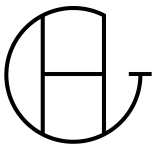


HoGent

Faculteit Bedrijf en Organisatie

React Native
Moderne webtechnologieën in native mobiele applicaties

Steven De Cock

Scriptie voorgedragen tot het bekomen van de graad van
Bachelor in de toegepaste informatica

Promotor:
Martine Van Audenrode
Co-promotor:
David Hemmerijckx

Academiejaar: 2015-2016

Eerste examenperiode

Faculteit Bedrijf en Organisatie

React Native
Moderne webtechnologieën in native mobiele applicaties

Steven De Cock

Scriptie voorgedragen tot het bekomen van de graad van
Bachelor in de toegepaste informatica

Promotor:
Martine Van Audenrode
Co-promotor:
David Hemmerijckx

Academiejaar: 2015-2016

Eerste examenperiode

Samenvatting

Deze scriptie onderzoekt het gebruik van het framework React Native voor het ontwikkelen van native mobiele applicaties. Sinds de komst van React is er een grote mentaliteitswijziging op gebied van webontwikkeling. Standaarden werden heruitgevonden en niet door iedereen op evenveel applaus onthaald. React heeft in zijn korte bestaan zijn waarde aan front-end ontwikkeling bewezen en wint op het internet aan populariteit. In deze scriptie behandel ik React dan ook uitvoerig en maak ik een korte vergelijking met populaire frameworks als AngularJS en Backbone. React werd namelijk ontwikkeld om het rendementsprobleem van webpagina's met veel, veranderende data op een elegante wijze weer te geven. Dit blijkt ook uit de resultaten van de vergelijking. React werd door Facebook ontwikkeld omdat er vooral voor de eigen applicaties veel nood was aan een nieuwe manier om views op te bouwen. In deze scriptie komt u gedetailleerd te weten hoe een applicatie in React wordt opgebouwd en wat React zo krachtig maakt.

In navolging van de populariteit van React op gebied van webontwikkeling werd een framework opgebouwd om native mobiele applicaties te ontwikkelen door gebruik te maken van de kracht van React. Het framework werd React Native gedoopt en uitgebracht in maart 2015 voor iOS ontwikkeling. De ondersteuning voor Android kwam er in september 2015. Ondanks het feit dat applicaties in React Native geschreven worden in Javascript is het toch zo dat de applicaties niet als hybride worden beschouwd maar als echte native applicaties. In de scriptie onderzoek ik hoe dit in zijn werk gaat en bespreek ik enkele voorbeelden van componenten en API's in React Native. Facebook wil met React Native het ontwikkelingsproces voor native applicaties vereenvoudigen en maakt hiervoor gebruik van reeds bestaande webtechnieken. Zo zullen webontwikkelaars sneller vertrouwd raken met het ontwikkelen van native applicaties en wordt debuggen en testen eenvoudiger en volledig analoog aan wat een webontwikkelaar gewoon is.

In de scriptie onderzoek ik ook hoe stabiel React Native is. Uit het opbouwen van een voorbeeld applicatie met React Native is gebleken dat nog niet alles volledig op punt staat en dat voor het ontwikkelen van Android applicaties nog heel wat moet gebeuren. Omdat React Native nog jong is, zijn er delen die nog niet volledig af zijn waardoor die vaak tot kleine irritaties leiden. Facebook is een grote firma die zich in de wereld van de webontwikkeling al verscheidene keren heeft laten gelden en die bewezen heeft kwalitatieve applicaties en tools af te leveren. Met React Native zal dat ook niet anders zijn. Samen met de community zal Facebook ervoor zorgen dat er in de toekomst nog veel rond React Native te doen zal zijn. Toch zouden softwarebedrijven moeten investeren in React Native ontwikkelingen, voor Android en iOS wordt immers veel code hergebruikt waardoor het ontwikkelingsproces aanzienlijk verkort.

Voorwoord

Deze scriptie schreef ik ter voltooiing van mijn studie Toegepaste Informatica met afstudeerrichting mobiele applicatie ontwikkeling. Het onderwerp lag dan ook in de lijn van mijn afstudeerrichting en was een leerrijke ervaring.

Bij het schrijven van deze scriptie kreeg ik steun van veel mensen. Ik zou een aantal van hen speciaal willen bedanken. Als eerste wens ik mijn co-promotor David Hemmerijckx te bedanken, hij heeft samen met mij naar een geschikt onderwerp gezocht en ik kon telkens bij hem terecht met inhoudelijke vragen. Daarnaast wens ik mijn collega Maarten Dhondt te bedanken die mij geholpen heeft met de LaTeX opbouw van mijn scriptie, mijn vader en zijn vrouw, Hugo en Ann De Cock- Van der Smissen voor nakijken op taalfouten. En uiteraard bedank ik ook mijn promotor, Martine Van Audenrode voor de ondersteuning van mijn bachelorproef.

Daarnaast bedank ik mijn familie en vrienden voor de morele steun tijdens mijn bacheloropleiding aan HoGent

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van code fragmenten

Hoofdstuk 1

Het huidige Javascript landschap

1.1 Wat is Javascript?

Javascript is een dynamische programmeertaal die gebruikt wordt om webpagina's interactief te maken. Het wordt door de browser van de gebruiker zelf aanvaard en gedraaid, waardoor het geen constante downloads van een bepaalde webpagina nodig heeft. Javascript mag echter niet verward worden met Java, enkel de namen zijn gelijkaardig. Toch heeft Javascript veel geleend bij andere talen zoals Java, Python en Perl :

- **Java:** Syntax, primitieve waarden tegenover objecten
- **Python en Perl:** strings, arrays en reguliere expressies

1.2 Client-side Javascript

Javascript wordt het vaakst gebruikt voor Client-side implementatie, de bedoeling is dat het script rechtstreeks in de HTML pagina aanwezig is of dat er verwezen wordt naar een Javascript document. In de body van een HTML pagina kan gerefereerd worden naar een stuk Javascript code, zoals in Code fragment 1.1. Wanneer er naar een aparte Javascript document verwezen moet worden kan dat zoals in Code

```
1 <script>
2 function myFunction() {
3     document.getElementById("demo").innerHTML = "Paragraph changed.";
4     }
5 </script>
```

Code fragment 1.1: Javascript in HTML body

```
<script src="myScript.js"></script>
```

Code fragment 1.2: Verwijzing in HTML naar Javascript file

```
function ValidateEmail(mail)
{
 if (/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(myForm.emailAddr.
     value))
  {
    return (true)
  }
    alert("U heeft een ongeldig e-mail adres opgegeven!")
    return (false)
}
```

Code fragment 1.3: Client-side validatie met Javascript

fragment 1.2. Het attribuut "src= .." verwijst dan naar het Javascript document die in de HTML pagina gebruikt moet worden.

1.3 Voordelen van Javascript

Het voornaamste voordeel van Javascript is dat er onmiddellijk interactie kan gemaakt worden met de UI op client-side niveau. Dit wil zeggen dat de interacties met server tot een minimum beperkt kunnen blijven. Het beste voorbeeld hiervoor is de validatie van gebruiker input. Javascript kan onmiddellijk nagaan of een gebruiker een correct e-mail adres of telefoonnummer heeft ingegeven op basis van reguliere expressies zoals in Code fragment 1.3. De waarde die werd ingegeven in het HTML document wordt onderworpen aan een test door deze functie, er wordt getest of de waarde overeenkomt met het opgegeven patroon van hoe een e-mailadres er moet uitzien (voorbeeld@domein.be). Hierdoor hoeft de gebruiker niet te wachten tot de pagina herladen is en de server op basis van domeinregels bepaald heeft dat een bepaalde invoer niet correct was doorgegeven. Javascript geeft dit dus onmiddellijk weer en belast de server niet onnodig met foutieve aanvragen.

Daarnaast heeft Javascript er ook voor gezorgd dat de interactiviteit en de ervaringen van de gebruiker aangenamer werden. HTML is een statische geheel die de inhoud van een webpagina beschrijft en die geen mogelijkheden biedt om de voor gedefinieerde tekst of afbeeldingen te gaan wijzigen zonder dat de pagina moet herladen worden. Javascript biedt verschillende en eenvoudige oplossingen om de inhoud van HTML domein objecten te gaan aanpassen. (Vijayweb Solutions, 2014) In Code fragment 1.4 wordt door middel van 13 lijnen code een afbeelding groter en terug kleiner gemaakt door met de muis over de afbeelding te gaan. Op die manier maakt Javascript het mogelijk om te werken met drag-en-drop componenten, afbeelding galerijen die automatisch

```
<img onmouseover="bigImg(this)" onmouseout="normalImg(this)" border="0" src="
    smiley.gif" alt="Smiley" width="32" height="32">

<script>
function bigImg(x) {
    x.style.height = "64px";
    x.style.width = "64px";
}

function normalImg(x) {
    x.style.height = "32px";
    x.style.width = "32px";
}
</script>
```

Code fragment 1.4: HTML inhoud wijzigen met Javascript

van afbeelding wijzigen, enz.. .

1.4 De beperkingen van Javascript

Javascript wordt niet beschouwd als een volwaardige programmeertaal omdat er een aantal belangrijke onderdelen ontbreekt :

- Client-side Javascript is niet instaat om bestanden te lezen, te wijzigen of te schrijven. Deze functionaliteit is vooral om veiligheidsredenen niet mogelijk.
- Javascript is singlethreaded.
- Javascript heeft geen ondersteuning voor netwerk applicaties.

1.5 Javascript frameworks en libraries

Javascript wordt al snel eentonig en repetitief, je moet heel vaak dezelfde code herhalen om bepaalde zaken uit te voeren in Javascript. Het is de bedoeling van een framework of library om de code herbruikbaar te maken en het voor de ontwikkelaar makkelijker te maken om een goede structuur in zijn applicaties te behouden, zodat deze later gemakkelijker worden uitgebreid met nieuwe functionaliteiten.

Een framework is een verzameling van source code of libraries die de mogelijkheid biedt om gemeenschappelijke functionaliteiten te voorzien aan een bepaalde soort van applicaties. Het bevat vaak voorgeprogrammeerde templates, helper klassen, interfaces, ... Een framework behandelt eerder hoe een applicatie opgebouwd moet worden volgens de regels die de makers ervan voorzien hebben. De ontwikkelaar kan zich, door gebruik te maken van een framework, meer focussen op de onderdelen die uniek zijn aan zijn of haar applicatie.

Een library daarentegen zal over het algemeen slechts één deel van een bepaalde functionaliteit voorzien. Het is een verzameling van voorgeprogrammeerde Javascript code die ervoor zorgt dat bepaalde onderdelen van functionaliteiten abstract benaderd kunnen worden en waardoor de ontwikkelaar veel minder code zelf hoeft te schrijven. (DocForge, 2015)

Voorbeelden van frameworks : AngularJS, Backbone.js, Ember, Knockout. Voorbeelden van libraries : JQuery, underscore.js.

Hoofdstuk 2

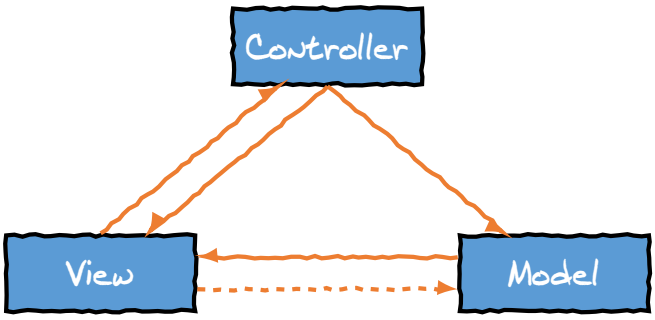
React

2.1 Wat is React?

ReactJS of React is een Javascript library gecreëerd door Facebook, React zou een oplossing bieden bij het ontwikkelen van complexe user interfaces met telkens veranderende datasets. Dit is geen onbelangrijke uitdaging en moet niet enkel onderhoudbaar zijn, maar ook schaalbaar zijn op het niveau van Facebook's datasets.

React werd geboren in de Facebook advertentie organisatie, waar men gebruik maakte van een traditionele client-side Model-View-Controller aanpak. Applicaties zoals deze maken normaal gezien gebruik van two-way data binding samen met renderen van het template. Telkens wanneer data objecten gewijzigd worden moet de pagina of een deel van de pagina opnieuw laden. In een kleine performante applicatie geeft dit geen merkbare problemen, maar hoe meer functionele uitbreidingen er aan de applicatie worden toegevoegd, hoe meer views en modellen de applicatie gaat bevatten. Deze zijn allemaal verbonden door een delicaat en onoverzichtelijk kluwen van code die de verbondenheid van views en modellen onderling bijhoudt. Dit wordt al snel enorm complex, moeilijk onderhoudbaar, moeilijk testbaar en niet-gebruiksvriendelijk omdat items die opgevraagd moeten worden vaak veel meer tijd vragen om gepresenteerd te kunnen worden.

React werd dus ontwikkeld om één welbepaald probleem te gaan oplossen, namelijk het tonen van data in de UI. Als u denkt dat dat probleem reeds werd opgelost, bent u zeker niet verkeerd. Het verschil is dat React werd gecreëerd om grote schaalbare user interfaces, zoals die van Facebook en Instagram, met veel data die continu wijzigt, te gaan behandelen. Dit soort interfaces kan gecreëerd worden en de data behandeling kan gebeuren via verschillende tools los van React, maar zorgen voor een hoop code en dus ook veel werk om te onderhouden en up-to-date te houden. Deze manier van werken werd reeds door Facebook zelf gebruikt want Facebook werd immers in 2004 gemaakt, 9 jaar voor React werd uitgebracht. React zorgt met andere woorden eerder



Figuur 2.1: MVC basis architectuur

voor een korte, overzichtelijke en onderhoudbare manier van werken.

React veranderde de manier waarop deze applicaties gecreëerd werden en voerde een aantal gewaagde veranderingen door op gebied van webontwikkeling. Toen React in mei 2013 op de markt werd gebracht, was er heel wat commotie in de webontwikkeling community. Heel wat ontwikkelaars waren geïnteresseerd en sterk onder de indruk van de React aanpak maar sommigen waren niet tevreden met wat React deed. React daagde namelijk een aantal van de conventies uit, die reeds standaarden geworden zijn voor Javascript frameworks en libraries.

De ingenieurs bij Facebook hebben met React een mentaliteitswijziging doorgevoerd op gebied van webontwikkeling en het maken van schaalbare en onderhoudbare Javascript applicaties. Ze voorzien React van een hele hoop nieuwe features die het bouwen van single-page applicaties toegankelijk maakt voor ontwikkelaars met verschillende kennisniveaus. (Gackenheimer, 2015)

2.2 React en frameworks

React wordt door velen vaak beschouwd als een volwaardig Javascript framework vergelijkbaar met andere frameworks zoals Backbone, KnockoutJS, AngularJS, Ember, of één van de vele MVC frameworks die bestaan.

Figuur 2.1 toont de basis van een typische structuur van een MVC framework. Elk onderdeel heeft een eigen verantwoordelijkheid.

Model-View-Controller:

- **Model:** het model bevat de staat van een applicatie en stuurt events naar de View wanneer de staat van bepaalde data objecten werd gewijzigd.
- **View:** de view is de presentatie laag, dit is hetgeen de gebruiker ziet en bevat alle UI componenten. De view stuurt events naar controller en in sommige gevallen naar het model om bepaalde data op te halen.

- **Controller:** de controller zorgt ervoor dat alle events van de view en het model opgevangen worden, eventueel behandeld en op correcte wijze doorgestuurd worden naar de view of het model.

Dit is slechts een oppervlakkige benadering van wat een MVC is, in realiteit zijn er verschillende varianten en implementaties, vandaar wordt er vaak verwezen naar een MV*- architectuur. Het is niet de bedoeling om uit te leggen wat MVC is, maar wat React juist niet is.

