wikipedia.org/wiki/OpenGL. [Geopend Maart 2018]. |
| [7] | „GLUT - The OpenGL Utility Toolkit,” [Online]. Available: https://www.opengl.org/resources/libraries/glut/. [Geopend Maart 2018]. |
| [8] | Qt, „Signals & slots,” [Online]. Available: http://doc.qt.io/archives/qt-4.8/signalsandslots.html. [Geopend Maart 2018]. |
| [9] | „Applications using Qt,” [Online]. Available: https://en.wikipedia.org/wiki/Qt\_(software)#Applications\_using\_Qt. [Geopend Maart 2018]. |
| [10] | „React (JavaScript library),” [Online]. Available: https://en.wikipedia.org/wiki/React\_(JavaScript\_library). [Geopend Maart 2018]. |
| [11] | M. Konicek, „React Native: A year in review,” [Online]. Available: https://code.facebook.com/posts/597378980427792/react-native-a-year-in-review/. [Geopend Maart 2018]. |
| [12] | B. Eisenman, Learning React Native, O'Reilly Media, Incc., 2016. |
| [13] | „Document Object Model,” [Online]. Available: https://en.wikipedia.org/wiki/Document\_Object\_Model. [Geopend Maart 2018]. |
| [14] | M. Tilley, „What is Flux?,” [Online]. Available: http://fluxxor.com/what-is-flux.html. [Geopend Maart 2018]. |
| [15] | Redux, [Online]. Available: https://redux.js.org/introduction/prior-art. [Geopend Maart 2018]. |
| [16] | D. Abramov, „Why use Redux over Facebook Flux?,” [Online]. Available: https://stackoverflow.com/questions/32461229/why-use-redux-over-facebook-flux. [Geopend Maart 2018]. |
| [17] | React Native, „Who's using React Native?,” [Online]. Available: http://facebook.github.io/react-native/showcase.html. [Geopend Maart 2018]. |
| [18] | „Xamarin,” [Online]. Available: https://en.wikipedia.org/wiki/Xamarin. [Geopend Maart 2018]. |
| [19] | D. Hermes, Xamarin Mobile Application Development - Cross-platform C# and Xamarin.Forms Fundamentals, Apress, 2015. |
| [20] | „Model-View-Viewmodel,” [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel. [Geopend April 2018]. |
| [21] | „Understanding MVC, MVP and MVVM Design Patterns,” [Online]. Available: http://www.dotnettricks.com/learn/designpatterns/understanding-mvc-mvp-and-mvvm-design-patterns. [Geopend April 2018]. |
| [22] | Microsoft, „MVVM,” [Online]. Available: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm. [Geopend April 2018]. |
| [23] | Xamarin, [Online]. Available: https://www.xamarin.com/customers. [Geopend Maart 2018]. |
| [24] | „Apache Cordova,” [Online]. Available: https://en.wikipedia.org/wiki/Apache\_Cordova. [Geopend April 2018]. |
| [25] | B. LeRoux, „PhoneGap, Cordova, and what's in a name?,” [Online]. Available: https://phonegap.com/blog/2012/03/19/phonegap-cordova-and-whate28099s-in-a-name/. [Geopend April 2018]. |
| [26] | S. A. Wilkins Fernandez, Beginning App Development with Parse and PhoneGap, Apress, 2015. |
| [27] | F. Cheng, Build Mobile Apps with Ionic 2 and Firebase. |
| [28] | Apache Cordova, „Overview Apache Cordova,” [Online]. Available: https://cordova.apache.org/docs/en/latest/guide/overview/. [Geopend April 2018]. |
| [29] | „Guide Install Apache Cordova,” [Online]. Available: https://cordova.apache.org/docs/en/latest/guide/cli/index.html. [Geopend April 2018]. |
| [30] | „Apache Cordova,” [Online]. Available: https://cordova.apache.org/. [Geopend Mei 2018]. |
| [31] | „What is a webview,” [Online]. Available: https://developer.telerik.com/featured/what-is-a-webview/. [Geopend April 2018]. |
| [32] | „Apps PhoneGap,” [Online]. Available: https://phonegap.com/app/. [Geopend Mei 2018]. |
| [33] | „Top application made with Ionic,” [Online]. Available: http://showcase.ionicframework.com/apps/top. [Geopend Mei 2018]. |
| [34] | „License React Native,” [Online]. Available: https://github.com/facebook/react-native/blob/master/LICENSE. [Geopend April 2018]. |
| [35] | [Online]. Available: https://blog.xamarin.com/xamarin-for-all/. [Geopend April 2018]. |
| [36] | „Pricing Visual Studio Licenses,” [Online]. Available: https://www.visualstudio.com/vs/pricing/. [Geopend April 2018]. |
| [37] | „Open source licenseQt,” [Online]. Available: http://doc.qt.io/qt-5/opensourcelicense.html. [Geopend April 2018]. |
| [38] | „Licensing comparison Qt,” [Online]. Available: https://www1.qt.io/licensing-comparison/. [Geopend April 2018]. |
| [39] | „License Apache Cordova,” [Online]. Available: https://github.com/apache/cordova-android/blob/master/LICENSE. [Geopend April 2018]. |
| [40] | „Open Source,” Sauce Labs, [Online]. Available: https://saucelabs.com/open-source. [Geopend Mei 2018]. |
| [41] | „iOS is twice as memory-efficient as Android. Here’s why.,” [Online]. Available: https://www.cultofmac.com/303223/ios-twice-memory-efficient-android-heres/. [Geopend Mei 2018]. |
| [42] | M. S. Jasmin Blanchette, C++ GUI Programming with Qt 4, Trolltech Press, 2006. |
| [43] | B. C. Daniels, „QT – Introduction C++ GUI Programming with Qt 4,” [Online]. Available: http://slideplayer.com/slide/7747920/. [Geopend Maart 2018]. |
| [44] | Microsoft, „The MVVM Pattern,” [Online]. Available: https://msdn.microsoft.com/en-us/library/hh848246.aspx. [Geopend April 2018]. |

