HO
GENT

Faculteit Bedrijf en Organisatie

Write once, render anywhere: een exploratief onderzoek naar React Native Web en Expo

Jay van Mook

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

Promotor:
Johan Van Schoor
Co-promotor:
Sander Goossens

Instelling: Endare BVBA

Academiejaar: 2019-2020

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Write once, render anywhere: een exploratief onderzoek naar React Native Web en Expo

Jay van Mook

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

Promotor:
Johan Van Schoor
Co-promotor:
Sander Goossens

Instelling: Endare BVBA

Academiejaar: 2019-2020

Tweede examenperiode

Woord vooraf

Zonder enige kennis en ervaring over de library React en het framework React Native, begon ik aan deze bachelorproef in opdracht van mijn stageplaats Endare BVBA. Het opende voor mij een nieuwe deur binnen de wereld van web- en mobiele applicatieontwikkeling, één van mijn interesses binnen de ICT sector. Doorheen de periode dat ik aan deze bachelorproef werkte, heb ik enorm veel bijgeleerd over React, React Native en uiteindelijk ook React Native Web, de library waar we ons in deze bachelorproef zullen verdiepen qua exploratie, mogelijkheden, limitaties, problemen, enz. Ik hoop dat dit onderzoek Endare BVBA en andere geïnteresseerde lezers zal inspireren en hen zal bijstaan bij het toepassen van React Native Web en Expo in hun toekomstige projecten.

Ik wil mijn promotor Johan Van Schoor en co-promotor Sander Goossens bedanken voor de kritische feedback op deze bachelorproef, evenals Stijn De Brauwer voor het beantwoorden van enkele vragen die ik had omtrent React Native.

Samenvatting

Op welke technologieën men zich richt voor de ontwikkeling van een applicatie is geen gemakkelijke keuze. Een mobile native applicatie biedt meer controle en betere performantie, maar is niet zo universeel als een webapplicatie. Waarom is het niet vanzelfsprekend dat ontwikkelaars voor beide technologieën een applicatie voorzien? Het is tenslotte een betere ervaring voor de eindgebruiker als de applicatie overal beschikbaar is, maar ten nadele van de tijd en kosten voor de ontwikkeling ervan. In deze studie doen we een exploratief onderzoek naar React Native Web: een cross-technology library dat, in combinatie met het framework Expo, het mogelijk maakt om met behulp van één codebase meerdere applicaties te ontwikkelen voor zowel mobile native platformen als het web. Dit wordt mogelijk gemaakt door het converteren van React Native componenten en API's naar React DOM, zodat een mobile native applicatie in essentie kan draaien in de webbrowser. We onderzoeken in welke mate React Native componenten en API's ondersteund worden binnen React Native Web. Uiteindelijk concluderen we dat de library effectief voor een 'write once, render anywhere' ontwikkelingservaring zorgt aan de hand van een proof-of-concept. Op basis van de bevindingen biedt React Native Web en Expo in theorie voordeel in tijd en kosten, maar vereist het extra inspanning op sommige vlakken wegens niet ondersteunde React Native onderdelen binnen React Native Web. Deze studie vormt een startpunt voor toekomstige onderzoeken binnen de wereld van cross-technology applicatieontwikkeling. Het biedt inzicht in de mogelijkheden en limitaties van React Native Web en Expo. In toekomstige studies kan men onderzoek doen naar de hoeveelheid tijd en kosten die effectief wordt bespaard bij gebruik van cross-technology in vergelijking met traditionele applicatieontwikkeling.

Inhoudsopgave

Lijst van figuren

1. Inleiding

Vandaag de dag wordt er voor bijna elke applicatie twee types ontwikkeld: een webapplicatie voor in de browser en één of meerdere mobile native applicaties voor platformen zoals Android, iOS, Windows, enz. Beide technologieën bieden voor- en nadelen ten opzichte van elkaar: een mobile native applicatie levert bijvoorbeeld meer controle en betere performantie, maar is niet zo universeel als een webapplicatie. Omdat men alle voordelen wil benutten en het grootste bereik van gebruikers wil garanderen, is het belangrijk om een applicatie voor beide technologieën beschikbaar te stellen. Als nadelig gevolg neemt het ontwikkelingsproces aanzienlijk meer tijd, kosten en inspanning in beslag.

Op gebied van mobile native applicaties moet men verschillende zaken overwegen. Android en iOS zijn de twee populairste platformen, maar beide hebben een verschillende programmeertaal en ontwikkelingsproces. Bouwt men een applicatie voor Android, iOS of beide? Hiervoor komt cross-platform of hybrid applicatieontwikkeling als oplossing naar voren. Voorbeelden van zo'n frameworks zijn Xamarin (de Icaza, Friedman & Microsoft, 2011), React Native (Walke & Facebook, 2015) en Ionic (Sperry, Lynch & Co., 2012). Deze maken gebruik van één codebase voor het ontwikkelen van meerdere applicaties voor verschillende mobile native platformen.

1.1 Probleemstelling

Toch lossen cross-platform of hybrid applicatieontwikkeling frameworks niet het totaalplaatje op. Ontwikkelaars zitten nog steeds met het probleem dat men twee aparte applicaties moet ontwikkelen voor twee verschillende werelden: mobile native applicatieontwikkeling én webapplicatieontwikkeling. De laatste jaren zijn enkele oplossingen boven

water gekomen die dit probleem aanpakken. Ten eerste is er Flutter (Google, 2017), ook een cross-platform framework dat sinds recent bezig is aan de ondersteuning voor het web. Daarnaast bestaat er ook React Native Web: een alleenstaand project van Gallagher (2015) dat React Native tot leven brengt op het web. Beide hebben geen officiële benaming zoals 'cross-platform' of 'hybrid', dus verwijzen we in deze studie naar 'cross-technology' als omvattende term voor zo'n framework of library en dit type applicatieontwikkeling.

Endare BVBA (Goossens, 2012) is een bedrijf dat gespecialiseerd is in het ontwikkelen van digitale producten zoals websites en mobiele applicaties, evenals innovatieve projecten waarin big data, artificiële intelligentie, chatbots, enz. aan bod komen. Kunnen Endare BVBA en andere ontwikkelaars, die actief zijn binnen de sector van web- en mobile native applicatieontwikkeling, voordeel halen uit een cross-technology framework of library dat zowel web- als hybrid applicatieontwikkeling samenvoegt? Op het eerste zicht zou er minder tijd, kosten en inspanning nodig zijn voor de ontwikkeling van een applicatie. Maar hoe precies zit zo'n framework of library in elkaar en zit er een addertje onder het gras?

