KU LEUVEN

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN

CAMPUS GEEL

Development in crossplatforms vergeleken met platform-specifieke APIs

Evaluatie van Qt, Xamarin, React Native en Apache Cordova

Lennert VAN LOOVEREN

Promotor: Prof. P. Karsmakers

Co-promotoren: Koen Swings, 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

Ik heb deze masterproef kunnen verwezenlijken dankzij mijn ouders, via hun morele en financiële steun. Alsook door mijn promotoren die mij gedurende de hele weg begeleidt hebben.

Samenvatting

De (korte) samenvatting, toegankelijk voor een breed publiek, wordt in het Nederlands geschreven en bevat **maximum 3500 tekens**. Deze samenvatting moet ook verplicht opgeladen worden in KU Loket.

Abstract

Het extended abstract of de wetenschappelijke samenvatting wordt in het Engels geschreven en bevat **500 tot 1.500 woorden**. Dit abstract moet **niet** in KU Loket opgeladen worden (vanwege de beperkte beschikbare ruimte daar).

Keywords: Voeg een vijftal keywords in.

INHOUD

Symbolenlijst

Lijst met afkortingen

API	Application programming interface
UI	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

1 INLEIDING

Zappware

Deze masterproef werd uitgevoerd bij Zappware in Hasselt. Een globaal bedrijf dat zich zowel bezig houdt met het ontwerpen van video UI design als client-software development. Deze masterproef kadert binnen de client-software kant. Hierbij werden er oplossingen gezocht om het proces van individuele APIs niet te moeten gebruiken bij het programmeren van applicaties.

Onderzoeksvraag

Het doel is om een crossplatform te vinden dat aan de eisen van performantie, useability en development effort voldoet. Deze door crossplatforms gecreëerde applicaties vergelijken we dan met elkaar en met applicaties die wel native zijn gebouwd. Uiteindelijk bepalen we dan welke crossplatforms geschikt bevonden zijn voor gebruik of eventueel voor gebruik als een prototype-platform.

2 CROSSPLATFORMS

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 moest runnen. Hiervoor bedachten ze een object-georiënteerd display systeem nodig te hebben. Hieruit resulteerde een basis voor de object-georiënteerde cross-platform GUI framework dat ze later zouden bouwen en de naam Qt, Q-toolkit, zou verkrijgen.

Na 5 jaar schrijven aan de nodige C++ klasses werd in 1995 Qt 0.90 uitgebracht, bruikbaar voor zowel Windows als Unix development en gebruikte voor beide platforms dezelfde API. In 2008 werd TrollTech overgenomen door Nokia en streefden zij om Qt development het voornamelijkste development platform te maken voor hun apparaten.

Vanaf 2014 werd Qt beheerd door “The QT Company”. Een dochterbedrijf van Digia, die de rechten van Qt overgekocht heeft van Nokia in 2012. Digia verzorgde de revolutionaire Qt 5.0 waarbij met behulp van JavaScript en QML de performantie en de gemakkelijkheid van UI te ontwerpen zwaar werd verbeterd. Alsook werd Qt vanaf toen open-source governance, waardoor ook ontwikkelaars buiten Digia verbeteringen konden voorstellen.

In 2016 werd Digia en “The Qt Company” gesplitst in 2 onafhankelijke bedrijven.

2.1.2 Basis

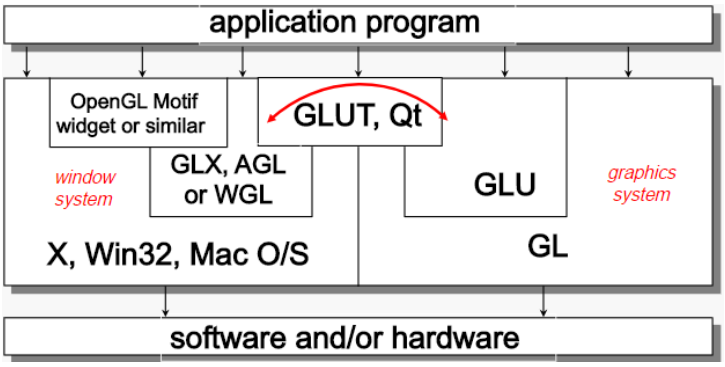
Qt is een cross-platform applicatie framework dat in C++ geschreven is en Andriod, iOS en WindowsPhone, … ondersteunt. Het doel van Qt was om zonder codewijzigingen op elk platform zonder performance-verlies te draaien en native te lijken. Voorbeelden van programma’s die in Qt Creator, het framework voor Qt, geschreven zijn: Skype, Google Earth, VirtualBox,...