Bijlagen

Bijlage A Javascript-code van de React Native applicatie

Bijlage B XAML-code van de Xamarin applicatie

Bijlage C C#-code van de Xamarin applicatie

Bijlage D Resultaten enquête

**A**

|  |
| --- |
| "use strict";  import React, {Component} from 'react';  import { StyleSheet, Image, View, ScrollView, ListView, Text, TouchableHighlight, AppRegistry, Modal, FlatList} from 'react-native';  import VideoPlayer from 'react-native-video-controls';  import DatePicker from 'react-native-datepicker';  import SearchBar from 'react-native-searchbar';  import Dimensions from 'Dimensions';  import {List, ListItem} from 'react-native-elements';  import Hr from 'react-native-hr-component';  var elements = [];  function parseJsonDate(jsonDateString){  return Date.parse(jsonDateString);    }  function calculatePercentage(start, end){  var starttijd = parseJsonDate(start);  var eindtijd = parseJsonDate(end);  var nu = parseJsonDate('2018-02-28T12:03:30Z');  var calc1 = eindtijd - starttijd;  var calc2 = eindtijd - nu;  var calc3 = calc1 - calc2;  var calcpercent = calc3/calc1;  return calcpercent;  }  function calculatePercentageAnder(start,end){  var starttijd = parseJsonDate(start);  var eindtijd = parseJsonDate(end);  var nu = parseJsonDate('2018-02-28T12:03:30Z');  var calc1 = eindtijd - starttijd;  var calc2 = eindtijd - nu;  var calc3 = calc1 - calc2;  var calcpercent = 1 - calc3/calc1;  return calcpercent;  }    export default class App extends React.Component {  constructor(){  super()    var today = new Date(),  today = today.getDate() + '-' + (today.getMonth() + 1) + '-' +today.getFullYear();  global.vandaag = today;      this.state = {  items,  results: [],  date: today,  count: 0,  modalVisible: false,  data: []  };  this.\_handleResults = this.\_handleResults.bind(this);    }    componentWillMount() {  this.fetchData();  }  fetchData = async () => {  const response = await fetch("https://api.myjson.com/bins/87f6f");  const json = await response.json();  this.setState({ data: json.data.channelList.channels.edges });    };    renderdate(){  if (this.state.date == vandaag){  return (  this.state.date  );  }  else{  return (  this.state.date  );  }}  \_handleResults(results) {  this.setState({ results });  }      render() {  let logo = {  uri: 'http://zappware.com/wp-content/uploads/2017/04/1.png'}  let zoek = {  uri: 'https://cdn4.iconfinder.com/data/icons/pictype-free-vector-icons/16/search-128.png'}  ;  var { width, height } = Dimensions.get('window')  return (  <View style={styles.overallcontainer}>  <Modal visible={this.state.modalVisible} animationType={'slide'} onRequestClose={() =>this.closeModal()}>  <View style={{ marginTop: 110 }}>  {  this.state.results.map((result, i) => {  return (  <Text key={i}>  {typeof result === 'object' && !(result instanceof Array) ? 