KU LEUVEN

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN

CAMPUS GEEL

Applicatieontwikkeling in cross-platform en platform-specifieke frameworks

Een vergelijkende studie

Lennert VAN LOOVEREN

Promotor: Dr. Ing. Peter Karsmakers

Co-promotor: Bram Schrijvers

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

Academiejaar 2017-2018

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 verdient een dankwoord, om 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 om de tijd te nemen 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 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 omvat, waarvoor weinig setup benodigd is en bovendien zijn functies en features gratis aanbiedt. Xamarin daarentegen heeft een meer uitgebreide IDE in Visual Studio waardoor het debuggingsproces aangenamer en duidelijker verliep. Desondanks bleek dit framework minder goed gedocumenteerd te zijn, waarbij niet alle features open-source te vinden zijn. Uit deze studie werd besloten dat Xamarin eerder een concept voor langetermijn-projecten is, terwijl React Native een ideale omgeving is voor prototypes op te bouwen. Nadien werden de gecreëerde applicaties getest op verschillende factoren: snelheid, geheugengebruik en user experience. Deze laatste factor zal de belangrijkste zijn aangezien de snelheid en geheugengebruik uiteindelijk de gebruikerservaring beïnvloedt.

Om deze ervaring te testen, werd een enquête afgenomen waarbij tien deelnemers verschillende vragen over de gecreëerde applicaties dienden in te vullen. De resultaten toonden aan dat de native applicatie, de meest complete en gebruiksvriendelijkste was. Wanneer vervolgens de resultaten van de React Native en Xamarin app vergeleken worden, blijkt dat slechts minimale verschillen kunnen waargenomen worden tussen beide applicaties. Het framework bleek niet aan de grondslag te liggen van deze verschillen, waardoor de impact van het framework op de gebruikerservaring niet significant verklaard werd. Wat wel opviel is de performantie van deze apps, aangezien ze geen perfecte resultaten behaalden, ondanks het feit dat ze basic ontworpen waren. Uit deze informatie werd geconcludeerd dat native applicatieontwikkeling toch de juiste keuze is om de gebruikerservaring te garanderen. Dit betekent echter niet dat een cross-platform framework geen plaats kan hebben in een development omgeving. Een framework zoals React Native kan immers perfect gebruikt worden om prototypes en demo’s op te ontwikkelen. Dit laatste heeft React Native te danken aan het feit dat het met Javascript een bekende taal gebruikt, over veel voorhanden documentatie beschikt en alle mogelijkheden open-source aangeboden krijgt.

INHOUD

LIJST MET FIGUREN

LIJST MET TABELLEN

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	Microsoft Visual C++
DOM	Document Object Model
IDE	Integrated development environment
APK	Android Package File
IPA	Iphone Application Archive
OS	Operating system
CLI	Command line interface

1 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 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.

2 CROSS-PLATFORM FRAMEWORKS

2.1 Qt

2.1.1 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 voor beide platforms wordt dezelfde API ingezet. 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]

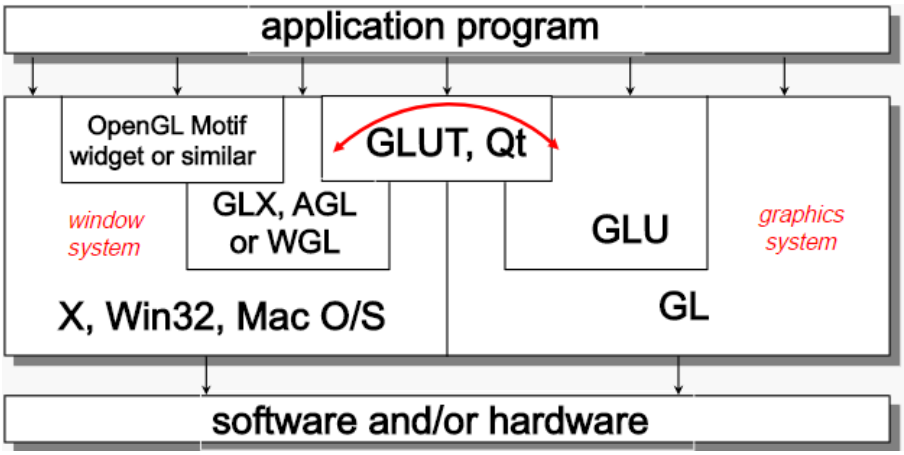
In 2016 is Digia en “The Qt Company” gesplitst in 2 onafhankelijke bedrijven. [3]

2.1.2 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 erin 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 om 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]

2.1.3 Grafische architectuur



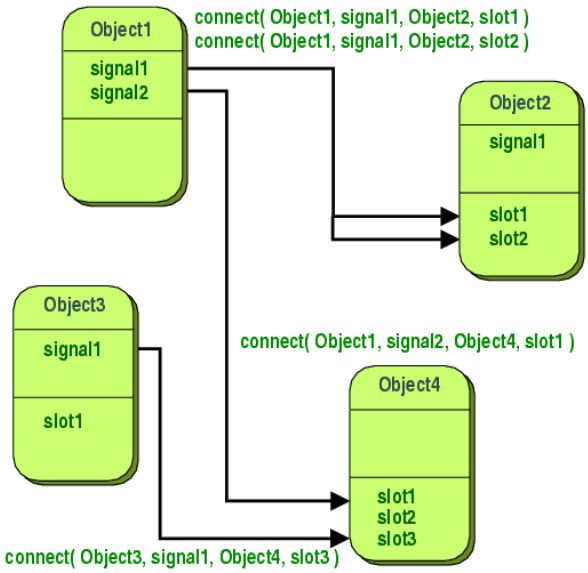
Figuur 2.1 de grafische architectuur van Qt [43]

De grafische architectuur die toegepast wordt op Qt is 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 grafisch 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.

2.1.4 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 *event* 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.

2.1.5 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 te 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.

Het installeren en opstarten wijst zichzelf uit. Om de omgeving te laten opereren moeten enkele omgevingsvariabelen aangepast worden aan Windows/MAC. Er dienen eveneens development kits geïnstalleerd te 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.

2.1.6 Portfolio

Noemenswaardige applicaties die Qt of QML gebruiken: [9]

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

2.2 React Native

2.2.1 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 erin 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 jaar lang ontwikkeling van het prototype, kreeg het framework zijn eerste taak toebedeeld: het ontwikkelen van een onafhankelijk werkende iOS app, met als doel 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 publiekelijk tentoongesteld en tegen het einde van het jaar werd het volledig open source. [10]

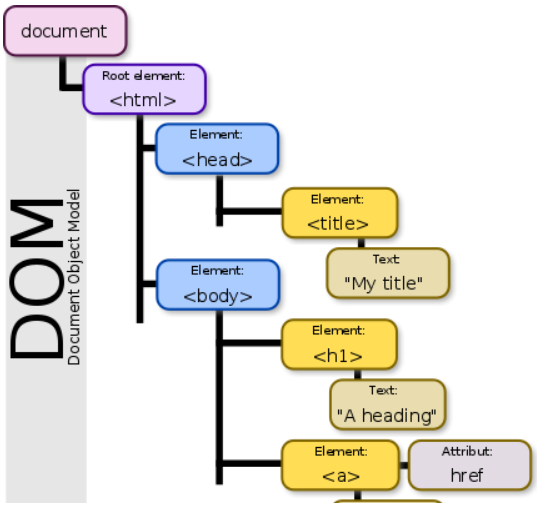
2.2.2 Basis

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 erg 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]

