HO GENT

Faculteit Bedrijf en Organisatie

Front end performantie in React gebaseerde applicaties

Matthias Tison

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Leen Vuyge
Co-promotor:
Arvid De Meyer

Instelling: Codifly

Academiejaar: 2018-2019

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Front end performantie in React gebaseerde applicaties

Matthias Tison

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Leen Vuyge
Co-promotor:
Arvid De Meyer

Instelling: Codifly

Academiejaar: 2018-2019

Tweede examenperiode

Woord vooraf

De scriptie werd geschreven ter voltooiing van mijn bachelor Toegepaste Informatica met als specialisatie mobiele applicaties. Het gekozen onderwerp is niet in lijn van alles wat ik aangeleerd kreeg in de voorbije jaren, maar het is een onderwerp dat mij intrigeert. De manier waarop je als frontend developer te werk moet gaan en alle factoren waarmee rekening moeten gehouden worden vindt ik oprecht interessant. Als frontend developer sta je toch net dat stapje dichter bij de eindgebruiker en heb je visuele confirmatie van alles dat wordt geïmplementeerd. Ik kan zeggen dat ik eerder thuis ben in het frontend aspect dan backend, wat dit een geschikt onderwerp maakte voor mij.

Verder zou ik graag iedereen bedanken die mij geholpen heeft bij het maken van deze scriptie. Mijn co-promotor Arvid De Meyer, maar ook alle collega's van het bedrijf waar ik mijn stage liep wil ik bedanken voor het toelichten van hun opinies en verfrissende kijk. Uiteraard wil ik ook mijn promotor Veerle Vuyge bedanken voor al het geduld en vertrouwen dat ze in mij vertoonde. Ten slotte bedank ik graag mijn familie, vrienden en vriendin voor de talloze momenten van steun en toeverlaat.

Samenvatting

React was sinds enkele jaren nog een nieuwkomer op het toneel voor frontend JavaScript development. Sindsdien is de populariteit van de flexibele library enorm toegenomen en is het één van de meest prominente frontend frameworks geworden. In deze scriptie wordt een onderzoek gedaan naar de aspecten binnen het React landschap dat performantie kan bevorderen op frontend niveau.
In de scriptie wordt React uitvoerig ontleed en de meest courante technieken voor het verbeteren van de performantie in het framework worden onderzocht, met bijhorende studie.
Dit onderzoek draagt bij tot het in kaart brengen van de belangrijkste factor voor het maken van moderne webapplicaties. De impact die websites op een bedrijf kunnen hebben en een betere kijk op wat het precies allemaal inhoud. Voor deze scriptie is een uitgebreid onderzoek gedaan naar die belangrijke factor, performantie. Hierbij is de scriptie opgedeeld in drie delen. In deel één wordt het framework onder de loop genomen om een beter begrip te krijgen van de structuur en het gebruik van React. Het tweede deel omvat de theorie achter performantie. Daartegenover staat het derde deel van de scriptie die ingaat op de praktische kan van performantie en wat er aan kan gedaan worden om het te optimaliseren als zijde developer.
Het resultaat van de scriptie is veelbelovend. Aangezien het internet zo snel evolueert, gaan ook de verwachtingen van mensen de hoogte in. Er is aangetoond dat met de juiste technieken en voldoende kennis over het framework optimalisaties kunnen toegepast worden.
Omtrent verdere onderzoeken zie ik dit in de toekomst nog meer gedaan worden. Het blijft veranderen en de standaarden van nu zullen binnen de kortste tijd weer veranderen en dan is er altijd plaats voor een volgend soortgelijk onderzoek. Daarom niet speciaal voor het React, maar voor de nieuwe opkomende frameworks zoals Preact, vue.js, Flutter,...

Inhoudsopgave

Lijst van figuren

Listings

Woordenlijst

AJAX Asynchronous JavaScript And HTML. 18
API Application Programming Interface. 41

CSS Cascading Style Sheet. 18

DOM Document Object Model. 18, 24, 27, 31, 32, 38, 42, 44

ECMA European Computer Manufacturers Association. 18, 32

fps Central processing unit. 42
fps Frames per second. 38

HTML HyperText Markup Language. 18, 19, 23, 24, 27, 38

JS javaScript. 19, 23
JSX JavaScript eXtension. 24

KMO Kleine of Middelgrote Onderneming. 17

MVC Model View Controller. 23

SEO Search Engine Optimization. 20, 36

UI User Interface. 20, 23, 25–29, 35, 43, 44
UX User eXperience. 19

WWW World Wide Web. 35, 37

1. Inleiding

De inleiding van deze scriptie licht de probleemstelling en onderzoeksvraag toe. Verder wordt de titel opgedeeld in verschillende delen. Elk deel wordt kort toegelicht om een algemene kijk te krijgen op wat de scriptie inhoudelijk bevat.

1.1 Probleemstelling

Het onderzoek is van meerwaarde voor elke React frontend web developer, zowel zelfstandig als binnen een onderneming. De developers kunnen het onderzoek gebruiken als toevoeging tijdens of vooraf het ontwikkelen van een React web applicatie. Op grotere schaal is het ook interessant voor ondernemingen, in het bijzonder start-ups, maar ook KMO's. De onderwerpen die onderzocht worden zijn van belang voor elk jong en/of nieuw startend team.
Dit wilt niet zeggen dat het geen meerwaarde kan bieden voor bijvoorbeeld backend developers, mobile developers of software architecten. Het is in belang van elk die interesse heeft in React frontend web development.

1.2 Onderzoeksvraag

De kernvraag van deze scriptie: Hoe performantie op frontend niveau op een doelgerichte en innovatieve manier verbeteren in React web applicaties?
Performantie is een gegeven dat in de loop der jaren in de software wereld enorm op de voorgrond is gekomen. Met hoe snel technologie en software zich ontwikkeld is het

moeilijk voor iedere software ontwikkelaar om bij te houden wat de best practices zijn op het gebied van performantie. In deze 'modern age' is het vanzelfsprekend dat webpagina's en apps binnen de twee seconden reageren op interactie van de gebruiker. Uit het artikel van Mazaika (2017) leiden we af dat de vraag naar developers en software bedrijven enorm toeneemt. Dit ten gevolge van de exponentiële groei in de vraag naar webapplicaties, mobiele applicaties en andere software. Elke dag worden zij op de proef gesteld om aan de noden van gebruikers te voldoen. Met de druk van steeds meer uitgebreide en evenement rijke gebruikersomgevingen is het noodzakelijk om stil te staan bij performantie.
Hoe meten we performantie? Welke oplossingen biedt React ons? Waar hangt performantie van af in webapplicaties? Wat zijn de veel voorkomende valkuilen? Bestaan er ondersteunende software? Welke innovatieve technieken kunnen we toepassen? Kunnen we een leidraad vormen voor frontend performantie binnen React web applicaties?

1.3 Javascript

De taal werd gecreëerd door Brendan Eich in 1995 in zijn tijd als werknemer bij Netspace Communications, zoals aangegeven in het artikel van Aston (2015). In 1997 werd het een ECMA standaard en nu behoort het tot één van de meest gebruikte programmeertalen voor het web. Zoals de naam al vrijgeeft behoort javaScript tot de scripttalen, elke scripttaal is een programmeertaal in zijn eigen recht. Voorbeelden zijn: Node js, bash, Ruby, Perl, Python, ...
Scripttalen worden veel gebruikt omdat ze eenvoudig, flexibel en gebruiksvriendelijk zijn. Het zijn talen geschreven voor een run-time omgeving en worden samen met de executie van de applicatie uitgevoerd. Ze moeten niet gecompileerd worden door een compiler.
JavaScript wordt veel gebruikt als client side programmeertaal. Alle geschreven javaScript documenten worden bij het navigeren naar een webpagina samen met alle HTML en CSS documenten opgehaald van de server. Alle geïmporteerde documenten worden daarna door de browser geïnterpreteerd aan de kant van de gebruiker.

