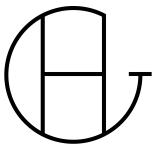


HoGent

Faculteit Bedrijf en Organisatie

Steven zijn witty titel

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

Instelling: HoGent

Academiejaar: 2015-2016

Eerste examenperiode

Faculteit Bedrijf en Organisatie

Steven zijn witty titel

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

Instelling: HoGent

Academiejaar: 2015-2016

Eerste examenperiode

Samenvatting

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Voorwoord

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

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

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.

Lijst van code fragmenten

Hoofdstuk 1

Het huidige JavaScript landschap

1.1 Wat is JavaScript?

JavaScript is een dynamische programmeer taal 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 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 onder heven aan een test door deze functie, er wordt getest of de waarde overeenkomt met het opgegeven patroon van hoe een e-mail adres 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. Dankzij JavaScript wordt dit onmiddellijk weergegeven en wordt de server niet onnodig belast met foutieve aanvragen.

```
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

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 bied om de voor gedefinieerde tekst of afbeeldingen te gaan wijzigen zonder dat de pagina herladen moet 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 van afbeelding wijzigen, enz.. .

1.4 De beperkingen van JavaScript

JavaScript wordt niet beschouwd als een volwaardige programmeertaal omdat het een aantal belangrijke onderdelen ontbreekt :

- Client-side JavaScript is niet instaat om bestanden te lezen, 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 gaan herhalen om bepaalde zaken uit te voeren in JavaScript. De bedoeling van een framework of library is dan ook om code herbruikbaar te maken en het voor de ontwikkelaar makkelijker te maken een goede

```
<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

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 mogelijkheid bieden 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 zorgen 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 applicatie geeft dit geen merkbare problemen voor de performantie van de applicatie, 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.

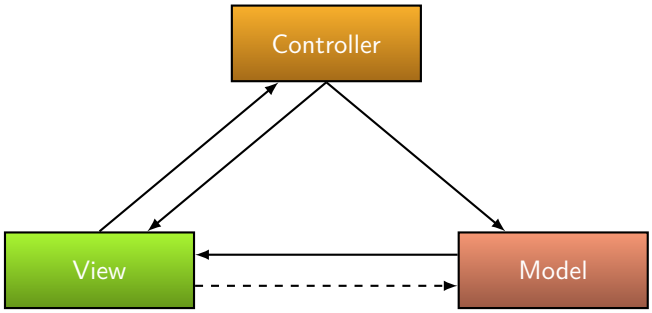
React werd dus ontwikkeld om één welbepaald probleem te gaan oplossen, namelijk het tonen van data in de UI. Als u denkt dat dat probleem reeds werd opgelost, bent u zeker niet verkeerd. Het verschil is dat React werd gecreëerd om grote schaalbare user interfaces, zoals die van Facebook en Instagram, met veel data die continu wijzigt te gaan behandelen. Dit soort interfaces kan gecreëerd worden en de data behandeling kan gebeuren via verschillende andere 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 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 Web ontwikkeling. Toen React in mei 2013 op de markt werd gebracht was er heel wat commotie in de Web ontwikkeling 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 mentaliteits wijziging doorgevoerd op gebied van web-ontwikkeling 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 kennis niveaus. (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 frame-



Figuur 2.1: MVC basis architectuur

works 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.

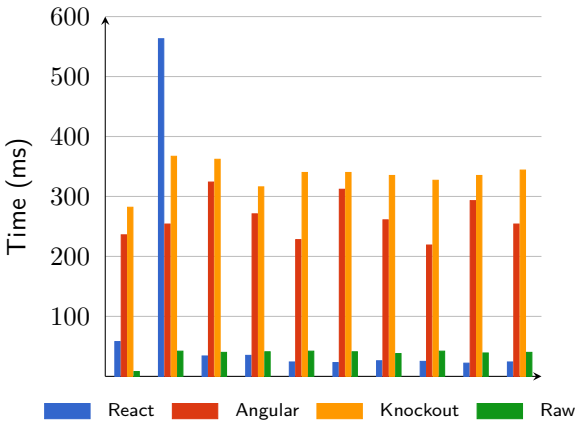
Resultaten:

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.