2.2.3 Features

2.2.3.1 Virtual DOM

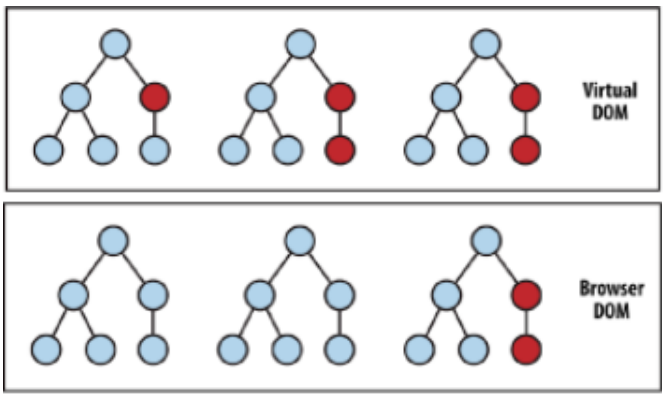
In een webapplicatie is één van de meest belastende operaties het veranderen van de DOM. Dit is een API dat een HTML-, XHTML- of XML-bestand behandelt als een boomstructuur waarin elke node een object voorstelt als deel van het document. [13] (Figuur 2.3)



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 courante 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



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

deze laatste stap wordt dan ook het browser DOM aangepast, zodat deze niet geheel opnieuw gerenderd hoeft te worden.

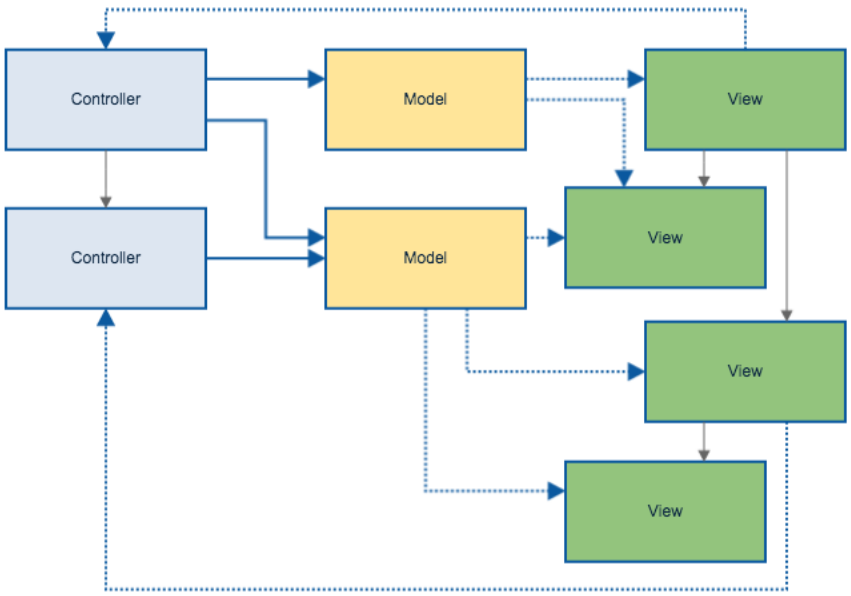
Het gebruik van Virtual DOM heeft dus zijn performance voordelen, maar het potentieel is echter veel groter dan bovengenoemde verbeteringen doen vermoeden. Wat als React een ander doel kon renderen dan de browser zijn bijhorende DOM? Met deze filosofie werd React Native gebouwd, waarbij in plaats van de browser DOM te renderen, React Native Objective-C APIs aangewend worden om iOS componenten te renderen. Analoog Java APIs worden aangewend 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]

2.2.3.2 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:

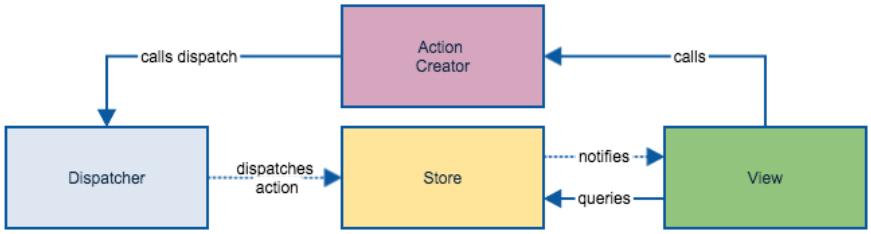
- De relaties tussen View, Models & Controllers worden te complex.
- De code is zeer moeilijk te debuggen.
- Oneindige loops worden te gemakkelijk geactiveerd.

Figuur 2-4: Voorbeeld MVC [14]

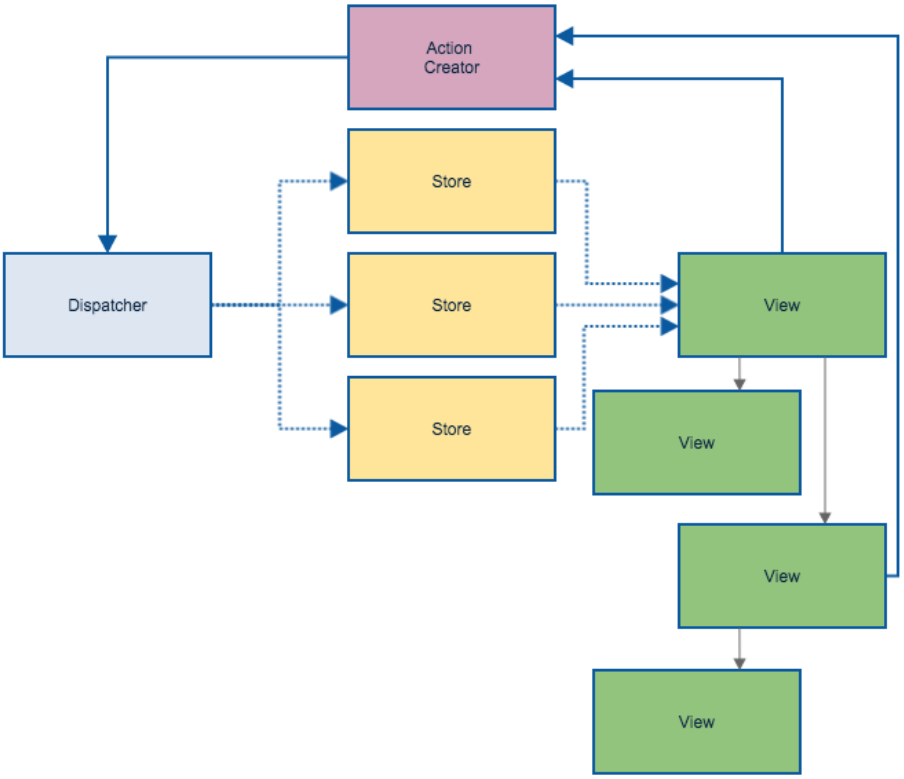


Figuur 2.5 Complexer voorbeeld MVC [14]

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 2.5 Voorbeeld Flux [14]



Figuur 2.4: Complexer 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]

2.2.4 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 een 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.