1.3.1 JavaScript-framework

Frameworks bezorgen de programmeur een duidelijke structuur en gestandaardiseerde code. Telkens opnieuw code moeten opbouwen en schrijven rooft tijd en productiviteit. Afhankelijk op welk niveau de programmeertaal wordt gebruikt bestaan er 2 typen frameworks, frontend en backend.
Door de immense opkom van webapplicaties begin jaren 2000 werd de vraag naar javaScript en AJAX enorm groot. Het vraag overschot zorgde voor de onvermijdelijke behoefte aan javaScript-frameworks. De frameworks brengen een oplossing voor de problemen die zuivere javaScript vormt:

> **Browserafhankelijkheid**: JavaScript zelf is niet afhankelijk van de browser, maar er worden wel DOM manipulaties gedaan via javaScript. De DOM is wel afhankelijk van de browser. Er moet op basis van de browser extra code voorzien worden.

Prototype programmeren: Maakt geen gebruik van klassen. Er wordt een prototype gemaakt van een object dat kan dienen voor overerving, het heeft limieten.

Geen code herbruikbaarheid: De code kan niet op een functionele manier worden gebundeld voor hergebruik, wat noodzakelijk is wanneer de complexiteit toeneemt.

Voorbeelden van JS-frameworks zijn ReactJS, AngularJS, vue.js, Ember.js, ...

1.3.2 Javascript library

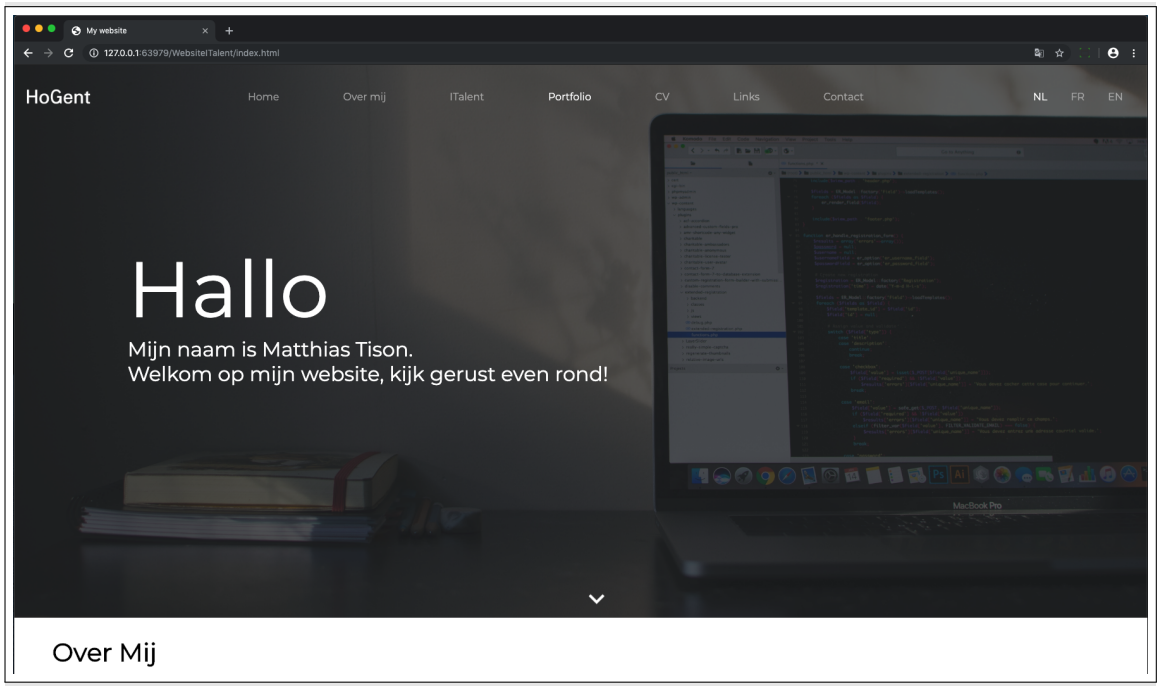
Libraries zijn een geheel van voorgeprogrammeerde functies die kunnen gebruikt worden om een aanvulling te bieden tot de reeds beschikbare functionaliteiten die het framework in kwestie aanbied. Het aspect dat ze een aanvulling bieden maakt het development eenvoudiger en makkelijker uit te breiden. In deze tijd is het simpel om een library te gebruiken met een installatie via de package manager, daarna enkel nog een import om deze in scope te plaatsen. In figuur 1.1 op pagina 19 wordt de *styled* library geïmporteerd voor het stylen van HTML elementen.
Sommige libraries worden aanzien als een eigen framework omdat zij full-stack capaciteiten hebben, wat wil zeggen dat ze zowel frontend als backend functionaliteiten voorzien. ReactJS is één van zo'n frameworks. Meningen verschillen over de interpretatie, maar beide kunnen geaccepteerd worden met elk juiste gegronde argumenten. Mensen uit de branche zoals Tom Dale, Senior Staff Software Engineer bij LinkedIn en co-creator van Ember.js, verwijzen naar React als een framework waar en tegen de online documentatie van React het als een library omschrijft.

```
import React from 'react';  8.5K (gzipped: 3.4K)
import styled from '@emotion/styled';  24.5K (gzipped: 9.8K)
import PropTypes from 'prop-types';  1.6K (gzipped: 838)
```

Figuur 1.1: Importeren van een library

1.4 Frontend

De letterlijke vertaling van het woord is 'voorkant', duidend op datgene wat rechtstreeks binnen de visuele waarneming valt. Alles waar de gebruiker een interactie mee kan hebben wordt aanzien als frontend. Het definieert het visuele van een webapplicatie, dat wat de gebruiker werkelijk ziet en op kan inwerken. Aan de andere kant, alles wat de gebruiker niet ziet of waarneemt wordt naar verwezen als de backend.
Het belang van een goede frontend is een prioriteit voor elk nieuw web project. Voor en in de loop van het project wordt er een UX-ontwerp gemaakt, afgestemd door een bijhorend onderzoek en voor opgestelde requirements.



Figuur 1.2: Frontend webpagina

1.5 Perfomantie

Om een duidelijk beeld te krijgen van de kwaliteit van een website worden verschillende criteria gebruikt, performantie is hier één van. Performantie is een maatstaf voor de laadtijden van een website die bepalen hoe snel web gebruikers de UI te zien krijgt.
De attentiespanne ven een persoon op het web is progressief gedaald met de tijd, wat te maken heeft met de verandering in cultuur en technologie. In het artikel van Carette (2011) worden er wetenschappelijke resultaten aangehaald die hierop duiden. De onderzoeken tonen aan dat laadtijden van enorm belang zijn voor de perceptie van mensen wanneer ze voor het eerst naar een bepaalde website surfen. Het artikel beschrijft hoe een slechte laadtijd kan leiden tot imago schade, verlies van cliënteel, fiscaal verlies en tot zelfs een verarming in SEO.
De meeteenheid bij uitstek voor performantie is tijd en is van cruciaal belang voor online succes. Performantie is afhankelijk van veel factoren, in deze scriptie wordt ingegaan op de frontend performantie, gebruikmakende van het React framework.

1.6 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt React ontleed voor een goed begrip te krijgen van het framework zelf.