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

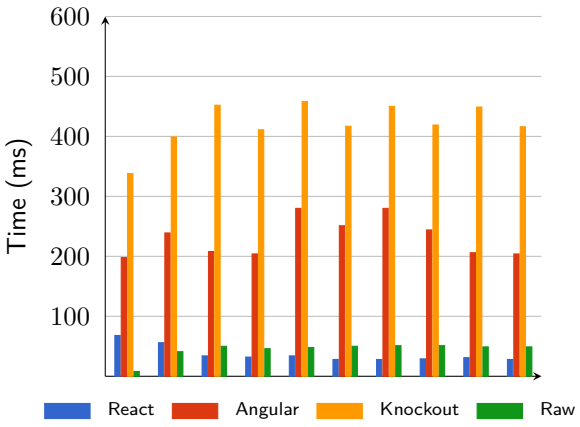
Tabel 2.1: Performance Comparison for React, Angular and Knockout, by Chris Harrington

	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



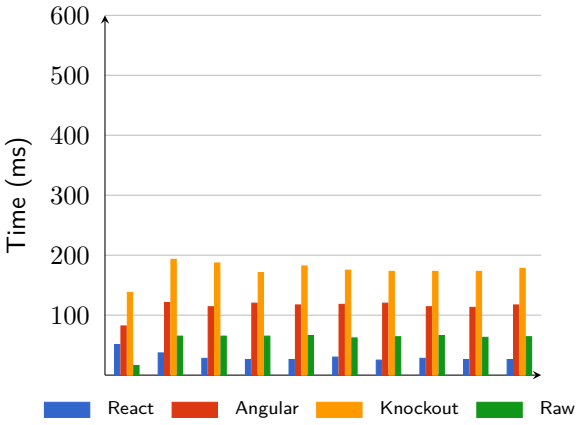
Figuur 2.2: Resultaat van performance test door Chrome 39.0.2171.95

	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: Resultaat van performance test door Chris 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: Resultaat van performance test door Chris 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

2.3 React concepten

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

2.3.1 Componenten

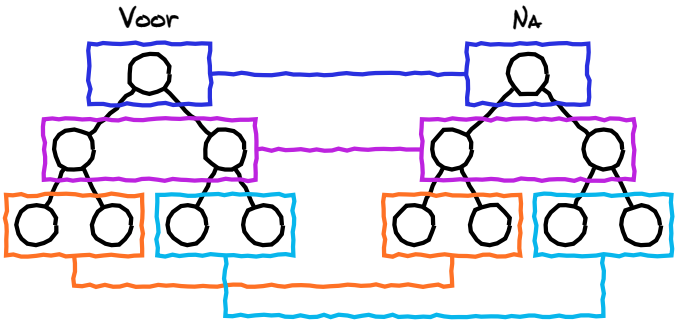
Componenten zijn de basis van React en de view van de applicatie. In Code fragment 2.1 wordt een voorbeeld gegeven van een component in React. Let goed op bij de *render* functie, deze geeft een HTML tag terug. Wanneer deze zal aangeroepen worden zal die de tekst "Hello world", laten verschijnen op de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

2.3.2 Virtual DOM

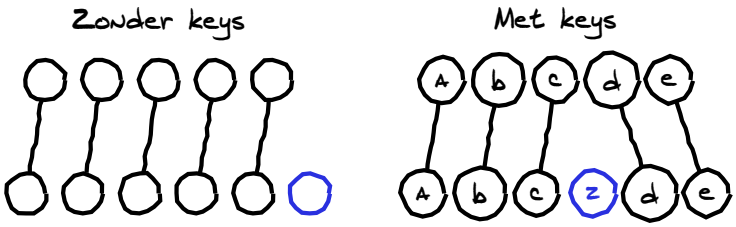
Dit is waarschijnlijk het meest belangrijke concept van React. Het virtuele DOM is de oplossing voor het probleem die het DOM stelt, namelijk dat het (nog) niet aangepast werd voor het weergeven van dynamische data. Met JavaScript en JQuery kon je als ontwikkelaar wel een oplossing voor dit probleem 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 gaat uiteraard het DOM gaan behandelen maar doet dit 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

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 omdat het zeer zeldzaam is in web applicaties dat componenten verplaatst worden naar een ander niveau in de boom structuur. Zie Figuur 2.5.