2.2.5 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

2.3 Xamarin

2.3.1 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]

2.3.2 Basis

Het Xamarin platform is een .NET-omgeving met iOS en Android C# verbonden libraries. Zoals te zien in Figuur 2.8 is deze 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]

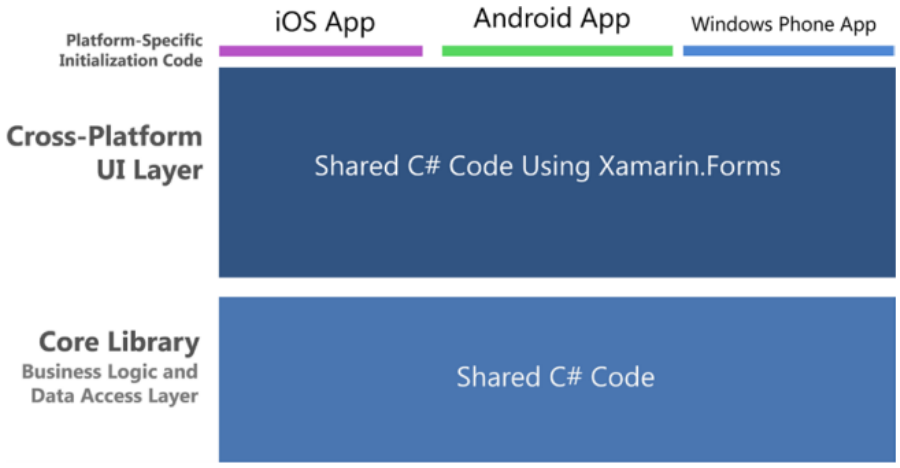
Figuur 2.6: Opbouw Xamarin [19]

2.3.3 Features

2.3.3.1 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 klassen zijn vooraf gemapt 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 toolkit 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 eindlayers zijn 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)

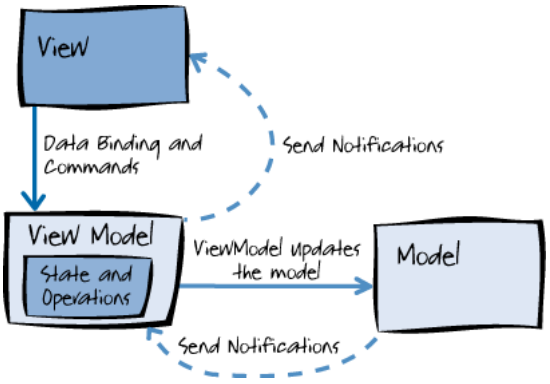


Figuur 2.7: Xamarin.Forms' oplossing architectuur [19]

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.

2.3.3.2 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 te presenteren door het View. [20]

Het Model bevat: een deel klassen die de business logica en data voorstellen. Het definieert ook de regels waarin staan 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 deze 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]

2.3.4 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 geïntegreerd. Het is eveneens evident om de build rechtstreeks op een mobiel device te testen.

2.3.5 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)

2.4 Apache Cordova/PhoneGap

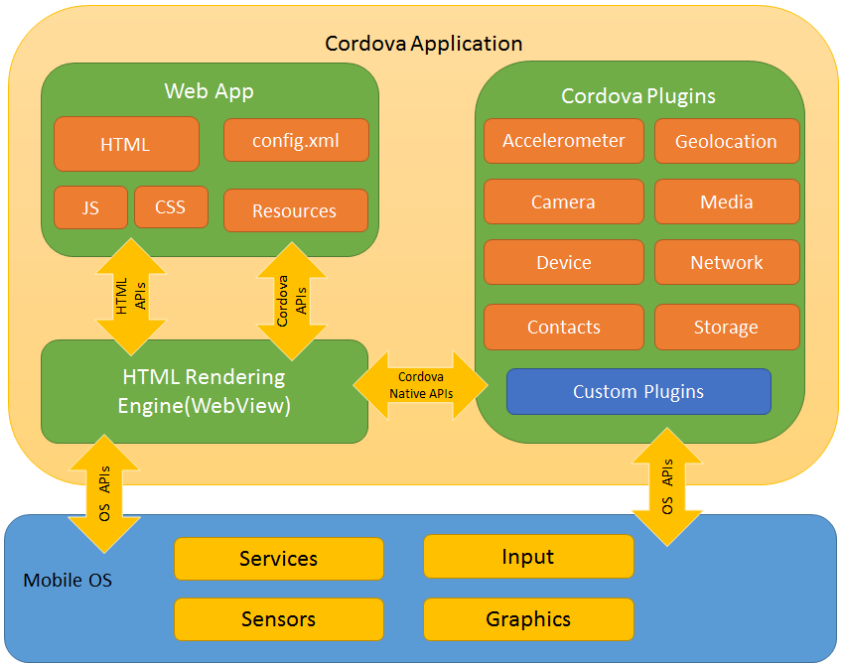
2.4.1 Oorsprong

Apache Cordova is een ontwikkelingsframework voor mobiele applicaties, ontwikkeld door het Canadese bedrijf Nitobi. Initieel werd dit framework benoemd als PhoneGap, vermits het onder deze naam in 2009 gecreëerd werd door Nitobi op een iPhoneDevCamp evenement in San Francisco. Adobe kocht in 2011 Nitobi over, waarna deze eerstgenoemde besliste dat de PhoneGap codebase gedoneerd zou worden aan de Apache Software Foundation. [24] Deze versie kreeg de naam Apache Cordova, dewelke bovendien open-source werd. Hierna focuste Adobe zich op het ontwikkelen een meer uitgebreide versie van PhoneGap, wat de naam Adobe PhoneGap kreeg. [25] Deze laatste omvat dus naast Cordova ook Adobe services en extensies die de capaciteiten verder verbeteren. Cordova kan echter nog steeds apart gebruikt worden, maar wordt vooral aangewend voor zijn mogelijkheden om componenten van mobiele devices te kunnen hanteren. [26] [27]

2.4.2 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 door het framework specifiek voor elk platform gebouwd worden. Hierop komt dan een WebView die toegang krijgt tot device-level APIs waardoor deze gebruik kan maken van sensoren, data, netwerk status, *etc*. Deze WebView voorziet de hele UI voor de applicatie. Voor sommige platformen kan deze ook een component vormen binnenin een grote, hybride applicatie die de WebView mixt met componenten van native applicaties. [28]

2.4.3 Architectuur



Figuur 2.9: Architectuur van een Cordova applicatie [28]

Wanneer de architectuur van een Cordova applicatie bekeken wordt, valt op dat de Web App het deel is wat de geschreven applicatie code bevat (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 vervolgens een WebView, ook wel een HTML Rendering Engine genoemd, gecreëerd. Deze WebView is dan geprojecteerd op het mobiel via een API van het OS zelf. De Cordova Plugins zijn eveneens van belang, vermits zij een interface voor Cordova, native componenten om met elkaar te communiceren en verbindingen naar standaard device APIs zoals de camera, contacten, *etc*. voorzien. [28]

2.4.4 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, waarna via de command line een project geïnitialiseerd kan worden. Om het project te bewerken kan een tekst-editor naar keuze gehanteerd worden.