Qt supportert elke standaard C++ compiler, zoals GCC, ICC, MSVC en Clang.
Wij schreven onze applicatie in een combinatie van C++ en QML, een declaratieve scriptie-taal die Javascript gebruikt voor de logica.

Qt dankt zijn cross-platform eigenschap aan het feit dat bij de C preprocessor (cpp) condities sommige code chuncks worden enabled/disabled afhankelijk van het platform. Het build proces gaat configuratie scripts runnen om preprocessor flags op te merken, afzetten en aan te passen.

2.1.3 Software architectuur

Het grafische systeem beheert de display hardware. Vooral OpenGL-GL, een library die 2D en 3D vectoren rendert, wordt als API gebruikt voor het grafische systeem. Dit terwijl het Windowing systeem de windows genereren, de events, window aanpassingen, … controleert. Het Windowing systeem heeft een eigen API voor programmeren, dat meestal in het OS is geïntegreerd zoals Microsoft Windows en MAC OS.

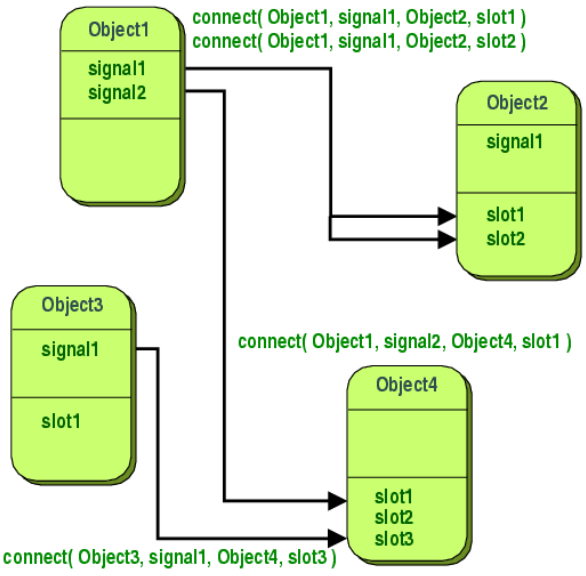


GLUT is een interface dat de communicatie tussen het window system en het graphic system verzorgt, deze gebruikt GLU en GL voor graphics en controleert operating en windowing systemen door voornamelijk GLX, AGL, … Dit zijn OpenGL extensies voor specifieke windowing systemen.

Qt zal op hetzelfde niveau als GLUT zowel het windowing als het graphics systeem kunnen raadplegen. Hierdoor kan het dus zowel Windowing functions als graphic functions oproepen.

2.1.4 Signals and slots

Maar Qt behandelt events, zoals quit(), onkeydown(),… , niet rechtstreeks. Er wordt een alternatief gebruikt op zogenaamde callback functions. Namelijk signals and slots.
Hierbij wordt een signaal verzonden wanneer een bepaald evenement gebeurt of een verandering van status plaatsvindt. Een slot is een functie die opgeroepen wordt als er een bepaald signaal verzonden wordt. Deze signal-slot relatie zit al in vele Qt-widgets (interface objecten) ingebouwd, maar zelf slots bepalen voor bepaalde signalen is ook veel gebruikt. Dit signal-slot gebruik is zeer veelzijdig, meerdere signalen kunnen aan meerdere sloten gekoppeld worden en ook signalen kunnen aaneengeschakeld worden.