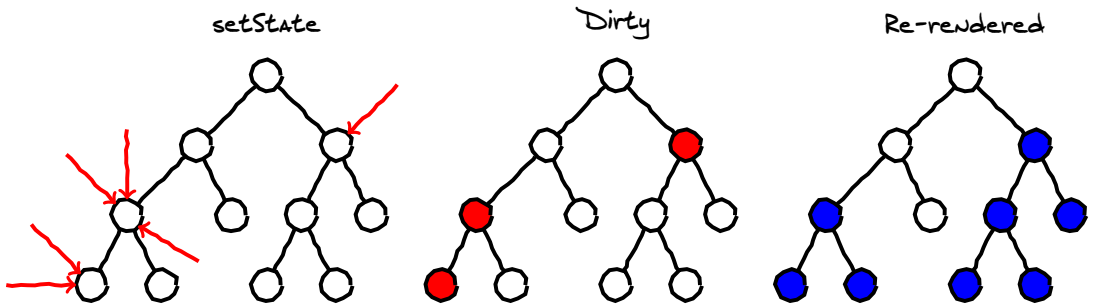
- Lijsten: Wanneer een component bij een éénmalige iteratie 5 andere componenten bevat wordt het moeilijk om in het midden van die lijst een nieuwe component toe te voegen. React gaat standaard een associatie maken tussen het eerste component van de eerste lijst en het eerste component van de tweede lijst, etc. Als ontwikkelaar kan je React helpen door een sleutel attribuut 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 genaamd "event delegation". Bij *event delegation* wordt een *event* getriggerd op de parent node en niet op elke child node. Bijvoorbeeld, een *Onclick listener* op een niet gesorteert 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, er slechts 1 moment is dat het DOM wordt ge-update. 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.



Figuur 2.7: Flow setState to render DOM

```
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

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

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 wanneer je als ontwikkelaar de logische volgorde volgt die nodig is om een applicatie te bouwen in React, je dit allemaal standaard krijgt en zelf hiervoor niets hoeft te doen.

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 renderen in JavaScript. Let op! Dit is geen verplicht onderdeel van React maar wordt sterk aangeraden om het te gebruiken. JSX maakt coderen veel eenvoudiger en leesbaarder, zo kan bijvoorbeeld HTML tags meegeven en aanvaard het aangepaste React klassen.

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.

```
1 var MyComponent = React.createClass({
2
3   render: function() {
4     return (<div>Hello world</div>);
5   }
6
7 });
8 React.render( <MyComponent / > , document.getElementById('container'));
```

Code fragment 2.3: CreateClass voorbeeld ReactJS

State

State is een set die geïnitialiseerd kan 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 wordt.

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 **??** 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 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 gaan we volgende 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

```
var MyComponent = React.createClass({

  render: function() {
    return (
      <div>
        {this.props.name}
      </div>)
  }
});

React.render(<MyComponent name= "Steven" / >, document.getElementById('container'));
```

Code fragment 2.4: Property aan een component toevoegen

```
var myComponent = React.createClass({

    render: function() {
      React.Children.map(this.props.children, function (child) {
        console.log(child);
      });
      return (
        <div>
        {this.props.name}
        </div>)
    }
});

React.render(<myComponent name="Steven">
  <p key="first">a child</p>
  <p key="second">another</p>
</myComponent>,document.getElementById('container'));
```

Code fragment 2.5: Voorbeeld van een map functie

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

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.

```
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

```
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