In Hoofdstuk 3 komt het theoretische aspect van de performantie in kaart en worden de

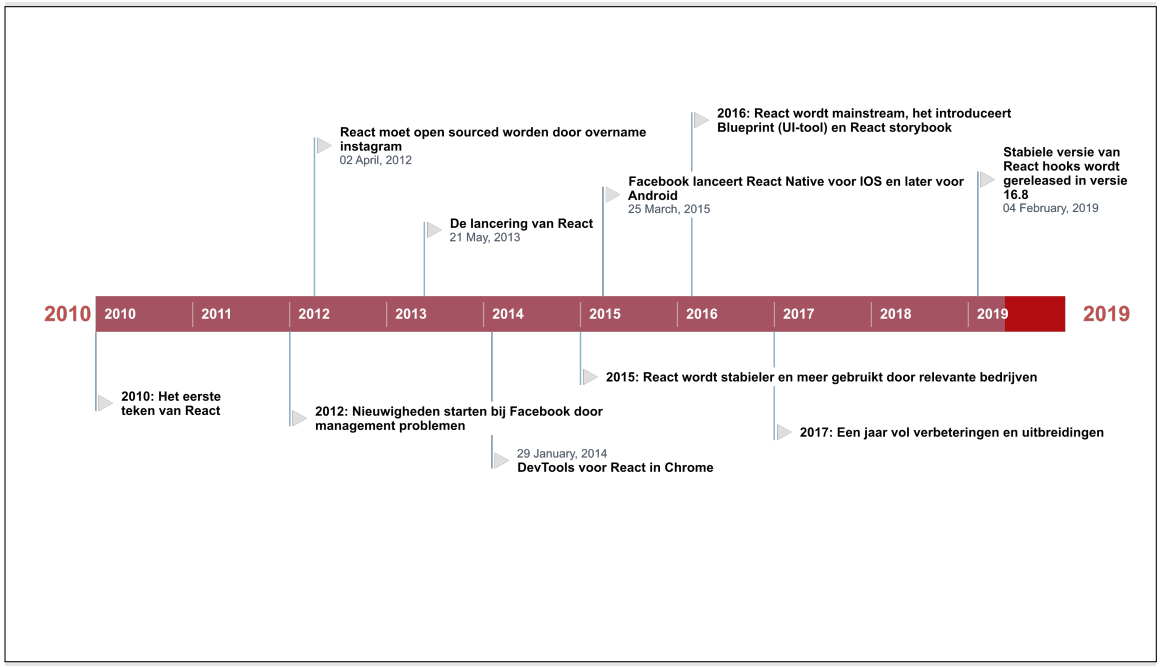
metrieken uitbundig besproken.

In Hoofdstuk 4 wordt het praktische van performantie op basis van de meest courante technieken uitgewerkt.

In Hoofdstuk 5, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. React

React is een open source JS-library dat werd geïntroduceerd in 2013 door het development team van Facebook. Figuur 2.1 op pagina 24 is een vereenvoudigde tijdlijn dat de evoluties van React aangeeft tot heden.
Vanuit het standpunt dat React een framework is, kan het geen volgend populair MVC framework genoemd worden. Eerder is React vervanger voor de V binnen MVC. Views in traditionele MVC-frameworks zijn aanzien als logica loze files die worden aangestuurd door een controller.
React maakt gebruik van componenten voor het definiëren van speciaal op maat gemaakte HTML-tags. Deze zijn een combinatie van HTML en plain javaScript, waarbij elk component gedefinieerd wordt in een aparte file, wat een makkelijk leesbare structuur oplevert. Het is bedoeld om alle UI elementen onder te verdelen in zo klein mogelijke componenten, zodat deze doorheen de codebase makkelijk hergebruikt kunnen worden. De UI van een React applicatie wordt opgebouwd door alle componenten samen te voegen tot één geheel.



Figuur 2.1: Product tijdlijn React

2.1 JSX

De JavaScript eXtension (JSX) wordt gebruikt voor het definiëren van React elementen in een HTML formaat. De compiler vertaalt de JSX code naar gewone javaScript tijdens runtime. JSX geeft een virtuele HTML visualisatie van hoe het component er in de DOM uitziet. Het is een vereenvoudigde presentatie van een nesting van React elementen die elk respectievelijk worden aangemaakt met ‘React.createElement()‘. In JSX wordt elk van die elementen voorgesteld in een HTML-tag.
Zoals wordt aangetoond in Code fragment 2.1 op pagina 24 is de syntax aanzienlijk eenvoudiger met JSX.

```
1   // HelloComponent.js met JSX syntax
2   export class HelloComponent extends React.Component {
3       render () {
4           return (
5               <div>Hello world!</div>
6           );
7       }
8   }
9
10  // HelloComponent.js zonder JSX syntax
11  export class HelloComponent extends React.Component {
12      render () {
13          return (
14              React.createElement(
15                  'div',
16                  null {/* Properties van de HTML-tag zoals 'name', 'alt', ... */},
17                  'Hello world!'
18              )
19          );
20      }
```

```
21      }
```

Code fragment 2.1: JSX naar gewone javaScript

2.2 Componenten

Componenten zijn de bouwstenen voor een React UI. Ze dienen om React elementen aan te maken die op hun beurt beschrijven hoe het specifieke deel van de UI er moet uitzien. Optioneel kan er input meegegeven worden aan een component in de vorm van properties. Naast optionele input kunnen componenten ook een lokale state bijhouden waar waarden worden in opgeslagen. De waarden kunnen veranderen doorheen de levenscyclus van het component.
Er bestaan twee soorten, functionele en klasse componenten. Vóór React versie 16.8 was er een duidelijk verschil tussen de twee soorten en door de komst van React hooks in versie 16.8 is dit helemaal veranderd.

2.2.1 Klasse componenten

Klasse componenten hebben dezelfde bouw als klassen in een object-georiënteerde programmeertaal zoals Java. Ze kunnen allebei erven van andere modules en bevatten functies en/of waarden die de inhoud kunnen veranderen. Het grote verschil tussen beiden zit in de methodologie. Bij klasse componenten wordt er altijd een specifiek deel van de UI geretourneerd via een render functie en er kunnen waarden in een lokale state worden geplaatst die invloed hebben op de geretourneerde UI. Ze bevatten lifecycle methoden die uitgevoerd worden doorheen de levenscyclus van een component, zoals de render, componentDidMount, componentWillMount,... functies.
In Code fragment 2.2 op pagina 26 is een simpele representatie uitgeschreven van een klasse component.

```
import React from 'react';
import logo from '../logo.svg';

// ListItem component als klasse component
export class ListItem extends React.Component {
    render () {
        return (
            <ListItemContainer>
                <LogoWrapper><Logo src={logo} alt="logo" /></LogoWrapper>
                <ContentWrapper>
                    <h2>Dit is de titel van het item</h2>
                    <Paragraph>
                        Dit is waar de inhoud normaal komt, maar er is geen..
                    </Paragraph>
                </ContentWrapper>
            </ListItemContainer>
        );
    }
}
```

Code fragment 2.2: Een klasse component

2.2.2 Functionele componenten

Vóór React hooks werden functionele componenten gebruikt voor een beknopte presentatie te maken van React elementen. Ze kunnen optioneel input krijgen waarmee ze een bepaald deel van de UI retourneren. Functionele componenten hadden geen logica en/of state waardoor ze heel voorspelbaar waren, dus werden deze componenten stateless genoemd. Code fragment 2.3 op pagina 26 toont een stateless functioneel component.
Met de komst van React hooks kunnen functionele componenten statefull gemaakt worden en logica bevatten. De bruikbaarheid is naar het niveau van klasse componenten getrokken. Door de veranderingen is het mogelijk om meer overzichtelijke en makkelijk leesbare componenten te maken. In het boek van Robert C. Martin (Martin, 2008) wordt aangehaald waarom dit voordelig is:

> *Indeed, the ratio of time spent reading versus writing is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ...[Therefore,] making it easy to read makes it easier to write.*

```
import React from 'react';
import logo from '../logo.svg';

// ListItem component als functioneel component
export const ListItem = () => (
    <ListItemContainer>
        <LogoWrapper><Logo src={logo} alt="logo" /></LogoWrapper>
        <ContentWrapper>
            <h2>Dit is de titel van het item</h2>
            <Paragraph>
                Dit is waar de inhoud normaal komt, maar er is geen..
            </Paragraph>
        </ContentWrapper>
    </ListItemContainer>
);
```