Een voorbeeld voor dit signal-slot gebeuren: een simpele quit() functie:

```
Int main (int argc, char *argv[])
{ QApplication app(argc, argv);  // applicatie object creëren
  QPushButton *button = new QPushButton(“Quit”); // Creëren gelabelde knop
  QObject::connect (button, SIGNAL(clicked()), &app, SLOT(quit());
  // als signal “clicked” op object “button” wordt verzonden, roep functie “quit” op in object “app”
  button->show();
  return app.exec();
}
```

2.2 React Native

2.2.1 Oorsprong

React (soms geschreven als react.js of ReactJS) is een JavaScript library die gemaakt is om UI's te bouwen. Gebouwd door Facebook en Instagram teams in 2013. Het doel van React was om grote schaalbare webapplicaties te maken waarvan de data constant veranderd kon worden zonder de pagina te herladen.

React Native startte als een hackathlon project in de zomer van 2013. Na een jaar werken aan een prototype, kreeg React Native zijn eerste job: een onafhankelijk werkende iOS app. Het doel was om een volledige React Native aangedreven app te maken die volgens user experience identiek was aan een app die in Objective-C geschreven was. Dit werd behaald en er werd beslist om RN cross-platform te maken, beginnend met een RN Android team die de basis Android Runtime, een applicatie runtime omgeving gebruikt door de Android OS, en componenten schreef. Begin 2015 werd RN publiek tentoongesteld en tegen eind 2015 werd React Native volledig open source.

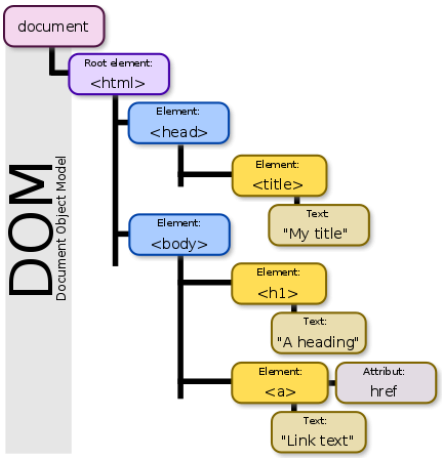
2.2.2 Basis

React Native is volledig gebaseerd op React. Enerzijds is het een open Source library die onderhouden wordt door individuen, 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 al veel gebruikt werd voor webapplicaties. Dit framework zorgt er voor dat de code die je schrijft voor de 3 mobiele platforms kan gebruikt worden. Deze applicaties zijn geschreven met een mix van JavaScript en JSX, een taal die hard lijkt op XML. React Native gebruikt het platform zijn eigen standaard rendering API zodat de mobiele applicatie dezelfde presentatie als voeling zal hebben als een native designed applicatie.

2.2.3 Features

2.2.3.1 Virtual DOM

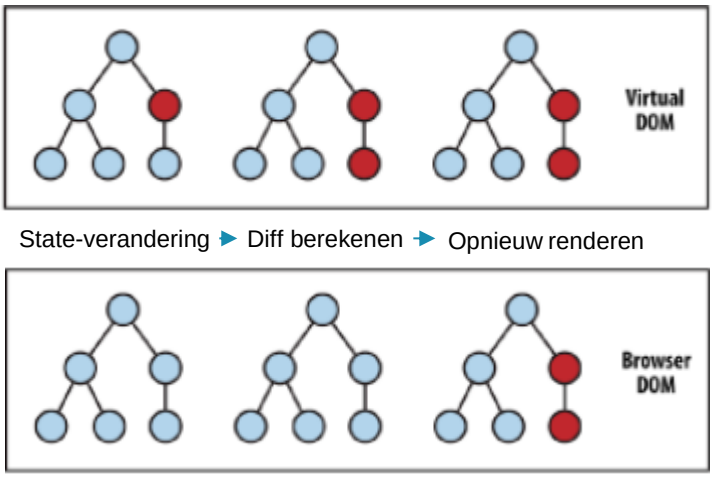
In een web applicatie is één van de meest belastende operaties het veranderen van de DOM, een applicatie programmerende interface dat een HTML, XHTML of XML bestand behandelt als een boomstructuur waarin elke node een object voorstelt als deel van het document.



React onderhoudt een virtuele representatie van deze DOM, Virtual DOM dus. Samen met een 'diffing' algoritme, deze vergelijkt twee trees, kan RN het verschil t.o.v. de oorspronkelijke DOM bepalen en enkel het deel updaten dat veranderd is.

Deze eigenschap is noodzakelijk voor real time applicaties die enige complexiteit bevatten.