'gold object!' : result.toString()}  </Text>  );  })  }  </View>  </Modal>  <View style={styles.container}>  <Image source={logo} style={{flex:1, resizeMode: Image.resizeMode.contain}} />  <View style={{flex: 2, backgroundColor: 'white'}} >  <DatePicker  style={{width: 200}}  date={this.renderdate() }  mode="date"  placeholder="select date"  format="DD-MM-YYYY"  minDate="01-06-2016"  maxDate="02-07-2019"  confirmBtnText="Confirm"  cancelBtnText="Cancel"  showIcon={false}  hideText={false}  customStyles={{  dateInput: {  borderWidth: 0,  marginLeft: 20,  },  dateText:  {  fontWeight: 'bold',  }  }}  onDateChange={(date) => {this.setState({date: date})}}  />  </View>  <TouchableHighlight underlayColor="white" onPress={() => this.doSearchopen() } style={{flex: 1, backgroundColor: 'white', marginBottom: 2, marginTop: 2}} >  <Image source={zoek} style={{flex:1, resizeMode: Image.resizeMode.contain,}} />    </TouchableHighlight>  </View>  <View style={styles.video} >  <VideoPlayer source={{ uri: 'http://d23dyxeqlo5psv.cloudfront.net/big\_buck\_bunny.mp4' }} navigator={ this.props.navigator }/>  </View>  <View style={styles.channel} >      <FlatList  data={this.state.data}  keyExtractor={(x, i) => i}  renderItem={({ item }) =>  (<ListItem  hideChevron={true}  avatar ={  <View style={{height: 75, width: 75 ,justifyContent: 'center', alignItems: 'center'}}>  <Image source={{uri: item.node.eventsAt.TVitem.smallImage.url}} style={{height:75, width:75, justifyContent: 'center', alignItems: 'center', backgroundColor: 'rgba(0,0,0,0.5)' }} >  <Image source={{uri: item.node.logo.url}} style={{resizeMode: 'center', height: 60, width: 60, alignItems: 'center', }} />  </Image>  </View>  }  title={item.node.eventsAt.TVitem.title}  subtitle={  <View style={styles.totalbox}>  <Text style={styles.textbox} > {item.node.eventsAt.TVitem.start.substring(11,16)}-{item.node.eventsAt.TVitem.end.substring(11,16)}</Text>  <View style={styles.viewboxoverhead}>  <View style={{flex: calculatePercentage(item.node.eventsAt.TVitem.start, item.node.eventsAt.TVitem.end), borderBottomColor: 'black', borderBottomWidth: 1.5,marginLeft: 5}} />  <View style={{flex: calculatePercentageAnder(item.node.eventsAt.TVitem.start, item.node.eventsAt.TVitem.end), borderBottomColor: 'grey', borderBottomWidth: 0.3, marginRight: 200}} />  </View>  </View>  }      />  )}  />      </View>  <SearchBar  ref={(ref) => this.searchBar = ref}  data={items}  handleResults={this.\_handleResults}  />  </View>    );}  doSearchopen(){  this.searchBar.show();  this.setState({modelVisible:true});  }  doSearchclose(){  this.searchBar.hide();  this.setState({modelVisible:false});  }  }  const items = [    ];    const styles = StyleSheet.create({  container:  {  height: 40  ,  backgroundColor: "white",  flexDirection: 'row',  },  video:  {  flex: 1,  backgroundColor: 'red',  },  channel:  {  flex: 2,  backgroundColor: 'white',  flexDirection: 'column',  },  overallcontainer:  {  flex: 1,  backgroundColor: 'blue',  },  scrollview:  {  flex: 1,    }  ,  totalbox:{  flexDirection: 'column'  },  textbox:  {  flex:1,  },  viewboxoverhead:  {  flexDirection: 'row',  flex:1,  },    }); |