Endare BVBA maakt onder andere gebruik van libraries en frameworks zoals React (Walke & Facebook, 2013) voor webapplicatieontwikkeling en React Native voor hybrid applicatieontwikkeling. Daarom doen we in deze studie een exploratief onderzoek naar React Native Web, zodat Endare op basis van dit gegeven te weten komt of er voordeel valt te halen uit het ontwikkelen van applicaties met behulp van React Native Web.

1.2 Onderzoeksvraag

We onderzoeken in welke mate een applicatie kan ontwikkeld worden als mobile native applicatie voor native platformen én als webapplicatie voor in de browser via één codebase met behulp van de library React Native Web.

Naast de onderzoeksvraag willen we ook weten hoeveel extra inspanning er nodig is om de applicatie volledig operationeel te maken op beide technologieën. Anders geformuleerd, heeft de React Native applicatie extra hulp nodig om te functioneren in de webbrowser? Zijn er hiervoor drastische wijzigingen nodig in de code? Worden er onderdelen van de applicatie beïnvloed?

We bekijken ook algemeen tot op welke hoogte cross-technology een oplossing is voor het probleem van twee aparte ontwikkeling technologieën: web- en hybrid applicatieontwikkeling.

1.3 Onderzoeksdoelstelling

In de vorm van een vergelijkende studie overlopen we de componenten en API's van React Native, of met andere woorden, de bouwstenen van een React Native applicatie, en vergelijken deze met de respectievelijke onderdelen van React Native Web. We onderzoeken of deze al dan niet ondersteund worden en indien er limitaties of gebreken aanwezig zijn.

Vervolgens passen we een proof-of-concept toe om de onderzoeksvraag te beantwoorden. Vertrekkende vanuit een React Native applicatie, proberen we deze met behulp van React Native Web operationeel te maken in een webbrowser.

1.4 Opzet van deze bachelorproef

In hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein. Op basis van een literatuurstudie controleren we of er al dan niet onderzoek is gedaan naar React Native Web en cross-technology in het algemeen.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

Hoofdstuk 4 voorziet een beknopte geschiedenis van de universele React ontwikkelingscultuur en React Native Web, evenals de werking van de cross-technology library. Vervolgens bekijken we twee manieren voor het opzetten van een project met React Native Web: Create React App of via Expo. Daarna vergelijken we alle componenten en API's van React Native ten opzichte van React Native Web. Uiteindelijk bestuderen we ook enkele cross-technology features dat Expo voorziet binnen zijn framework.

In hoofdstuk 5 verkennen we de applicatie die als proof-of-concept zal bepalen of React Native Web het mogelijk maakt om via één codebase meerdere applicaties te ontwikkelen voor verschillende technologieën. Onderweg voorzien we oplossingen voor enkele problemen die zich voordoen.

Tenslotte wordt een antwoord geformuleerd op de onderzoeksvragen en een conclusie toegelicht op het onderzoek in hoofdstuk 6. Daarbij wordt er ook een aanzet gegeven voor toekomstig onderzoek als gevolg op deze studie.

2. Stand van zaken

Kosten, tijd en inspanning zijn belangrijke factoren voor de ontwikkeling van een applicatie voor ICT bedrijven zoals Endare BVBA. Een eerste probleem zijn de verschillende mobile native platformen zoals Android, iOS en Windows waarvoor mobiele applicaties kunnen ontwikkeld worden. Cross-platform of hybrid applicatieontwikkeling lost dit probleem gedeeltelijk op. Dit is aangetoond in de studie 'Real Challenges in Mobile App Development' van Joorabchi, Mesbah en Kruchten (2013) en reduceert als gevolg de kosten en tijd voor de ontwikkeling van een mobile native applicatie zoals bewezen in 'An Introduction to Hybrid Platform Mobile Application Development' door Khandeparkar, Gupta en Sindhya (2015).

Cross-platform of hybrid applicatieontwikkeling frameworks zoals Xamarin, React Native en Ionic worden hierdoor als waardige keuzes beschouwd door ontwikkelaars voor het bouwen van een applicatie in de afgelopen jaren. Voor React Native zijn bijvoorbeeld verscheidene artikels zoals 'React Native for Android: How we built the first cross-platform React Native app' van Facebook (2015) te vinden op het internet en allerlei handleidingen, documentatie en API's beschikbaar op de officiële website (Facebook, 2020). Talloze studies hebben ook onderzoek gedaan naar de performantie en bruikbaarheid van cross-platform frameworks waaronder React Native door Hansson en Vidhall (2016). Uit dit onderzoek kan men besluiten dat React Native een aanbevolen framework is om snel applicaties te bouwen voor meerdere native platformen. 75% van React Native kan gebruikt worden voor zowel Android als iOS. Op gebied van performantie presteren native applicaties wel beter dan hybrid applicaties. Andere studies vergelijken verschillende hybrid applicatieontwikkeling frameworks zoals het onderzoek van Dalmasso, Datta, Bonnet en Nikaein (2013). Hieruit concludeerde men dat Adobe PhoneGap, op gebied van performantie minder geheugen, processor capaciteit en energie gebruikt tegenover andere cross-platform frameworks en algemeen ook een betere ontwikkelingservaring bezorgt.

Een tweede probleem is dat ontwikkelaars naast een hybrid applicatie ook een webapplicatie moeten ontwikkelen om alle voordelen van beide technologieën te benutten en een zo groot mogelijk bereik van gebruikers te garanderen. Als we verder kijken naar libraries en frameworks die vallen onder onze categorie 'cross-technology', dan is er verrassend weinig materiaal te vinden op het internet aangezien deze wereld nog volop moet openbloeien. Zowel Flutter als React Native Web beschikken niet over exploratieve onderzoeken naar de achtergrond, mogelijkheden en problemen hiervan. Beide bevinden zich dan ook nog steeds in een ontwikkelingsfase. Artikels zoals 'Flutter For Web: A Complete Guide to Create & Run a Web Application' geven een beknopt inzicht in het opzetten van een project en het creëren van een simpele applicatie met behulp van de web ondersteuning van Flutter (Rehman, 2019). Deze maken gebruik van de documentatie die te vinden is op de officiële website van Flutter (Flutter, 2020b) (Flutter, 2020a). Hetzelfde geldt voor React Native Web met talloze artikels zoals 'Build Mobile-Friendly Web Apps with React Native Web' van Nwamba (2019) en de presentaties van Rayzis (2017) tijdens ReactNext 2017 en Gallagher (2017). Deze bronnen geven slechts een introductie tot de library met een korte uitleg en minimale code voorbeelden.