Simpelweg beschouwd men React als de view in MV* - frameworks. Zoals eerder vermeld is React een manier om de UI van een applicatie te beschrijven en een mechanisme om die UI aan te passen wanneer de data wijzigt. React bestaat uit declaratieve componenten die samen een interface beschrijven en een data-binding mechanisme die niet observeerbaar is (niet two-way binding). React is ook heel makkelijk te manipuleren omdat je de gecreëerde componenten kan nemen en combineren met andere componenten, die telkens werken zoals je verwacht. Dit komt omdat React enorm schaalbaar is.

In het artikel Harrington (2015) worden 2 Javascript frameworks vergeleken met React in verschillende browsers. Harrington schreef in elk framework een gelijkaardige applicatie (zie Tabel 2.1) die door een lijst van 1000 items loopt en elk item in de lijst weergeeft. De items worden vervolgens in een niet-gesorteerde lijst geplaatst op een webpagina. Wanneer op de *run* link (tekst-label met milliseconden in) geklikt wordt, wordt de tijd opgeslagen en wanneer de 1000 elementen geladen zijn, wordt opnieuw de tijd opgeslagen. De laatste tijd wordt van de eerste afgetrokken en de uitkomst wordt weergegeven op het scherm. Voor de test werd bij elk van de twee frameworks en React, 10x de lijst geladen en werd de tijd bijgehouden in tabellen, dit werd voor drie browsers herhaald.

Resultaten:

Het resultaat is dat over het gemiddelde React en Raw sneller laden dan Angular en Knockout. Dit resultaat is niet verassend, React en Raw JS hebben achterliggend niets meer te doen, terwijl Angular en Knockout eerst een aantal componenten eigen aan het framework moeten laten in haken zoals bv. Services, Directives,.... React en Raw JS moeten dit niet doen waardoor ze sneller kunnen laden. Met deze test wil Harrington niet bewijzen dat React beter is dan Angular of Knockout maar wil hij de kracht van de schaalbaarheid van React aantonen.

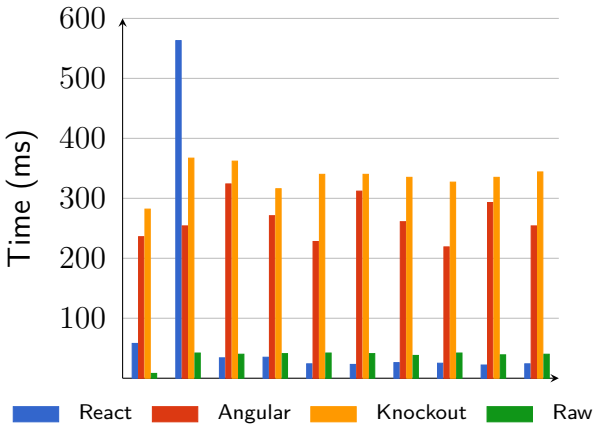
2.3 React concepten

De React API is klein en daardoor gemakkelijk aan te leren. Gemakkelijk wil niet zeggen dat het herkenbaar is. Eerst moet er een aantal concepten en terminologie

React 40 ms	Angular 100 ms	Knockout 161 ms	Raw 14 ms
pretty orange desk	odd brown house	short black bbq	fancy brown cookie
helpful yellow house	fancy red keyboard	fancy white sandwich	fancy brown mouse
big pink cookie	long black burger	long black car	handsome brown pony
helpful orange burger	big pink mouse	tall white pizza	important brown pony
clean orange house	important orange mouse	plain pink desk	odd brown mouse
easy red cookie	handsome orange mouse	cheap brown house	mushy orange house
tall red desk	plain orange sandwich	tall red bbq	small white burger
crazy pink car	mushy blue sandwich	unsightly brown table	fancy yellow pizza
crazy purple chair	expensive black bbq	long brown pony	inexpensive yellow bbq
important orange sandwich	quaint orange chair	large pink chair	fancy purple chair
mushy green table	clean blue pony	odd white chair	cheap brown desk
plain yellow burger	odd yellow sandwich	mushy brown mouse	cheap black pizza
small pink sandwich	large purple table	important yellow burger	pretty blue keyboard
easy purple keyboard	important yellow pony	helpful white keyboard	handsome pink mouse
inexpensive purple desk	unsightly red desk	clean black bbq	big orange cookie
expensive red pizza	short brown table	small blue house	long pink house
pretty yellow burger	expensive green bbq	angry green mouse	pretty yellow pizza
adorable white cookie	quaint green mouse	small white cookie	quaint purple keyboard

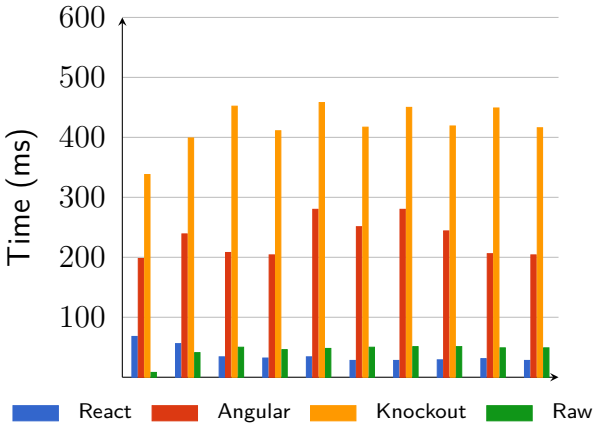
Tabel 2.1: Performance vergelijking voor React, Angular and Knockout, (Harrington, 2015)

	React	Angular	Knockout	Raw
	58	236	282	8
	563	254	367	42
	34	324	362	40
	35	271	316	41
	24	228	340	42
	23	312	340	41
	26	261	335	38
	25	219	327	42
	22	293	335	39
	24	254	344	40
Average	83.4	265.2	334.8	37.3



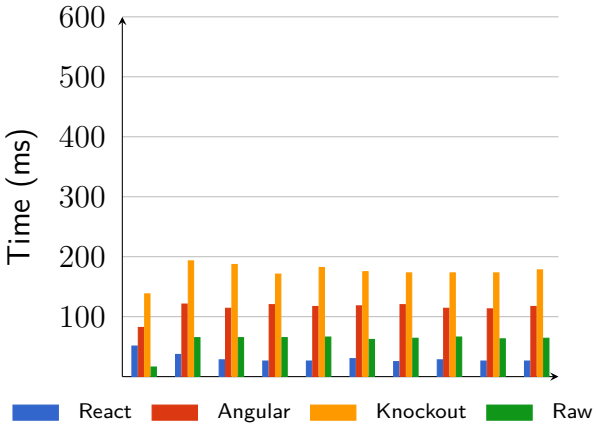
Figuur 2.2: Performance test door Harrington Chrome 39.0.2171.95

	React	Angular	Knockout	Raw
	68	198	338	8
	56	239	399	41
	34	208	452	50
	32	204	411	46
	34	280	458	48
	28	251	417	50
	28	280	450	51
	29	244	419	51
	31	206	449	49
	28	204	416	49
Average	36.8	231.4	420.9	44.3



Figuur 2.3: Performance test door Harrington voor Firefox 34.0.5

	React	Angular	Knockout	Raw
	51	82	138	16
	37	121	193	65
	28	114	187	65
	26	120	171	65
	26	117	182	66
	30	118	175	62
	25	120	173	64
	28	114	173	66
	26	113	173	63
	26	117	178	64
Average	30.3	113.6	174.3	59.6



Figuur 2.4: Performance test door Harrington voor Safari 7.0.2

```
1 var MyClass = React.createClass({
2
3   render : function() {
4     return (<div>Hello world</div>);
5   }
6
7 });
```

Code fragment 2.1: Component in ReactJS

worden uitgelegd.

2.3.1 Componenten

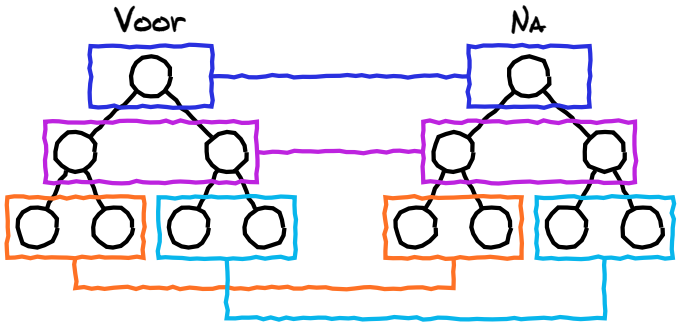
Componenten zijn de basis van React en de view van de applicatie. In Code fragment 2.1 wordt een voorbeeld gegeven van een component in React. Let goed op bij de *render* functie, deze geeft een HTML tag terug. Wanneer deze zal aangeroepen worden, zal die de tekst "Hello world", laten verschijnen op de HTML pagina.

2.3.2 Virtual DOM

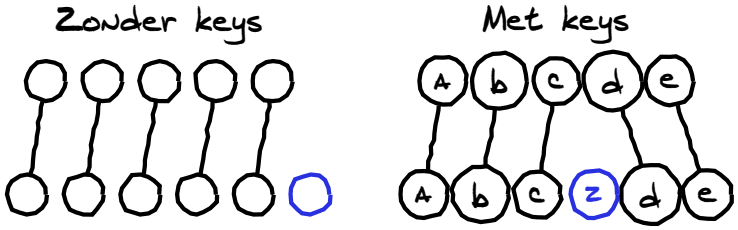
Dit is waarschijnlijk het meest belangrijke concept van React. Het virtuele DOM is de oplossing voor het probleem die het DOM stelt, namelijk dat het (nog) niet aangepast werd voor het weergeven van dynamische data. Met Javascript en JQuery kon je als ontwikkelaar wel een oplossing voor dit probleem vinden, maar dat bleek zowieso een hele hoop werk en er kwam heel wat code en denkwerk bij kijken. Dit was uiteraard het probleem waarvoor Facebook React heeft ontwikkeld. Het virtuele DOM is geen nieuw concept en bestaat al eerder dan React. React is wel ontwikkeld met het concept van het virtuele DOM in het achterhoofd.

Het virtuele DOM wordt ontwikkeld bovenop het DOM, het zal uiteraard het DOM gaan behandelen. Het doet dit echter zo weinig en zo efficiënt mogelijk, door te werken met een light weight kopie van het DOM. Wanneer de data wordt gewijzigd in de kopie wordt er gekeken welke delen in het DOM vervangen moeten worden. Er wordt opzoek gegaan naar de kortste manier om de gewijzigde delen te gaan vervangen. Deze manier gaat veel sneller dan direct te werken met het DOM, omdat niet alle delen van het DOM die belastend zijn voor de browser her-in-ge-laden moeten worden.(Gackenheimer, 2015)

Om dit te realiseren maakt React gebruik van diff algoritmes, event delegation en rendering.



Figuur 2.5: Level per level aanpassen



Figuur 2.6: Lijsten aanpassen

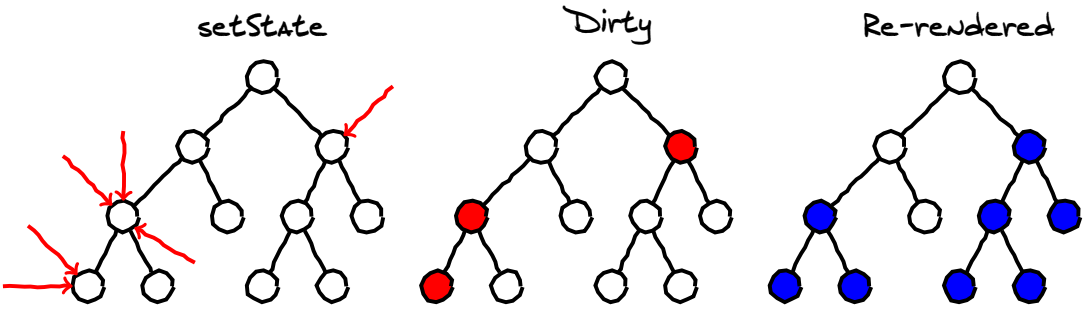
Diff algoritmen

React gaat via een aantal algoritmen op zoek naar het minimum aantal stappen die nodig zijn om van de vorige versie van het DOM naar een nieuwere versie te gaan.

- Level per level: React gaat proberen om op level niveau wijzigingen aan te brengen. Dit gaat de complexiteit drastisch verminderen en brengt weinig verlies met zich mee. Het is in web-applicaties zeer zeldzaam dat componenten verplaatst worden naar een ander niveau in de boom structuur. Zie Figuur 2.5.
- Lijsten: Wanneer een component bij een eenmalige iteratie 5 andere componenten bevat, wordt het moeilijk om in het midden van die lijst een nieuwe component toe te voegen. React gaat standaard een associatie maken tussen het eerste component van de eerste lijst en het eerste component van de tweede lijst, etc. Als ontwikkelaar kan je React helpen door een sleutelattribuut toe te voegen zodat React een beter begrip krijgt over de mapping van beide lijsten.

Event delegation

Toevoegen van event listeners aan DOM nodes is enorm traag en gaat veel geheugen gebruiken. React gebruikt hiervoor een populaire techniek:“event delegation”. Bij *event delegation* wordt een *event* getriggerd op de parent node en niet op elke child



Figuur 2.7: Flow setState to render DOM

node. Bijvoorbeeld, een *Onclick listener* op een niet-gesorteerde lijst (<ul>) element in plaats van op elk lijst item (<li>) element in de lijst . Wanneer het event wordt getriggerd gaat de parent nagaan welk child element wordt aangeklikt en op basis van het *id*, die React automatisch toevoegd aan elk child element, de nodige wijzigingen doorvoeren.

Rendering

Wanneer *setState* wordt aangeroepen op een component, zal React dat component aanduiden als "dirty". Aan het einde van elke event loop zal React alle "dirty" componenten opnieuw laden. Dit betekent dat er tijdens een event loop, slechts 1 moment is dat het DOM wordt geüpdatet. Dit is een belangrijke eigenschap om performante applicaties te schrijven, maar is extreem moeilijk te benaderen in gewone Javascript code. React regelt dit standaard voor de ontwikkelaar.

Het voordeel is dat *setState* niet alleen op het root element kan opgeroepen worden, maar op elk child element dat aanwezig is in de applicatie. Wanneer hierop een wijziging gebeurt, moet niet het volledige DOM worden overschreven maar slechts een deel van de tree.(Freed, 2015)

Deze technieken zijn niet nieuw en worden gebruikt door andere Javascript libraries en frameworks:

- **Virtual-dom by Matt Esch:** Een Javascript virtual DOM algoritme
- **Mithril:** Javascript Framework
- **Bobril:** component georiënteerd framework geïnspireerd door React en Mithril.

Wat React speciaal maakt hierin is dat als je als ontwikkelaar de logische volgorde respecteert die nodig is om een applicatie te bouwen in React, je dit allemaal standaard krijgt en zelf hiervoor niets hoeft te doen.

```
1  //JSX versie
2
3  React.render(
4      <div>
5        <h1> Hello </h1>
6      </div>
7    );
8
9  //wordt vertaald naar
10
11 React.render(
12     React.createElement('div', null,
13       React.createElement('h1', null, 'Hello')
14     );
15   );
```

Code fragment 2.2: JSX component in React

2.3.3 JSX

JSX is de transformatie laag van React om XML syntax die gebruikt wordt om React componenten te schrijven, om te zetten naar syntax die door React gebruikt wordt om elementen te laten renderen in Javascript. Let op! Dit is geen verplicht onderdeel van React maar het gebruik wordt sterk aangeraden. JSX maakt coderen veel eenvoudiger en leesbaarder, zo kunnen HTML tags rechtstreeks worden aangemaakt en aanvaardt het aangepaste React klassen. In Code fragment 2.2 wordt een component aangemaakt met JSX-syntax.