Code fragment 2.3: Een functioneel component

2.2.3 Componenten exporteren

Componenten worden gemaakt in hun specifieke file en geïmporteerd in een andere om ze daar in scope te plaatsen en te gebruiken. Om een component te importeren in een file moet deze dus eerst geëxporteerd worden uit zijn eigen file.
Er zijn twee manieren om een component te exporteren. De klasse of functionele constante kan expliciet of impliciet geëxporteerd worden, dit is te zien in Code fragment 2.4 op pagina 27.
Bij expliciet exporteren wordt het component ook expliciet geïmporteerd onder die naam. Impliciet exporteren geeft de mogelijkheid om het component onder een ander naam te importeren. Een impliciete export kan enkel één keer per file gedaan worden.

```
1   //Expliciet exporteren en importeren van een klasse component
2   export class HelloComponent extends React.Component {
3       render () {
4           return (
5               <div>Hello world!</div>
6           );
7       }
8   }
9
10  import { HelloComponent } from './helloComponent.js';
11
12
13  //Impliciet exporteren en importeren van een klasse component
14  class HelloComponent extends React.Component {
15      render () {
16          return (
17              <div>Hello world!</div>
18          );
19      }
20  }
21  export default HelloComponent;
22
23  import HelloComponent from './helloComponent.js';
```

Code fragment 2.4: Exporteren van een component

2.3 Fragments

Een React component retourneert altijd een (klein) deel van de UI. Hetgeen dat wordt geretourneerd is een nesting van React elementen die samen gebonden zijn in één ouder element. Fragments voorkomen de vervuiling van het DOM met nutteloze omhullende ouder tags, voornamelijk tags, die dienen voor het samenbinden van React elementen. De fragment tag retourneert al zijn kinderen zonder aanvullende HTML-tags, dus is er minder vervuiling van het DOM en voorkomt men het overvloedig nesten van elementen. In Code fragment 2.5 op pagina 28 wordt een onderscheid gemaakt tussen het gebruik zonder en met fragment tag.

```
// HelloComponent.js zonder Fragment als ouder tag
import React from 'react';

export class HelloComponent extends React.Component {
    render () {
        return (
            <div>
                <div>Hallo,<div>
                <div>Groetjes aan iedereen!</div>
            </div>
        );
    }
}

// HelloComponent.js met Fragment als ouder tag
import React from 'react';

export class HelloComponent extends React.Component {
    render () {
        return (
            <React.Fragment>
                <div>Hallo,<div>
                <div>Groetjes aan iedereen!</div>
            </React.Fragment>
        );
    }
}
```

Code fragment 2.5: Gebruik van de Fragment tag

2.4 Props

De properties, of props genaamd, van een component zijn een gewoon javaScript object die alle input waarden bevat die aan de component worden meegegeven. Het meegeven van props aan een component zorgt ervoor dat deze dynamisch aan te maken zijn. De props bepalen hoe het te retourneren deel van de UI er zal uitzien. Ze zorgen ervoor dat er data kan worden doorgegeven naar andere componenten, dit in de meeste gevallen van een ouder naar zijn kind. Deze toepassing zorgt ervoor dat de unidirectional data flow, waar React op steunt, wordt behouden.
Props zijn bovendien read-only, wat wil zeggen dat ze niet aanpasbaar zijn doorheen de levenscyclus van een component. Wanneer data moet veranderen tijdens de levenscyclus van een component gebruiken we state.

2.4.1 Prop types

Het is een best practice om voor alle componenten de data die binnen komt te definiëren. Door het expliciet definiëren wordt het duidelijk wat componenten verwachten om goed te kunnen functioneren.

2.4.2 Default props

Wanneer een bepaalde property in de component visuele waarde heeft, is bij afwezigheid van die property een deel van de UI onvolledig. Om dergelijke scenario's te vermijden stellen we een standaard waarde in voor properties die onvoorzien kunnen zijn. Door properties een standaard waarde te geven maken we de componenten veerkrachtig.

```
import React from 'react';
import PropTypes from 'prop-types';

import logo from '../logo.svg';

// ListItem component als klasse component
export const ListItem = (props) => (
    <ListItemContainer>
        <LogoWrapper><Logo src={logo} alt="logo" /></LogoWrapper>
        <ContentWrapper>
            <h2>{props.title}</h2>
            <Paragraph>{props.text}</Paragraph>
        </ContentWrapper>
    </ListItemContainer>
);

ListItem.defaultProps = {
    text: 'Er werd geen inhoud tekst meegegeven',
}

ListItem.defaultProps = {
    title: PropTypes.string.isRequired,
    text: PropTypes.string
}
```

Code fragment 2.6: Functioneel component met properties

2.5 State

State is, net zoals de properties, een javaScript object waar data wordt in opgeslagen. De data in de state kan doorheen de levenscyclus van een component veranderen door middel van 'setState()'. Wanneer setState wordt aangeroepen gaat het volledige component worden herladen, ook wel rerender genaamd. Alle elementen in het component die de aangepaste data uit de state gebruiken gaan die nieuwe data invullen.
Binnen een component is state aanzien als immutable, wat wil zeggen dat de state niet rechtstreeks mag aangepast worden door gebruik te maken van 'this.state'. Het aanpassen van de state met setState zorgt voor de immutability van de state. Bij componenten met een grote state is een goede benadering een kopie maken van het reeds bestaande state object, dit aan te passen en het nieuwe object mee te geven aan setState.
In Code fragment 2.7 op pagina 30 is een statefull klasse component uitgewerkt die met setState de zijn interne state gaat aanpassen. Code fragment 2.8 op pagina 31 toont datzelfde klasse component omgezet naar een functioneel component, waar met hooks de interne state wordt beheert.

```
import React from 'react';
import PropTypes from 'prop-types';
import styled from '@emotion/styled';

import logo from '../logo.svg';

const BACKGROUND_COLORS = ['green', 'yellow', 'blue', 'orange', 'red', 'white'];

// ListItem als klasse component
// ////////////////////////////////////////////////////////
export class ListItem extends React.Component {
    state = {
        backgroundColor: 'white',
    }

    static propTypes = {
        text: PropTypes.string,
        title: PropTypes.string.isRequired,
    }

    setColor = () => {
        const random =
            Math.floor(0 + Math.random() * (0 + BACKGROUND_COLORS.length));
        this.setState({
            backgroundColor: BACKGROUND_COLORS[random]
        });
    }

    render () {
        return (
            <ListItemContainer
                color={this.state.backgroundColor}
                onClick={this.setColor}
            >
                <LogoWrapper><Logo src={logo} alt="logo" /></LogoWrapper>
                <ContentWrapper>
                    <h2>{this.props.title}</h2>
                    <Paragraph>{this.props.text}</Paragraph>
                </ContentWrapper>
            </ListItemContainer>
        );
    }
}

ListItem.defaultProps = {
    text: 'Er werd geen inhoud tekst meegegeven',
}
```

Code fragment 2.7: Statefull klasse component

```
import React from 'react';
import PropTypes from 'prop-types';
import styled from '@emotion/styled';

import logo from '../logo.svg';

const BACKGROUND_COLORS = ['green', 'yellow', 'blue', 'orange', 'red', 'white'];

export const ListItem = (props) => {
    // Maak een specifiek state object met de useState hook
    const [backgoundColor,setBackgroundColor] = React.useState('white');

    // Aanmaken van een callback functie met de useCallback hook
    const setColor = React.useCallback(() => {
        const random =
            Math.floor(0 + Math.random() * (0 + BACKGROUND_COLORS.length));
        setBackgroundColor(BACKGROUND_COLORS[random]);
    }, [setBackgroundColor]);

    return (
        <ListItemContainer
            color={backgoundColor}
            onClick={setColor}
        >
            <LogoWrapper><Logo src={logo} alt="logo" /></LogoWrapper>
            <ContentWrapper>
                <h2>{props.title}</h2>
                <Paragraph>{props.text}</Paragraph>
            </ContentWrapper>
        </ListItemContainer>
    );
};

ListItem.defaultProps = {
    text: 'Er werd geen inhoud tekst meegegeven',
}

ListItem.defaultProps = {
    title: PropTypes.string.isRequired,
    text: PropTypes.string
}
```

Code fragment 2.8: Statefull functioneel component

2.6 Lifecycle methoden

De lifecycle methoden stellen de levenscyclus van een component voor, van wanneer hij 'geboren' wordt (mounten op het DOM) tot hij 'sterft' (unmounten van het DOM). Er bestaan veel verschillende lifecycle methoden en ze kunnen opgedeeld worden in vier fasen:

Initialization: De fase waarin het component zichzelf klaarmaakt door zijn properties binnen te halen en state te initialiseren

Mounting: Het component wordt gecreëerd en in het DOM geplaatst. De render methode wordt voor de eerste keer uitgevoerd

Updating: Props en/of State data worden geüpdatet door interactie van de gebruiker met het component. Er gebeurd een rerender van het component

Unmounting: De laatste fase in het proces waarin het component van het DOM wordt gehaald. Duid het einde van de levenscyclus aan

Alle methoden samengevoegd vormen de levenscyclus van een component en veranderen doorheen de tijd samen met de state van een component. Lifecycle methoden worden bijvoorbeeld gebruikt om bepaalde effecten uit te voeren wanneer de state veranderd.