Algemeen kunnen we uit deze literatuurstudie besluiten dat cross-technology libraries en frameworks nog in hun kinderschoenen staan. Er is hier haast geen onderzoek naar gedaan qua exploratie, bruikbaarheid, performantie, business use cases, enz. Ook vergelijkende onderzoeken die cross-technology applicatieontwikkeling middelen zoals Flutter en React Native Web tegenover elkaar zetten, konden we niet terugvinden. Hetgeen hier het dichtst tegenaan leunt is het artikel van Ancheta (2020) dat enkele informatieve verschillen beschrijft in onder andere de werking en het ontwikkelingsproces.

3. Methodologie

Het onderzoek start met enkele methodes voor het opzetten van een project waarin we React Native Web kunnen benutten. Vervolgens zetten we alle componenten en API's van React Native op een rijtje en onderzoeken of deze al dan niet ondersteund worden binnen React Native Web. Worden ze volledig ondersteund, gedeeltelijk ondersteund of helemaal niet ondersteund? We overlopen en testen alle properties en methodes van alle React Native componenten en API's in een React Native Web applicatie. We valideren of de functionaliteit hetzelfde is op beide ontwikkelingstechnologieën. We bekijken eveneens de documentatie van beide kanten en valideren of er verschillen en gebreken aanwezig zijn. Uiteindelijk verzamelen en categoriseren we de resultaten zodat we een kort besluit kunnen vormen.

Daarnaast overlopen we ook enkele features, grotendeels op hardware niveau, van de Expo SDK. We onderzoeken of deze al dan niet ondersteuning bieden voor het web aan de hand van voorbeeldcode die we uitvoeren in de webbrowser.

Vervolgens proberen we, als proof-of-concept, een React Native applicatie operationeel te maken in de webbrowser met behulp van React Native Web. Op deze manier onderzoeken we of React Native Web het mogelijk maakt om een applicatie te bouwen voor zowel mobile native platformen als het web via één codebase. We overlopen daarbij enkele problemen en oplossingen die onderweg opduiken.

4. React Native Web en Expo

React is een open-source JavaScript library waarmee interactieve UI's (User Interface) kunnen gebouwd worden voor webapplicaties. Het gebruik van herbruikbare en ingekapselde componenten die hun eigen staat beheren, laat de ontwikkelaar toe om gemakkelijk complexe webapplicaties te bouwen. De library update en rendert de componenten ook op een efficiënte manier wanneer hun staat wijzigt. De bouwstenen van componenten zijn React elementen zoals , <p> en <img> die in de React DOM (Document Object Model) worden gerenderd. Algemeen heeft React de denkwijze en manier van ontwikkelen van webapplicaties volledig veranderd.

Twee jaar later werd het framework React Native uitgebracht, dat de principes van React naar mobile native applicatieontwikkeling bracht voor Android en iOS. Dit type frameworks staat ook bekend onder de term 'cross-platform' of 'hybrid' applicatieontwikkeling frameworks. React Native bevat een set van primitieve componenten zoals <View>, <Text> en <Image>. Deze vormen de bouwstenen voor de creatie van eigen componenten en het samenstellen van een applicatie. Deze primitieven worden omgezet naar code en UI onderdelen, specifiek voor het native platform.

Rond dezelfde periode, in de schaduw van React Native, werd de library React Native Web ontwikkeld door Nicolas Gallagher, dat de cirkel compleet maakt en een React Native applicatie terug in de webbrowser doet draaien.

4.1 React Native Web

React Native Web maakt het mogelijk om React Native componenten en API's te gebruiken in een webbrowser met behulp van React DOM. Dit wordt mogelijk gemaakt door React Native primitieven zoals <View>, <Text> en <Image> te converteren naar React elementen of DOM nodes. Hier respectievelijk , <p> en <img>.

Maar waarom maakt React Native Web gebruik van React Native voor het bouwen van een webapplicatie terwijl React specifiek ontwikkeld is voor dit doel? Als we cross-technology willen benutten voor de ontwikkeling van een applicatie die op zowel mobile native platformen als het web draait, dan moeten we ons houden aan de meest strikte en beperkte set van bouwstenen (Marquez, 2019). In dit geval zijn dat de componenten en API's van React Native.

Een tweede reden, wat React Native superieur maakt ten opzichte van React voor het ontwikkelen van universele applicaties, is dat React Native een taal puur en alleen voor UI vormt. De basisonderdelen van deze taal, de React Native primitieven, zijn logisch voor elke visuele UI en onafhankelijk van de technologie waarin deze worden gebruikt. React's primitieven daarentegen zijn niet gemaakt voor een visuele taal te definiëren. Deze zijn eerder bedoeld voor de hypertext structuur van een webbrowser op te stellen en hebben dus geen nut als universele taal voor een UI buiten de browser.

Om het samengevat en eenvoudig te verwoorden, is React Native de betere kandidaat als algemene taal voor de UI op zowel mobile native platformen als het web. Het is ook makkelijker om React Native primitieven te converteren naar React DOM elementen, dan omgekeerd.

React Native Web bevindt zich op het tijdstip van deze studie in beta versie 0.12.2 en is compatible met versie 0.60 of hoger van React Native.

4.1.1 React Native Web applicatie opzetten

Een universele React applicatie opzetten vereist enkele hulpmiddelen waaronder een module bundler zoals Webpack (Koppers, Larkin, Ewald, Vepsäläinen & Kluskens, 2012), een JavaScript compiler zoals Babel (McKenzie, 2014) en een package manager zoals npm (Schlueter, Turner & Marchán, 2010). Daarbovenop moeten deze allemaal geconfigureerd worden. De creatie van een universele React applicatie vergt dus enkele complexe stappen. Gelukkig bestaan er verschillende methodes die dit automatisch voor ons regelen, zodat wij als ontwikkelaar direct kunnen beginnen aan het bouwen van een applicatie.

Een eerste methode is Create React App, een door React aanbevolen hulpmiddel voor het creëren van React applicaties voor beginners en single-page applicaties (Facebook, 2016a). Deze vereist enkele extra manuele stappen voor de integratie van React Native Web ten opzichte van de tweede methode die we daarna bekijken, namelijk Expo (Cheever & Ide, 2013).

Voor eender welke methode is NodeJS een vereiste. We kunnen deze JavaScript runtime environment downloaden en installeren van de officiële website: https://nodejs.org/en. Daarnaast zullen we ook gebruik maken van npm voor de installatie van bepaalde zaken. npm is ten eerste een online repository voor het publiceren van open-source NodeJS projecten of zogenaamde 'packages' en ten tweede een CLI (Command Line Interface) voor het installeren en beheren van deze packages, evenals de versies en dependencies (NodeJS, 2011).

Voor het opzetten van de Android en iOS simulators, waarop de mobile native applicaties draaien, kunnen we afwijken naar de handleiding van React Native die te vinden is op https://reactnative.dev/docs/environment-setup.