2.3.4 Properties

Properties zijn een set van opties die een component bevat en worden aangeroepen in React als *this.props*, een gewoon Javascript object in React. Properties zullen tijdens de lifecycle van een component niet wijzigen, ze zijn immutable. Wanneer een ontwikkelaar iets wil wijzigen aan een component zal hij de *state* van een object moeten wijzigen.

2.3.5 State

State is een set die kan geïnitialiseerd worden op elke component en die gewijzigd kan worden doorheen de lifecycle van van dat component. De *state* mag niet gewijzigd worden buiten het component tenzij door een parent component. Het is aangewezen om zo weinig mogelijk *state* objecten te gebruiken op de componenten, hoe meer state objecten geïnitialiseerd en aangepast worden hoe groter de complexiteit van de componenten.

```
var MyComponent = React.createClass({

  render: function() {
    return (<div>Hello world</div>);
  }

});
React.render( <MyComponent / > , document.getElementById('container'));
```

Code fragment 2.3: CreateClass voorbeeld ReactJS

2.3.6 Flux

Flux is een project dat zeer nauw verbonden is met React. Het is belangrijk om te weten hoe het werkt met React. Flux is de Facebook applicatie architectuur voor hoe data werkt met React componenten op een logische en georganiseerde manier. Flux is geen MVC architectuur omdat het niet werkt met bi-directionele data flow (Two-way data binding). Flux is daarentegen wel essentieel voor React omdat het helpt met het gebruik van React componenten op de manier waarop ze voorzien zijn om gebruikt te worden. Flux doet dit door een uni-directionele data flow (one-way data binding) te gaan creëren die door drie stukken van de flux architectuur wordt geleid namenlijk de *dispatcher*, de *stores* en uiteindelijk de React View. Flux is essentieel in het bouwen van **webapplicaties** met React en zal niet verder aan bod komen in deze scriptie.

2.4 React Core

Tot hiertoe werd een beeld gecreëerd over hoe React werkt en zich profileert ten opzichte van andere frameworks. Er werden een aantal concepten verklaard die hieronder in detail worden uitgediept. In dit hoofdstuk wordt de syntax van React belangrijk en worden enkele basis begrippen in detail uitgelegd aan de hand van React code, vooral in de JSX syntax.

2.4.1 React *createClass*

In het hoofdstuk over React concepten (2.3) werd aangehaald dat componenten de basis vormen van een React applicatie. De methode *React.createClass* zorgt ervoor dat een nieuwe component wordt aangemaakt, deze methode moet verplicht de methode *render()* overschrijven omdat deze het object, meestal in HTML vorm weergeeft. De *render()* methode zal in 2.5.3 verder toegelicht worden.

In Code fragment 2.3 wordt een basis object *'MyComponent'* aangemaakt via de methode *React.createClass* door de *render()* methode te overschrijven zal een * * element aangemaakt worden die de tekst "Hello world" op het scherm afprint. Op

```
1  var MyComponent = React.createClass({
2
3    render: function() {
4      return (
5        <div>
6          {this.props.name}
7        </div>)
8    }
9  });
10
11 React.render(<MyComponent name= "Steven" / >, document.getElementById('
       container'));
```

Code fragment 2.4: Property aan een component toevoegen

```
Object {$$typeof: Symbol(react.element), type: "p", key: "first", ref: null, props: Object...}    script.js:62
Object {$$typeof: Symbol(react.element), type: "p", key: "second", ref: null, props: Object...}   script.js:62
```

Figuur 2.8: Console.log function on child

lijn 8 wordt *React.render()* methode opgeroepen die het object zal creëren en aan het DOM zal toevoegen als child node van de node met *id = container*.

Wanneer een waarde aan het component moet worden doorgegeven krijgen we syntax zoals Code fragment 2.4. Wanneer in de methode *React.render* het object *myComponent* wordt aangemaakt geven we een *property 'name'* mee. In *myComponent* wordt verwezen naar de *property name* door gebruik te maken van "{this.props.name}" om een property op te roepen.

2.4.2 React *Children*

React.Children is een object die een aantal helper functies bevat om gemakkelijker te werken met de properties van een component (*this.props.children*). Deze functies zullen uitgevoerd worden op elk child object die de component bevat en geven een object terug.

- *React.Children.map(children, myFn, [, context])*: Deze functie zal voor elk child object in *children* de functie *myFn* uitvoeren, waarbij (optioneel) een context kan toegevoegd worden. Code fragment 2.5 toont aan hoe de map functie werkt door twee child elementen toe te voegen wanneer een object van *MyComponent* wordt geïnitialiseerd. In de *render* functie van *MyComponent* wordt elk child object van *this.props.children* overlopen en voor elk child object wordt de info in de console uitgeprint (zie Figuur 2.8).
- *React.Children.forEach(children, myFn [, context])*: *forEach* functie werkt net zoals *map* functie maar geeft geen object terug. Deze functie zal enkel de lijst doorlopen en een actie uitvoeren op elk child object.

```
1  var myComponent = React.createClass({
2
3      render: function() {
4        React.Children.map(this.props.children, function (child) {
5          console.log(child);
6        });
7        return (
8          <div>
9          {this.props.name}
10         </div>)
11     }
12 });
13
14 React.render(<myComponent name="Steven">
15   <p key="first">a child</p>
16   <p key="second">another</p>
17 </myComponent>,document.getElementById('container'));
```

Code fragment 2.5: Voorbeeld van een map functie

```
1  var MyComponent = React.createClass({
2      displayName: "MyComponent",
3      render: function() {
4        return React.createElement(
5          "div",
6          null,
7          this.props.name);
8      }
9  });
10 React.render(React.createElement(MyComponent, {name: "Steven"}),
11 document.getElementById('container'));
```

Code fragment 2.6: Create element in component

- *React.Children.count(children)*: de *count* methode zal het aantal child objecten in de lijst children teruggeven.

2.4.3 React *CreateElement*

De *createElement* methode wordt gebruikt om een *ReactElement* te creëren, deze elementen verhogen de performantie van een React applicatie, omdat ze kunnen bestaan uit *ReactElementen*, *ReactComponenten*, HTML tags, ...

React.createElement(type, [props [, children ...]])

Een element wordt gecreëerd met minstens één en optioneel drie argumenten: een string type, optioneel een lijst van properties en optioneel een lijst van child objecten.

In Code fragment 2.6 wordt op lijn 4 een * * element expliciet gecreëerd terwijl eenzelfde functie op lijn 11 wordt aangeroepen met als type een object van *MyComponent*. In dit voorbeeld lijkt het overbodig om de functie op te roepen om een * *

```
class MyComponent extends React.Component {

  render(){
    return (<div> Hello World </div>);
  }
}
```

Code fragment 2.7: Component als super klasse

```
var MyComponent = React.createClass({

    render: function() {
      return (<div> Hello World </div>);
    }
});
```

Code fragment 2.8: Component als variabele

element aan te maken maar wanneer de elementen gaan nesten en de complexiteit van de UI begint te stijgen, is het aangeraden om met *ReactElementen* te gaan werken.

2.5 React componenten

React componenten zijn de bouwstenen van een React applicatie. React componenten hebben een eigen API die een aantal methodes en helpers bevat.

2.5.1 Creatie

Een React component kan op verschillende manieren aangemaakt worden, hieronder vindt u twee voorbeelden van de meest gebruikte manieren.

- Voorbeeld 1: *React.Component* als super klasse, maakt een klasse aan die overerft van *React.Component*. Overschrijf **altijd** de *render()* methode (zie Code fragment 2.7).
- Voorbeeld 2: maakt een variabele die de methode *React.createClass* aanroept, deze methode geeft een object weer van type *React.Component* met eigen implementatie (zie Code fragment 2.8). Deze manier is de meest gebruikte om een component aan te maken.

2.5.2 State

De State van een property kan gewijzigd worden door de functie *setState* aan te roepen in een component. Deze functie kan een andere functie aanroepen of een object

weergegeven die de state van de component wijzigt. Wanneer *setState* aangeroepen wordt, komt het nieuwe object in de React update queue, het mechanisme die React gebruikt om wijzigingen te behandelen.

Door deze manier van werken moet men als ontwikkelaar rekening houden met een aantal zaken wanneer *setState* wordt aangeroepen. Zo is er geen enkele garantie dat de updates in een bepaalde volgorde zullen gebeuren. Wanneer er een afhankelijkheid is voor het updaten van een bepaalde property kan dit via een Callback functie worden doorgegeven die optioneel wordt meegegeven aan *setState*. Daarnaast is het aangewezen om de state niet te wijzigen met de "dot-notatie" *this.state*, en deze enkel te gebruiken om de state op te roepen. Het idee hierachter is dat de ontwikkelaar het state object als immutable moet beschouwen en enkel het *setState* proces van React de wijzigingen aan de state controleert.

2.5.3 Specificatie functies en component Lifecycle

Specificatie functies zijn functies die verplicht of optioneel kunnen worden toegevoegd aan componenten wanneer ze worden geïnitialiseerd, sommigen ervan zijn lifecycle functies. Er zijn drie soorten lifecycles mogelijk op een component:

- De eerste keer wanneer een component aan het DOM wordt toegevoegd.
- Wanneer de state wijzigt.
- Wanneer de properties wijzigen.

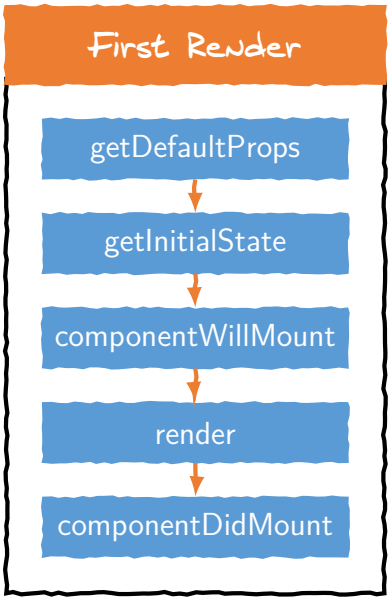
Render

Zoals reeds een aantal keren is vermeld,moet elke component een render functie bevatten. Deze functie aanvaardt een *ReactElement* en voorziet een container locatie waar het component zal toegevoegd worden aan het DOM.

getInitialState

Deze functie zal een object teruggeven, de inhoud van dit object zal de *state* van het component *setten* wanneer het voor de eerste keer geïnitialiseerd wordt.

Code fragment 2.9 geeft een object *MyComponent* weer waarbij een * * element met de tekst "Hello Steven" wordt toegevoegd aan het DOM. Door *getInitialState* aan te roepen, kan de *name property* een initieële waarde mee krijgen.



Figuur 2.9: Lifecycle: eerste maal op het DOM

```
1  var MyComponent = React.createClass({
2
3      getInitialState: function() {
4        return (
5          {name : "Steven"}
6        )
7      },
8      render: function() {
9        return (
10         <div>Hello {this.state.name}</div>
11       );
12     }
13
14 });
15 React.render(<MyComponent/ >, document.getElementById("root"));
```

Code fragment 2.9: GetInitialState voorbeeld

Figuur 2.10: Screenshot resultaat mixin voorbeeld

Get Default Props

Wanneer een *ReactClass* (component of element) voor het eerst wordt aangemaakt, wordt de methode *getDefaultProps* éémalig aangeroepen om de properties op de component met default waarden te bevolken. Wanneer deze methode niet opgeroepen wordt, zullen *this.props* voorzien worden van *null* waarden of de waarde die tijdens de initialisatie worden meegegeven. (<MyComponent name = "Steven"/>).

Component Will Mount

ComponentWillMount is een lifeCycle event die React gebruikt wanneer React het component neemt en op het DOM plaatst. De methode wordt éénmaal aangeroepen vlak voor de *render* methode van de component. Het unieke aan *componentWillMount* is dat wanneer de ontwikkelaar *setState* aanroept in de methode en de *state* van het component wijzigt, er niet opnieuw wordt ingeladen, maar dat de gewijzigde *state* onmiddellijk zichtbaar is.

Component Did Mount

ComponentDidMount is een methode die enkel aan de kant van de client wordt aangeroepen. De component bestaat en staat in het DOM.

Mixin

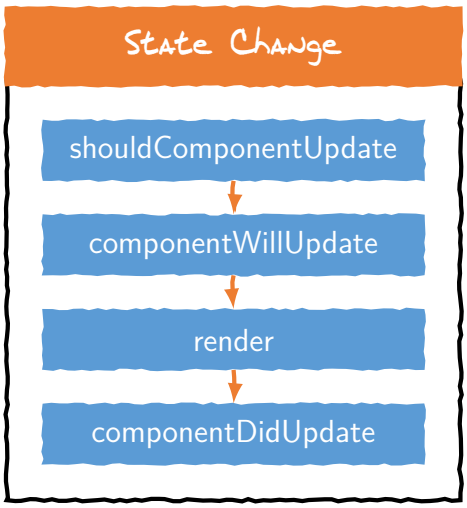
Een *Mixin* is een array, het kan de lifecycle events van een component monitoren zodat de ontwikkelaar met zekerheid weet dat functionaliteit op de correcte manier en tijdstippen van de lifecycle kan uitgevoerd worden. Code fragment 2.10 toont hoe een *Mixin* een component kan monitoren. Deze code zal een *timer* genereren die de levensduur van de component op de pagina weergeeft in het aantal seconden (zie Figuur 2.10).

Should Component Update

Elke keer wanneer er een wijziging plaats vindt in de *state* of de properties wordt deze functie aangeroepen. Deze functie is een mechanisme dat kan gebruikt worden om

```
var SetIntervalMixin = {
  componentWillMount: function() {
    this.intervals = [];
  },
  setInterval: function() {
    this.intervals.push(setInterval.apply(null, arguments));
  },
  componentWillUnmount: function() {
    this.intervals.map(clearInterval);
  }
};

var MyComponent = React.createClass({
    mixins: [SetIntervalMixin], // Use the mixin
    getInitialState: function() {
      return ({seconds: 0, name: "Steven"});
    },
    componentDidMount: function() {
      this.setInterval(this.tick, 1000); // Call a method on the mixin
    },
    tick: function() {
      this.setState({seconds: this.state.seconds + 1});
    },
    render: function() {
      return (
        <div>
        <h1>Hello {this.state.name}</h1>
        <p>
          MyComponent has been running for {this.state.seconds} seconds.
        </p>
        </div>
      );
    }
});

React.render(<MyComponent / >, document.getElementById("root"));
```

Code fragment 2.10: Voorbeeld Mixin



Figuur 2.11: LifeCycle : state gewijzigd

het opnieuw laden (*render()*) van de component over te slaan, wanneer de wijzigingen geen update nodig hebben.

Component Will Update

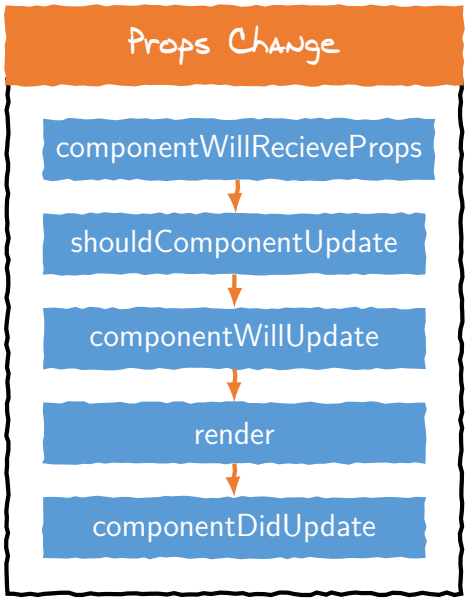
ComponentWillUpdate wordt vlak voor de render functie opgeroepen, het is niet mogelijk om hier setState aan te roepen. Deze functie dient slechts als voorbereiding op de update zelf.

Component Did Update

componentDidUpdate is net als *componentDidMount* een functie die enkel aan de client kant wordt opgeroepen. Deze functie wordt aangeroepen nadat een update volledig werd doorgevoerd.

