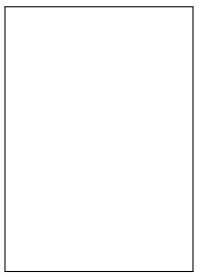


Faculteit Bedrijf en Organisatie

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

ADHS	Adaptive Dynamic Heuristic Set
BI	Best Improving
CHeSC	Cross-domain Heuristic Search Challenge
LLH	Low Level Heuristiek
OI	Only Improving
PAC	Probabilistically Approximately Complete
SAT	Boolean Satisfiability problem

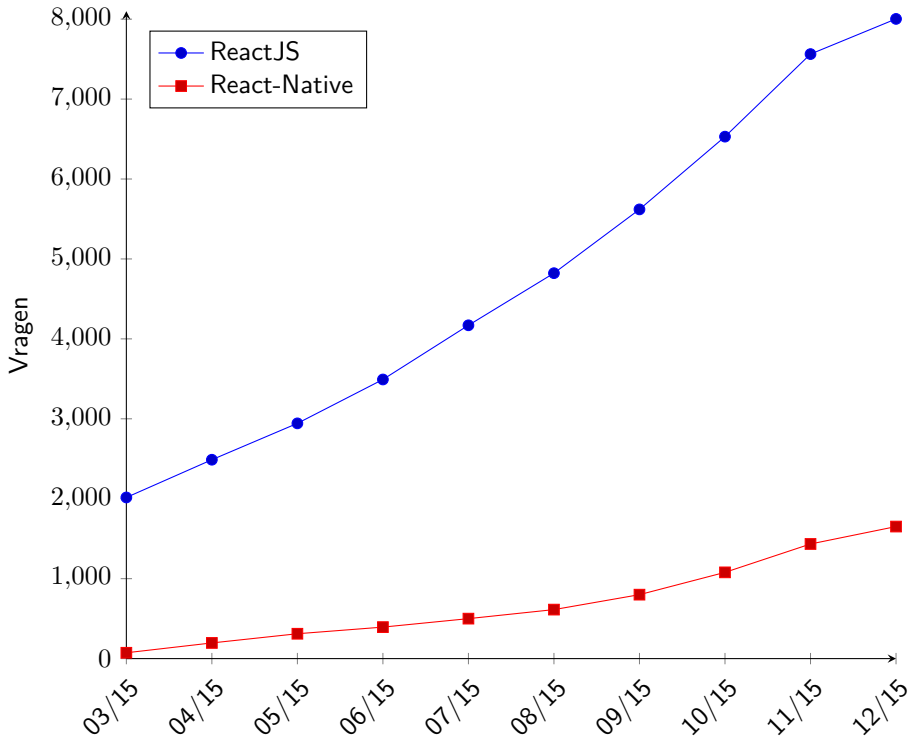
Woorden

woord	uitleg

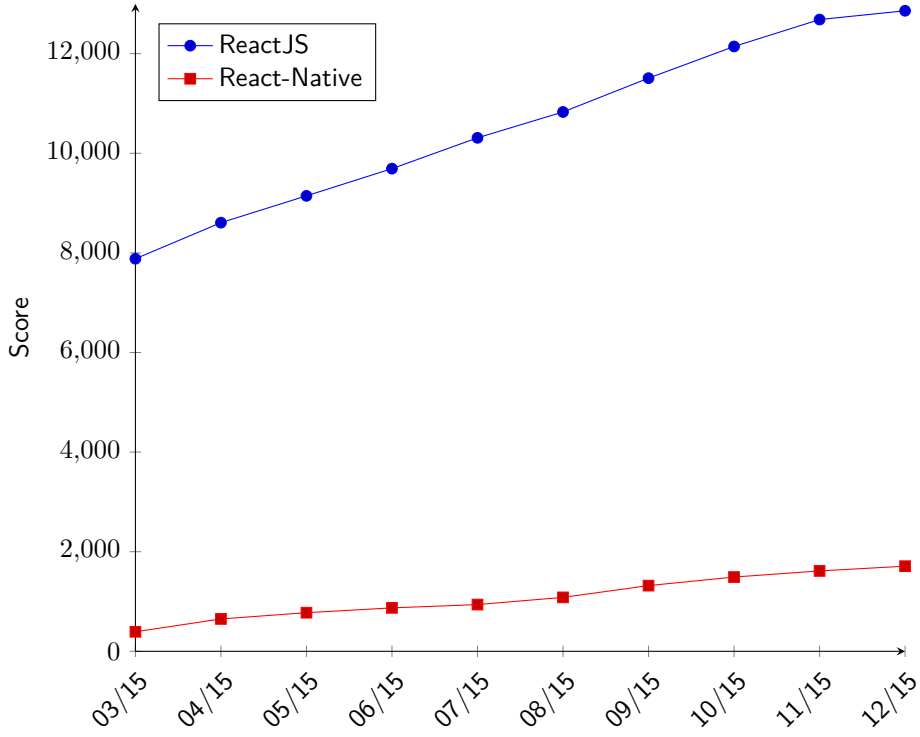
Lijst van code fragmenten

Hoofdstuk 1

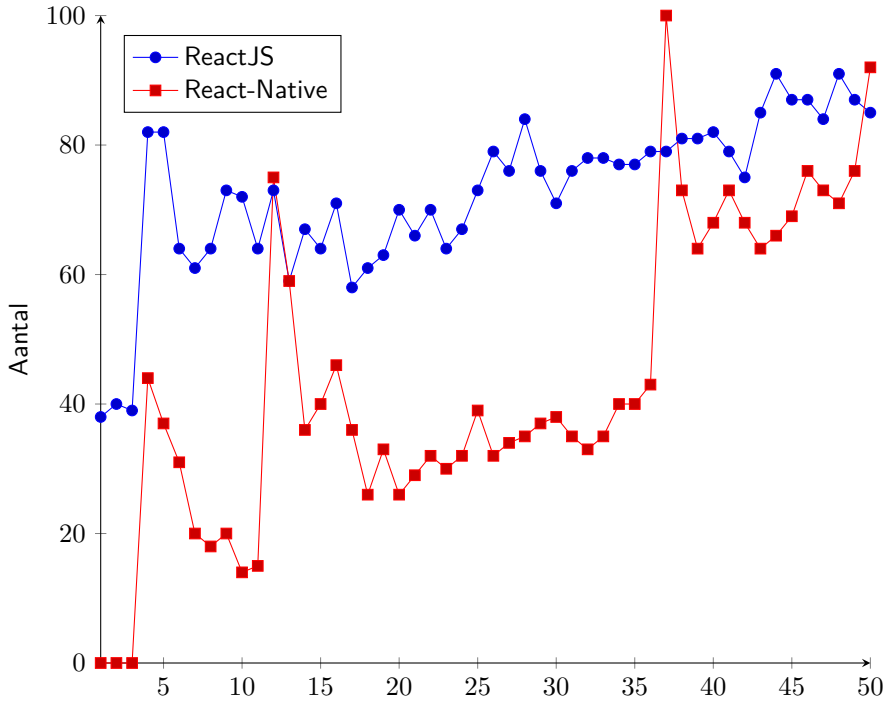
Grafiekskes



Figuur 1.1: Vragen



Figuur 1.2: Score



Figuur 1.3: Google Week

Hoofdstuk 2

React

2.1 Wat is React?

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.

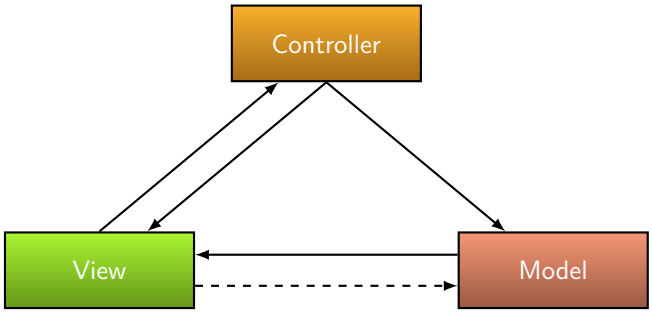
React werd dus ontwikkeld om één welbepaald probleem te gaan oplossen, nl. het tonen van data in de user interface. Als u denkt dat dat probleem reeds werd opgelost, bent u zeker niet verkeerd. Het verschil is dat React werd gecreëerd om grote schaalbare user interfaces, zoals Facebook en Instagram, 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 in 2004 al 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.