4.1.1.1 Create React App

De eerste methode is het genereren van een React applicatie via Create React App om vervolgens de React Native Web library toe te voegen aan het project. We gebruiken hiervoor de CLI van React dat inbegrepen is in npx: een package runner tool die wordt meegeleverd met npm versie 5.2 of hoger. Telkens men het commando uitvoert, wordt de meest recente versie van de CLI gedownload (indien niet aanwezig op het systeem) en uitgevoerd. We geven 'my-app' als naam aan de applicatie en gebruiken de flag –use-npm om gebruik te maken van npm in plaats van standaard yarn, in geval dat dit ook geïnstalleerd is op het systeem. Yarn is ook een package manager en alternatief voor npm. Daarna navigeren we naar de gegenereerde folder en installeren we de React Native Web library via npm vanuit een terminal.

```
1 npx create-react-app my-app --use-npm
2 cd my-app
3 npm install react-native-web
```

We hebben succesvol een React Native Web applicatie opgezet en toegang tot het gebruik van React Native componenten en API's in de code. Als voorbeeld vervangen we de inhoud van App.js met onderstaand voorbeeld.

my-app/src/App.js

```
1  import React from 'react';
2  import { Button, Image, StyleSheet, Text, View } from 'react-native';
3
4  export default function App() {
5      return (
6          <View style={styles.container}>
7              <Image style={styles.image} source={require('./images/reactnativelogo.
                   png')}></Image>
8              <Text style={styles.title}>React Native Web</Text>
9              <Button title='Button'></Button>
10         </View>
11     );
12 }
13
14 const styles = StyleSheet.create({
15     container: {
16         flex: 1,
17         alignItems: 'center',
18         justifyContent: 'center'
19     },
20     ...
21 });
```

Vervolgens moeten we aangeven in welk HTML element het geregistreerde component als root kan draaien met behulp van de AppRegistry.runApplication methode. We vervangen de inhoud van index.js met onderstaande code.

my-app/src/index.js

```
1  import { AppRegistry } from 'react-native';
2  import App from './App';
3
4  AppRegistry.registerComponent('App', () => App);
5  AppRegistry.runApplication('App', {rootTag: document.getElementById('root')});
```

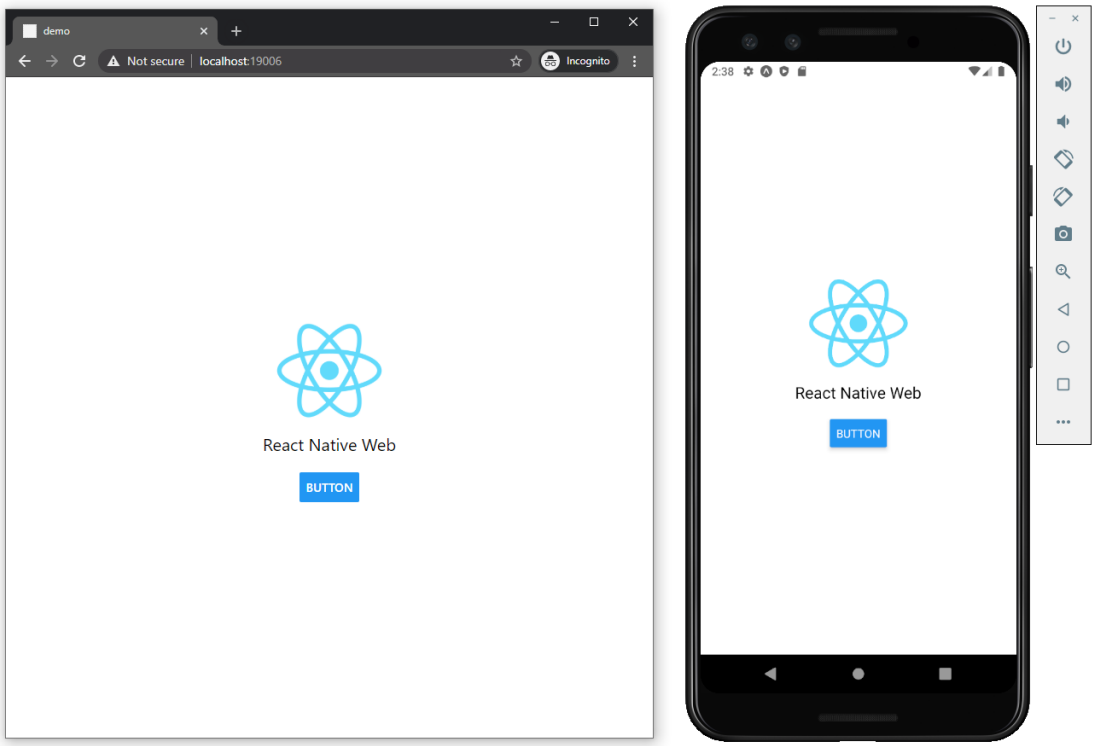
Merk op dat deze bestanden en hun inhoud, letterlijk een kopie zouden kunnen zijn van een React Native applicatie. React Native Web laat ons dus toe om een React Native applicatie in de webbrowser te draaien.

Wanneer we het commando npm start uitvoeren in de terminal, compileert de React Native Web applicatie en krijgen we het resultaat van het voorbeeld zonder problemen te zien in de webbrowser op http://localhost:3000/, op één detail na. Het View component staat niet verticaal gecentreerd in het midden van de webbrowser, ondanks de toewijzing van enkele style properties aan het component. De oorzaak hiervan is het root element dat niet de volledige hoogte van de webbrowser aanneemt. Als laatste stap kunnen we dit oplossen

door onderstaande styling toe te passen op de HTML elementen in index.html, zoals aangegeven in het GitHub issue op https://github.com/necolas/react-native-web/issues/940. Het eindresultaat kunnen we bezichtigen in figuur 4.1.

my-app/public/index.html

```
<style>
    /* These styles make the body full-height */
    html, body {
        height: 100%;
    }
    /* These styles disable body scrolling if you are using <ScrollView> */
    body {
        overflow: hidden;
    }
    /* These styles make the root element full-height */
    #root {
        display: flex;
        height: 100%;
    }
</style>
```



Figuur 4.1: Een voorbeeld van een React Native Web applicatie dat gebruik maakt van React Native componenten op het web (links) en op een native platform (rechts).

4.1.1.2 Expo

Expo is een framework en platform voor universele React applicaties, bestaande uit een set van hulpmiddelen die gebouwd zijn rond React Native en native platformen voor het ontwikkelen van Android-, iOS- en webapplicaties vanuit één codebase (Expo, 2020l).