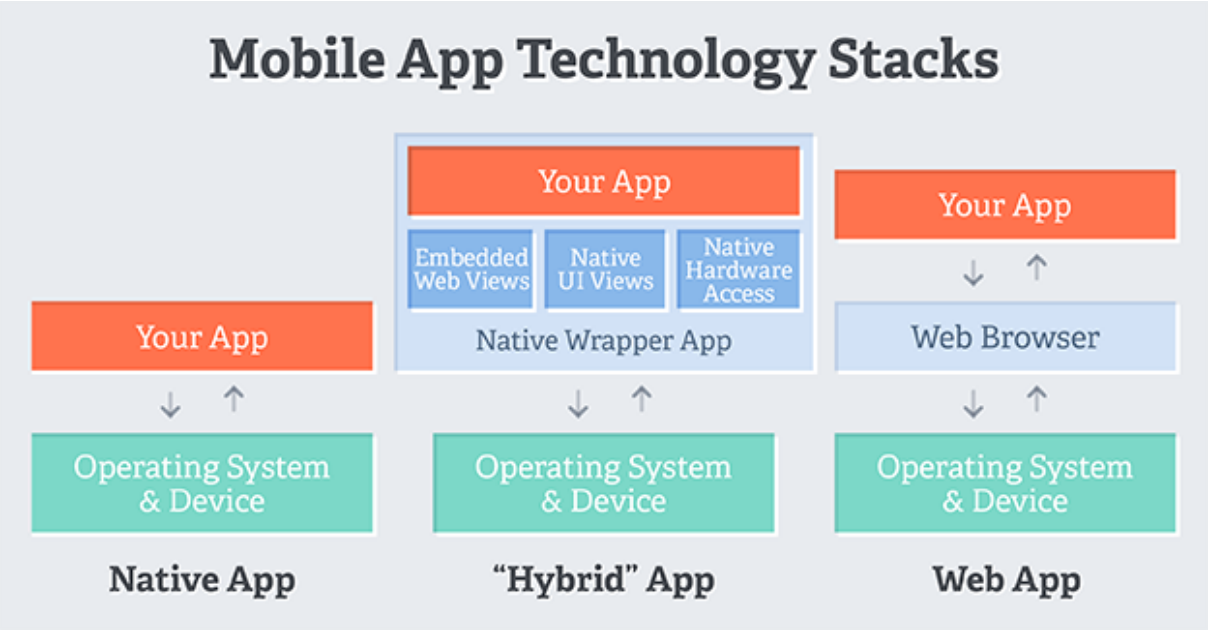
Om de applicaties te testen, kunnen via de command line emulators gedownload worden. Deze zijn eenvoudig te gebruiken door het project te builden op een Android/iOS emulator via commands in de command line, waarbij de optie om de app te builden op een reëel apparaat eveneens mogelijk is. Alles verloopt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

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

2.4.5 Main feature: WebView

Over de details van Apache Cordova is weinig gerapporteerd en beschreven. Wat echter wel duidelijk vastgesteld wordt, is de Webview-gebaseerde aanpak van dit framework om een 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 zowel een native als web app. Hierop is duidelijk waar te nemen hoe de hybride app een wrapper app gebruikt die bestaat uit een WebView, een native UI view en een native hardware toegang in de back-end.

Een WebView omvat niet meer dan een iframe of tab in een vermomde browser. Het Cordova framework zet de HTML5 code in een WebView en biedt vervolgens een *'foreign function interface' (FFI)* aan. Deze interface overbrugt het obstakel naar het native OS en zorgt bovendien dat de native mogelijkheden (bijvoorbeeld toegang tot de camera) doorgegeven worden aan het Cordova-framework. Zo kan het framework aan de hand van deze middelen communiceren en ze gebruiken. Het gebruik van de FFI impliceert echter een tragere verbinding dan een native gebouwde applicatie, wat nadelig is. [31]



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

2.4.6 Portfolio

Een aantal applicaties gebouwd met behulp van systemen met als fundering Cordova (Ionic, PhoneGap, *etc.*) [32] [33]

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

2.5 Kwaliteitscontrole van de applicaties

2.5.1 Theoretisch

Om de platformen van elkaar te onderscheiden wordt onderzocht voor welke eigenschappen de platformen van elkaar verschillen (zie Tabel 2-1). Hieruit kunnen vervolgens 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 vergelijking van de programmeertalen wordt niet veel informatie gehaald. Hierbij is het vooral belangrijk met welke talen de gebruiker het meest ervaren is en wordt de keuze eerder gemaakt op basis van comfort en reeds verworven kennis. Interessant om op te merken 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, waaruit geconcludeerd wordt dat de programmeertalen enkel vanuit een persoonlijke voorkeur een rol

zullen spelen, waarbij het enige platform zonder leercurve voor ervaren programmeurs Apache Cordova zal zijn.

Wanneer gekeken wordt hoe elk van de platformen zijn UI rendert, dus hoe het zijn componenten gebruikt om een UI op het scherm te krijgen, wordt vastgesteld dat zowel Xamarin als React Native echt native UI-controllers hanteren. Dit betekent dat bij deze platformen de applicatie niet enkel Android/iOS aanvoelt, maar effectief is. Omwille hiervan werd de voorkeur gegeven aan het uitdiepen en verder onderzoeken van deze twee platformen.

Apache Cordova zal zijn applicatie starten door een WebView op te zetten die daarop dan zijn UI toont, terwijl bij alle andere platformen effectief Android/iOS componenten aan te pas zullen 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 bijgevolg theoretisch minder interessant geacht.

Alles is open-source te vinden, wat een belangrijk voordeel is van al deze cross-platform frameworks. Bovendien biedt React Native al zijn features onder een courante licentie gratis aan, wat een van de redenen zal zijn waarom dit framework de hoogste positie zal bekleden op de finale theoretische lijst. Het gratis aanbieden van al deze features is immers zeer interessant zijn voor een bedrijf als Zappware, dat 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 grootste potentieel van de platformen, wanneer deze vanuit een theoretische ooghoek bekeken worden, bevindt zich in een combinatie van free-to-use, native UI-controllers en een veelgebruikte bekende taal. Eén van de belangrijkste onderdelen van deze criteria wordt gevormd door de *look and feel.* Aangezien **Xamarin** eveneens de native UI-controllers gebruikt wordt deze qua potentieel als 2de meest geschikte crossplatform framework beschouwd.

Tot slot resteren Qt en Apache Cordova. 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, waardoor deze laatste twee applicaties niet effectief gebouwd zullen worden. Dit komt tevens omdat Qt in het vorige jaar reeds onderzocht werd, waaruit geconcludeerd werd dat dit platform toch geen ideaal cross-platform was waarbij bovendien de *look and feel* van geïmplementeerde applicaties toch tekortkwam.

2.5.2 Tools

Om de praktische controles uit te voeren, werden een aantal tools gehanteerd, die helpen bij het vergelijken van de frameworks in termen van performantie. Ten eerste worden deze algemeen besproken. Ten tweede zal er naar deze hulpmiddelen verwezen worden wanneer ze in het proces zijn toegepast.

2.5.2.1 Testobject

Www.testobject.com is een online test-API die via een cloud verzekert dat de ontwikkelde Android of iOS applicaties, vlot werken op een bereik van verschillende devices. Deze API werd ontwikkeld door een Amerikaans bedrijf genaamd Sauce Labs, dat eerder al implementaties van cloud-hosted, web en mobiele applicaties tests automatiseerde. De

feature is enkel tegen betaling verkrijgbaar, 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, werden geüpload op de cloud, waarna deze bestanden 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 afgeleid kunnen worden door de timestamps van het begin en het einde van het proces met elkaar te vergelijken.

2.5.2.2 Activity monitor

Om het geheugengebruik van een Android applicatie te monitoren, kan simpelweg het real device aangewend worden. De Android software bezit namelijk vanuit het eigen OS de potentie om het RAM-geheugen van apps afzonderlijk te bekijken.