In plaats van de veranderingen die gebeuren op een pagina direct te renderen zal React de benodigde veranderingen berekenen in zijn memory en het minimaal mogelijke van de pagina rerenderen, zo zal niet de gehele pagina moeten worden herladen.

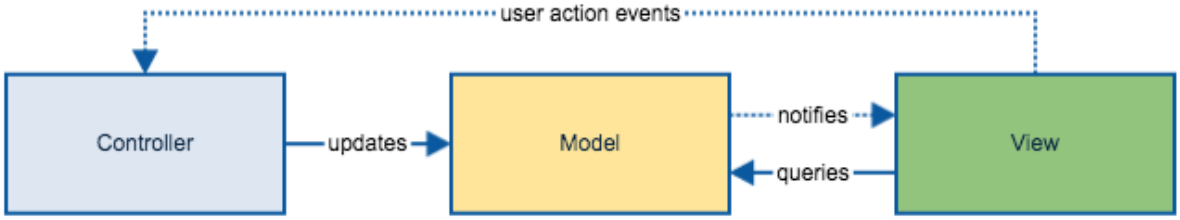


De Virtual DOM heeft dus zijn performance voordelen. Maar het potentieel is veel groter dan enkel performance voordelen. Wat als React een ander doel kon renderen dan de browser z'n DOM?

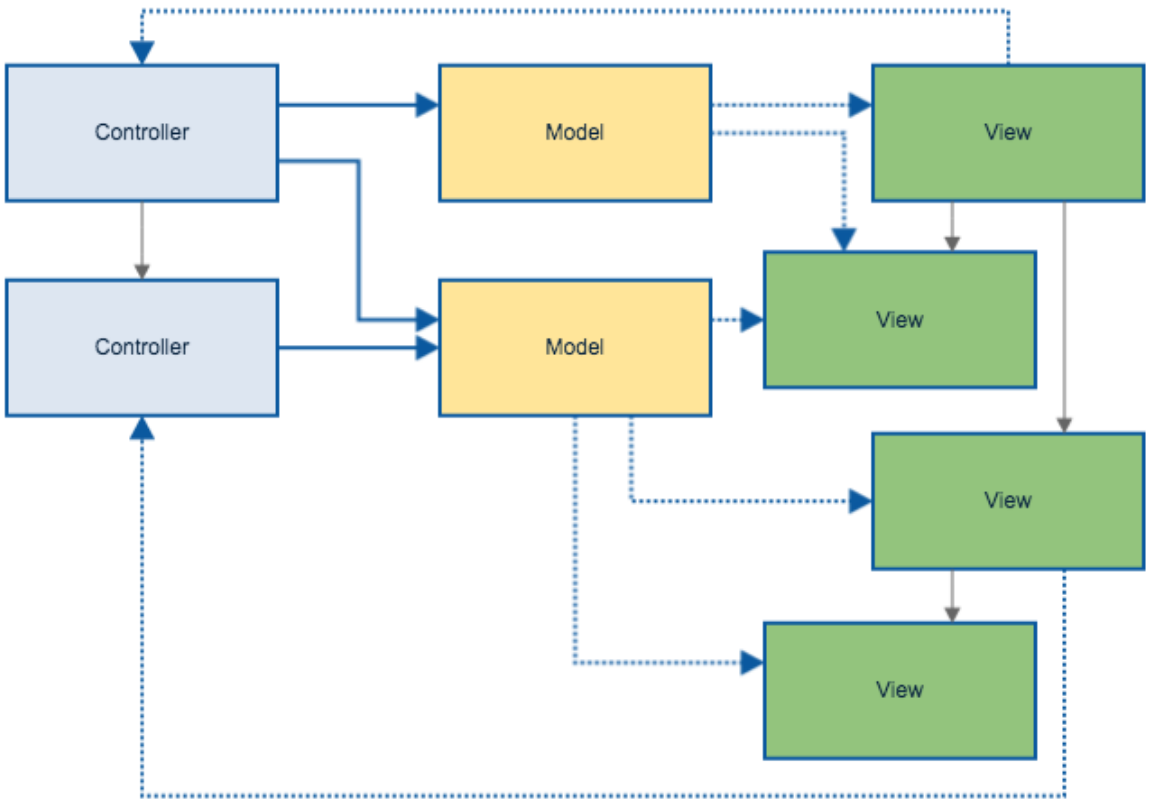
React Native werkt zo.In plaats van de browser z'n DOM te renderen, zal React Native Objective-C APIS gebruiken om iOS componenten te renderen en analoog Java APIs om Android componenten te renderen. Met deze eigenschap onderscheidt RN zich van andere cross-platform development opties, die meestel een web-based view renderen.