B

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  xmlns:local="clr-namespace:XamarinProjectWerkNuIs"  xmlns:xamarians="clr-namespace:Xamarians.MediaPlayer;assembly=Xamarians.MediaPlayer"  x:Class="XamarinProjectWerkNuIs.MainPage">  <ActivityIndicator VerticalOptions="Center" HorizontalOptions="Center" x:Name="activity\_indicator" Color="#4D7EE1" />  <StackLayout Spacing="0" x:Name="layout">  <Grid HorizontalOptions="FillAndExpand" VerticalOptions="Start" ColumnSpacing="0" >  <Grid.RowDefinitions>  <RowDefinition Height="40" />    </Grid.RowDefinitions>  <Grid.ColumnDefinitions>  <ColumnDefinition Width="\*"/>  <ColumnDefinition Width="\*"/>  <ColumnDefinition Width="\*"/>  </Grid.ColumnDefinitions>  <Image Source="http://zappware.com/wp-content/uploads/2017/04/1.png" Grid.Column="0" VerticalOptions="FillAndExpand" HorizontalOptions="FillAndExpand"/>  <DatePicker Grid.Column="1" VerticalOptions="End" HorizontalOptions="Center"/>  <Button Text="Image nog aanpassen" Grid.Column="2" VerticalOptions="End" HorizontalOptions="End" />  </Grid>      <xamarians:VideoPlayer Source="http://d23dyxeqlo5psv.cloudfront.net/big\_buck\_bunny.mp4" AutoPlay="True" HeightRequest="200" VerticalOptions="Fill"/>  <ListView x:Name="myList" HasUnevenRows="true" ItemsSource="{Binding Items}" >  <ListView.ItemTemplate>  <DataTemplate>  <ViewCell>  <!-- Here we make a Horizontal orientation with the help of StackLayout-->  <StackLayout Orientation="Horizontal" Margin="5" HeightRequest="90">  <Image Source="{Binding smallimageURL}" WidthRequest="100" HeightRequest="200" Aspect="AspectFit" >  <!--<Image Source="{Binding imageURL}" WidthRequest="20" HeightRequest="40" Aspect="AspectFit" VerticalOptions="Center"/>-->  </Image>  <StackLayout VerticalOptions="Center">  <Label Text="{Binding title}" TextColor="#1C5AD8" />  <StackLayout Orientation="Horizontal" Padding="0">  <Label Text="{Binding end}" />    </StackLayout>  </StackLayout>  </StackLayout>    </ViewCell>  </DataTemplate>  </ListView.ItemTemplate>  </ListView>  </StackLayout>  </ContentPage> |

C

|  |
| --- |
| using Newtonsoft.Json;  using Plugin.Connectivity;  using System;  using System.Collections.Generic;  using System.Collections.ObjectModel;  using System.Diagnostics;  using System.Linq;  using System.Net.Http;  using Xamarin.Forms;  using XamarinProjectWerkNuIs.ViewModels;  namespace XamarinProjectWerkNuIs  {  public partial class MainPage : ContentPage  {  public int Count = 0;  public short Counter = 0;  public int SlidePosition = 0;  int heightRowsList = 90;  private const string Url = "https://api.myjson.com/bins/e7jqf";  // This handles the Web data request  private HttpClient \_client = new HttpClient();  public MainPage()  {  InitializeComponent();  BindingContext = new ItemListViewModel();  }    }  } |