Indien het geheugengebruik voor een iOS applicatie gemonitord moet worden, werd een tool gebruikt waarbij de applicatie vanuit een MAC geanalyseerd kan worden. Xcode heeft instrumenten ter beschikking om allerlei parameters op te meten, vertrekkend van CPU-gebruik, batterijgebruik tot geheugengebruik. Dit laatste gebeurt met het Activity monitor instrument, waarbij de applicatie normaal wordt gebruikt terwijl de app “opgenomen” wordt. Hieruit kunnen vervolgens verschillende parameters afgelezen worden die onder andere het *leaked*, het *abandoned* geheugen en het algemene geheugengebruik omvatten, waarbij deze laatste nog verder kan onderverdeeld worden in het cache- en RAM-geheugengebruik. Enkel deze laatste twee zullen effectief besproken worden, vermits de applicaties niet de complexiteit omvatten die benodigd is om een hoeveelheid geheugen te lekken, i.e. het geheugen wordt statisch gealloceerd en kan dus niet in een *loop* geheugen blijven alloceren.

2.5.3 Praktisch

Ten eerst wordt besproken hoe de controles in het algemeen worden aangepakt, waarna de performantie van elk van de applicaties behandeld wordt.

2.5.3.1 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.

Uit deze gegevens 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 voor het construeren van prototypes 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 verscheidene conclusies worden getrokken.

2.5.3.2 Geheugengebruik

Android

Het geheugengebruik wordt getest op een eigen echt apparaat, geen simulatie dus. Het gehanteerde apparaat is 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 werd het cache-geheugen, het RAM-geheugen en het interne geheugen met elkaar vergeleken. Hoe minder geheugen er gebruikt werd, hoe beter. Aangezien iOS geen optie heeft om het RAM-geheugen te determineren, moet dit opgemeten worden via *Monitor Memory Usage*, zoals beschreven werd in sectie 2.5.2.2.

2.5.3.3 Snelheid

Android

Om de snelheid van deze applicaties te testen is *testobject.com* benut (*vide supra*, sectie 2.5.2.1.). Als resultaat worden er logs vanuit het device verkregen. Er is gezocht naar volgende lijn output op deze 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 beschikbaar gesteld wordt op de gratis test-API, is de Motorola Moto E (2nd generation) die bovendien dicht aanleunt 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 wordt een console command van Xcode ingeschakeld. Dit is een onderdeel dat Testobject niet in zijn resulterende logs weergeeft., waardoor dit manueel diende uitgevoerd te worden in Xcode. Door een omgevingsvariabele mee te geven aan dit console command, 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, waarbij eveneens de duur van bepaalde delen van de initialisatie weergegeven wordt. De installatiesnelheid bezorgt Testobject achteraf in zijn kwaliteitsrapport.

Het device dat hiervoor gehanteerd wordt is de Apple iPhone SE. Dit is eveneens hetzelfde dat ons ter beschikking werd 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, de opstarttijd en/of de tijd die het nodig heeft om de geïnstalleerde app op te starten zonder dat er iets in het cachegeheugen geplaatst is.

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

2.5.3.4 User experience

Voor het user experience-onderdeel is het niet eenvoudig om harde cijfers te vinden over bepaalde parameters (vlotheid, uitstraling, functionaliteit, *etc*.), vermits deze eerder een subjectief karakter bezitten. Elk van deze eigenschappen zal immers anders ervaren worden door verschillende personen, waardoor besloten werd een beeld te vormen van de gebruikerservaring aan de hand van een enquête. Tien personen zullen de applicaties beoordelen op bepaalde factoren, door het stellen van gerichte vragen. Deelnemers duiden als antwoord een score op een schaal van van één tot vijf aan. De steekproefomvang is tien, die bestaat uit 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, waardoor in totaal zes applicaties getest zullen worden.

De tabel met 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.			

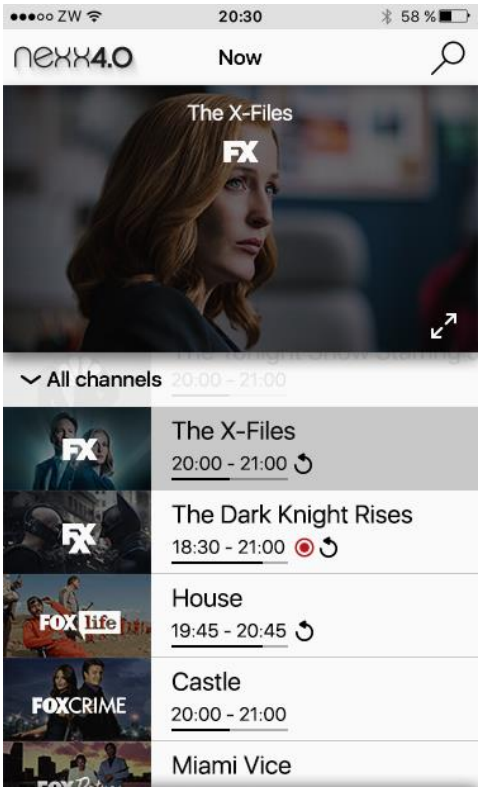
3 APPLICATIE

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:

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.



Figuur 3.1: Einddoel applicatie

Om een voorbeeldapplicatie 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.

4 VERGELIJKING

4.1 Development effort

4.1.1 React Native

4.1.1.1 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, kan een React Native project worden opgezet. Dit project kan via een tekst-editor worden aangepast en uitgebreid. 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 gebruikt voor een voorbeeld hoe de applicatie er uit zal zien.

Documentatie over 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 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 opgezet worden en dit zorgt er voor dat 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 aangemaakt 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 aangesproken worden 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 hieromtrent geregeld worden en de benodigde

pakketten voor onze applicatie installeren. De geschreven code voor de Android app is hierna direct bruikbaar in Xcode.

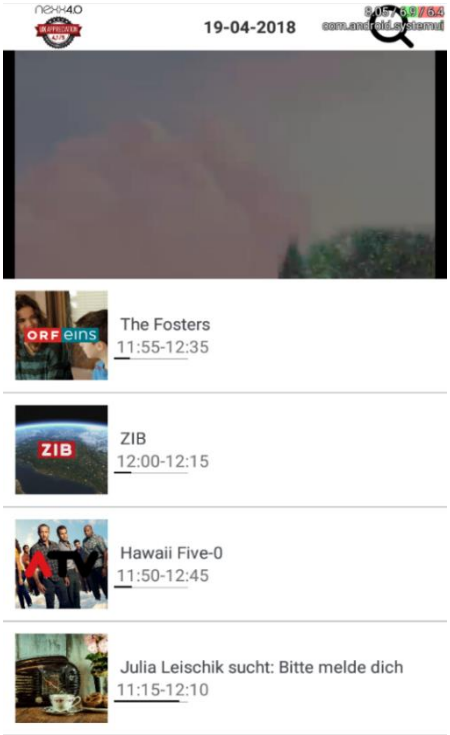
Het deployment van deze iOS applicatie loopt eveneens analoog. In XCode wordt de React Packager ook gebruikt om de applicatie te bouwen naar een .ipa vorm. Deze moet gesigneerd worden door een geverifieerd Apple ID. Dit was echter een uitdaging, enkele aanpassingen – die allemaal online te vinden waren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

4.1.1.2 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

4.1.2 Xamarin

4.1.2.1 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 Visual Studio verwerkt via emulators waarvan de installatie omslachtig is en veel geheugen in gebruik neemt, daarom werd er een real device gehanteerd.

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 geëvalueerd 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 *on hold* gezet. Desondanks was de structuur bruikbaar en moesten enkel details afgewerkt worden. Maar deze konden potentieel zeer tijdrovend zijn.

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

Om de app op iOS uit te rollen was de connectie maken met een MAC eenvoudig en logisch. Visual Studio kon zelf connecteren met de MAC die is gebruikt en zodanig Xcode zelf gebruiken om de applicatie te renderen en uit te rollen op een iPhone. De .ipa-file instellen was eveneens een analoge zaak als de .apk-file. De documentatie hierrond was duidelijk en gestructureerd.