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.

2.2 Recent framework

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 een 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 user interface van een applicatie te beschrijven en een mechanisme om die user interface 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. 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 drie JavaScript Frameworks vergeleken 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 en in een listitem plaatst van een "unsorted list" in HTML. 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 drie frameworks 10x de lijst geladen en werd de tijd bijgehouden in tabellen, dit werd voor drie browsers herhaald.

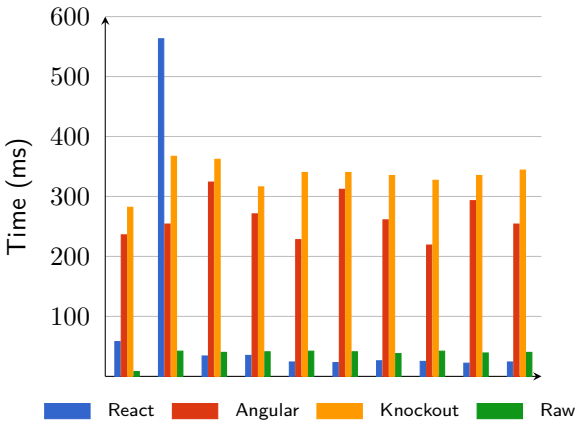
Resultaten:

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	

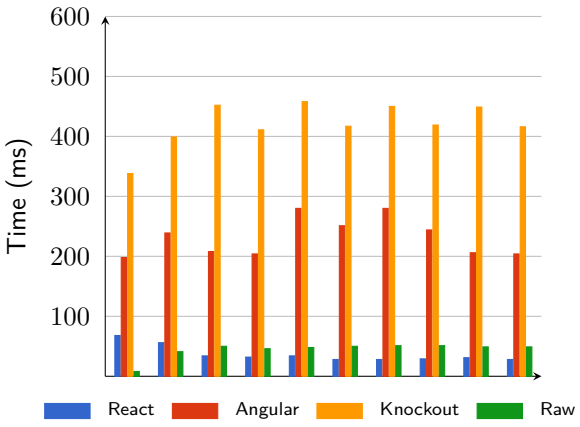
Tabel 2.1: Performance Comparison for React, Angular and Knockout, by Chris Harrington

	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



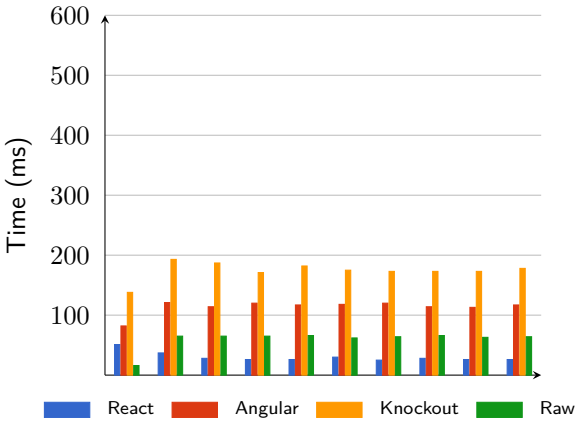
Figuur 2.2: Resultaat van performance test door Chrome 39.0.2171.95

	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

```
var MyClass = React.createClass({

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

});
```

Code fragment 2.1: TODO: add caption

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.