Component Will Recieve Props

Deze functie wordt aangeroepen vlak voor de component properties zal ontvangen, en wordt uitgevoerd elke keer als een property wijzigt, behalve bij de eerste keer dat de component aan het DOM wordt toegevoegd. Hier kan men steeds setState aanroepen zonder een extra update te moeten doen.



Figuur 2.12: Lifecycle : properties gewijzigd

Hoofdstuk 3

React Native

In maart 2015 bracht Facebook het open-source framework React Native uit voor iOS. React Native is sinds maart verkrijgbaar via GitHub en kan genieten van een hoge populariteit bij ontwikkelaars. Toen Facebook in maart het nieuws openbaar maakte, werd het Android gedeelte van de app met een vertraging van 6maanden aangekondigd. Op 14 september was het dan zo ver, React Native kan nu ook gebruikt worden om Android applicaties te schrijven op de React manier.

> *"React Native brings what developers are used to from React on the Web – declarative self-contained UI components and fast development cycles – to the mobile platform, while retaining the speed, fidelity and feel of native applications."*
>
> — (Witte & von Weitershausen, 2015)

De aanpak van Facebook om applicaties te schrijven in Javascript is niet nieuw maar React Native heeft een heel eigen manier ontwikkeld om dit te kunnen realiseren.

3.1 Native applicatie

Een native applicatie is een applicatie die ontwikkeld is voor een specifiek platform. De maker van het platform heeft hiervoor een specifieke programmeertaal aangeduid, met API om de platform specifieke componenten, views, enz.. te kunnen gebruiken. Vaak zijn hieraan ook duidelijke en verplichte richtlijnen verbonden om te mogen/kunnen programmeren voor dit specifieke platform. De applicaties worden verkrijgbaar gemaakt in stores die door de aanbieders van het OS worden aangeboden.

Positieve eigenschappen van native ontwikkeling :

- Er kan maximaal gebruik gemaakt worden van alle beschikbare functionaliteiten van het apparaat waarop de applicatie draait.

- Een native applicatie heeft toegang tot de bibliotheek voor het gebruik van media.
- Het is niet noodzakelijk om met het internet verbonden te zijn.
- Een native applicatie heeft een snelle en gebruiksvriendelijke UX.

Toch zijn er ook negatieve eigenschappen aan verbonden :

- De applicatie moet ontwikkeld worden op elk platform afzonderlijk.
- Er is een goedkeuring nodig om de applicatie te publiceren in de applicatie stores.
- Een update in de software van de OS betekent vaak dat een applicatie moet aangepast worden.
- Hoge ontwikkelingskost.

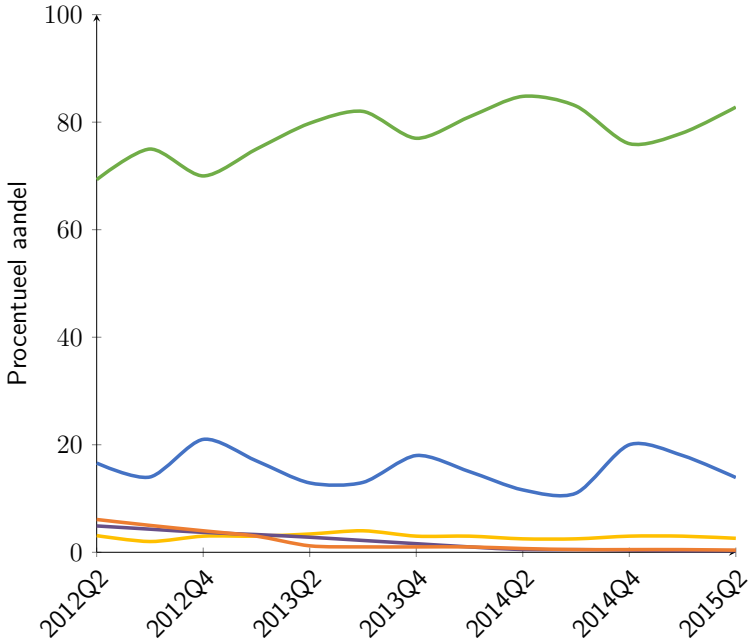
Op niveau van mobiele applicaties zijn er verschillende OS, dit zijn de belangrijkste en meest voorkomende :

- **Android** : ontwikkeld door Google.
- **iOS** : ontwikkeld door Apple.
- **Windows Phone** : ontwikkeld door Microsoft.
- **BlackBerry OS** : ontwikkeld door RIM (Research In Motion).
- Symbian, Java ME, Kindle, Bada, ...

Android en iOS hebben volgens IDC (International Data Corporation)(Figuur 3.1) samen een marktaandeel van 96.7%. Windows phone is daarna de grootste met slechts 2.6%. (IDC, 2015)

3.2 Hybride applicaties

Hybride applicaties zijn een combinatie van een native component, vaak een webview, en een web applicatie in HTML en Javascript. Deze aanpak wordt weleens de "best of both worlds" -aanpak genoemd omdat ze de voordelen van een native applicatie combineren met de voordelen van een web applicatie, en het grootste voordeel is natuurlijk dat de code slechts één keer moet geschreven worden. Bij een hybride applicatie is het dus meestal zo dat de ontwikkelaar de applicatie gaat programmeren zoals een web applicatie maar deze wordt als het ware gewrapt in een native webview component van het specifieke platform waarvoor de applicatie bedoeld wordt. Wanneer een ontwikkelaar de native API componenten, zoals camera, gebruikerslocatie, ...



	Android	iOS	Windows Phone	BlackBerry OS	Others
2012Q2	69.3 %	16.6 %	3.1 %	4.9 %	6.1 %
2013Q2	79.8 %	12.9 %	3.4 %	2.8 %	1.2 %
2014Q2	84.8 %	11.6 %	2.5 %	0.5 %	0.7 %
2015Q2	82.8 %	13.9 %	2.6 %	0.3 %	0.4 %

Figuur 3.1: IDC - worldwide smartphone OS Market Share, Aug 2015

wenst te gebruiken in zijn applicatie dan kan die daarvoor gebruik maken van een framework zoals PhoneGap, Ionic of Cordova. Deze frameworks maken gebruik van HTML5, CSS en Javascript om hybride applicaties voor mobiel te ontwikkelen.

Positieve eigenschappen van hybride applicaties :

- Een hybride applicatie moet slechts éénmaal geschreven worden om op verschillende platformen te kunnen draaien. Figuur 3.2
- Een hybride applicatie heeft mogelijkheden om native componenten te gebruiken.
- De inhoud van een hybride applicatie kan snel ge-update worden.
- Hybride applicaties kunnen in de stores geplaatst worden.
- De ontwikkelingskost voor een hybride applicatie is minder dan die van een native applicatie.

Negatieve eigenschappen van hybride applicaties :

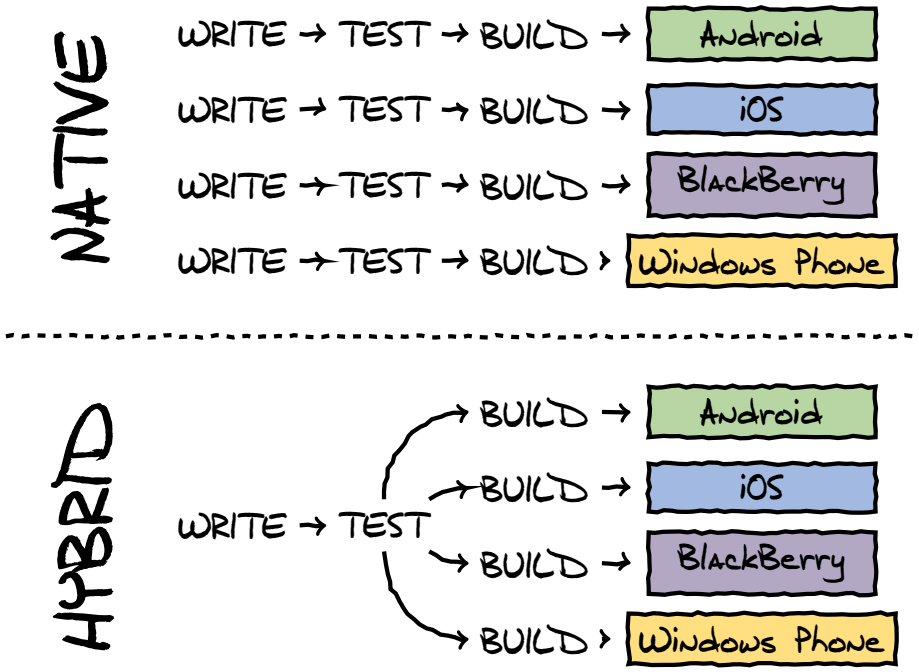
- De ontwikkelaar heeft een brede kennis nodig, zowel web als native kennis zijn vereist.
- Hybride applicaties zijn niet geschikt voor games of zware grafische applicaties.
- Een hybride applicatie loopt over het algemeen trager dan een native applicatie.
- Een hybride applicatie is niet altijd offline beschikbaar.

De populariteit van hybride applicaties groeit en de voordelen zijn voor vele applicatie ontwikkelaars een doorslaggevend argument om te kiezen voor een hybride oplossing. Toch moeten ontwikkelaars telkens afwegen hoe men de applicatie zal ontwikkelen, hybride of native. De vraag die telkens gesteld moet worden is hoe een applicatie de beste gebruikerservaring kan bekomen.

Een aantal voorbeelden van populaire hybride applicaties :

- **Uber** : applicatie voor een soort taxiservice.
- **Apple's app store** : Apple heeft zijn liefde voor HTML5 nooit onder stoelen of banken gestoken, en de appstore is inderdaad een hybride applicatie.
- **Twitter** : sociaal netwerk applicatie.

Uit deze voorbeelden kan men concluderen dat wanneer men een applicatie wenst te bouwen die veel http aanvragen (post, get, ...) zal moeten uitvoeren, best hybride ontwikkeld wordt.



Figuur 3.2: Hybride vs. Native ontwikkeling

3.3 Waarom React Native?

Er zijn verschillende redenen waarom native mobiele omgevingen moeilijker zijn om mee te werken dan het web. Allereerst is het moeilijk om componenten weer te geven op het scherm, deze moeten vaak manueel op de view geplaatst en voorzien worden van een grootte. Daarnaast was er geen mogelijkheid om React code te integreren, die het proces voor bouwen van websites met telkens veranderende datasets had vereenvoudigd.

De grootste ergernis bij Facebookingenieurs was dat als er native applicaties worden gebouwd ze rekening moesten houden met het telkens heropstarten van de applicatie nadat er iets werd gewijzigd. De wijzigingen van een knop die bijvoorbeeld een aantal centimeter werd verplaatst, zorgde ervoor dat als men de wijziging op het scherm wou weergeven, de applicatie volledig moest herstart worden. Dat vertraagde sterk het ontwikkelingsproces. Bij Facebook wordt er 2 maal per dag (!) een nieuwe versie van de website afgeleverd zodat resultaten van een bepaald experiment onmiddellijk worden waargenomen en eventuele fouten opgelost kunnen worden. Bij de native applicatie van Facebook moet er vaak weken of maanden gewacht worden om een wijziging te kunnen doorvoeren. De aanbieders van de stores moeten deze immers eerst nakijken en goedkeuren. Dit brengt een hoop tijdsverlies en dus ook geldverlies met zich mee.

"Move fast" zit bij Facebook in het DNA, maar men was gebonden aan de regels van de mobiele OS aanbieders om de wijzigingen van hun applicaties te kunnen doorvoeren.

Toch wou men bij Facebook niet inbinden op de gebruikerservaring en het uitzicht van de applicatie, deze moesten native blijven. (Occhino, 2015)

3.4 Wat is React Native?

React Native is een Javascript framework die gebruik maakt van de React library, om native applicaties voor iOS (iPhone, iPad) en Android te schrijven, hierdoor kunnen ze het grootste deel van de smartphone- en tabletgebruikers bereiken. Toch gebruikt Facebook in React Native een hybride aanpak, want het maakt gebruik van Javascript. De applicatie zal wel volledig native beschouwd worden.

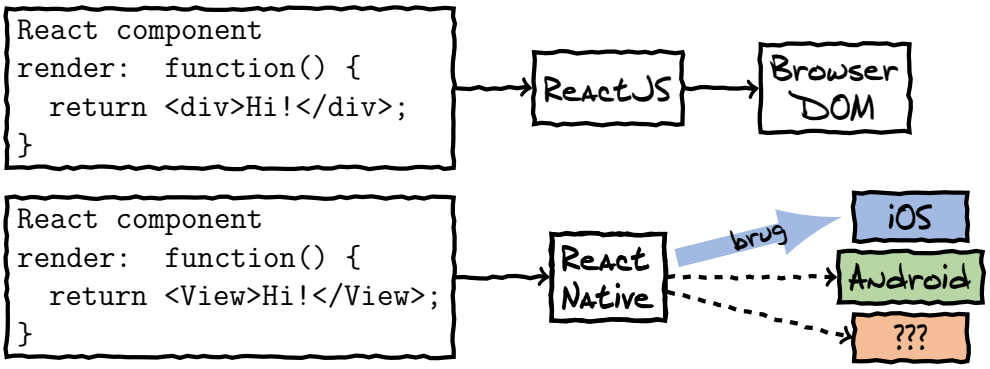
Dankzij dit framework heeft Facebook het mogelijk gemaakt voor web ontwikkelaars om applicaties te schrijven voor mobiele platformen in de reeds beheerste Javascript taal en omdat het grootste deel van de code kan gedeeld worden tussen beide platformen wordt een gekend probleem van mobiele applicatieontwikkeling geëlimineerd, namelijk dat eenzelfde applicatie in meerdere programmeertalen geschreven moest worden.

Net zoals React worden React Native applicaties geschreven in een mix van Javascript en JSX. Onderliggend is er een soort brug voorzien die de componenten van de API's in Objective C (iOS) of Java (Android) gaan aanspreken en gebruiken in de applicatie. Met andere woorden, de applicaties die gemaakt worden in React Native maken gebruik van echte mobiele UI componenten en geen webviews zoals vele hybride applicaties, zodat de applicatie aanvoelt zoals elke andere mobile applicatie geschreven in de native taal. Daarnaast zijn er ook interfaces voorzien die de hardware componenten zoals camera, gebruikerslocatie, ... kunnen aanspreken. (Eisenman, 2015)

3.5 Hoe werkt React Native?

Om de werking van React Native te kunnen begrijpen, moet de werking van React gekend zijn, namelijk de manier waarop React omgaat met het virtuele DOM (2.3.2). React plaatst zoals eerder vermeld een abstractie laag tussen de code van de ontwikkelaar en datgene wat zichtbaar is voor de gebruiker. Maar wanneer gesproken wordt over native mobiele applicaties dan wordt er niet gewerkt met de browser dus moest Facebook een andere oplossing zoeken om deze manier van werken ook te kunnen toepassen.

Facebook heeft hiervoor met React Native een gepaste oplossing bedacht. In plaats van gebruik te maken van het DOM van de browser, gaat React Native Objective C API's gebruiken om iOS componenten weer te geven, of Java API's gebruiken om Android componenten weer te geven. In Figuur 3.3 wordt het verschil tussen een React component en React Native component op gebied van weergave getoond. In



Figuur 3.3: Virtual DOM vs bridge

plaats van het Virtuele DOM wordt de "bridge" aangesproken om de juiste weergave te tonen in de juiste OS.

De gebouwde bridge of brug voorziet React van een interface in de platform specifieke UI elementen. De voorziene layout door de render functie van een React component zal met andere woorden vertaald worden naar een gepaste component. Namelijk <view> kan vertaald worden naar een iOS specifieke UIView. De brug is niet uniek aan React, het PhoneGap framework gebruikt ook een brug om API's aan te spreken, het verschil met React Native is dat de brug niet enkel de hardware componenten zal aanspreken maar ook de componenten die de layout van een native applicatie zal samenstellen.