```
1  class MyComponent extends React.Component {
2
3    render(){
4      return (<div> Hello World </div>);
5    }
6  }
```

Code fragment 2.7: Component als super klasse

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 * * 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.

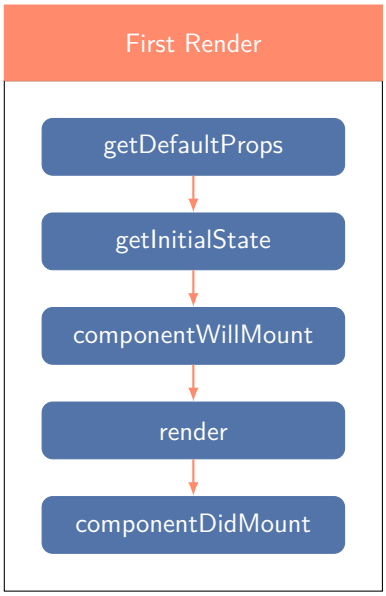
- Voorbeeld 1: *React.Component* als super klasse, maak een klasse aan die overerft van *React.Component*. Overschrijf **altijd** de *render()* methode (zie Code fragment 2.7).
- Voorbeeld 2: maak een variabele die de methode *React.createClass* aanroept, deze methode geeft een object weer van type *React.Component*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

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

Code fragment 2.8: Component als variabele



Figuur 2.9: Lifecycle first render

update queue, het mechanisme die React gebruikt om wijzigingen te behandelen. (Zie Virtual DOM, Rendering)

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 geinitialiseerd, 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 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 geinitialiseerd 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.

```
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 client kant wordt aangeroepen. De component bestaat en staat in het DOM.

Mixin

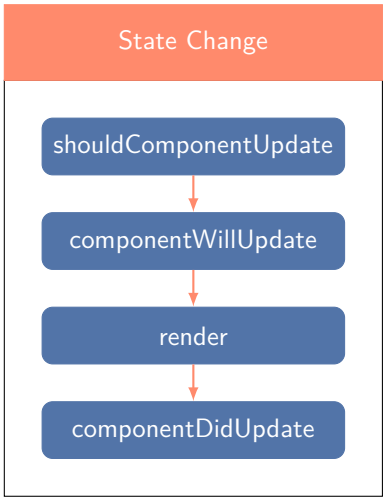
Een *Mixin* is een array, het kan de lifecycle events van een component monitoren zodat de ontwikkelaar met zekerheid weet dat functionaliteit op de correcte manier en tijdstippen van de lifecycle kan uitgevoerd worden. Code fragment 2.10 toont hoe een *Mixin* een component kan monitoren. Deze code zal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