Voor de installatie van Expo vereist het framework, naast NodeJS, ook Git voor de generatie van een project. Voor beide wordt de meest recente stabiele versie aanbevolen. Git kunnen we downloaden en installeren van de officiële website https://git-scm.com/downloads. Met onderstaande commando's kunnen we vervolgens Expo installeren op het systeem en een nieuw project initialiseren onder de naam myapp.

```
1 npm install --global expo-cli
2 expo init myapp
```

Naast React Native, voorziet Expo sinds SDK versie 33 ook standaard React Native Web integratie in het project. We hoeven deze dus niet manueel te installeren via npm. Indien we andere packages willen toevoegen aan het Expo project, is het aanbevolen om deze te installeren via het commando expo install in plaats van npm install zodat een versie wordt benut die compatibel is met de gebruikte Expo SDK.

Initialisatie

Vooraleer de initialisatie van het Expo project start, krijgen we de keuze uit enkele configuraties, onderverdeeld in twee categorieën: 'managed workflow' en 'bare workflow'. De managed workflow beheert het grootste deel van de Expo hulpmiddelen, de configuraties en het complexe proces achter het bouwen van een applicatie. De bare workflow daarentegen geeft volledige controle over het ontwikkelingsproces aan de gebruiker. Een groot verschil is de toegang tot de native project folders van de Android en iOS applicatie bij de bare workflow, evenals toegang tot het index.js bestand voor het registreren, draaien en stoppen van root components via de AppRegistry API. Verder kunnen we ook opteren voor het gebruik van TypeScript als programmeertaal in plaats van JavaScript. Als men deze optie gebruikt bij de initialisatie van een Expo project, wordt er geen App.js maar een App.tsx bestand gegenereerd. Als voorbeeld kiezen we hier de optie 'blank' van de managed workflow.

Applicatie opstarten

Na generatie van het Expo project, voeren we het commando npm start uit om deze op te starten. Expo CLI start de Metro Bundler op, een JavaScript bundler net zoals Webpack, maar dan specifiek gemaakt voor React Native (Facebook, 2016b). Deze draait op een lokale HTTP server die het project compileert en serveert aan binnenkomende aanvragen. Daarnaast opent Expo CLI ook de Expo GUI in een webbrowser. Vanuit zowel de GUI (Graphical User Interface) als de CLI kunnen we applicaties opstarten voor het web en mobile native platformen waaronder Android en iOS.

Indien we de applicatie willen testen op de lokale computer, kunnen we gebruik maken van de voorziene knoppen en commando's in respectievelijk de GUI en CLI. Deze openen een nieuwe tab in de webbrowser in geval van een webapplicatie en de simulators van Android of iOS in geval van een mobile native applicatie.

Wanneer we de applicatie willen openen op een fysiek apparaat, moeten we de Expo app installeren van Google Play of de App Store (zie links onderaan). Vervolgens scannen we de gegeven QR code via de Expo app voor Android en de Camera app voor iOS om de applicatie te laten draaien op het fysieke apparaat. In geval van een webapplicatie, kunnen we simpelweg surfen naar de voorziene URL met behulp van een webbrowser. Deze bestaat hoogstwaarschijnlijk uit een lokaal IP-adres met poort zoals bijvoorbeeld https://192.168.0.100:19006. Let op dat het protocol HTTPS is in plaats van HTTP.

Expo behandelt alle complexe stappen omtrent de bundlers, compilers, package managers, enz. Helaas hebben we als ontwikkelaar geen toegang tot deze configuraties. Deze zitten diep verborgen in de packages van Expo. Om deze configuraties, evenals andere bestanden zoals index.html, beschikbaar te stellen, voeren we het commando expo customize:web uit in de Expo CLI. Een lijst van beschikbare bestanden wordt weergegeven waartussen we kunnen navigeren met behulp van de bovenste en onderste pijl-toets. Vervolgens drukken we op de rechter pijl-toets om het gewenste bestand te selecteren en tenslotte de ENTER-toets om deze te genereren.

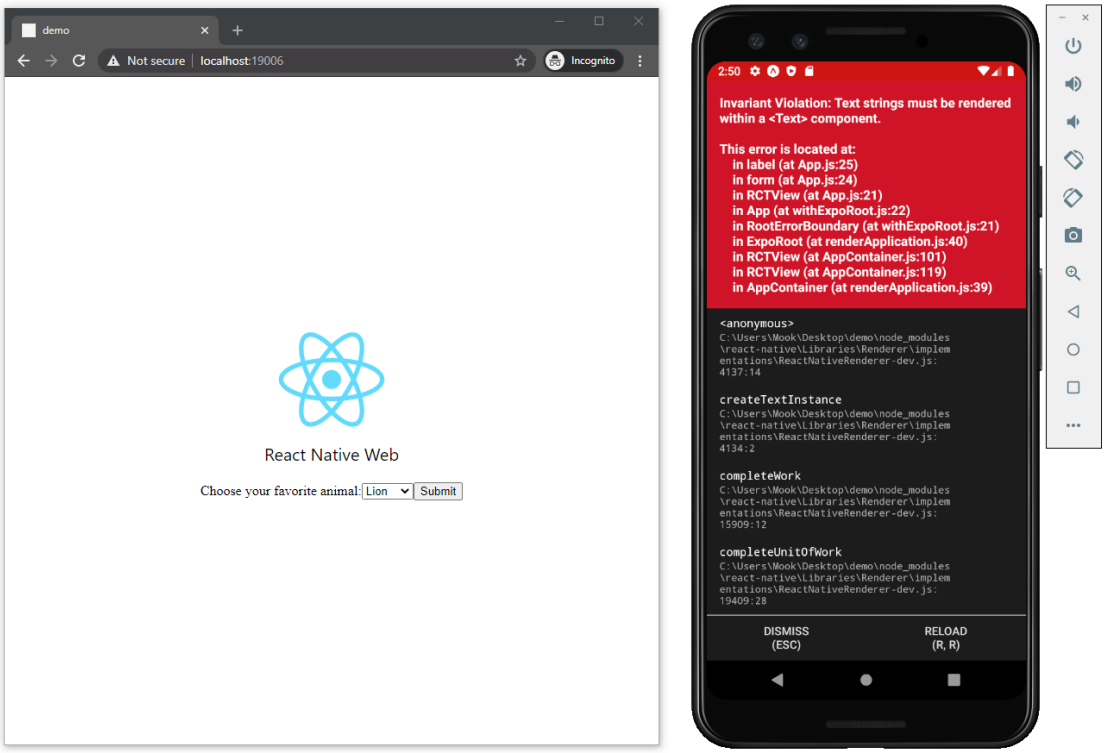
Google Play: https://play.google.com/store/apps/details?id=host.exp.exponent
App Store: https://apps.apple.com/us/app/expo-client/id982107779

4.1.1.3 React gebruiken naast React Native

React Native Web maakt het mogelijk om React Native componenten te benutten op het web, zoals we konden vaststellen bij het voorbeeld in sectie 4.1.1.1. Maar hoe zit het met React dat origineel bedoeld was voor het bouwen van webapplicaties? Na uitvoering van onderstaand voorbeeld tonen we aan dat React elementen nog steeds bruikbaar en functioneel zijn naast React Native componenten en API's binnen een React Native Web applicatie dat draait op het web. Op een native platform compileert de applicatie echter niet, aangezien React Native componenten alleen geconverteerd kunnen worden naar React elementen en niet omgekeerd (zie figuur 4.2).