De levensloop van de componenten in React Native is dezelfde als die van React voor webontwikkeling, maar het weergaveproces is anders. Door de brug te gebruiken, zal een React Native applicatie niet op de main thread maar op een andere thread lopen, op deze manier zal de applicatie niet lang moet laden en voorkomt het dat de applicatie ongewenst vastloopt.

3.5.1 Developer experience

De wijze waarop React Native werkt, wordt vaak gelinkt aan cross-platform applicatie ontwikkeling zoals PhoneGap, Ionic of Cordova. Toch onderscheid React Native zich van hybride ontwikkeling doordat het de platform standaard weergave API gebruikt. Hybride tegenhangers maken gebruik van een webview wrapper om hun applicatie weer te geven. Wanneer React Native een component aanspreekt wordt dit ook de echte native component van dat platform en geen HTML5 component. Op deze manier worden de nadelen van hybride ontwikkeling op gebied van gebruikerservaring volledig geëlimineerd.

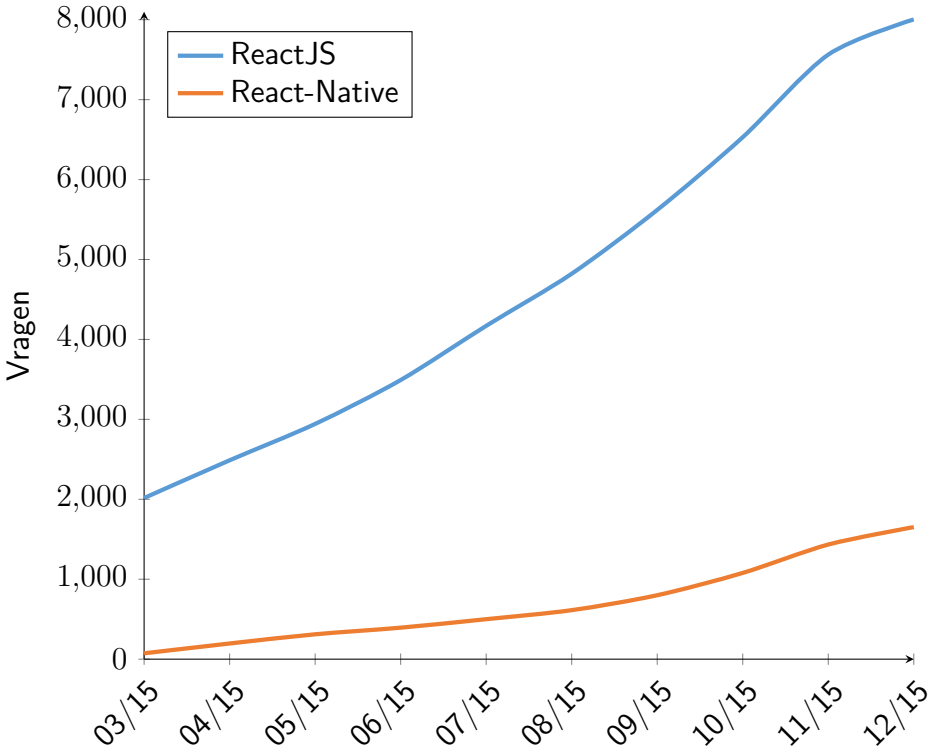
Eén van de grootste voordelen van React Native is de "Developer Experience", want die kan nu door gewoon Javascript (+ React) een native applicatie maken, zonder echt

veel te moeten leren over mobiele applicatie ontwikkeling. Dankzij de uitgebreide documentatie die Facebook voorziet voor ontwikkelaars van React Native kan op eenvoudige en correcte wijze gebruik gemaakt worden van de native componenten. Maar het ingenieursteam van Facebook heeft naast een goede documentatie ook nog een aantal developer tools ontwikkeld die het leven van een ontwikkelaar moeten vereenvoudigen. Zo maakt men het mogelijk dat tijdens het ontwikkelingsproces een ontwikkelaar de applicatie kan refreshen zoals een browser dat ook kan. Dit wordt technisch mogelijk gemaakt doordat de applicatie op een node.js server draait en in de code een pad wordt gegenereerd naar de lokale server, die werkt zoals een gewone webserver. Dit brengt natuurlijk met zich mee dat de ontwikkeliteraties aanzienlijk korter worden. Dankzij het concept van een webserver voor het runnen van de applicatie in ontwikkeling, kan een ontwikkelaar de tools van Google's Chrome en Mozilla's Firefox ook gebruiken voor zijn applicatie. De applicatie zal niet zichtbaar zijn in de browser, maar debug boodschappen, errors, http requests, ... zijn net zoals bij webtesten mogelijk om te gebruiken. De ingenieurs van Facebook hebben er ook voor gezorgd dat elke mogelijke error voorzien wordt van een gedetailleerde boodschap.

Het andere probleem van Facebook was de duurtijd, vaak enkele weken of maanden, om een update door te voeren. Apple en Google laten toe dat wijzigingen betreffende Javascript code onmiddellijk worden doorgevoerd, zonder eerst een controle door te voeren. Hierdoor zijn de wijzigingen in een React Native applicatie onmiddellijk zichtbaar. Daarnaast is de code herbruikbaarheid van React Native applicaties ook nog een enorm groot voordeel. Wanneer een applicatie volledig native ontwikkeld wordt, moet die zowel in Java als in Objective-C (of Swift) geschreven worden, dubbel werk. Omdat React Native, native applicaties maakt, is het niet altijd mogelijk om alle code te gaan hergebruiken. Sommige componenten zijn heel specifiek voor het platform en moeten dus ook anders benaderd worden. Volgens Christopher Chedeau, heeft de Facebook Ads applicatie een hergebruik van codebase tussen de iOS en Android applicatie van ongeveer 86%. (Chedeau, 2015)

> *"If you are an iOS developer considering playing with React Native, know that you aren't alone. React Native is wonderful, and you should try to embrace it with an open mind. Don't pigeonhole yourself into what is comfortable like I did."*
>
> — (Shilling, 2015)

Mark Shilling benadrukt in zijn bevindingen over werken met React Native als doorwinterde iOS developer, dat React Native nog niet volledig op punt staat maar dat er in een paar maanden tijd reeds veel gebeurd is door de community en Facebook om de kleine irritaties weg te werken.

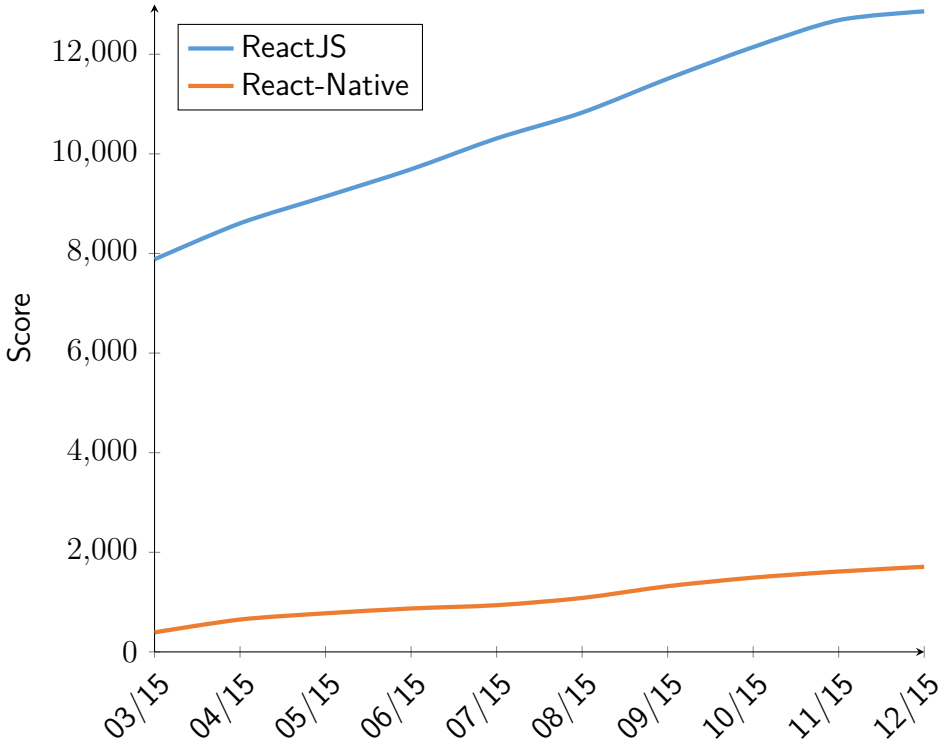


Figuur 3.4: StackOverflow vragen (StackOverflow, 2015)

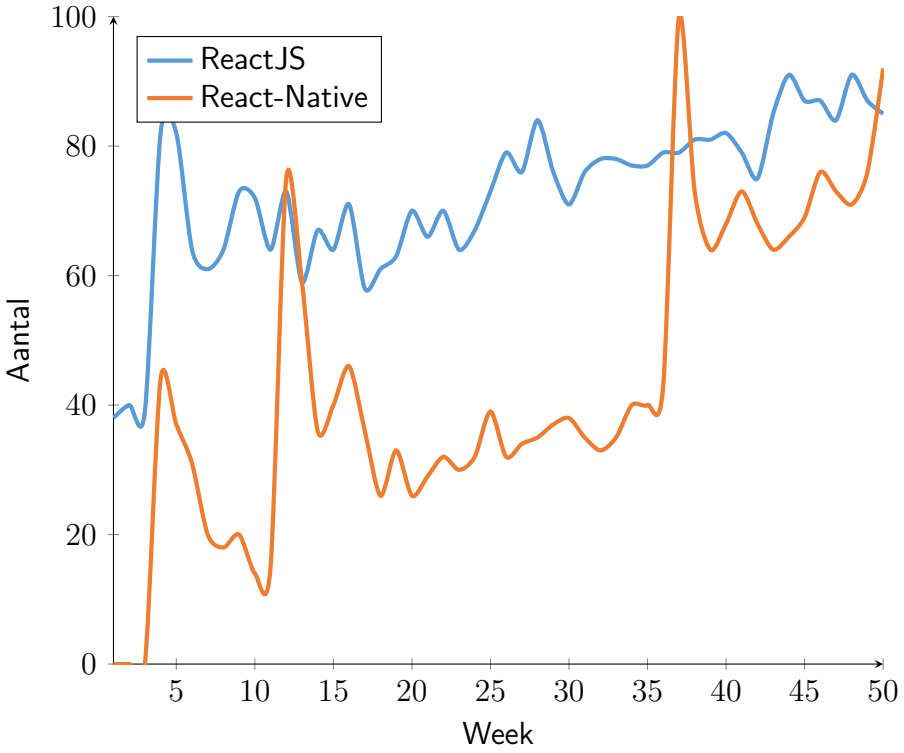
3.6 React Native populariteit op het internet

React en React Native zijn nog niet zo lang actief op het internet, toch zijn er een aantal trends zichtbaar. De grafiek Figuur 3.4 toont het aantal vragen die bij *StackOverflow* worden gesteld over de Thema's ReactJS en React Native, per maand in cumulatieve vorm, (StackOverflow, 2015). Aangezien React Native vrij jong is, zijn het aantal vragen dus ook nog vrij klein maar belangrijk om op te merken is dat de lijn sterk stijgend is zowel voor ReactJS als voor React Native. ReactJS kreeg ook een boost wanneer React Native werd aangekondigd en React Native krijgt een piekmoment wanneer Facebook de Android versie ervoor uitbracht. Hetzelfde geldt voor de score die toegekend wordt aan een bepaalde vraag of antwoord over een bepaald onderwerp (Figuur 3.5). De trend is in stijgende lijn wat betekent dat de community van React Native actief bezig is om ontwikkelaars te ondersteunen bij het bouwen van applicaties, en opstellen van *best practices*.

Dezelfde trends kan men terug vinden bij *Google Trends* Figuur 3.6 die de zoekopdrachten op basis van kernwoorden *ReactJS* en *React Native* weergeeft week na week, startend van begin 2015, (Google, 2015). React Native heeft overduidelijke piekmomenten tijdens de eerste periode in maart 2015 (week 16 - 17) toen het voor het eerst



Figuur 3.5: StackOverflow scores (StackOverflow, 2015)



Figuur 3.6: Google Trends zoekopdrachten met kernwoorden ReactJS en React Native, Google (2015)

werd uitgebracht en tijdens de periode dat React Native ook voor Android beschikbaar werd in september 2015 (week 37 - 38). Toch is de algemene trend voor zowel ReactJS als voor React Native een stijgende lijn in zoekopdrachten bij *Google Trends* .

3.7 React Native componenten

Net zoals in React, werkt React Native op een exacte wijze om componenten aan te maken. Toch zijn er een aantal verschillen in syntax, zoals eerder vermeld, werkt native niet met het DOM maar met platform specifieke componenten. Deze moeten dan ook op correcte wijze worden aangesproken.

3.7.1 Views

In ReactJS plaatsen we HTML componenten op het DOM (* *, *<p>*, **), bij React Native gaan we de platformcomponenten weergeven. Het meest gebruikte basiscomponent in UI elementen is het *<View>* component. Deze is voor een we-

ReactJS	React Native
	<View>
	<Text>
,	<ListView>
<img>	<Image>

Tabel 3.1: ReactJS componenten vs React Native componenten

```
var FirstProject = React.createClass({
  render: function() {
    return (
      <View>
        <Text>
          Welcome to React Native!
        </Text>
      </View>
    );
  }
});
```

Code fragment 3.1: Basis component React Native

bontwikkelaar best te vergelijken, met het * * element en het vertaalt zich naar respectievelijk iOS en Android als *UIView* en *View*. In tabel 3.1 wordt een vergelijking van HTML componenten, gebruikt in ReactJS gemaakt met React Native tegenhangers.

In Code fragment 3.1 toont men hoe een component wordt samengesteld in React Native, het enige verschil met het web is zoals eerder vermeld enkel de UI componenten. Deze componenten uit tabel 3.1 worden aanzien als de basiscomponenten. Deze zijn aanwezig in de standaard package en worden automatisch geïmporteerd wanneer een nieuwe project wordt gestart. Als de ontwikkelaar andere componenten wenst te gebruiken in zijn applicatie zullen die eerst geïmporteerd moeten worden, zoals Code fragment 3.2. Let hierbij goed op de naamgeving van het component *DatePickerIOS*. Platform specifieke componenten en API's krijgen speciale tags, waarbij de suffix de platform naam is.

```
var React = require('react-native');
var {
    DatePickerIOS
    } = React;
```

Code fragment 3.2: Import een component in React Native

```
var style = {

    backgroundColor: 'white',
    fontColor: '16px'

}

var MyComponent = React.createClass({

    render : function()
    {
        return (
            <View>
            <Text style={style}>Hello World</Text>
    </View>

        );
    }

})
```

Code fragment 3.3: Stijlen van componenten in React Native

3.7.2 Stijlen van componenten

Op het web wordt een stijl meegegeven aan componenten door middel van CSS, aanbieders van mobiele OS hebben vaak een eigen manier om componenten vorm te geven. Bij Android gebeurt dit door XML en bij iOS heeft men een eigen variant gemaakt, React Native heeft één standaard aanpak gecreëerd om componenten te gaan stijlen. De wijze waarop in React Native stijl aan componenten wordt gegeven, is een vereenvoudigde versie van CSS die door de brug, steunend op Flexbox modaliteiten vertaald wordt naar de platform eigen stijlen. Het is in React Native niet gebruikelijk om aparte stijlpagina's te maken zoals bij het web, maar om de stijl als een Javascript object te omschrijven. In Code fragment 3.3, zie je hoe een stijl aan een component wordt meegegeven.