2.7 Framework concepten

Dit hoofdstuk haalt enkele veel gebruikte concepten aan die belangrijk zijn binnen React of veralgemenend javaScript.

2.7.1 Destructuring

Een functionaliteit die werd geïntroduceerd met de komst van ES6 (ECMAScript 6 of javaScript 6) is destructuring. Het staat toe om waarden uit arrays te halen en ze toe te kennen aan lokale of globale variabelen. Hetzelfde principe geldt voor objecten, waar de properties van een object kunnen worden toegekend aan variabelen.
Het principe wordt in React veelal gebruikt voor leesbaarheid en het ontleden van props en state. In Code fragment 2.9 op pagina 32 is te zien hoe de syntax wordt gebruikt.

```
1   const array = ['value1', 1, 'value2', 2];
2   // Je kan waarden overslaan met de ',' operator
3   const [one, two,, three] = array;
4
5   console.log(one); // output: value1
6   console.log(three); // output: 2
7
8   const object = {a: 'this', b: 'is', c: 'amazing'};
9   // Bij destructuren van een object spreek je de properties expliciet aan
10  const {one, two, three} = object;       // Fout
11  const {a, b, c} = object;               // Juist
12
13  console.log(a, b, c); // output: this is amazing
14  console.log(b, a, c); // output: is this amazing
```

Code fragment 2.9: Destructuring van een array en object

2.7.2 Spread operator

Samen met destructuring werd ook de spread operator geïntroduceerd met de komst van ES6. Op dit moment is het enkel ondersteund voor arrays. Het geeft de mogelijkheid om arrays te kopiëren, waarden toe te voegen en uit te lezen op een eenvoudig makkelijk leesbare manier.
Spread operator wordt veelal gebruikt voor het benoemen van state, waarden uit te lezen en het doorgeven van properties naar een kind component door zijn ouder. Code fragment 2.10

op pagina 33 licht het gebruik en syntax toe.

```
    const array = ['value1', 1, 'value2', 2];
    const extendedArray = [...array, 'newValue'];

    console.log(array); // output: ['value1', 1, 'value2', 2]
    console.log(extendedArray); // output: ['value1', 1, 'value2', 2, 'newValue']

    // Uitlezen van de array waarden
    console.log(...extendedArray); // output: value1 1 value2 2 newValue

    // Doorgeven van properties uit een object aan een component
    const props = {firstName: 'John', lastName: 'Walker', gender: 'Male'};
    return (
        <PersonComponent {...props} />
    );
```

Code fragment 2.10: Gebruik van de spread operator

3. Theoretische performantie

Doorheen de jaren is er een steeds hogere standaard gelegd voor sites en daarbij ook veralgemenend, het internet. Gebruikers verwachten snelheid wanneer ze zich op het web begeven en dat reikt ook door tot de hedendaagse moderne webapplicaties. Performantie is een fundamenteel aspect geworden voor het leveren van een goede UI.
Alles wat visueel aan de gebruiker wordt getoond heeft invloed op hem, zowel direct als indirect. Er bestaan talloze aspecten waarmee rekening kunnen gehouden worden. In dit hoofdstuk wordt het subjectieve onderworpen aan het objectieve waarbij de meest prominente aspecten aan bod komen.

3.1 Waarom performatie?

Voor iets kan weerlegt worden over de daadwerkelijk te creëren performantie is het goed om stil te staan bij de impact. Bedrijven met een eigen webportaal zijn er zich van bewust dat verandering nodig is om aan de eisen van gebruikers te voldoen. Er wordt veel gezegd over performantie, maar waarom is het net zo belangrijk voor hun?

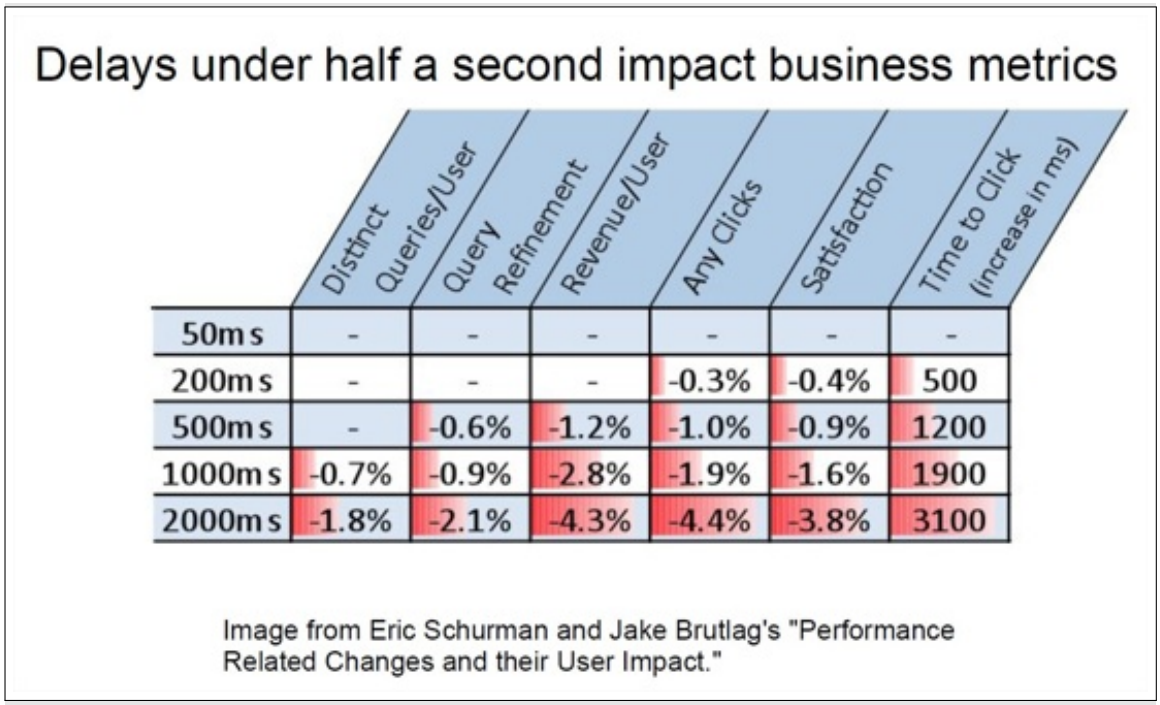
3.1.1 Verkoopcijfers

De enorme groei van het WWW en dagelijkse gebruik binnen de maatschappij zorgt voor een nieuw medium voor bedrijven om hun boodschap aan het doelpubliek over te brengen. Websites zijn een vaste waarde geworden binnen het businessmodel van een onderneming. Elke bezoeker van een website kan gezien worden als een potentiële klant, afhankelijk van de aard onderneming wiens website wordt bezocht.