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

React gebruikt 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 gebruikt. MVC heeft als architectuur het nadeel dat als de applicatie complexer en groter wordt dat de relaties tussen View, Models & Controllers te complex worden, de code zeer moeilijk te debuggen valt en oneindige loops te gemakkelijk getriggered worden.

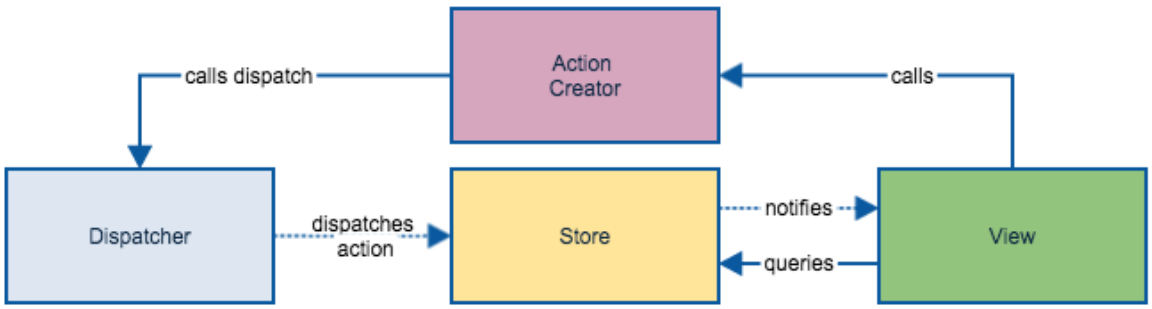


MVC

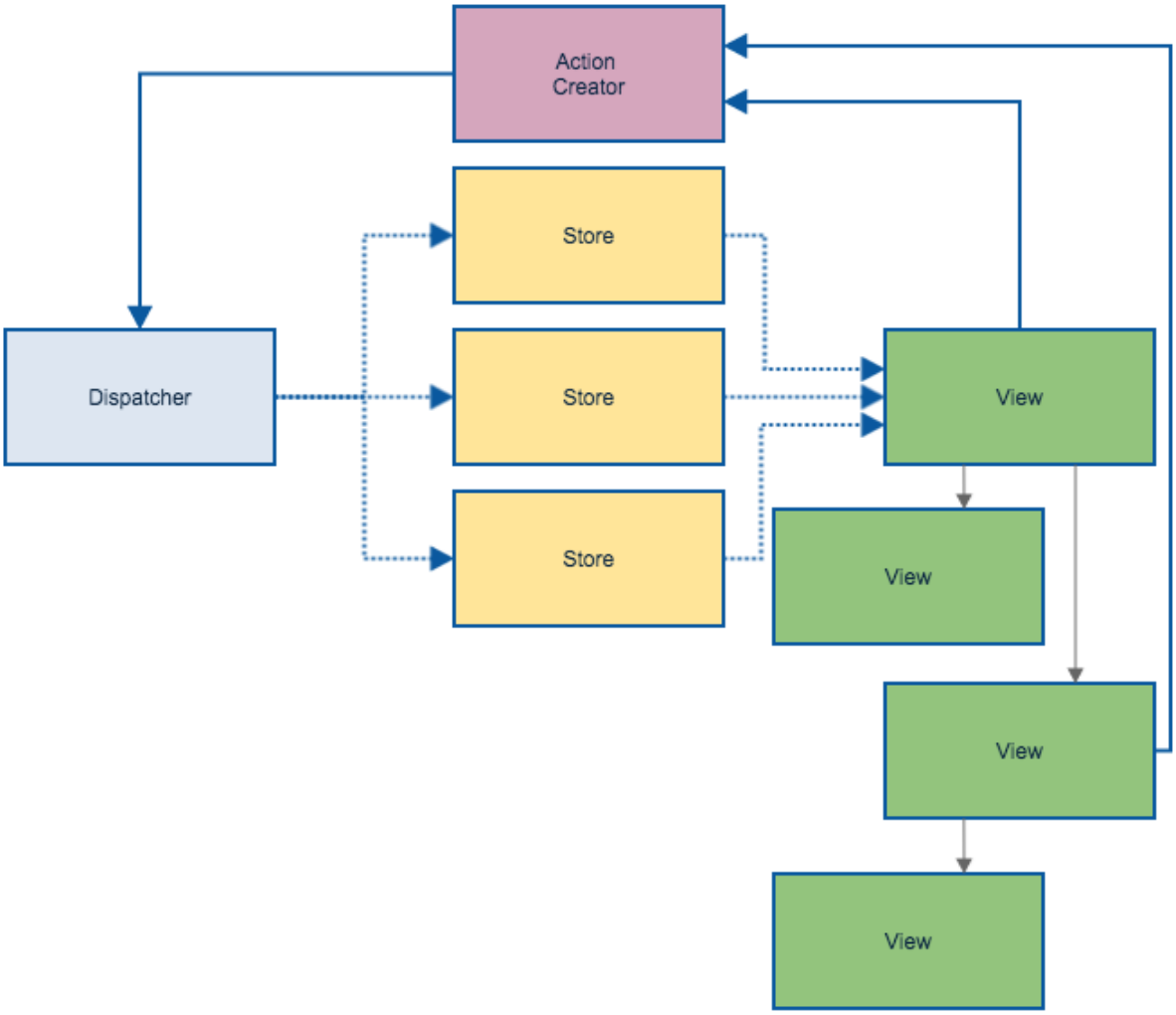


MVC: complexer

De Flux architectuur zorgt ervoor dat elke interactie met de View via één enkel pad naar de dispatcher gaat, deze is dan een centraal punt voor alle acties naar de alle Stores te sturen. De Stores zelf bezitten dan de logica om te bepalen of deze actie dan iets veranderd in de eigen data. Elke Store is verantwoordelijk voor een domein van de applicatie en 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, hiermee vermijd je complexiteit, infinite loops, debug problemen, … Eveneens is het veel makkelijker om de flow van data uit te leggen aan iemand die er nog geen kennis van had.



Flux



Flux: complexer

Tegenwoordig wordt voor de React library veelal de Redux architectuur gebruikt, dit is een uitbreiding op Flux. Redux heeft als core dezelfde architectuur maar lost nog enkele complexiteit issues op dat Flux bevatte: de callback registration