Hoofdstuk 4

Werken met React Native

Facebook en de community werken hard om het React Native framework grondig uit te bouwen en alle platform specifieke componenten en API's te integreren. In dit hoofdstuk worden een aantal van deze componenten en API's besproken die resulteren in een applicatie als voorbeeld.

4.1 Werkomgeving

4.1.1 MAC OSX

Applicaties ontwikkelen voor iOS met React Native kan momenteel enkel gebeuren in een MAC OSX omgeving, omdat native iOS applicaties enkel ontwikkeld kunnen worden in *XCode*, de ontwikkelomgeving van MAC OSX. Om deze omgeving compatibel te maken met React Native zijn een aantal stappen vereist.

Eerst en vooral raadt Facebook aan om *Homebrew* te installeren. *Homebrew* installeert pakketten en ontwikkelaars tools die Apple is vergeten. Deze pakketbeheerder installeren kan op eenvoudige manier in de *terminal* door het commando :

> *ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew /install/-master/install)"*

Wanneer *Homebrew* is geïnstalleerd, kan het commando *brew* gebruikt worden. Voor React Native is het ook verplicht om *Node.js v4.0* of recenter te gaan installeren. Hiervoor is de *Node Version Manager of nvm* nodig. Deze wordt standaard geïnstalleerd wanneer XCode wordt geïnstalleerd. Om *Node.js* te installeren gebruik het commando :

> *nvm install node && nvm alias default node*

Deze zal de laatste nieuwe versie van *Node.js* installeren. *Node.js* is nodig om de applicatie te runnen op de simulator en zorgt er ook voor dat de Javascript ontwikkelaar tools beschikbaar worden. Om te voorkomen dat er fouten zijn tijdens het ontwikkelingsproces wanneer *Node.js* server draait en de code wordt aangepast, werd *Watchman* door Facebook ontwikkeld. *Watchman* gaat het project observeren en de nodige scripts oproepen wanneer de code wordt aangepast tijdens het ontwikkelingsproces. Installeer *Watchman* door gebruik te maken van het volgende commando :

brew install watchman

Optioneel raden de ingenieurs van Facebook aan om *Flow* te gaan gebruiken. *Flow* gaat type errors detecteren in Javascript programma's en werd door Facebook ontwikkeld. Het kan via het *Homebrew* commando worden geïnstalleerd :

brew install flow

Wanneer deze stappen zijn uitgevoerd, is de iOS omgeving klaar om React Native te installeren. Voor Android is de installatie van Android SDK uiteraard verreist. Wanneer ook de Android omgeving werd geconfigureerd volgens de stappen die Google voorziet in de documentatie, kan er ook een emulator van een Android compatibel toestel worden aangemaakt dat nodig is om de applicaties te laten draaien.

Het installeren van React Native gebeurt door het volgende commando :

npm install -g react-native-cli

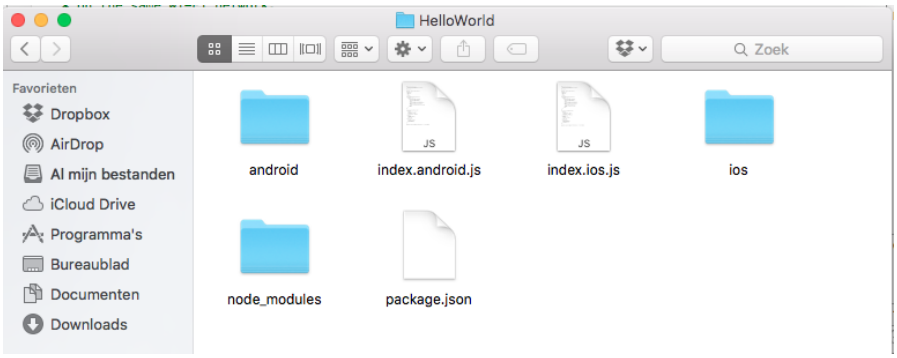
4.1.2 Windows en Linux

Ontwikkelen van applicaties met React Native kan in beperkte mate ook op Windows en Linux maar enkel voor Android-applicaties. Facebook besteedt hiervoor het beheer en de opvolging uit aan de community van React Native. De reden hiervoor is dat de ingenieurs bij Facebook gebruik maken van MAC OSX en dus krijgt de ondersteuning voor OSX prioriteit. Men gelooft bij Facebook dat de beste Linux en Windows ondersteuning dan ook enkel kan komen van mensen die dagelijks met deze OS werken. (Facebook, 2015c) Ook in deze scriptie wordt gebruik gemaakt van MAC OSX en zal verder niet meer verwezen worden naar het gebruik voor Windows of Linux.

4.1.3 Start React Native

Om een nieuw project te starten in React Native wordt volgende commando gebruikt :

react-native init HelloWorld



Figuur 4.1: Schermafbeelding map HelloWorld project

Dit zal een map plaatsen in het filesysteem met de naam *HelloWorld*. Deze map bevat reeds 2485 directories en 10660 files. Op Figuur 4.1 wordt de inhoud van de map weergegeven. De mappen *android* en *ios* bevatten boilerplate code relevant aan het platform, deze zijn ook de respectievelijke startpunten van de applicaties. In de *node_modules* map worden de *node* afhankelijkheden die werden geïstalleerd via *npm* opgeslagen. De documenten *index.ios.js* en *index.android.js* bevatten de React code die de views van de applicaties beschrijven, Code fragment 4.1. Wanneer in XCode het project uit de map *ios* wordt geopend en de iPhone emulator opgestart, start de applicatie en zal de view die in de *render* functie van de *HelloWorld* component weergegeven worden, Figuur 4.2. Hetzelfde kan met een Android emulator ook gebeuren door in de *terminal* naar de *android* map te gaan en daar het commando te gebruiken om de android applicatie te starten :

react-native run-android

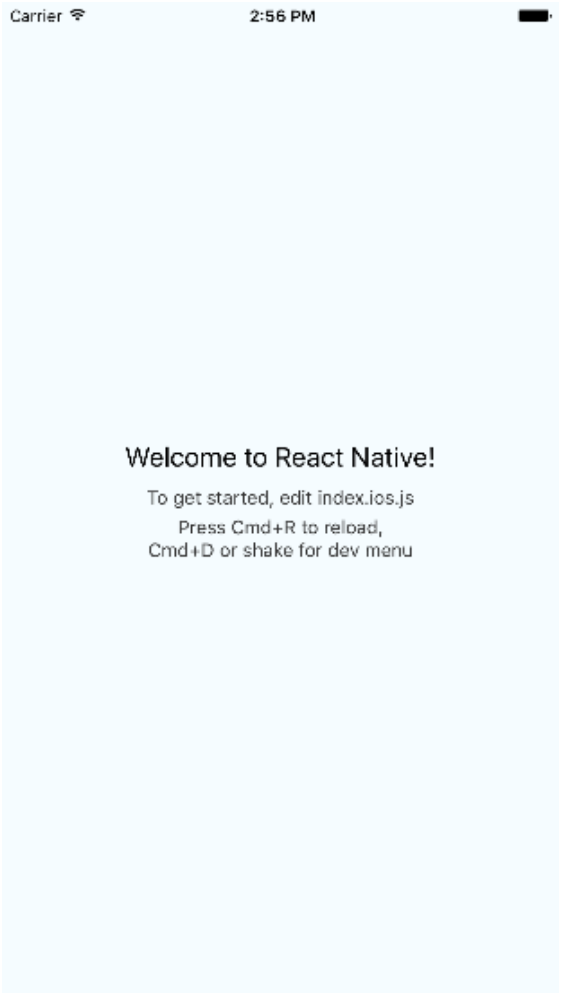
De applicaties kunnen nu aangepast worden in de *index.xxx.js* files. Door de emulators te laten herladen worden de wijzigingen zichtbaar. (Facebook, 2015b)

4.2 Componenten

UI Componenten vormen de basis van een applicatie in React Native en verwijzen naar platform specifieke elementen. Er wordt dan ook een onderscheid gemaakt tussen *cross-platform componenten* en *platform specifieke componenten* met elk hun eigen manier van werken. Het is ook gebruikelijk om componenten in eigen *.js* documenten te plaatsen om bepaalde principes van OO te kunnen gebruiken. Componenten kunnen herbruikt worden en het geheel blijft overzichtelijk, Figuur 4.3. Om een component te importeren uit een ander Javascript document moet *require* worden aangeroepen gevolgd door de locatie van het document, Code fragment 4.2.

```
/**
 * Sample React Native App
 * https://github.com/facebook/react-native
 */
'use strict';

var React = require('react-native');
var {
  AppRegistry,
  StyleSheet,
  Text,
  View,
} = React;

var HelloWorld = React.createClass({
  render: function() {
    return (
      <View style={styles.container}>
        <Text style={styles.welcome}>
          Welcome to React Native!
        </Text>
        <Text style={styles.instructions}>
          To get started, edit index.android.js
        </Text>
        <Text style={styles.instructions}>
          Shake or press menu button for dev menu
        </Text>
      </View>
    );
  }
});

var styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  welcome: {
    fontSize: 20,
    textAlign: 'center',
    margin: 10,
  },
  instructions: {
    textAlign: 'center',
    color: '#333333',
    marginBottom: 5,
  },
});

AppRegistry.registerComponent('HelloWorld', () => HelloWorld);
```

Code fragment 4.1: HelloWorld gegenereerde code



Figuur 4.2: HelloWorld op iOS simulator

```
var CameraRoll = require('./camera_roll_example');
var LocalImage = require('./local_image');

var App = React.createClass({

  render: function(){
    return (
        //Componenten kunnen gebruikt worden
        <CameraRoll />
        <LocalImage />
   )
  }

});
```

Code fragment 4.2: Importeren van componenten

```
MacBook-Pro-van-Steven:App Steven$ tree
.
├── Button
│   ├── index.js
│   └── style.js
├── Forecast
│   └── index.js
├── LocationButton
│   ├── index.js
│   └── style.js
├── PhotoBackdrop
│   ├── camera_roll_example.js
│   ├── index.js
│   ├── local_image.js
│   └── style.js
└── styles
    └── typography.js
```

Figuur 4.3: Mappen structuur voor applicatie

4.2.1 Cross-platform componenten

Cross-platform componenten zijn componenten die voor beide platform gelijkwaardig worden behandeld en worden als basis componenten aanzien.

Text

Een voorbeeld hiervan is het *<Text>* component. Zoals de naam doet vermoeden is dit een wrapper component die een tekst zal bevatten om af te printen op het scherm. Werken met tekst in React Native is eenvoudig maar verschillend in hoe het werkt in webontwikkeling. Het is mogelijk om daar tekst te plaatsen zonder een specifieke wrapper maar in native ontwikkeling is dit een verplichte tag om tekst te gaan gebruiken. In Code fragment 4.3 wordt een voorbeeld weergegeven hoe in React Native een *text* tag gebruikt wordt. In React Native bestaan er geen componenten zoals *<h1> of <h2>*, deze kunnen gemakkelijk nagebootst worden door een stijl klasse mee te geven en de *fontsize* aan te passen.(Facebook, 2015a)

Image

Het *text* component is het meest eenvoudige component in React Native gevolgd door het *image* component. Met *image* kan een afbeelding weergegeven worden op het scherm, dit kan op een aantal verschillende manieren. In Code fragment 4.4 wordt getoond hoe een afbeelding uit het file systeem van de Android of iOS *package* gebruikt kan worden. Deze moeten volgens de richtlijnen van respectievelijk Android en iOS ontwikkeling worden toegevoegd aan de correcte folders. In hetzelfde voorbeeld wordt ook getoond hoe een afbeelding kan gebruikt worden als achtergrond met bijvoorbeeld een tekst op de voorgrond. Dit wordt mogelijk gemaakt doordat React

```
1  //niet geldig
2  <View>
3        Hier is geen tekst toegelaten/
4     </View>
5  //geldig
6  <View>
7    <Text>
8      Hier is wel tekst toegelaten.
9    </Text>
10 <Text>
11   Ook dit is geldig met een {this.props.child}!
12 </Text>
13   <Text>
14       Of met een state object zoals  : {this.state.child}!
15     </Text>
16 </View>
```

Code fragment 4.3: Voorbeeld gebruik van Text

```
1  <Image source={require('image!imagefromfile')} >
2    <Text style={styles.h1}>titel h1</Text>
3        <Text style={styles.h2}>Ondertitel</Text>
4    </Image>
```

Code fragment 4.4: Voorbeeld gebruik van Image met require

Native werkt met componenten, en deze als een container kan gebruikt worden om andere componenten in te gaan wrappen.

Een andere manier om een afbeelding te gaan gebruiken, is om het te downloaden van het internet. Hiervoor wordt een uri meegegeven zoals in Code fragment 4.5. Let wel, wanneer een afbeelding wordt gedownload, moet er verplicht een dimensie meegegeven worden, anders wordt de afbeelding niet zichtbaar.(Facebook, 2015a)

ListView

Naast *text* en *image* wordt in native ontwikkeling heel veel gebruik gemaakt van lijsten die weergegeven worden dankzij een *ListView* component in React Native. In Code fragment 4.6 wordt een basis implementatie van een *ListView* getoond. Er zijn een aantal verplichtingen bij het gebruik van *ListView*, zo wordt in het voorbeeld in *getInitialState* eerst een *ListView.DataSource* object aangemaakt die een functie *rowHasChanged* meekrijgt. Deze zal als een Observer fungeren op de lijst en detecteren

```
1       <Image source={{uri: 'https://facebook.github.io/react/img/logo_og.png'
            }}
2       style={{width: 400, height: 400}}/>
```

Code fragment 4.5: Voorbeeld gebruik van Image met Uri

```
getInitialState: function() {
  var ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});
  return {
    dataSource: ds.cloneWithRows(['row 1', 'row 2']),
  };
},

render: function() {
  return (
    <ListView
      dataSource={this.state.dataSource}
      renderRow={(rowData) => <Text style={styles.h1}>{rowData}</Text>}
      />
    );
}
```

Code fragment 4.6: Voorbeeld gebruik van basis ListView

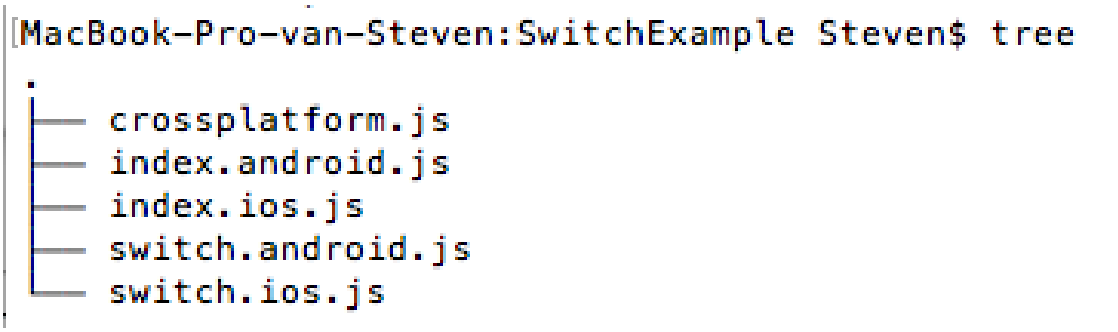
```
<TouchableHighlight
            onPressIn={this.state.pressing: 'Pressed IN'}
            onPressOut={this.state.pressing: 'Pressed OUT'}
            style={styles.touchable}>
                <View style={styles.button}>
                    <Text style={styles.welcome}>
                        {this.state.pressing}
                    </Text>
                </View>
</TouchableHighlight>
```

Code fragment 4.7: Voorbeeld gebruik van basis TouchableHighLight

wanneer een lijstitem moet aangepast worden. Daarnaast wordt een *state* object, *dataSource* geïnitialiseerd met het *ListView.DataSource* object met een array in de *cloneWithRows* functie. In de *render* functie wordt de *ListView* vervolgens opgevuld met een *dataSource* die uit de state gehaald wordt en wordt in *renderRow* aangegeven welk item er rij per rij wordt weergegeven.(Facebook, 2015a)

Touch

Een van de basis componenten die men kan gebruiken om een *TouchEvent* op te vangen, is het *TouchableHighlight* component. Deze kan rond een *View* of andere component worden geplaatst en zal één of meer van volgende methoden moeten bevatten : *onPressIn, onPressOut, onLongPress, onPress, ...* . Het gebruik ervan is simpel en wordt getoond in Code fragment 4.7. In dit voorbeeld zal de tekst die verschijnt op de knop wijzigen wanneer een bepaald drukbeweging wordt uitgevoerd.(Facebook, 2015a)



Figuur 4.4: Map structuur van SwitchExample

4.2.2 Platform specifieke componenten

Platform specifieke componenten kunnen herkend worden aan de prefix, meestal wordt het platform aangegeven als deel van de naam van het component. Platformspecifieke componenten zijn heel vaak nodig omdat niet alle componenten eenzelfde tegenhanger hebben op elk platform. Het kan ook zijn dat een bepaald component wel cross-platform is maar een bepaalde *property* niet. Zo kan je in iOS op component *TextInput*, *property : maxLength* gebruiken en op Android niet. Dit wordt in de API van React Native weergegeven door een label. (Facebook, 2015a).