De oude technieken voor het maken van sterke feature rijke sites kunnen niet meer het verwachte resultaat bieden in een steeds evoluerende niche. Een bedrijf heeft alle belang bij het onderhouden van zijn potentiële klanten en hun de beste ervaring te bieden wanneer ze naar hun website surfen.
In een artikel van Sam Meder (2017) voor Pintrest engineering worden optimalisaties in kaart gebracht op basis van een vooraf onderzochte metriek als vertrekpunt. The Telegraph deed, zoals verklaard in het artikel van Palmer (2016), een inverse studie waarbij een synthetische vertraging van de laadtijd een daling aantoont van het aantal pagina bezoeken wanneer de laadtijd trager wordt.

3.1.2 SEO

Search engine optimization is een prominent begrip in de bedrijfswereld sinds iedereen steeds meer onderling verbonden is door het internet en websites een groot deel uitmaken van de omzet voor bedrijven. Het principe van SEO is om een aantal doelstellingen op te stellen voor het verbeteren van de website en zo eerder in zoekopdrachten te verschijnen van zoekmachines zoals Google, Yahoo!, Bing,...
Google maakte in 2010 bekend dat er rekening zal gehouden worden met de snelheid van een website in het Google algoritme. Wat voor bedrijven de performantie van hun website nog zo belangrijk maakt. In het artikel van McGee (2010) over de aanpassing van het Google algoritme wordt aangetoond dat laadtijd vertragingen van meer dan een halve seconde al invloed kunnen hebben op business waarden, zoals te zien is in figuur 3.1 op pagina 36. Dit maakte het voor Google relevant genoeg om op te nemen in hun algoritme.



Figuur 3.1: Laadtijd vertragingen met impact, Bron: McGee (2010)

3.2 Metingen zijn belangrijk

Metingen zijn de basis waarop optimalisaties worden uitgevoerd, dit is in het geval van performantie niet anders. Weten waar werkpunten liggen en op basis daarvan een plan van aanpak opstellen zorgt voor doelgerichte optimalisatie. Arbitrair verandering aanbrengen zonder een duidelijke kijk te hebben op wat noodzakelijk is stelt zichzelf in vraag.

3.2.1 De grootste factor

In de sectie 3.1 op pagina 35 werd er al nadruk gelegd op de impact die een slechte performantie kan achterlaten voor een bedrijf, maar wat de gebruiker vindt geeft het uiteindelijke verdict. Waar het om draait voor de hedendaagse gebruiker van het WWW is snelheid. Gebruikers willen zoveel mogelijk informatie op hun beeld krijgen op een snelle en efficiënte manier.
In het artikel van Everts (2016) werden conclusies getrokken uit Google Analytics en data van derde partijen die vrijwillig bereid waren hun specifieke data te delen. De conclusies waren unaniem, te lange laadtijd zorgt voor ontevredenheid bij de gebruiker. Trage sites (19 seconden laadtijd) hebben zeer korte gebruikerssessies en vervolgens ook een hoger debouncepercentage, waarbij snelle sites (5 seconden laadtijd) 70 procent langere gebruikerssessies en 35 procent minder debouncepercentage ondervinden.

3.2.2 Metrieken

Wat zij de metrieken waarmee rekening gehouden worden in welzijn van de gebruiker? Dit kan gaan van hoe snel de eerste pixels op het scherm verschijnen tot de reactietijd van een interactie die de gebruiker uitvoert.

FCP (First Contentful Paint)

De tijd totdat een gebruiker iets visueel op het scherm te zien krijgt wordt is the frist contentful paint. Het is een maatstaf om een idee te geven wanneer de gebruiker voor het eerst inhoud zal te zien krijgen die hij of zij kan consumeren.

FMP (First Meaningful Paint)

In aanvulling op een first contentful paint wordt ook rekening gehouden met de tijd tot het eerste deel van de website waar een gebruiker iets aan heeft, deze belangrijke delen van een website worden hero elementen genoemd. In tegenstelling tot first contentful, wat nog maar een enkel woord kan zijn dat op het scherm verschijnt, is the first meaningful veel verfijnder. Wanneer een gebruiker het belangrijkste deel van een website eerst te zien krijgt aan een verwachte snelheid zal er minder aandacht geschonken worden wanneer de rest van diezelfde pagina verschijnt. Waardoor een website veel sneller oogt.

Interactie

Interactie is een algemeen begrip en wordt onderverdeeld in verschillende criteria. Met het steeds sneller evolueren van de niche, dat het internet is, worden doelstelling opgelegd die de industrie verwacht behaald te zien.
In figuur 3.2 op pagina 38 worden de doelstellingen in verband met interactie aangekaart, bron van deze informatie komt is het artikel van Taub (2017).

Time to interact: Geeft een indicatie weer van de tijd die nodig is vooraleer de gebruiker interactie kan voeren met delen van de website, zoals bv. een input veld selecteren, doorklikken op een link, animaties starten,...

Interaction frame rate: Meeste schermen hebben een refresh snelheid van 60fps, wanneer er onder 60 wordt gegaan kan dit snel aanzien worden als *laggy* voor een gebruiker. De oorzaak hiervoor is vaak teveel schrijven/lezen of een verrommeling van het DOM.

Interaction response time: Een abstracte maatstaf voor de snelheid waarmee een interactie word waargenomen door de gebruiker. Hoe lager deze waarde ligt, hoe sneller een website zal waargenomen worden.

Industry defined targets

METRICS		
Time to interact	Interaction frame rate	Interaction response time
< 1.5s	60fps	< 100ms

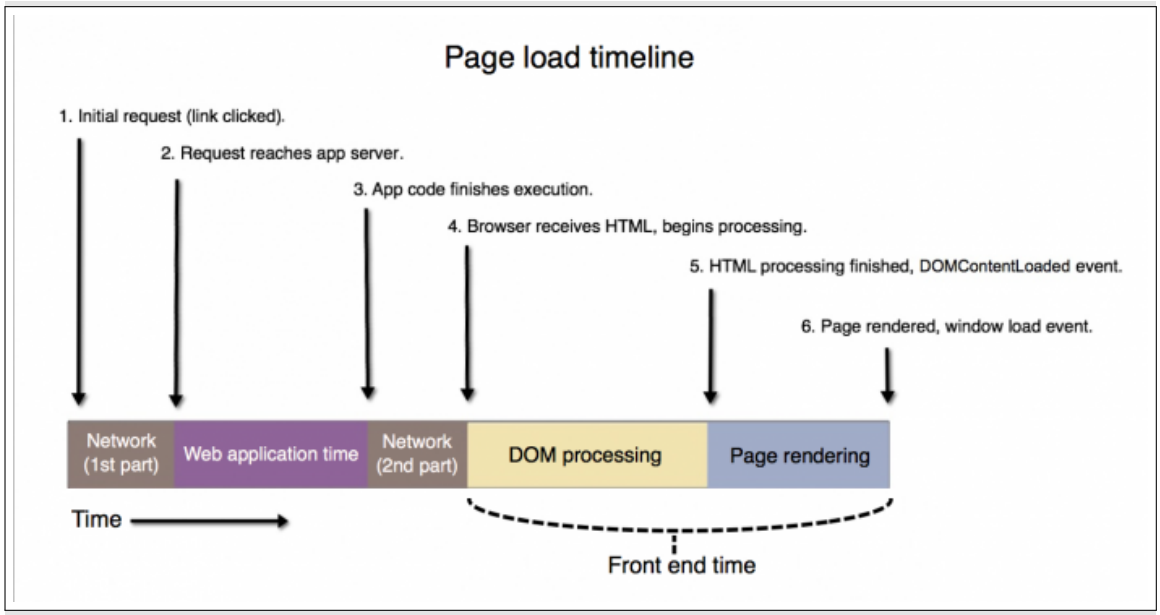
Figuur 3.2: Verwachte waarden opgelegd door de industrie

Laadproces

Wanneer een website wordt geladen worden er een bepaald aantal fasen doorlopen vooraleer de eerste pagina op het scherm tevoorschijn kan komen. In figuur 3.3 op pagina 39 worden de fasen aangetoond op een as doorheen de tijd. Voor deze scriptie ligt de focus op frontend tijd.