Should Component Update

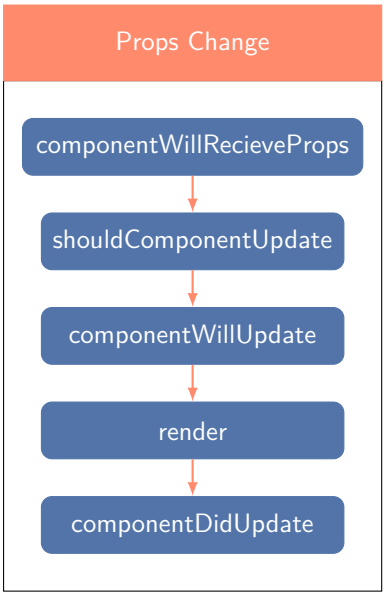
Elke keer wanneer er een wijziging gebeurd in de *state* of de properties wordt deze functie aangeroepen. Deze functie is een mechanisme die gebruikt kan worden om het opnieuw laden (*render()*) van de component over te slaan, wanneer de wijzigingen geen update nodig hebben.

```
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: Component state changed lifeCycle



Figuur 2.12: Pops change lifecycle

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 er steeds setState aanroepen zonder een extra update te moeten doen.

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 aan 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.

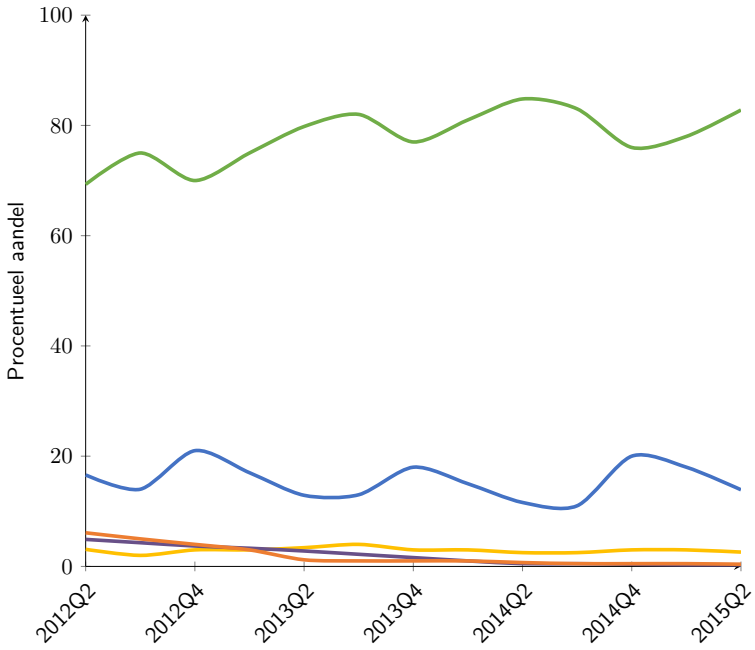
Positieve eigenschappen van native ontwikkeling :

- 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 betekend 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	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

- **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 geschreven hoeft te 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, ... 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.

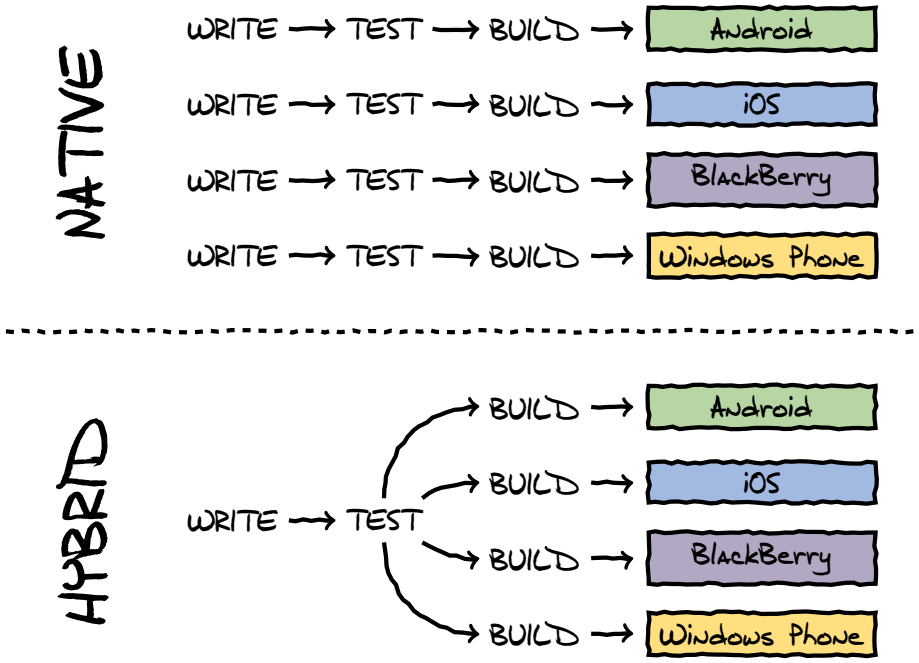
Uit deze voorbeelden kan de conclusie getrokken worden dat wanneer men een applicatie wenst te bouwen die veel http aanvragen (post, get, ...) zal moeten uitvoeren, best hybride ontwikkeld wordt.

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 worden en ze moeten 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 Facebook ingenieurs was dat wanneer er native applicaties werden gebouwd ze rekening moesten houden met het telkens heropstarten van de applicatie wanneer er iets werd gewijzigd. De wijzigingen van een knop die aantal centimeter werd verplaatst, bijvoorbeeld zorgde ervoor dat als men de wijziging op het scherm wou weergeven de applicatie volledig herstart moest worden. Wat het ontwikkelingsproces sterk vertraagde. 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 per direct kunnen worden opgelost. Bij de native applicatie van Facebook moet er vaak weken of maanden gewacht worden om een wijziging te kunnen doorvoeren, omdat de aanbieders van de stores deze eerst moeten 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)