Wanneer platformspecifieke componenten worden aangemaakt, moet er nauw opgelet worden dat deze in de correcte file worden geïmporteerd, zoniet zal de applicatie stoppen met werken. Componenten *<SwitchAndroid> en <SwitchIOS>* zijn 2 componenten die in elk platform specifiek moeten worden aangemaakt, op Figuur 4.4 wordt de file structuur afgebeeld. Hier wordt een onderscheid gemaakt tussen *switch.android.js*, Code fragment 4.8 en *switch.ios.js*, Code fragment 4.9 welke respectievelijk een *switch* component aanmaken voor Android en iOS. In *crossPlatform.js*, Code fragment 4.10 wordt de *switch* geïmporteerd, merk op dat er geen *.android of .ios* na *switch* wordt geplaatst. De native compiler van React Native gaat dit voor de ontwikkelaar bepalen welke *switch* wordt ingeladen afhankelijk van het platform.

Switch is een voorbeeld van een component die zowel voor Android als voor iOS kan gebruikt worden op dezelfde manier en toch verschillend geïmplementeerd worden door de *native compiler* van React Native. Sommige componenten zijn uiteraard enkel op een specifiek platform te verkrijgen, zoals de *<TabBarIOS>* die een navigatie item onderaan de applicatie plaatst, Figuur 4.5. Voor Android kan de ontwikkelaar in dit geval kiezen voor een *<ToolBarAndroid>* component of voor een *<DrawerLayoutAndroid>* component. Deze werkwijze zorgt ervoor dat sommige code niet herbruikbaar is en dat de ontwikkelaar goed moet opletten dat de code van de platform-specifieke componenten gescheiden blijft.

```
var React = require('react-native');
var { SwitchAndroid } = React;

var Switch = React.createClass({
  getInitialState() {
    return {value: false};
  },

  _onValueChange(value) {
    this.setState({value: value});
    if (this.props.onValueChange) {
      this.props.onValueChange(value);
    }
  },

  render() {
    return (
      <SwitchAndroid
        onValueChange={this._onValueChange}
        value={this.state.value}/>
      );
  }
});

module.exports = Switch;
```

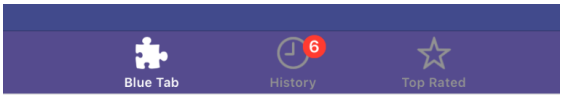
Code fragment 4.8: Switch voorbeeld specifiek voor Android

```
var React = require('react-native');
var { SwitchIOS } = React;

var Switch = React.createClass({
  getInitialState() {
    return {value: false};
  },

  _onValueChange(value) {
    this.setState({value: value});
    if (this.props.onValueChange) {
      this.props.onValueChange(value);
    }
  },

  render() {
    return (
      <SwitchIOS
        onValueChange={this._onValueChange}
        value={this.state.value}/>
      );
  }
});

module.exports = Switch;
```

Code fragment 4.9: Switch voorbeeld specifiek voor iOS

```
1
2  var Switch = require('./switch');
3
4  var CrossPlatform = React.createClass({
5
6    render: function() {
7      return (
8        <View>
9
10         <Switch onValueChange={//do something}/>
11       </View>
12     );
13   }
14 });
```

Code fragment 4.10: Crossplatform import Switch



Figuur 4.5: TabBarIOS voorbeeld

Native UI componenten

Er zijn momenteel heel wat native UI componenten aanwezig in het React Native platform en ook als *third-party libraries* aangeboden door leden uit de community. Maar soms kunnen ontwikkelaars eigen UI componenten ontwikkelen die applicatie- of bedrijfs specifiek zijn, dan biedt React Native nog een ander alternatief aan, namelijk het *custom* component zelf aanmaken in Objective C of Java en exporteren naar Javascript. Dit kan door een standaard Objective C of Java object aan te maken die een *View* voorstelt. De Java klasse zal moeten overerven van *SimpleViewManager* en de Objective C klasse van *RTCViewManager*, om de export mogelijk te maken. Methoden en properties worden geannoteerd volgens de annotaties voorzien in de superklasse, zodat ze kunnen gebruikt worden in Javascript. Wanneer de ontwikkelaar zijn componenten wenst te gebruiken, hoeft hij die enkel nog te importeren. De brug van React Native voorziet hiervoor een aantal klassen die dit voor de ontwikkelaar gaat opvangen, aan de hand van de annotaties die werden geplaatst.

Wanneer is het een goed idee om native Objective-C of Java code te gaan gebruiken?

- Wanneer er reeds een bestaande native applicatie is, kan het nuttig zijn bepaalde modules te exporteren om zo dubbel werk te voorkomen.
- Wanneer de applicatie gespecialiseerde taken met een hoog CPU gebruik moet gaan uitvoeren, wordt het aangeraden om deze in de Objective-C of Java te laten uitvoeren en het resultaat ervan naar de Javascript view doorgegeven.

- En wanneer er tenslotte nog geen ondersteuning is voor een platform specifiek component of API. React Native is momenteel in actieve ontwikkeling en ook Android en iOS zullen blijven innoveren.

4.3 API

Het grote voordeel van native ontwikkeling is het gebruik van de hardware componenten van het toestel waarop de applicatie draait, hiervoor zal de ontwikkelaar gebruik willen maken van de verschillende API's die voor het platform worden aangeboden. Die API's worden toegankelijk voor React Native door modules (reeds aanwezig in de *node_modules* map) die de ontwikkelaar voorzien van Javascript interfaces die de API's asynchroon gaan aanspreken. Uiteraard zijn er ook API modules die cross-platform zijn en die platform specifiek zijn. Door de complexiteit van sommige API's heeft React Native niet altijd alle functionaliteiten in het platform geïntegreerd, sommige onderdelen worden door de community aangereikt en in sommige gevallen zal de ontwikkelaar de modules zelf moeten ontwikkelen in Objective-C of Java. Het spreekt dan ook voor zich dat er zowel cross-platform API modules bestaan als platform specifieke API modules.

4.3.1 Cross-platform API modules

De lijn tussen cross-platform en platformspecifieke API modules is erg dun. Een module die nu momenteel slechts op één platform beschikbaar is, kan binnen enkele weken (of zelfs dagen) cross-platform gemaakt worden aangezien Facebook en de community nog steeds actief bezig zijn om het platform te verbeteren.

AsyncStorage

Een eerste en belangrijke module is de *AsyncStorage*, een simpel persistentie systeem die over de globale applicatie gebruikt kan worden. *AsyncStorage* kan vergeleken worden met *LocalStorage* die gebruikt wordt in webontwikkeling. Aangezien de werkwijze van *AsyncStorage* gelijkaardig is met die van *LocalStorage* kan een webontwikkelaar hieraan snel gewoon raken. De data wordt via een Javascript systeem opgeslagen in het intern geheugen als json-object. In Code fragment 4.11 en Code fragment 4.12 wordt getoond hoe data wordt opgeslagen en terug opgevraagd aan de hand van de *key*. Dit gebeurt volledig asynchroon waardoor de applicatie niet nodeloos belast wordt.

PanGesture

Naast data opslaan, kan het voor een ontwikkelaar interessant zijn om gebruik te maken van de aanrakingsmogelijkheden van smartphones en tablets. Hiervoor heeft

```
1    saveData: function(value) {
2        AsyncStorage.setItem("myKey", value);
3    }
```

Code fragment 4.11: AsyncStorage: opslaan van data

```
1        AsyncStorage.getItem(myKey).then((value) = {
2            this.setState({myKey value});
3        }).done();
```

Code fragment 4.12: AsyncStorage: opvragen van data

React Native een *PanGesture* module ontwikkeld. Deze gaat de aanrakingen van de gebruiker kunnen behandelen en laten resulteren in gepaste wijzigingen op het scherm. Omdat aanrakingen op mobiel anders zijn dan die op het web voorziet React Native een interface voor *PanGesture* die de methoden nodig voor de implementatie onmiddellijk meegeeft, deze kunnen door de ontwikkelaar overschreven worden om het gewenste resultaat te bekomen. In Code fragment 4.13 wordt de *_handlePanResponderMove* overschreven, in dit voorbeeld zal een cirkel, wanneer die versleept wordt door de gebruiker, van plaats veranderen. Dankzij het *gestureState* object die de locatie van de aanraking op het scherm bijhoudt, kan de locatie van de cirkel worden aangepast in de update functie.

4.3.2 Platform specifieke API modules

Uiteraard zijn de meeste API modules platform specifiek, dit komt doordat Apple en Google andere functionaliteiten in hun platform integreren. Zo heeft Android bijvoorbeeld een *Toast* API die de gebruiker onderaan het scherm een bericht kan geven. Hiervoor kan bij React Native de *ToastAndroid* module gebruikt worden, deze kan op eenvoudige wijze worden aangesproken, Code fragment 4.14. Deze code zal bijvoorbeeld na een *onPress()* functie van een knop een toast op het scherm afbeelden, Figuur 4.6. Hetzelfde kan voor iOS door middel van bijvoorbeeld een *AlertIOS* te implementeren. Bij Code fragment 4.15 zal de *onPress* functie een standaard iOS *alert* venster op het scherm afbeelden, Figuur 4.7. In dit geval gebeurt dat met een *default button* die het venster opnieuw terug afsluit. Wanneer er aan de *alert* een array met een

```
1   _handlePanResponderMove: function(e: Object, gestureState: Object) {
2     this._circleStyles.style.left = this._previousLeft + gestureState.dx;
3     this._circleStyles.style.top = this._previousTop + gestureState.dy;
4     this._updatePosition();
5   }
```

Code fragment 4.13: PanGesture voorbeeld

```
1 ToastAndroid.show('Hello, this is a toast', ToastAndroid.LONG)
```

Code fragment 4.14: ToastAndroid voorbeeld

```
1   onPress={() =>AlertIOS.alert(
2             'Alert title',
3             'Alert message, with a default button!!'
4           )}>
```

Code fragment 4.15: AlertIOS voorbeeld

reeks van objecten, die een title en een *onPress* functie bevatten, wordt meegegeven, zal op basis hiervan een reeks knoppen aan het *alert* venster worden meegegeven.

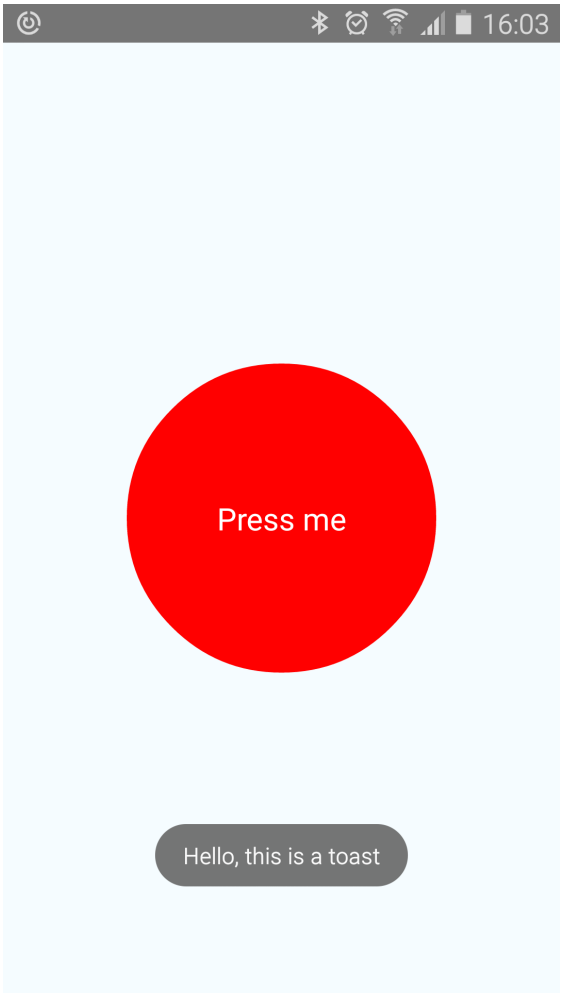
4.4 Debuggen en testen

React Native werd ontwikkeld om webontwikkelaars de kans te geven om native applicaties te schrijven, Facebook heeft er dan ook voor gezorgd dat het debuggen van de applicatie lijkt op het debuggen van een webapplicatie. Hiervoor zijn een aantal mogelijkheden, de combinatie van meerdere debug manieren is dan ook aangeraden net zoals in webontwikkeling.

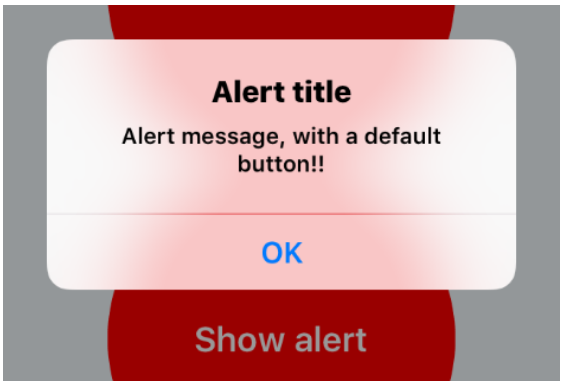
4.4.1 Ontwikkelaars opties

Wanneer een applicatie wordt afgespeeld op een simulator of op een echt toestel dan kan er een menu worden aangesproken die een aantal opties geeft aan de ontwikkelaar, Figuur 4.8. In dit menu kan de ontwikkelaar elementen inspecteren, net zoals dat kan in de chrome of safari ontwikkelingstools, de applicatie opnieuw laten inladen of de chrome Javascript debug tool gebruiken om breakpoints te detecteren. Wanneer de applicatie om een bepaalde reden vastloopt is er een *Red Screen of Death* te zien voor de ontwikkelaar, dit scherm geeft de reden van het vastlopen van de applicatie op een betekenis volle manier terug zoals op Figuur 4.9.