2.4 Componenten

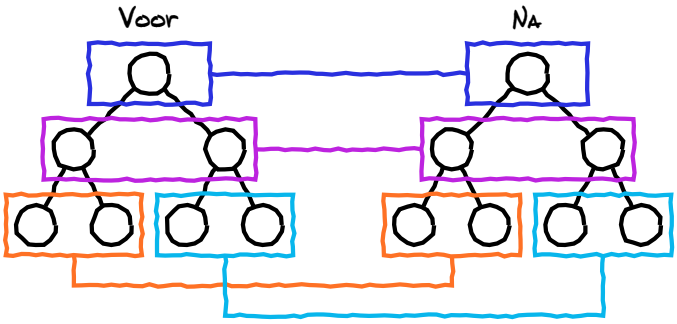
Componenten zijn de basis van React en de view van de applicatie. Hieronder vindt je een simpel voorbeeld van zo'n component. Let goed op bij de 'render' functie, deze geeft een HTML tag terug. Wanneer deze zal aangeroepen worden zal die de tekst "Hello world" laten verschijnen op de HTML pagina.

2.5 Virtual DOM

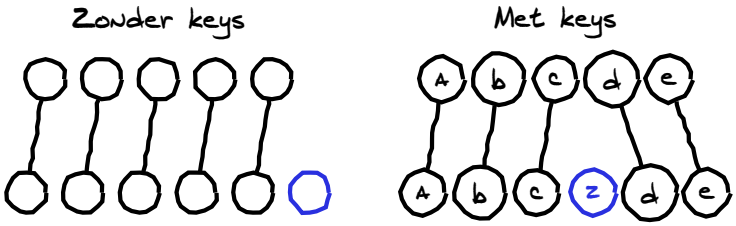
Dit is waarschijnlijk het meest belangrijke concept van React. Het virtuele DOM is de oplossing voor het probleem die het DOM stelt, namelijk dat het (nog) niet aangepast werd voor het weergeven van dynamische data. Met JavaScript en JQuery kon je als ontwikkelaar wel een oplossing voor dit probleem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

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

2.5.1 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.

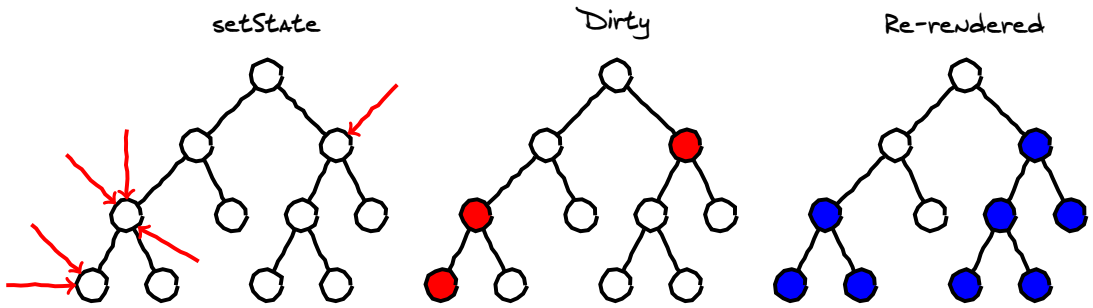
- Lijsten: Wanneer een component bij een éénmalige iteratie 5 andere componenten bevat wordt het moeilijk om in het midden van die lijst een nieuwe component toe te voegen. React gaat standaard een associatie maken tussen het eerste component van de eerste lijst en het eerste component van de tweede 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.

2.5.2 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 bv. Onclick listener op een unsorted list (<ul>) element in plaats van op elk list item (<li>) element in de list . 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.

2.5.3 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

```
//JSX versie

React.render(
    <div>
      <h1> Hello </h1>
    </div>
  );

//wordt vertaald naar

React.render(
    React.createElement('div', null,
      React.createElement('h1', null, 'Hello')
    );
  );
```

Code fragment 2.2: TODO: add caption

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.

2.5.4 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 je bijvoorbeeld HTML tags meegeven en aanvaard het aangepaste React klassen.

2.5.5 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: TODO: add caption

2.5.6 State

State is een set op elke component 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.

2.5.7 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. 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 te gaan creëren die door drie stukken van de flux architectuur wordt geleid nl. de dispatcher, de stores en uiteindelijk de React View. Flux is essentieel in het bouwen van webapplicaties met React en zal dus niet verder aan bod komen in deze scriptie.