Figuur 3.2: Hybride vs. Native ontwikkeling

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 tablet gebruikers bereiken. Toch gebruikt Facebook in React Native een hybride aanpak, want het maakt gebruik van JavaScript maar de applicatie zal volledig native beschouwd worden.

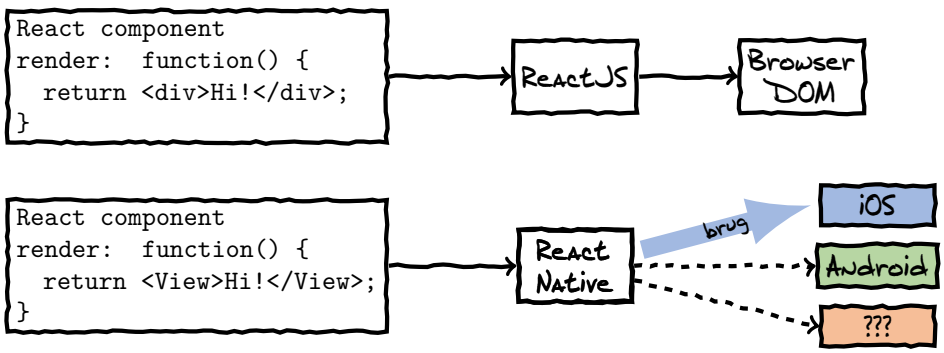
Dankzij dit framework heeft Facebook het mogelijk gemaakt voor web ontwikkelaars om applicaties te schrijven voor mobiele platformen in de reeds beheerste JavaScript taal en omdat het grootste deel van de code gedeeld kan worden tussen beide platformen wordt een gekend probleem van mobiele applicatie ontwikkeling geëlimineerd, nl. dat eenzelfde applicatie in meerdere programmeer talen 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 voor de gebruiker 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.

3.5 Hoe werkt React Native?

Om de werking van React Native te kunnen begrijpen moet de werking van React gekend zijn, nl. 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 hetgeen dat zichtbaar is voor de gebruiker. Maar wanneer gesproken wordt over native mobiele applicaties dan wordt er niet gewerkt met de browser, en 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: Virtual DOM vs bridge

Figuur 3.3 wordt het verschil tussen een React component en React Native component op gebied van weergave getoond. In plaats van het Virtuele DOM wordt de "bridge" aangesproken om de juiste weergave te tonen in de juiste OS.

De bridge of brug die gemaakt wordt voorziet React van een interface in de platform specifieke UI elementen. De layout die voorzien wordt door de render functie van een React component zal met ander woorden vertaald worden naar een gepaste component. Bv. <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 gaan aanspreken maar ook de componenten die de layout van een native applicatie gaat samenstellen.