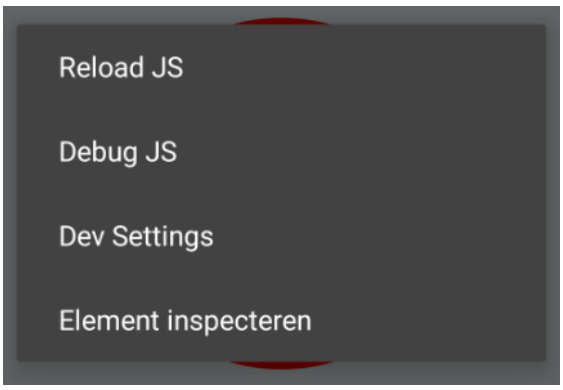
Naast de mogelijkheid om te debuggen in de applicatie zelf, kunnen de chrome en safari development tools ook gebruikt worden om de applicatie te debuggen. Aangezien de applicatie draait op een *node.js* server kan de ontwikkelaar via *http://localhost:8081* de applicatie debuggen in de browser. Er is ook geen enkele aanpassing nodig om te werken met *console.log* waarvan de output in de Javascript console wordt weergegeven. Maar ook in Xcode zal de output van *console.log* zichtbaar zijn voor de iOS applicatie. Voor de Android applicatie kan de ontwikkelaar *logcat* activeren door in de hoofdmap van het project volgend commando te activeren :



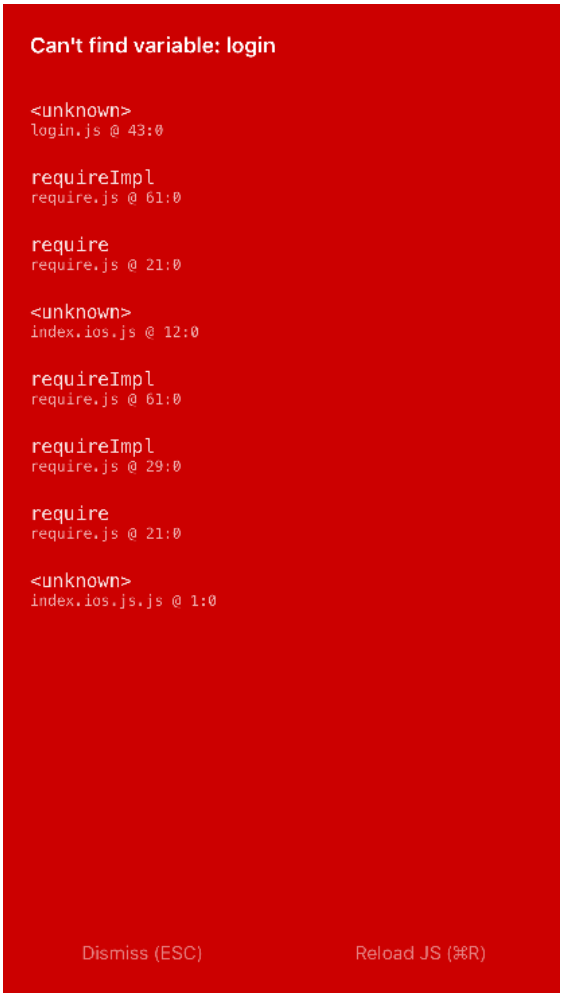
Figuur 4.6: ToastAndroid schermafbeelding



Figuur 4.7: AlertIOS schermafbeelding



Figuur 4.8: In-app ontwikkelaars opties



Figuur 4.9: The Red Screen of Death

adb logcat

De logs zullen in de *terminal* verschijnen. Deze laatste optie voor Android geeft een onoverzichtelijk geheel van logs terug omdat de logs van platform-specifieke zaken hier ook worden weergegeven. (Eisenman, 2015)

Het is ook mogelijk gemaakt om, net zoals bij webontwikkeling, CSS componenten te gaan wijzigen wanneer de applicatie loopt en dit rechtstreeks in de browser, hiervoor hoeft de ontwikkelaar enkel de React plugin te downloaden voor chrome of safari. Deze functie is voor React Native momenteel nog in opbouw waardoor sommige functionaliteiten hiervoor nog niet helemaal of volledig niet werken.

4.4.2 Testen

Debuggen van de applicatie wil zeggen dat de ontwikkelaar een fout in zijn code zal moeten zoeken en oplossen, het is in applicatie ontwikkeling dan ook aan te raden om te voorkomen dat bepaalde *bugs* voorkomen. Dit kan door testen te schrijven. Veel van de React Native code die geschreven wordt, is niet altijd code die uitsluitend gebruikt wordt voor mobiele applicatie ontwikkeling. Veel van de business logic code kan geïsoleerd worden van de render code. Op deze manier kan men Javascript code testen door gebruik te maken van om het even welke Javascript development tool.

React Native raadt een aantal tools aan voor ontwikkelaars die door de ingenieurs van Facebook zelf gebruikt (of ontwikkeld) zijn tijdens het ontwerpen van React Native.

Flow

Flow is een Javascript library voor *static-type checking*. *Flow* gaat de ontwikkelaar helpen met het vinden van errors op gebied van type interferentie. Wanneer een variabele een string object meekrijgt en er later een bewerking op gebeurt zoals op een nummer type, zal *Flow* de ontwikkelaar waarschuwen. Om een check te doen van een React Native project hoeft de ontwikkelaar in de *command line* volgend commando in te geven :

flow check

De check zal op gepaste wijze aangeven of er al dan niet errors gevonden werden.

Jest

React Native ondersteunt testen van React componenten door middel van *Jest*, een unit testing framework gebouwd op *Jasmine*. Het voorziet aggresieve automocking van afhankelijkheden, en werkt perfect samen met de ontwikkelaars tools van React Native. Om *Jest* te gebruiken moet dit uiteraard eerst geïnstalleerd worden :

```
{
...
    "scripts": {
        "test": "jest"
        }
...
}
```

Code fragment 4.16: Voeg test script toe aan package.json

```
'use strict';
describe('test', function() {
    it('expects true to be true', function() {
        expect(true).toBe(true);
    });
});
```

Code fragment 4.17: Voorbeeld van Jest test

npm install jes-cli –save-dev

Daarna kan de *package.json* aangepast worden door een test script toe te voegen, Code fragment 4.16. Wanneer *npm test* wordt uitgevoerd zal deze met behulp van *Jest* de tests doorlopen. In de hoofdmap moet een map komen, genaamd *tests*. *Jest* zal recursief opzoek gaan naar test documenten in *tests/* map. Code fragment 4.17 geeft een voorbeeld van een simpele test case.

7

Hoofdstuk 5

Conclusie

Toen ik voor het eerst over het concept van React Native leerde, stond ik er heel sceptisch tegenover, het was Javascript ontwikkeling maar toch geen Hybride applicatie ontwikkeling. Ik kon mij hier moeilijk bij neerleggen, mijn ervaringen met hybride ontwikkeling zijn heel kort geweest en ik deed de applicaties hierin vaak af als minderwaardige applicaties die zich willen voordoen als een echte native applicatie. Ik ben steeds van het idee geweest dat een echte native applicatie telkens in de taal wordt geschreven die de aanbieders van een OS voorleggen en die zij ondersteunen. Groot was echter mijn verbazing toen ik voor het eerst met React Native een applicatie ging ontwikkelen, want het voelde echt native aan en was het ook.
In mijn onderzoek ben ik uiteraard eerst met React voor webontwikkeling begonnen om de syntax onder de knie te krijgen. Ik heb nooit echt graag met Javascript gewerkt waardoor ik bevooroordeeld was ten opzichte van React. Toch was het een aangename ervaring en het maken van een webapplicatie ging enorm vlot. Omdat React een verkorte versie is van Javascript heb ik de syntax heel snel onder de knie gekregen. Het is natuurlijk ook zo dat als een ontwikkelaar in React codeert hij enkel rekening moet houden met de view, waardoor het vaak kinderspel lijkt om een applicatie te bouwen. React is een goede manier om de view van een webapplicatie op te bouwen, maar voor server-side ontwikkeling zal er steeds een back-end programmeur nodig zijn.
In al mijn enthousiasme ben ik aan React Native begonnen, de stukken die werden onderzocht in hoofdstuk 4 resulteren in een gebundelde applicatie voor zowel iOS als voor Android. In het begin lag mijn focus op iOS, door mijn ervaring met Android en vooral Java ontwikkeling was ik heel gerust dat die stukken sneller zouden verwerkt worden. Het ontwikkelen voor iOS was enorm aangenaam en verliep heel vlot, de integratie met Xcode is volledig uitgewerkt door Facebook en hier heb ik geen problemen ondervonden. Het debuggen in chrome en in Xcode was zeer duidelijk en ik verstond onmiddellijk waarom mijn applicatie niet wou verdergaan. De applicatie liep heel vlot en ik kon het verschil niet merken met een applicatie die ik in Objective-C of Swift zou geschreven hebben. Toch is de manier om stijl te geven aan componenten niet

echt een verbetering. Een webontwikkelaar is natuurlijk vertrouwd met CSS en zal dit waarschijnlijk sneller onder de knie krijgen als een mobiele applicatie ontwikkelaar. Ik miste toch een visuele tool om schermen op te bouwen zoals in Xcode voor iOS ontwikkeling of in Android studios voor Android. Ook is het zo dat de standaard schermopbouw voor IPhone of IPad applicaties, zoals aangeleverd door Apple, in native iOS ontwikkeling wordt voorzien voor de ontwikkelaar. Bij React Native is dit een moeilijker gegeven, je zou als ontwikkelaar de schermpbouw volledig zelf met stijlcomponenten moeten nabouwen om de standaard na te bootsen. Anderzijds, heb je als ontwikkelaar hierdoor veel meer vrijheid.
Nadat ik een werkende applicatie van 1 pagina had, heb ik geprobeerd om die ook op Android te laten draaien. Ik maakte gebruik van een Genymotion emulator om mijn applicatie te laten draaien, dit volledige proces liep niet zo heel vlot. Volgens de richtlijnen zou door middel van een aantal commando's de applicatie moeten starten op de emulator. Dit ging niet onmiddellijk en ik heb via Stackoverflow een aantal configuraties moeten uitvoeren die Facebook niet had vermeld. Na een aantal keer proberen, is dit dan toch gelukt. Net als bij iOS was ik ook hier overtuigd van het resultaat en had ik dezelfde bemerkingen op gebied van stijlcomponenten. De ondersteuning van Android voor React Native is nog heel jong en dat merk je als ontwikkelaar. Er zijn heel wat componenten en API's die nog niet geïntegreerd werden in React Native en je moet goed zoeken om een antwoord te vinden op bepaalde problemen. Uiteraard wordt hier door Facebook en de community nog volop aan gewerkt.
Toch zou ik alle bedrijven die zich met mobiele applicatie ontwikkeling bezighouden React Native aanbevelen. Er zijn inderdaad voorlopig nog een aantal mindere punten, maar de voordelen van React Native geven toch de doorslag. React is een Javascript library waarvan de syntax en de concepten heel snel aangeleerd zijn, zo kan iemand die Javascript kan coderen met een minimum aan inspanning onmiddellijk beginnen met React. Wanneer een ontwikkelaar React onder de knie heeft, is de stap naar React Native een kleine moeite, want veel van de concepten en werkwijzen worden hierin overgenomen. Een native applicatie ontwikkelaar zal dus met een kleine inspanning om React aan te leren, direct applicaties kunnen bouwen voor iOS en Android. Anderzijds geldt voor een webontwikkelaar dat wanneer hij vertrouwd geraakt met concepten van native applicatie ontwikkeling hij ook productief kan worden ingezet in native ontwikkeling, dankzij React Native. De opleiding van bekwame mensen is dus geen grote financiële kost voor een bedrijf maar een investering die snel rendeert.
Daarnaast zijn er ook de voordelen om het ontwikkelingsproces te versnellen, zoals code herbruikbaarheid op zowel Android als iOS maar ook de mogelijkheid van onmiddellijk herladen van de applicatie zorgt voor tijdswinst. Doordat een React Native applicatie in Javascript geschreven wordt, zal een update van een applicatie die al in de applicatie stores aanwezig is onmiddellijk doorgevoerd worden. Dit zal de tevredenheid van de klant van de applicatie verhogen. Er is nog steeds geen ondersteuning voor Windows

Phone en er is voorlopig ook geen enkele indicatie dat dit ook in de nabije toekomst zal gebeuren. Het is ook een nadeel maar omdat het grootste deel van smartphone gebruikers met Android of iOS werkt, weegt dit niet heel zwaar door. Ik heb er ook alle vertrouwen in dat Facebook de ondersteuning voor Windows Phone alsnog zal integreren in React Native.

Hoofdstuk 6

Lijst van afkortingen en woorden

Afkortingen

HTML	Hyper Text Markup Language
UI	User Interface
MVC	Model-View-Controller
OS	Operating System
CSS	Cascading Style Sheets
XML	Extensible Markup Language
API	Application Programming Interface
JSX	JavaScript Syntax Extension
DOM	Document Object Model
SDK	Software Development Kit
OO	Object Oriented
json	JavaScript Object Notation

Woorden

Script	lijst van instructies die worden uitgevoerd door een computer of computer programma.
Template	Een sjabloon voor een stuk code.
Two-way data binding	Bi-directionele wijzigingen in het model als in de UI.
Flexbox	CSS model voor responsive design van websites.
Immutable	Niet wijzigbaar.
Emulator	Programma dat hardware of software nabootst.
Boilerplate code	Code die op verschillende plaatsen voorkomt om hetzelfde te realiseren en vaak voorgeprogrammeerd wordt.
Package	Bundeling van klassen.
Observer	Ontwerppatroon om toestandsveranderingen efficiënt te detecteren.
Callback functie	Functie die wordt meegegeven aan een andere functie en pas zal worden uitgevoerd wanneer de eerste functie volledig is doorlopen.
Breakpoint	Opzettelijk stopzetten van een applicatie om te debuggen.

Bibliografie

Chedeau, C. (2015). Keynote at react-europe 2015. Geraadpleegd op 19 november 2015 via https://www.youtube.com/watch?v=PAA9O4E1IM4.

DocForge (2015). Framework. Geraadpleegd op 23 november 2015 via http://docforge.com/wiki/framework.

Eisenman, B. (2015). *Learning React Native: building native mobile apps with Javascript*. Sebastopol, CA , USA: O'Reilly Media, Inc., 1st ed.

Facebook (2015a). Documentation on react native. Geraadpleegd op 23 november 2015 via http://facebook.github.io/react-native/docs.

Facebook (2015b). Getting started. Geraadpleegd op 23 november 2015 via https://facebook.github.io/react-native/docs/getting-started.html.

Facebook (2015c). Linux and windows support. Geraadpleegd op 12 november 2015 via https://facebook.github.io/react-native/docs/linux-windows-support.html.

Freed, T. (2015). What is virtual dom? Geraadpleegd op 04 november 2015 via http://tonyfreed.com/blog/what_is_virtual_dom.

Gackenheimer, C. (2015). *Introduction to React: using react to build scalable and efficient user interfaces*. Apress, 1st ed.

Google (2015). Google trends. Geraadpleegd op 04 november 2015 via https://google.be/trends.

Harrington, C. (2015). React vs angularjs vs knockoutjs : a performance test. Geraadpleegd op 19 oktober 2015 via https://www.codementor.io/blog/reactjs-vs-angular-js-performance-comparison-knockout.

IDC (2015). Smartphone os market share. Geraadpleegd op 23 oktober 2015 via http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

Occhino, T. (2015). React native: Bringing modern web techniques to mobile. Geraadpleegd op 23 november 2015 via https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/.

Shilling, M. (2015). An ios developer on react native. Geraadpleegd op 06 oktober 2015 via https://medium.com/ios-os-x-development/an-ios-developer-on-react-native-1f24786c29f0#.f12aqtlqr.

StackOverflow (2015). Stackoverflow data exchange. Geraadpleegd op 04 november 2015 via https://data.stackexchange.com/stackoverflow/queries.

Vijayweb Solutions, I. P. L. (2014). Why you should avoid using javascript in your websites? Geraadpleegd op 23 november 2015 via http://www.vijaywebsolutions.com/why-you-should-avoid-using-javascript-in-your-websites/.

Witte, D., & von Weitershausen, P. (2015). React native for android: How we built the first cross-platform react native app. Geraadpleegd op 13 november 2015 via https://code.facebook.com/posts/1189117404435352/react-native-for-android-how-we-built-the-first-cross-platform-react-native-app/.