vervangen met 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 1 store te bevatten die gemanaged wordt door vele reducers en deze is dan ook zeer makkelijk de data te refreshen, verschillende states van de store op te slaan, … Door dit laatste is Redux dan ook zeer redundant.

2.3 Xamarin

2.3.1 Inleiding

2.4 Apache Cordova

3 APPLICATIE

4 VERGELIJKEN PERFORMANCE

4.1 Development effort

4.1.1 React Native

4.2 Snelheid en geheugen

4.3 User experience

4.4 Conclusie

5 BESLUIT

Referenties

Hier komt de volledige referentielijst in de gekozen stijl APA of IEEE.

https://en.wikipedia.org/wiki/Qt_(software)

https://zappware.com/

https://nl.wikipedia.org/wiki/Qt-toolkit

https://en.wikipedia.org/wiki/React_(JavaScript_library)

Bijlagen

Bijlagen worden bij voorkeur enkel elektronisch ter beschikking gesteld. Indien essentieel kunnen in overleg met de promotor bijlagen in de scriptie opgenomen worden of als apart boekdeel voorzien worden.

Er wordt wel steeds een lijst met vermelding van alle bijlagen opgenomen in de scriptie. Bijlagen worden genummerd het een drukletter A, B, C, …

Bijlage A Detailtekeningen van de proefopstelling

Bijlage B Meetgegevens (op US

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
CAMPUS GEEL
Kleinhoefstraat 4
2440 GEEL, België
tel. + 32 14 80 22 40
iiw.geel@kuleuven.be
www.iiw.kuleuven.be