De levensloop van de componenten in React Native is dezelfde als die van React voor web ontwikkeling, maar het weergave proces is een beetje anders, door het gebruik van de brug. Door de brug te gebruiken, is React Native verplicht om op een andere thread te gaan werken dan de main thread, op deze manier kan er voor gezorgd worden dat 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, eerder vermeld bij (3.2), 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 te gaan weergeven. 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, en de voordelen ervan behouden.

Eén van de grootste voordelen van React Native is de "Developer Experience", want die kan nu door gewoon JavaScript (+ ReactJS) 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. Men heeft het bv. Mogelijk gemaakt 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 nog eens voor gezorgd dat elke error die kan voorkomen nog eens voorzien wordt van een gedetailleerde boodschap.

Het andere probleem dat Facebook had was dat het weken of maanden duurde om een update door

te voeren. Aangezien Apple en Google, JavaScript gebaseerde wijzigingen toelaat om doorgevoerd te worden zonder dat er controle wordt op uitgevoerd zoals bij native applicatie updates, zijn de wijzigingen in React Native onmiddellijk zichtbaar bij de gebruikers van de applicatie. Daarnaast is de code herbruikbaarheid van React Native applicaties ook nog een enorm groot voordeel, wanneer een applicatie volledig native ontwikkeld moet worden 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, omdat sommige componenten heel specifiek voor het platform zijn en dus ook anders benaderd moeten worden. (zie hoofdstuk IOS vs Android in React Native). 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 van ontwikkelaars weg te werken.

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. 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 piek moment 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 React Native applicaties, en samen op zoek gaan naar *best practices*.

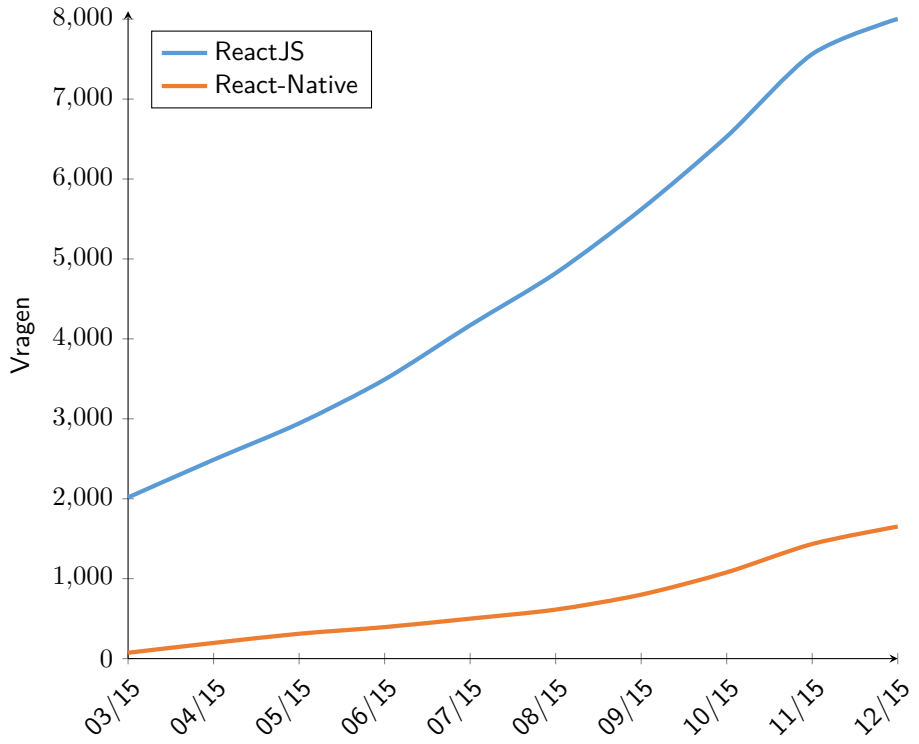
Dezelfde trends kan men terug vinden bij *Google Trends* Figuur 3.6 die de zoekopdrachten op basis van kernwoorden *ReactJS* en *React Native* weergeeft week na week, startend van begin 2015. React Native heeft overduidelijke piekmomenten tijdens de eerste periode, Maart 2015 (week 16 - 17) toen het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

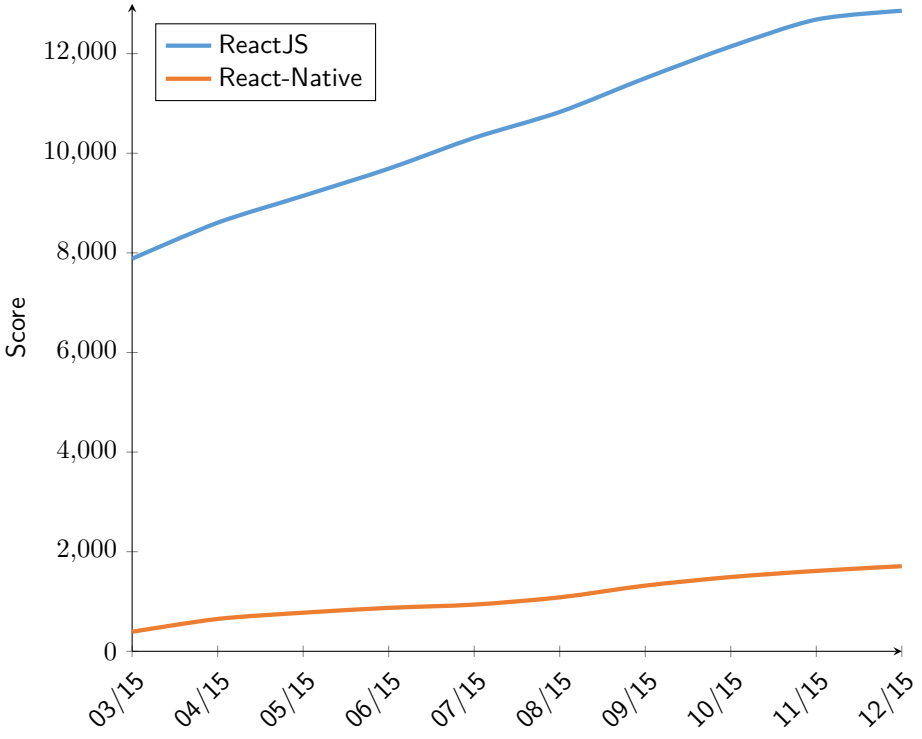