DOM processing: Na het verkrijgen van de HTML, die via requests van de server werd gehaald, wordt deze omgezet naar een DOM.

Pagina rendering: Wanneer het DOM is aangemaakt begint de browser met het renderen van de pagina op basis van het DOM. In deze fase wordt de inhoud verwerkt en nadien kan de browser het window load event initialiseren.



Figuur 3.3: Tijdlijn voor het laden van een pagina, Bron: Relic (g.d.)

3.2.3 Tools

Zoals voordien al aangegeven in sectie 3.2.1 op pagina 37 is snelheid alles bepalend en dit is te meten op basis van verschillende metrieken, welke al reeds besproken werden in sectie 3.2.2 op pagina 37. Voor het daadwerkelijk meten van deze maatstaven zijn er verschillende tools op de markt. Zo zijn er tools voor performantie tests manueel of geautomatiseerd kunnen zijn. Ook de Chrome DevTools bieden een geïntegreerde tool voor het testen van performantie. Hieronder enkele bekende voorbeelden van dergelijke tools:

MachMetrics: Een tool die geautomatiseerde testresultaten voor snelheid dagelijks opmaakt en doorstuurt. Het is een best practice om de statistieken van je website op gepaste tijdstippen te analyseren. Het internet en daarbij ook websites zijn in continue verandering.

WebPageTest: Online platform voor het testen van allround performantie in verschillende browsers op populaire besturingssystemen.

Lighthouse: Open-source tool geïntegreerd in Chrome DevTools, maar kan ook als node module of in command line uitgevoerd worden. De tool stelt verschillende audits ter beschikking om de kwaliteit van websites na te gaan. Het genereert rapporten voor de prestaties van uitgevoerde audits.

4. Praktische performantie

React biedt talloze oplossing voor het bevorderen van performantie, om een kijk te krijgen op enkele van de meest courante technieken wordt er in dit hoofdstuk een overzicht opgesteld. Enkele van de oplossingen die worden aangehaald zijn nagebootst binnen een testomgeving, waarna de resultaten worden geanalyseerd. Neem op voorhand aan dat dit niet de enige mogelijkheden zijn voor het optimaliseren van performantie. De voorgestelde technieken zijn onderverdeeld in 3 categorieën:

Native: De oplossingen die React ons zelf bied als zijnde deel van de core functionaliteiten

Framework: Mogelijkheden die we specifiek kunnen toepassen door het gebruik van React

Tools: Functionaliteiten afkomstig van een derde partij die we kunnen integreren in de codebase

4.1 Native

Door React in scope te plaatsen is er toegang tot de React top-level API. Zoals wordt aangegeven in figuur 4.1 op pagina 42 is de import het aanspreekpunt voor de API en maakt alle React functionaliteiten aanspreekbaar. Zoals wordt aangegeven in figuur 4.1 op pagina 42.

```
import React from 'react';  8.5K (gzipped: 3.4K)
import styled from '@emotion/styled';  24.5K (gzipped: 9.8K)
import PropTypes from 'prop-types';  1.6K (gzipped: 838)
```

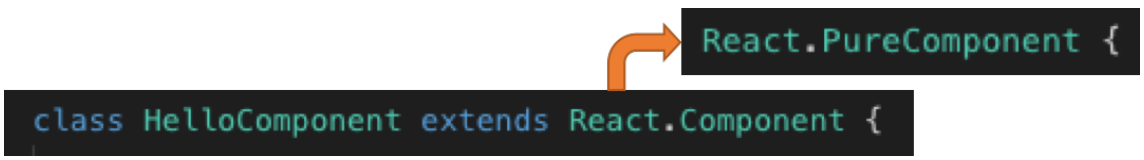
Figuur 4.1: React in scope

4.1.1 Gebruiken van de Fragment tag

Een probleem dat vaak voorkomt wanneer het gaat om performantie is het lezen en schrijven naar het DOM. DOM manipulaties zijn op zich zeer belastend en vragen veel fps-tijd. Als het DOM vol wordt gestoken met nestingen van nodes ontstaat er vanzelfsprekend vertraging.
In sectie 2.3 op pagina 27 werd uitgelegd hoe React fragments het DOM vrijhouden van een overrompeling aan nutteloze tags, waaronder vooral tags.

4.1.2 Extenden van React.PureComponent

Pure component is exact hetzelfde als een gewoon component in React. Het verschil tussen beiden ligt bij het aanroepen van de lifecycle methode shouldComponentUpdate(), die op een andere manier uitgevoerd wordt. Bij een normaal component zal er altijd een re-render uitgevoerd worden wanneer er iets aan props of state veranderd. In een pure component wordt er in de shouldComponentUpdate functie aan shallow comparison gedaan van props en state.
Bij shallow comparison worden de waarden en referenties van de vorige props en state vergeleken met de volgende. Deze controle is goedkoop, zeker in vergelijking met het telkens re-renderen van een component. Het nadeel is dat er geen controle wordt gedaan voor de waarden in geneste objecten.
Het omzetten gaat ervoor zorgen dat er minder re-renders gebeuren. Dit wordt ook overgedragen naar de kinderen van een pure component, daardoor is het af te raden voor componenten met kinderen, tenzij al deze ook pure components worden.
Het is een best practice om een gewoon klasse component om te zetten wanneer het eenvoudige state en props bevat. Het gebruik van een pure component geeft een performantie boost zonder dat er veel aanpassingen moeten worden gedaan aan de initiële code.
Figuur 4.2 op pagina 42 geeft aan waar de aanpassing plaats vindt.



Figuur 4.2: Veranderen van component naar pure component

4.1.3 Wikkelen in React.Memo

Memoization is een techniek voor het optimaliseren van snelheid in computer programma's. Door de dure functies, die regelmatig worden uitgevoerd, op te slaan in het cache geheugen kunnen ze veel sneller uitgevoerd worden in de toekomst. React memo werkt volgens hetzelfde principe.
React Memo is gelijkaardig aan het extenden van een pure component, het geeft controle over het renderen van functionele componenten. Als een functioneel component wordt gewikkeld in React.memo() worden bij elke re-render enkel de UI-elementen opnieuw gerenderd die geüpdatet props nodig hebben.

4.1.4 React lazy

Het integreren van code-splitting helpt voor het lazy loaden van componenten. Alles tegelijk in één keer downloaden voor het inladen van de pagina is niet optimaal. UI elementen worden ingeladen zonder dat er ooit interactie is met de gebruiker. Tijdens lazy loading wordt gewacht om code in te laden tot wanneer het wel degelijk nodig is.
Figuur 4.1 op pagina 43 toont een voorbeeld voor het lazy loaden van een component.

```
1   // MyComponent.js
2   class MyComponent extends Component{
3       render() {
4           return (<div>MyComponent</div>);
5       }
6   }
7
8   // App.js
9   import React from 'react';
10
11  const MyComponent = React.lazy(()=>import('./MyComponent.js'))
12      function AppComponent() {
13          return (
14              <div>
15                  <MyComponent />
16              </div>
17          );
18  }
```

Code fragment 4.1: Lazy loading a component

Suspense

Suspense is een concept dat werd geïntroduceerd met de komst van React lazy in React versie 16.6. Het bied een fallback functie voor componenten die lazy loaded zijn. Tijdens het dynamisch inladen van een component kan er een laad tijd ontstaan afhankelijk van de document grootte. Figuur 4.2 op pagina 44 geeft aan hoe die laadtijd wordt opgevangen met suspense.

```
// MyComponent.js
class MyComponent extends Component{
    render() {
        return (<div>MyComponent</div>);
    }
}

// App.js
import React, { Suspense } from 'react';

const MyComponent = React.lazy(()=> import('./MyComponent.js'));
function AppComponent() {
    return (
        <Suspense fallback={<div>Loading ...</div>} >
            <MyComponent />
        </Suspense>
    );
}
```

Code fragment 4.2: Lazy loading met suspense

4.2 Framework

4.2.1 Propagatie

Bij propagatie geven we properties mee aan een component, om het een bepaalde vorm te geven aan het deel van de UI die geretourneerd wordt. In praktijk is het mogelijk om alle props die een ouder bezit door te geven aan zijn kinderen door alle props te spreaden in de component tag. Sectie 2.7.2 op pagina 32 geeft aan hoe de spread operator functioneert en hoe props worden gespread in een component.
Het spreaden van het volledige prop object in de component tag is niet altijd een best practice. Wanneer het props object exact de gewenste waarden bevat om door te geven aan een volgend component is er geen probleem. Als er geen zekerheid is over de inhoud van het props object is het een bad practice om deze te spreaden. Door het doorsturen van een inhoudelijk onbekend props object is er grote kans dat we het DOM vervuilen door properties zonder afhankelijkheid toe te kennen aan componenten.