App.js

```
export default class App extends React.Component {
    constructor(props) {
        super(props);
        this.state = { value: 'Lion' };
    }

    handleChange = (event) => {
        this.setState({ value: event.target.value });
    }

    handleSubmit = (event) => {
        console.log(this.state.value);
        event.preventDefault();
    }

    render() {
        return (
            <View style={styles.container}>
                <Image style={styles.image} source={require('./images/
                    reactnativelogo.png')}></Image>
                <Text>Random text goes here</Text>
                <form onSubmit={this.handleSubmit}>
                    <label>
                        Choose your favorite animal:
                        <select value={this.state.value} onChange={this.
                            handleChange}>
                            <option value="Dog">Dog</option>
                            <option value="Lion">Lion</option>
                            <option value="Horse">Horse</option>
                        </select>
                    </label>
                    <input type="submit" value="Submit" />
                </form>
            </View>
        );
    }
}
```



Figuur 4.2: Het gebruik van React elementen naast React Native componenten en API's is mogelijk in een React Native Web applicatie dat draait op het web (links). Op een native platform compileert deze echter niet (rechts).

4.1.2 Alle componenten en API's op een rijtje

De componenten en API's vormen de bouwstenen van een React Native applicatie. In deze sectie sommen we de onderdelen op en onderzoeken we of deze dezelfde eigenschappen en functionaliteiten bevatten als de componenten en API's van React Native Web.

Elk component en API krijgt een symbool: een v indien het volledig ondersteund wordt door React Native Web, een ~ wanneer het ondersteund wordt maar enkele gebreken bevat en een x als het onderdeel niet ondersteund wordt. In geval dat een component of API het symbool - toegewezen krijgt, betekent dit dat deze niet van toepassing is voor het web of er geen web API's bestaan om dit onderdeel te implementeren op het web. Onderdelen die geen v symbool bezitten, worden verder in detail onderzocht.

Vele componenten en API's binnen React Native bevatten properties en functionaliteiten die specifiek voor native platformen zijn. Het Button component bevat bijvoorbeeld volgende properties in tegenstelling tot het React Native Web component:

- hasTVPreferredFocus
- nextFocusDown
- nextFocusForward
- nextFocusLeft
- nextFocusRight
- nextFocusUp
- touchSoundDisabled

Deze zijn niet van toepassing voor het web en worden dus niet vermeld in de verschillen in componenten en API's tussen de twee technologieën.

Sommige onderdelen bevatten geen documentatie op de website van React Native Web ondanks dat ze toch ondersteund worden en werken binnen een webapplicatie. Deze krijgen ook een gepaste aanduiding.

Components:

- v ActivityIndicator
- v Button
- v FlatList
- ~ Image
- v ImageBackground
- v KeyboardAvoidingView (geen documentatie)
- x Modal
- x RefreshControl
- v SafeAreaView (geen documentatie)
- ~ ScrollView
- v SectionList (geen documentatie)
- - StatusBar
- v Switch
- ~ Text
- ~ TextInput
- v TouchableHighlight
- v TouchableOpacity
- ~ TouchableWithoutFeedback
- v View
- v VirtualizedList (geen documentatie)

API's:

- v AccessibilityInfo (geen documentatie)
- x Alert
- ~ Animated (geen documentatie)
- x Appearance
- ~ AppRegistry
- v AppState
- x DevSettings
- v Dimensions
- v Easing (geen documentatie)
- v InteractionManager (geen documentatie)
- x Keyboard
- x LayoutAnimation
- v Linking
- v PanResponder (geen documentatie)
- v PixelRatio
- v Share
- v StyleSheet
- x Systrace
- v Transforms
- v Vibration

4.1.2.1 Image component

Het Image component van React Native Web beschikt over de properties onLoad en onError die respectievelijk aangeroepen worden wanneer de afbeelding succesvol is geladen en indien er een error is opgetreden. Het Image component van React Native heeft een uitgebreider assortiment aan functionaliteiten. Deze bevat ook de properties onLoadStart en onLoadEnd die respectievelijk aangeroepen worden bij de start en het einde van het laden van een afbeelding, onafhankelijk van een al dan niet opgetreden error.

Daarnaast mist het Image component van React Native Web ook het property onLayout dat wordt uitgevoerd wanneer er wijzigingen gebeuren aan de layout van het component. Althans dat veronderstellen we aangezien deze niet aanwezig is in de documentatie van het Image component. Wanneer we het property effectief benutten in de praktijk, zoals in onderstaand voorbeeld, dan stellen we vast dat deze perfect functioneert en dus wel ondersteund wordt binnen React Native Web.

App.js

```
1  export default function ImageComponent() {
2      const [bool, setBool] = useState(true);
3
4      return (
5          <View>
6              <Image style={bool ? styles.image1 : styles.image2} source={{uri: ...}}
                   onLayout={() => console.log('Layout changed!')}></Image>
7              <Button title='Change layout' onPress={() => setBool(!bool)}></Button>
8          </View>
9      );
10 }
11
12 const styles = StyleSheet.create({
13     image1: {
14         width: 100,
15         height: 100
16     },
17     image2: {
18         width: 200,
19         height: 200
20     }
21 });
```

Verder ondersteunt React Native Web niet het property loadingIndicatorSource dat, net zoals het property source, een databron bevat dat wordt weergeven tijdens het laden van het Image component. React Native Web biedt ook een property extra aan ten opzichte van React Native, namelijk draggable dat aangeeft of een afbeelding al dan niet versleepbaar is en alleen van toepassing is in de webbrowser. Als laatste mist React Native Web de

methode resolveAssetSource.

Een belangrijke laatste limitatie van het React Native Web component is de ondersteuning voor meerdere databronnen. In React Native kan men meerdere verwijzingen, ofwel een URL of een lokaal pad naar een bestand, meegeven aan het property source. Op basis van argumenten in deze bronnen zoals breedte, hoogte, schaal, enz. wordt de beste kandidaat gebruikt om weer te geven in het Image component, rekening houdend met de dimensies hiervan in de layout. Het property source van het React Native component biedt ook ondersteuning aan voor headers en body zodat de ontwikkelaar gegevens kan meesturen met het HTTP request dat de afbeelding ophaalt van een URL. React Native Web ondersteunt dit voorlopig allemaal niet.