**D**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Persoon 1 | Persoon 2 | Persoon 3 | Persoon 4 | Persoon 5 | Persoon 6 | Persoon 7 | Persoon 8 | Persoon 9 | Persoon 10 |  | Gemiddelde | Mediaan |
| RNANsnel | 4 | 4 | 5 | 5 | 3 | 4 | 5 | 4 | 4 | 3 |  | 4,1 | 4 |
| RNANscroll1 | 3 | 3 | 4 | 4 | 4 | 3 | 2 | 3 | 4 | 3 |  | 3,3 | 3 |
| RNANvideo | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| RNANdatum | 4 | 4 | 3 | 5 | 5 | 4 | 5 | 3 | 5 | 5 |  | 4,3 | 4,5 |
| RNANscroll2 | 4 | 4 | 4 | 5 | 4 | 4 | 3 | 4 | 5 | 4 |  | 4,1 | 4 |
| RNANscore | 3 | 4 | 3 | 4 | 4 | 4 | 4 | 3 | 4 | 4 |  | 3,7 | 4 |
| RNiOSsnel | 4 | 4 | 1 | 4 | 1 | 2 | 3 | 4 | 2 | 3 |  | 2,8 | 3 |
| RNiOSscroll1 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| RNiOSvideo | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| RNiOSdatum | 4 | 5 | 3 | 5 | 1 | 3 | 2 | 5 | 3 | 5 |  | 3,6 | 3,5 |
| RNiOSscroll2 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| RNiOSscore | 3 | 4 | 2 | 4 | 5 | 4 | 3 | 2 | 4 | 4 |  | 3,5 | 4 |
| NAANsnel | 3 | 3 | 3 | 4 | 3 | 4 | 4 | 3 | 3 | 4 |  | 3,4 | 3 |
| NAANsroll1 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| NAANvideo | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| NAANdatum | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| NAANscroll2 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| NAANscore | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| NAiOSsnel | 5 | 5 | 4 | 5 | 4 | 5 | 5 | 5 | 4 | 5 |  | 4,7 | 5 |
| NAiOSscroll1 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| NAiOSvideo | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| NAiOSdatum | 4 | 5 | 5 | 5 | 5 | 4 | 4 | 5 | 5 | 4 |  | 4,6 | 5 |
| NAiOSscroll2 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| NAiOSscore | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| XAANsnel | 2 | 2 | 1 | 3 | 2 | 1 | 3 | 2 | 1 | 2 |  | 1,9 | 2 |
| XAANscroll1 | 3 | 2 | 2 | 3 | 4 | 2 | 4 | 3 | 2 | 2 |  | 2,7 | 2,5 |
| XAANvideo | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| XAANdatum | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 4 | 5 | 4 |  | 4,5 | 4,5 |
| XAANscroll2 | 3 | 3 | 2 | 5 | 2 | 1 | 3 | 4 | 4 | 3 |  | 3 | 3 |
| XAANscore | 3 | 3 | 4 | 4 | 4 | 3 | 4 | 3 | 4 | 3 |  | 3,5 | 3,5 |
| XAiOSsnel | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| XAiOSscroll1 | 4 | 5 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 4 |  | 4,7 | 5 |
| XAiOSvideo | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |  | 5 | 5 |
| XAiOSdatum | 4 | 5 | 3 | 5 | 1 | 5 | 1 | 3 | 4 | 1 |  | 3,2 | 3,5 |
| XAiOSscroll2 | 4 | 5 | 4 | 5 | 2 | 5 | 4 | 5 | 5 | 4 |  | 4,3 | 4,5 |
| XAiOSscore | 3 | 4 | 3 | 5 | 4 | 4 | 5 | 4 | 4 | 3 |  | 3,9 | 4 |

**faculteit INDUSTRIËLE INGENIEURSWETENSCHAPPEN**

CAMPUS GEEl

Kleinhoefstraat 4

2440 GEEL, België

tel. + 32 14 72 13 00

iiw.geel@kuleuven.be

[www.iiw.kuleuven.be](http://www.iiw.kuleuven.be)