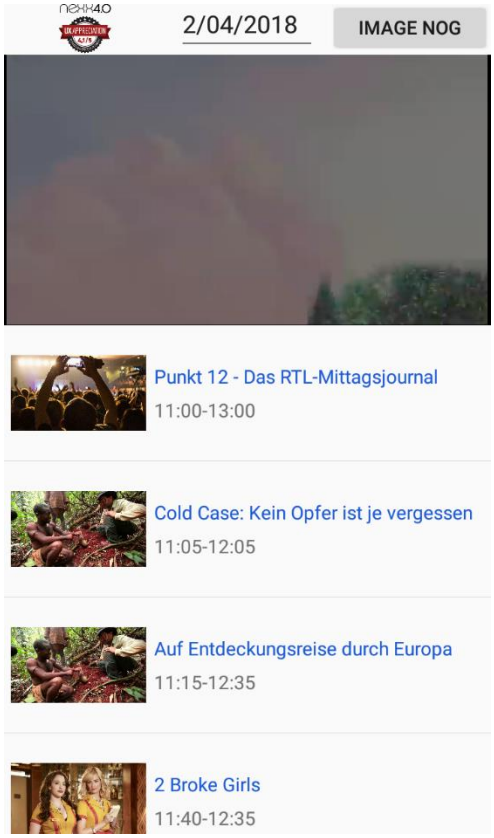
4.1.2.2 Details

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 geïmplementeerd).
- 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

4.1.3 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 bij 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 naar Android was bij React Native en Xamarin, die beide goed gedocumenteerd waren, een compleet identiek proces. 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 dan te 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 beide voor- en nadelen hebben. 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. Nog een voordeel is de meer uitgebreide debug-mogelijkheden van Visual Studio. Dit tegenover de ready-to-use eenvoudigheid die React Native aanbiedt door geen IDE te hanteren, 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 bedrijven geen financiële middelen willen inzetten om een framework te bezitten dat niet altijd zal gebruikt worden.

Het gebruik zelf is uiteindelijk aan de ontwikkelaar; hierdoor is een zekere persoonlijke voorkeur van toepassing. Daarom is een eindconclusie nooit aan de orde.

4.2 Geheugen

Het minimaliseren van alle soorten geheugens dat een applicatie inneemt, is belangrijk zowel om de 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 geschapen worden over het feit dat de apps niet dezelfde 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.

4.2.1 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 4-2 Vergelijking geheugengebruik Android applicaties

	Native (Android)	React Native	Xamarin
Interne opslagruimte[MB]	49,45	17,86	26,84
RAM-geheugen[MB]	343	252	169
Cachegeheugen[MB]	0,492	3,2	0,02

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 4-3 Vergelijking geheugengebruik iOS applicaties

	Native (iOS)	React Native	Xamarin
Interne opslagruimte[MB]	94,5	24,6	24,6
RAM-geheugen[MB]	288,1	126,3	97,8
Cachegeheugen[kB]	115	4	3000

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 hebben de .ipa files exact dezelfde grootte terwijl voor Android de React Native applicatie toch significant kleiner was.

4.2.2 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.

4.3 Snelheid

4.3.1 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 4-4 Vergelijking snelheden Android applicaties

	Native (Android)	React Native	Xamarin
Installatietijd [s]	26,59	9	10,75
Opstarttijd [s]	1,193	0,348	6,601

Als de installatietijden van de applicaties bekeken worden is de correlatie duidelijk. Dit is makkelijk te verklaren door het feit dat de grootte van de .apk file exact correleert met 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 4-5 Vergelijking snelheden iOS applicaties

	Native (iOS)	React Native	Xamarin
Installatietijd [s]	15,13	7,46	7,46
Opstarttijd [s]	0,3916	0,4639	0,3404

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.

4.3.2 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. Het bewijs hiervan zijn de iOS-tijden, die slechts een klein verschil aangaven. Aangaande kan verklaard worden door het verschil in aantal features. De voelbaarheid en significantie van deze specifieke opstarttijden zullen bij het onderdeel user experience nog geëvalueerd worden.

4.4 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.

4.4.1 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

4.4.2 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 behaalt 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 datum-component werken allemaal naar behoren. Enkel de datum-component 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 datum-component. 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.

4.4.3 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 deze app een goede maatstaaf zal zijn om de andere applicaties tegen af te wegen.

De Xamarin applicatie behaalt in het algemeen hogere scores als het over de iOS versie gaat. 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 betreffende de hybride apps kan geoptimaliseerd worden tot in het kleinste detail. Hier hang je vast aan bepaalde componenten die variabelen beperkter bevatten 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 het bouwen van native applicaties.

5 BESLUIT

Een cross-platform framework is in theorie een mooi concept , vermits deze de huidige, eerder omslachtige procedure van het schrijven van twee volledige projecten voor het creëren van een applicatie kunnen omzeilen. In realiteit bleken deze frameworks echter niet in staat alle gewenste oplossingen te bieden, die benodigd zijn om de native ontwikkeling van applicaties te kunnen vervangen. dragen immers bij tot verlaging van de algemene prestatie, wat samen met de beperkte mogelijkheden er toe leidt dat de hybride alternatieven ondermaats zijn t.o.v. de klassieke versies. Dit betekent echter niet dat er geen enkel potentieel is om die alternatieven nuttig in te zetten. Zo kunnen deze zeker ingeschakeld worden als prototype-framework om demo's op te bouwen voor bepaalde toepassingen.

De besproken ontwikkelingsplatformen zijn aan elkaar gewaagd wanneer de werking van de twee gecreëerde apps geanalyseerd werd. De algemene voorkeur om prototype-applicaties op te lanceren gaat vooralsnog uit naar React Native, vermits dit een eenvoudiger en beter gedocumenteerd open-source framework is in vergelijking met zijn concurrent Xamarin. Deze laatste is een alternatief waarbij de focus eerder gelegd wordt op producten met een lange houdbaarheid, maar doch vormt deze een dure optie om als parttime framework te gebruiken. Ondanks deze bevindingen is de keuze uit de voorgestelde alternatieven veeleer een persoonlijke keuze. Deze is afhankelijk van de specifieke eigenschappen van het framework waaraan de ontwikkelaar het meeste belang hecht.

6 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 Wetenschappelijke paper

Bijlage B Javascript-code van de React Native applicatie

Bijlage C XAML-code van de Xamarin applicatie

Bijlage D C#-code van de Xamarin applicatie

Bijlage E Resultaten enquête

Comparison of cross-platform and platform-specific frameworks for mobile development

Lennert Van Looveren, Peter Karsmakers[1], Bram Schrijvers[2],
[1]KU Leuven, Faculty of Engineering Technology, Technology Campus Geel
[2] Zappware, Ilgatlaan 19, 3500 Hasselt, Belgium

Abstract Mobile phones are everywhere. They are used as cellphones, communication channels and as a personal computer. Therefore, they function more and more as a source of entertainment and information. This paper contains the process of the development of a tv guide application for mobile. Since there are many options on which framework an app can be developed, a research for which framework is the most compatible for such development could provide useful information. This year we've created this mobile application for two different frameworks, while Zappware provided the native iOS and Android version. Those applications were written in React Native and Xamarin. We document specific differences in various areas and weigh them against the native ones. This document also provides information about two other cross-platform frameworks: Apache Cordova and Qt. In the search of a possible replacement for native designed applications, both frameworks lacked potential in terms of user experience. As a prototype-application React Native was the preferred choice.