4.1.2.2 ScrollView component

Het ScrollView component van React Native Web biedt de meeste en belangrijkste properties aan van het React Native component. Naast enkele kleinschalige niet ondersteunde properties, mist het web component toch een aantal grote mogelijkheden.

Het momentum feature zorgt ervoor dat het scrollen door een ScrollView een versnelling krijgt bij een haastige swipe gesture door de gebruiker. Dit feature staat in verbinding met enkele properties zoals disableIntervalMomentum, onMomentumScrollBegin, onMomentumScrollEnd en decelerationRate dat aangeeft hoe snel de momentum vertraagt in de ScrollView nadat de gebruiker het touchscreen niet meer aanraakt. React Native Web biedt voorlopig geen ondersteuning voor het momentum scroll feature. Het is ook niet logisch dat een momentum feature bestaat voor een webapplicatie aangezien deze vaak wordt bestuurd door een computermuis met scrollwiel. Het is een ander verhaal als de applicatie draait op een apparaat met besturing via touchscreen of touchpad, dan zou het momentum feature wel praktisch zijn.

Een tweede ontbrekende mogelijkheid is het snapping feature dat ervoor zorgt dat het scherm altijd in plaats valt nadat de gebruiker stopt met scrollen zodat er bijvoorbeeld één volledig element zichtbaar is en niet twee gedeeltelijke elementen. De properties van React Native die dit onderbouwen worden niet ondersteund in React Native Web, namelijk snapToInterval, snapToOffsets, snapToEnd, snapToStart en disableScrollViewPanResponder.

In tegenstelling tot React Native, bevat React Native Web naast het property onScroll ook geen extra properties onScrollBeginDrag en onScrollEndDrag. De web library bezit ook niet de properties showsHorizontalScrollIndicator en showsVerticalScrollIndicator, evenals de instantie methode flashScrollIndicators voor de weergave van scroll indicaties.

Tenslotte mist React Native Web de properties refreshControl (aangezien dit niet wordt ondersteund binnen de web library), keyboardShouldPersistTaps dat aangeeft of het toetsenbord verborgen wordt of niet na een aanraking van het touchscreen, evenals invertStickyHeaders dat aangeeft of headers aan de top of bodem van de ScrollView blijven plakken. Binnen React Native Web kunnen headers alleen aan de top blijven plakken via het property stickyHeaderIndices.

4.1.2.3 Text component

In vergelijking met het Text component van React Native, mist React Native Web een groot aantal properties. Ten eerste bevat het niet de mogelijkheid tot het schalen van de tekst naargelang de grootte van het scherm via de properties allowFontScaling en maxFontSizeMultiplier. Het React Native Web component bevat ook geen property ellipsizeMode dat bepaalt hoe tekst wordt afgekapt bij gebruik van het property numberOfLines. Ten tweede mist het enkele functionaliteit properties waaronder onTextLayout en onLongPress. Opnieuw is het property onLayout niet vermeld in de documentatie terwijl deze perfect functioneert binnen React Native Web zoals aangetoond in sectie 4.1.2.1. Als laatste biedt de library ook geen ondersteuning voor het gesture responder feature dat onderbouwd wordt door de properties onMoveShouldSetResponder, onResponderGrant, onResponderMove, onResponderRelease, onResponderTerminate, onResponderTerminationRequest en onStartShouldSetResponderCapture.

4.1.2.4 TextInput component

Net zoals het Text component voorziet React Native Web hier niet de properties allowFontScaling en maxFontSizeMultiplier voor het schalen van de tekstgrootte. Daarnaast bevat het ook geen properties zoals caretHidden voor het verbergen van de flikkerende verticale streep tijdens het typen. Het mist ook een groot aantal functionaliteit properties waaronder onFocus, onBlur, onLayout, onScroll, onTextInput en onEndEditing. Althans zo beweert de documentatie door middel van deze niet te vermelden. Opnieuw geldt hetzelfde verhaal voor de properties onFocus, onBlur en onLayout als bij het ScrollView en Text component. Aan de hand van onderstaand voorbeeld uit te voeren, bewijzen we dat deze properties wel degelijk ondersteund worden.

App.js

```
1 export default function TextInputComponent() {
2     return (
3         <TextInput onFocus={() => console.log('TextInput focused!')} onBlur={() =>
              console.log('TextInput blurred!')}></TextInput>
4     );
5 }
```

Verder worden de properties selectionColor en contextMenuHidden ook niet ondersteund binnen React Native Web. De library mist ook het property textAlign, maar dit kan ingesteld worden via een style object {textAlign: 'center'} binnen het property style. Tenslotte bevat het React Native Web component ook niet de properties selectionColor en contextMenuHidden. Deze laatste zou een belangrijke functionaliteit op het web verwijderen zoals het kopiëren en plakken van geselecteerde tekst via het contextmenu.

4.1.2.5 TouchableWithoutFeedback component

De properties hitSlop, onFocus, onBlur, onLayout, onPressIn en onPressOut van het React Native TouchableWithoutFeedback component worden niet vermeld in de documentatie van React Native Web. Desondanks worden deze toch ondersteund zoals we kunnen vaststellen na uitvoering van onderstaande code.

App.js

```
export default function TouchableWithoutFeedbackComponent() {
    const [bool, setBool] = useState(true);

    return (
        <View>
            <TouchableWithoutFeedback
                hitSlop={{top: 0, left: 0, bottom: 100, right: 0}}
                onFocus={() => console.log('Touchable focused!')}
                onBlur={() => console.log('Touchable blurred!')}
                onLayout={() => console.log('Touchable layout changed!')}
                onPressIn={() => console.log('Touchable pressed in!')}
                onPressOut={() => console.log('Touchable pressed out!')}
            >
                <View style={bool ? styles.layout1 : styles.layout2}></View>
            </TouchableWithoutFeedback>
            <Button title='Change layout' onPress={() => setBool(!bool)}></Button>
        </View>
    );
}

const styles = StyleSheet.create({
    layout1: {
        width: 100,
        height: 100,
        backgroundColor: '#AAAAAA'
    },
    layout2: {
        width: 200,
        height: 200,
        backgroundColor: '#DDDDDD'
    }
});
```

4.1.2.6 Animated API

Het enige dat de Animated API binnen React Native Web ontbreekt ten opzichte van React Native is ondersteuning voor de native driver. Informatie over de native driver en wat deze precies vervult kan men terugvinden in het artikel 'Using Native Driver for Animated' van Duplessis (2017).