4.2.2 Function calls in de render functie

Figuur 4.3 op pagina 45 toont twee voorbeelden voor het uitvoeren van een functie binnen in de render functie van een component. Het eerste voorbeeld zal telkens een nieuwe referentie aanmaken voor de functie wanneer hij opnieuw gerenderd wordt. Constant nieuwe referenties aanmaken is een bad practice in het performance handboek.
Voor het tegengaan van het steeds opnieuw aanmaken van een referentie maken we een arrow functie aan buiten de render. Wanneer de functie nu wordt aangeroepen wordt de referentie naar de arrow functie bewaard.

```
    // Functie uitvoeren in de render fucntie
    class MyComponent extends React.Component {
        render() {
            return (
                <button onClick={() => { console.log('Hey, you clicked!'); }}>Click me
!</button>
            );
        }
    }

    // Functie uitvoeren buiten de render functie
    class MyComponent extends React.Component {
        const handleClick = () => {
            console.log('Hey, you clicked!');
        }

        render() {
            return (
                <button onClick={this.handleClick}>Click me!</button>
            );
        }
    }
```

Code fragment 4.3: Lazy loading met suspense

5. Conclusie

Uit het onderzoek dat gevoerd is kunnen we concluderen dat er wel degelijk vele technieken bestaan voor het verbeteren van de frontend performantie. Er zijn oplossingen die niet van het framework zelf komen, maar van een derde partij zoals bepaalde libraries.
Het is een goede zaak om een overzicht te hebben van optimalisatie technieken voor een specifiek framework, hier in dit geval React. Waar het onderzoek wat vast loopt is het uitdagen van de reeds bestaande technieken. Alle mogelijke optimalisaties zijn reeds uitgevoerd en onderzocht en om een onderzoek te voeren die werkelijk performantie gaat optimaliseren op een innovatieve manier is een proces van maanden.
Deze scriptie is wel een leidraad voor developers die beginnen met React voor hun frontend uitwerkingen. Er kunnen lessen getrokken worden uit het gevoerde onderzoek en zorgen voor een optimaal gebruik van het framework vanaf het begin.

A. Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

A.1 Introductie

Performantie is voor een bedrijf dat zich specialiseert in web- en native applicaties een absoluut werkpunt bij elk project dat ze aangaan. Dit wordt nog groter wanneer het gaat over complexe applicaties. Codifly is een start-up en specialiseert zich net op dit gebied. Het heeft er alle baad bij een grondig onderzoek naar performantie te doen binnen het framework waar ze hun applicaties op baseren, dit zijnde React.

Voor een bedrijf als Codifly is het belangrijk om consistent sterke, nauwkeurige, dynamische en vooral performante producten aan hun klanten af te kunnen leveren. Om dit niveau van applicaties te kunnen garanderen is het belangrijk om grondig onderzoek te doen. Een specifiek, maar toch breed onderzoek, waar we ons afvragen: Hoe meten we de performantie binnen het framework? Welke effecten hebben diverse performantie verbetering methoden? Kunnen meerdere methoden gecombineerd worden? Hoe herwerken we onze React codebase hiernaar? Zijn er bepaalde 'regels' te hanteren? Zoja, welke regels moeten gehanteerd worden? Bestaan er specifieke technologieën? ...

A.2 State-of-the-art

De performantie binnen React gebaseerde applicaties is goed ontvangen door de meeste developers toen het bijna 3 jaar geleden op de voorgrond kwam. Het is nog steeds een veel gebruikt framework om sterke en complexe applicaties te maken, zowel web als native. Aangezien React een steeds verder uitbouwend framework is, zijn onderzoeken naar performantie binnen React meer dan een must.

Een 3 jaar terug deed De Cock (2016) aan de HoGent een soortgelijke studie, daar noemde hij React een moderne webtechnologie. In het betreffende onderzoek kwam hij tot de conclusie dat het een verfrissend concept is, maar nog niet helemaal op punt staat voor fulltime developers. Nu, na de goede opkomst van React over de voorbije jaren, wordt in deze studie nieuw onderzoek gedaan en analyseert het de vorderingen van het framework.

Er werden al reeds performantie onderzoeken uitgevoerd in React, maar het zijn telkens vergelijkende studies. Deze proberen aan te tonen waarom React, of het andere onderzochte framework, beter zou zijn in bepaalde situaties. Een echt onderzoek naar hoe zwakke performantie aan te pakken, of simpel weg te verbeteren, binnen React zelf wordt niet gedaan. Hierin onderscheid het onderzoek zich van de rest. De studie neemt het probleem concreet aan binnen React.

A.3 Methodologie

Het onderzoek dat wordt uitgevoerd ligt in de loop met de projecten die worden aangegaan binnen de stage. Stage vindt plaats in het hierboven vernoemde stagebedrijf Codifly. De verschillende experimenten die worden uitgevoerd zijn gebaseerd op eigen test applicaties en projecten waar aan bijgedragen wordt binnen het bedrijf. Door observaties, studies en experimenten te doen naar verschillende aspecten binnen React performantie kunnen er antwoorden geformuleerd worden op aangehaalde vragen.

Om de performantie te meten bestaan er tools binnen React zelf, zoals 'react-benchmark' die aangewezen werd door de co-promotor. Er is ook een performance tool ingebouwd in React zelf. Simulaties voor kleine testen worden uitgevoerd op zelf ontwikkelde test applicatie.

A.4 Verwachte resultaten

Een optimalisatie van de codebase die zorgt voor een verbetering van de performantie. Waarbij het hanteren van de methoden een positieve invloed hebben op het optimaliseren van laadtijden, minimaliseren van bundle sizes, code splitting, ...

De resultaten zorgen voor een betere aanpak en sterke basis voor consistente performantie in toekomstige projecten.

A.5 Verwachte conclusies

React is een volwaardig framework dat een goed gebalanceerde performantie biedt voor het bouwen van stabiele en complexe web- en native applicaties. Een framework waar performantie kan verbeterd worden met de juiste hanteringen.

Bibliografie

Aston, B. (2015). A brief history of javaScript. *Medium*.

Carette, P. (2011). Onderzoek: een snelle website = meer bezoekers. *SITEOPTIMO*.

De Cock, S. (2016). React Native Moderne webtechnologieën in native mobiele applicaties. Verkregen van https://catalogus.hogent.be/catalog/hog01:000700301

Everts, T. (2016). Mobile Load Time and User Abandonment. *Akamai Developer*.

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.

Mazaika, K. (2017). Will the demand of developers continue to increase? *Forbes*.

McGee, M. (2010). It's Official: Google Now Counts Site Speed As A Ranking Factor. *Search Engine Land*.

Palmer, O. (2016). How Does Page Load Time Impact Engagement? *Optimizely blog*.

Relic, N. (g.d.). *Page load timing process*. New Relic.

Sam Meder, J. C., Vadim Antonov. (2017). Driving user growth with performance improvements. *Medium*.

Taub, C. (2017). Key front-end performance metrics and how to capture them. *Medium*.