***Index Terms*— Cross-platform frameworks, Android, iOS, Xamarin, React Native, Apache Cordova, Qt, Mobile development**

I. Introduction

Mobile applications can be created in various ways. The most natural way to develop them is by writing the iOS app through XCode in Swift, and the Android app trough Android Studio in e.g. Java or Kotlin. This means there is a need for two separate fully developed applications. This research is dedicated to finding a good alternative for that scenario. The so-called cross-platform frameworks that are being researched: Qt, Apache Cordova, React Native and Xamarin. Their pros and cons get valued against each other, in both a theoretical and a practical way. The goal is to find the most practical framework, build a comparable application and compare those with the native applications Zappware have built. The parameters by which those will get evaluated are: speed, memory, development effort and user experience.

II. Cross-platform frameworks

Qt is a framework developed in C++ that among other things Android, iOS and Windows Phone supports. Applications are written in a combination of C++ and QML, which is a declarative scripting-language that uses JavaScript to build its logic with. Qt is a cross-platform framework, this means it can render one code on Android, iOS and Windows Phone. This cross-platform property exists since the C-preprocessor chunks of code enabled or disabled depending on the platform the application is running on. [1]

React Native is completely based on React, a JavaScript library build by Facebook to make big scalable web applications. React Native is a JavaScript framework build to render mobile application for iOS, Android and Universal Windows Programs (UWP). It is written in combination of JavaScript and JSX, which is a language comparable with XML. React Native uses the platform its own standard rendering API to create an application which will contain the same elements as a native designed app. Through this procedure the framework aims to possess the same look and feel as those native designed applications. [2]

Xamarin is a development framework which uses a .NET environment with iOS and Android C# libraries. While single-platform development is possible with Xamarin through those C# connections with native Android and iOS APIs, the cross-platform alternative is more interesting. Xamarin.Forms is a toolkit consisting of cross-platform UI-classes which can be considered as a bridge between the .NET environment and the single-platform libraries. This toolkit locates which of those libraries must be used depending on which platform its application is rendered on. The applications are coded in C# and XAML. [3]

Apache Cordova is an alternative which uses standard web technologies like HTML5, CSS3 and JavaScript for its cross-platform development. Applications run inside native

application wrappers specifically build for each platform. A WebView renders onto that wrapper, which gets access to the device-level APIs like sensors, data, ... This WebView is basically a borderless browser where a UI gets rendered upon. This UI is made off native application components which the WebView can access through the OS of the device. [4]

III. Theoretical comparison

To distinguish the frameworks from each other a comparison is made of various property's. These include: The programming language, the way the framework renders the UI and the commercial use and its restrictions. After that a conclusion is made about the theoretical usefulness of the specific platforms. This conclusion gets used to determine which practical applications effectively get build. React Native uses JavaScript and JSX for its code. Xamarin C# and XAML. Qt applications get developed in a combination of C++ and QML while Apache Cordova uses standard web application languages: HTML5, CSS3 and JavaScript. From this summary the conclusion can be made that only Apache Cordova exclusively uses languages which are already known and widely utilized. The rest of the frameworks has one language (JSX, XAML and QML) which is developed for the use in that specific framework only. Although it should be said those languages are all derived from a well-known alternative (respectively HTML, XML and CSS).

Looking at the way those frameworks render their UI, React Native and Xamarin both use native UI-controllers. Qt uses a mix of native components and the OpenGL API, this is a library that renders 2D and 3D vectors. Apache Cordova is the only framework which operates using a WebView.

A company is also interested in the financial side of such development platforms. All frameworks have an open-source version for developers to use. React Native and Xamarin have one under the MIT-license. [5] [6]

But while Xamarin has a paid Visual Studio IDE version, React Native does not. This means all features are enabled for React Native. Qt has an open-source LGPL v3-license or a GPLV2/GPLV3-licence, with an option for a commercial license. [7] Apache Cordova us an Apache licence version 2.0. Some platforms build upon Cordova like Adobe PhoneGap have paid versions though.

All things considered is React Native the framework with the most potential. Its combination of free-to-use, native UI-controllers and with JavaScript a very known widely used language potentially makes it a very viable framework. Since the look and feel of an application is one of the most important features, the Xamarin app is considered the second-best option since it also utilizes the native UI-controllers. This feature trumps the other options because it doesn't use a form of bridge between the device and application, this would automatically mean some performance-issues e.g. delays, connection problems, ...

Because of time-related issues not all frameworks could be tested, and thus only two implementations were made. Due to this theoretical comparison the options who were chosen are React Native and Xamarin.

IV. Application

The application that shall be built through the chosen frameworks is a basic tv-guide application as seen in Figure 1. This is the app that Zappware has built for Android and iOS use. Although the layout will be different on the two frameworks, since the frameworks will sometimes use different components, the overall look stays the same. The goal will be to mimic this example as good as possible and after that conclude the limits and possibilities of the used cross-platform frameworks. After the applications are finished developing, those will get tested on various elements including: development effort, speed, memory and

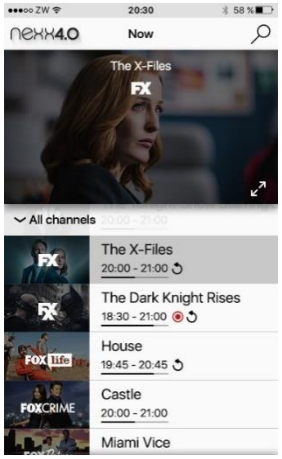


Figure 1 Target

user experience.

V. Tools

To test the created applications, some tools were used to determine some variables. These tools will be referenced upon whenever they are used later in the process. Testobject.com is an online test-API where applications can run through a cloud to determine their use on variant devices. The feature has a paywall for extended use, but gives a trial to new users. To use Testobject the applications their .ipa or .apk file(respectively for iOS and Android) get uploaded unto the cloud. These files undergo an automatic test procedure in an environment which depends on the devices the user chooses. After the calculations are done, Testobject provides among other things crash reports, quality reports and the option to manually test your application online. These reports are provided in the shape of logs, these logs get used to determine the installation time and the startup time of the applications.

To monitor the memory usage of the Android application, the OS provides information in the device itself. iOS on the

other hand needs a tool to monitor such things. XCode offers an instrument which can analyze the apps on memory usage. The instrument is called Activity monitor and provides information about leaked memory, abandoned memory and general memory usage including cache- and RAM-memory.

VI. PRACTICAL COMPARISON

As stated before, the application will undergo testing to determine various important elements in the making-of and results of the process. This paper reviews the most important conclusions which have been made about those elements for the two used frameworks.
The development effort which the framework needs to release an application from start to finish is the first important element. React Native doesn't use an IDE like Xamarin does with Visual Studio. This means a lot of tools meant for debugging are absent, which Visual Studio does provide though. In React Native there are tools for debugging with online sources, but are not as extensive as those inside Visual Studio. The coding languages in which the applications are written both feel familiar. Although the JavaScript that React Native uses does feel less complicated and more straight-forward than the combination of XAML and C# that Xamarin uses. This feeling gets amplified because the document around Xamarin aren't as obvious as those around React Native. While for React Native a lot of answers can be found online, Xamarin doesn't have the same active community. The processes for the installation and use for Xamarin itself was well-documented though.
The structures in React Native and Xamarin are different but can both be used for big complex applications. Also, the deployment of the finished application on iOS and Android for both frameworks was a clear process without problems.
If we conclude the statements just made, there's a case to be made that React Native thrives in an environment where there's a quick, simple prototype needed for an application. Because of how the setup works without an IDE, the ready-to-use simplicity, a programming language that requires less experience and less insight and an active community that solves problems with extensively documented solutions. Xamarin is more the framework that needs a longer learning curve and is harder to use. That way it would probably serve better as a stand-alone platform where a lot of experience and insight is needed to make it work. As a reference to the theoretical comparison, all features are available for React Native. This in contrast with Xamarin which has a paid version of Visual Studio, React Native is more suitable as a prototype platform than Xamarin is.
Analyzing the memory usage of the applications is next. The result of the test can be found inside Table 1 and Table 2 for iOS and Android respectively. The most notable thing was that the React Native application took significantly more RAM-memory than its Xamarin counterpart. These differences could be allocated on the fact that React Native version had more features since the Xamarin application wasn't finished. This is a reason why a conclusion couldn't be made.