Net zoals in ReactJS (**??**), werkt React Native op een exacte wijze om componenten aan te maken. Toch zijn er een aantal verschillen in syntax, zoals eerder vermeld werkt native niet met het DOM maar met platform specifieke componenten. Deze moeten dan ook op correcte wijze worden aangesproken.

3.7.1 Views

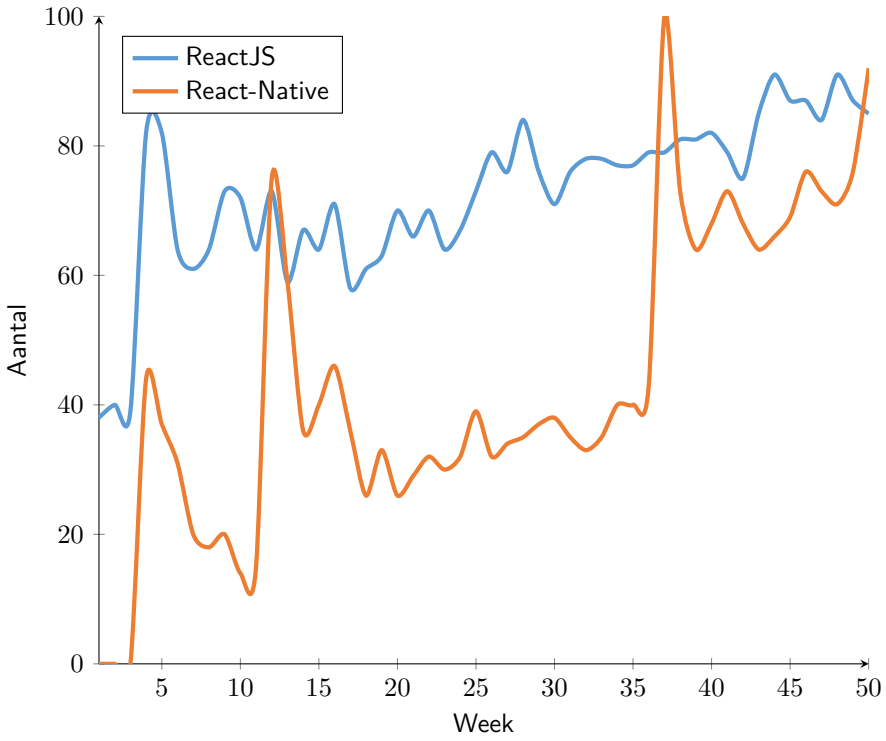
In ReactJS plaatsen we HTML componenten op het DOM (* *, *<p>*, **), bij React Native gaan we de platform componenten gaan weergeven. Het meest basis component in UI elementen is *<View>* die voor een webontwikkelaar best te vergelijken is met het * * element, en zich vertaald naar respectievelijk iOS en Android als *UIView* en *View*. In 3.1 wordt een vergelijking van HTML componenten, gebruikt in ReactJS gemaakt met React Native tegenhangers.



Figuur 3.4: StackOverflow vragen



Figuur 3.5: Score



Figuur 3.6: Google Trends zoekopdrachten met kernwoorden ReactJS en React Native

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

```
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

In Code fragment 3.1 wordt er getoond 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 3.1 worden aanzien als de basis componenten en worden automatisch geïmporteerd vanuit het standaard package die automatisch wordt ingeladen wanneer een nieuwe applicatie wordt gestart. Wanneer 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.

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.

Bibliografie

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

DocForge (2015). Framework. http://docforge.com/wiki/framework

Gackenheimer, C. (2015). *Introduction to React*. Apress.

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

IDC (2015). Smartphone os market share. http://www.idc.com/prodserv/smartphone-os-market-share.jsp

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

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

Vijayweb Solutions, I. P. L. (2014). Why you should avoid using javascript in your websites? 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. https://code.facebook.com/posts/1189117404435352/react-native-for-android-how-we-built-the-first-cross-platform-react-native-app/