2.6 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.

2.6.1 React.createClass

In het hoofdstuk over React Concepten 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 later 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 krijgen:

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.

```
var MyComponent = React.createClass({

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

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

Code fragment 2.4: TODO: add caption

```
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: TODO: add caption

2.6.2 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 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.
- *React.Children.count(children)*: de *count* methode zal het aantal child objecten in de lijst children terug geven.

2.6.3 React.createElement

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

```
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: TODO: add caption

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

Code fragment 2.7: TODO: add caption

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

2.7 React componenten

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

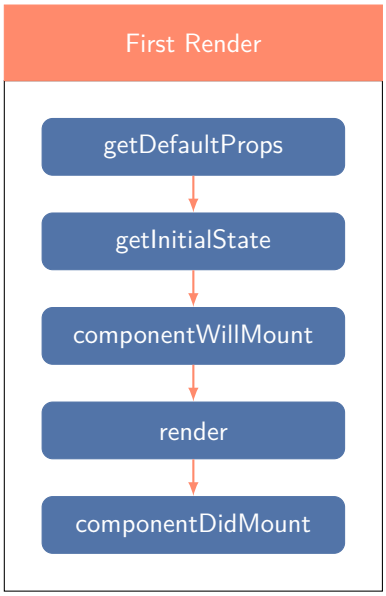
2.7.1 Creatie

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.

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

Code fragment 2.8: TODO: add caption



Figuur 2.9: Lifecycle first render

2.7.2 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. (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.

2.7.3 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.

2.7.4 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.

```
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: TODO: add caption

2.7.5 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 start waarde mee krijgen.

2.7.6 getDefaultProps

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"/>).

2.7.7 componentWillMount

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.

2.7.8 componentDidMount

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

2.7.9 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 pagina weergeeft in het aantal seconden (zie Figuur 2.10).

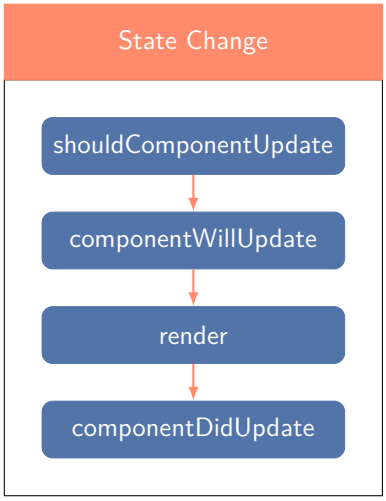
2.7.10 shouldComponentUpdate

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.

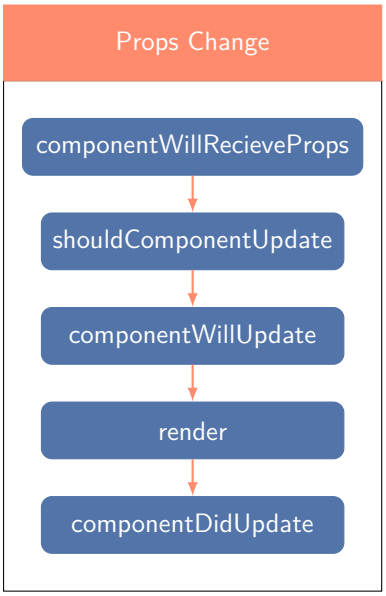
```
var SetIntervalMixin = {
  componentWillMount: function() {
    this.intervals = [];
  },
  setInterval: function() {
    this.intervals.push(setInterval.apply(null, arguments));
  },
  componentWillUnmount: function() {
    this.intervals.map(clearInterval);
  }
};

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: TODO: add caption

Figuur 2.10: Screenshot resultaat mixin voorbeeld



Figuur 2.11: Component state changed lifeCycle



Figuur 2.12: Pops change lifecycle

2.7.11 ComponentWillUpdate

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.

2.7.12 componentDidUpdate

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.

2.7.13 componentWillRecieveProps

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.

Bibliografie

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