Table 1 Android memory results

	Native (Android)	React Native	Xamarin
Internal memory $[MB]$	49,45	17,86	26,84
RAM-memory $[MB]$	343	252	169
Cache-memory $[kB]$	492	3200	20

Table 2 iOS memory results

	Native (iOS)	React Native	Xamarin
Internal memory $[MB]$	94,5	24,6	24,6
RAM-memory $[MB]$	288,1	126,3	97,8
Cache-memory $[kB]$	115	4	3000

In terms of speed (Table 3 and 4), the installation time was immediately declared a useless variable, since this is proportional with the file-size of the application. The startup time had some interesting insight though. On Android the Xamarin app performed significantly worse than the React Native alternative. But on iOS the React Native app performed worse than its counterpart. An explanation could be the difference in hardware inside the used devices and the structure of the application. If the reason was the framework, those problems would be consistent on Android and iOS. The significant startup time of the Xamarin Android app gets explained by the way it setups its list inside the application. It is a static reference, which Android had a hard time handling.
Because of these nuances there has been judged that there is no significant difference in the startup time between the cross-platform framework itself, this conclusion gets derived off the fact that the iOS-timings are only a little different from each other, which in itself could be explained due to the difference in features.

Table 3 Android speed results

	Native (Android)	React Native	Xamarin
Installation time [s]	26,59	9	10,75
Startup time [s]	1,193	0,348	6,601

Table 4 iOS speed results

	Native (iOS)	React Native	Xamarin
Installation time [s]	15,13	7,46	7,46
Startup time [s]	0,3916	0,4639	0,3404

The last and most important feature of a mobile application is its user experience. Previous elements like memory and speed all add to the user experience, but can't be finally judged by numbers alone. Since this is a subjective matter, this element will be tested by a survey. This survey will consist of parameters which the participant must give a score within the range of one and five. The participant will rate these parameters for each application in iOS and Android, including the native apps made by Zappware's development team. The survey has a sample size of ten participants with different backgrounds and various expertise levels. The results of the survey are shown in Table 5 and Table 6 with the first result being the mean and the second being the median of each score.

Looking at the results, it is obvious the native applications gets (almost) perfect scores the whole survey. This makes sense since this application is developed by a whole team of developers who realized this after a lot of hours and effort. The only thing that is noticeable: the startup time doesn't score as good in Android. This can be explained, like previously stated, by the difference in hardware. But the startup time of this application can't be compared with the other applications considering the native app is more complex and comprehensive.

The questions about the components, like the video player and date picker, were answered in positive fashion. Although the date picker in the iOS applications of the cross-platform frameworks get worse results. This is because in the iOS application the date picker gets blocked by the iOS standard clock. This is a small inconvenience which results in the fact that both React Native and Xamarin applications both need some form of platform-specific code to solve such issues.

The results in startup time are consistent with the startup times tested in Table 3 and Table 4. The explanation for these results have already been discussed.

Scrolling through a list on an application should run as smooth as possible. The scores on this part are based on the number of jitters happen. Those are creaks that happen when scrolling through such a list. There is a difference between the first time a list gets scrolled and the second time. This phenomenon occurs when the data is loaded in the cache-memory of the device the first time. The second time the list gets scrolled through the data will be obtained from that cache-memory, this results in a smoother experience. The results confirm that this phenomenon happens in the applications, with better results the second time opposite to the first time on Android. The results show that the React Native version runs smoother on Android and that they both score the highest on iOS.

The general look and feel is also judged by the participants. This results in a conclusion that the Xamarin and React Native applications feel quite similar and don't look different. Making a judgment about this element is hard, since they are so similar. The React Native application gets an average score 0.2 higher than its Xamarin counterpart on Android. The reverse is true for iOS with a difference of 0.4. This concludes that a React Native app has a slight preference in Android, while Xamarin is the choice in the OS of Apple.

All this information accumulates to the fact that determining which application is the best option is hard. It is obvious the native application is the best option on all accounts, which makes sense. On the native platforms every little detail can be adapted and everything is possible. On a hybrid framework the application gets limited since not all components are available and not every variable can be changed. But the options to build a good working application in both frameworks exist. To use such a framework to build a prototype for an application sure is possible. For this purpose, React Native seems the better choice of the two discussed frameworks. As stated earlier, React Native has the financial and simplicity advantage over Xamarin. For these reasons the conclusion is made that the React Native is the preferred framework to create hybrid applications. Although it's not good enough to replace native designed applications if user experience is the most important element of the app.

Table 5 Android results survey user experience

	React Native	Native Android	Xamarin
Startup Speed	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
Date	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

Table 6 iOS results survey user experience

	React Native	Native iOS	Xamarin
Startup Speed	2,8 / 3	4,7 / 5	5 / 5
Scroll 1	5 / 5	5 / 5	4,7 / 5
Video	5 / 5	5 / 5	5 / 5
Date	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

VII. Conclusion

A cross-platform framework is an alluring idea in theory. In practice it will not satisfy all requirements to fully replace native development. Too many factors ensure that the general performance and possibilities of the hybrid alternatives perform inadequate opposite to the native option. This doesn't mean there is no place for cross-platform development in the developing world. It could function perfectly as a prototype-framework to build demos for various purposes. The recommendation of this paper consists of using React Native for this exact purpose.

Although this decision is still a personal one, depending on a lot of various factors that have a different importance developer to developer.

VIII. References

IX.

1. [1] 2. B. Starynkevitch, „How is QT cross-platform,” [Online]. Available: [Online]. Available: https://www.quora.com/How-is-QT-cross-platform. [Opened 2018 March].
3. [2] 4. B. Eisenman, Learning React Native, O'Reilly Media, Incc., 2016.
5. [3] 6. D. Hermes, Xamarin Mobile Application Development - Cross-platform C# and Xamarin.Forms Fundamentals, Apress, 2015.
7. [4] 8. Apache Cordova, „Overview Apache Cordova,” [Online]. Available: https://cordova.apache.org/docs/en/latest/guide/overview/. [Opened April 2018].
9. [5] 10. „License React Native,” [Online]. Available: https://github.com/facebook/react-native/blob/master/LICENSE. [Opened April 2018].
11. [6] 12. [Online]. Available: https://blog.xamarin.com/xamarin-for-all/. [Opened April 2018].
13. [7] 14. „Open source licenseQt,” [Online]. Available: http://doc.qt.io/qt-5/opensourcelicense.html. [Opened April 2018].
15. [8] 16. B. Starynkevitch, „How is QT cross-platform,” [Online]. Available: https://www.quora.com/How-is-QT-cross-platform. [Opened March 2018].

```
"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,
  },

});
```

C

```
<?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>
```

D

```
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();
        }


        }
        }
```

E

	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