4.1.2.7 AppRegistry API

De AppRegistry API is het commandocentrum voor het registreren en uitvoeren van de root components. Deze moeten geregistreerd en uitgevoerd worden via respectievelijk de methodes registerComponent en runApplication. Met behulp van de methode unmountApplicationComponentAtRootTag kunnen draaiende componenten gestopt worden. Binnen Expo projecten met een managed workflow initialisatie is deze API niet van toepassing aangezien er slechts één root component automatisch geregistreerd en uitgevoerd wordt binnen AppRegistry, namelijk het geëxporteerde component uit het App.js bestand.

Als men gebruik maakt van de bare workflow initialisatie, dan zal men enkele methodes van de AppRegistry API moeten missen binnen React Native Web waaronder enableArchitectureIndicator, methodes met betrekking tot secties zoals registerSection, getSectionKeys en getSections, overige get-methodes zoals getRunnable en getRegistry en tenslotte de methodes die zorgen voor de HeadlessTask zoals registerHeadlessTask, registerCancellableHeadlessTask, startHeadlessTask en cancelHeadlessTask.

4.1.3 Deprecated en extra componenten en API's

4.1.3.1 CheckBox component

Het CheckBox component van React Native is deprecated en het react-native-checkbox component van React Native Community neemt zijn plaats in als opvolger (Community, 2019a). Desondanks heeft dit geen invloed op het CheckBox component van React Native Web. Dit staat los van het feit dat deze deprecated is of niet en werkt dus perfect in de webbrowser. Het bevat dezelfde properties als het originele CheckBox component van React Native waaronder color, disabled, onValueChange en value. Dit zijn slechts enkele basis functionaliteiten in vergelijking met het react-native-checkbox component.

Het is onduidelijk hoe React Native Web omgaat met deprecated componenten en API's van React Native in de toekomst. Blijft de library gebruik maken van het deprecated onderdeel of zal men uiteindelijk hiervan afstappen en zorgen voor een eigen implementatie of, zoals React Native, gebruik maken van externe middelen? Helaas biedt het react-native-checkbox component van React Native Community geen ondersteuning voor het web. Voor de ontwikkelaar is dit dus niet evident aangezien men twee verschillende componenten moet benutten bij de ontwikkeling van een cross-technology applicatie. Aan de hand van de Platform module kunnen we de twee varianten onderscheiden en via een conditie bepalen welke wordt uitgevoerd op welke technologie (Native, 2020c). Normaal kunnen we niet ondersteunde componenten en API's van React Native perfect importeren in React Native

Web, op voorwaarde dat deze niet uitgevoerd worden in de applicatie. We komen hier later op terug in het proof-of-concept. Helaas hebben we hier te maken met een uitzondering. Het react-native-checkbox component is niet afkomstig van React Native, maar van React Native Community en dus krijgen we een compilatiefout bij het draaien van de applicatie op het web. De enige manier om dit probleem te omzeilen is het gebruik van specifieke bestand-extensies.

Specifieke bestand-extensies

Om de onoverzichtelijkheid van meerdere herhalende condities met behulp van de Platform module te vermijden en technologie-specifieke code van elkaar te onderscheiden, kan de ontwikkelaar gebruik maken van specifieke bestand-extensies. Bestanden met de .native.js extensie worden opgenomen door de React Native bundler Metro en genegeerd door de web bundler Webpack. De webapplicatie blijft zoals gewoonlijk gebruik maken van de standaard bestands-extensie .js. Dit laat de ontwikkelaar toe om stukken code en componenten, die specifiek voor een bepaalde technologie zijn, op te splitsen van elkaar zodat de leesbaarheid en overzichtelijkheid verbeteren.

Met behulp van specifieke bestand-extensies kunnen we het probleem van het CheckBox component oplossen, zoals aangetoond wordt in onderstaande code.

App.js

```
1 import React from 'react';
2 import CheckBoxComponent from './components/CheckBoxComponent';
3
4 export default function App() {
5     return <CheckBoxComponent></CheckBoxComponent>;
6 }
```

CheckBoxComponent.native.js

```
1 import React, { useState } from 'react';
2 import CheckBox from '@react-native-community/checkbox';
3
4 export default function CheckBoxComponent() {
5     const [toggle, setToggle] = useState(false)
6     return <CheckBox value={toggle} onValueChange={() => setToggle(!toggle)}></
          CheckBox>;
7 }
```

CheckBoxComponent.js

```
1 import React, { useState } from 'react';
2 import { CheckBox } from 'react-native';
3
4 export default function CheckBoxComponent() {
5     const [toggle, setToggle] = useState(false)
6     return <CheckBox value={toggle} onValueChange={() => setToggle(!toggle)}></
          CheckBox>;
7 }
```

Deze bevindingen gelden ook voor andere deprecated componenten en API's die later worden onderzocht.

4.1.3.2 Picker component

Hetzelfde verhaal als het CheckBox component geldt tevens voor het Picker component van React Native. Ook deze is deprecated en vervangen door het react-native-picker component van React Native Community (Community, 2019b). Het React Native Web component werkt opnieuw op dezelfde manier als het originele React Native component en bevat dezelfde properties. Zoals beschreven in alinea 'Specifieke bestand-extensies' van sectie 4.1.3.1, kunnen we in dit geval ook gebruik maken van specifieke bestand-extensies om onderscheid te maken tussen native platformen en het web wanneer elke technologie een specifiek component vereist.

4.1.3.3 ProgressBar component

Het ProgressBar component werd geïnspireerd door het ProgressBarAndroid en ProgressViewIOS component van React Native. Opnieuw zijn deze deprecated binnen de library en overgenomen door React Native Community alternatieven (Community, 2019c) (Community, 2019d). Met behulp van de bevindingen in sectie 4.1.3.1 kunnen we ook hier gebruik maken van het React Native Web component op het web en de React Native Community componenten voor de native platformen.

4.1.3.4 Clipboard API

Het clipboard is een soort geheugenplaats waarin data zoals tekst of een afbeelding tijdelijk wordt bijgehouden wanneer de gebruiker deze kopieert met behulp van het contextmenu of de toetsencombinatie CTRL + C. De inhoud van deze geheugenplaats kan terug opgevraagd worden via de optie 'plakken' in het contextmenu of de toetsencombinatie CTRL + V.

De Clipboard API is gebaseerd op de deprecated API van React Native. Met behulp van de versie van React Native Web kunnen we tekst in de clipboard plaatsen via de methode setString. Het ophalen van deze tekst is helaas nog niet mogelijk op het web. De methode isAvailable controleert of de webbrowser de Clipboard API ondersteunt.

4.1.3.5 Geolocation API

De Geolocation API van React Native Web is gebaseerd op de deprecated API van React Native. Voor deze reden, evenals de ontbrekende documentatie, laten we deze API voor wat het is. In sectie 4.1.5.3 bekijken we het Location feature van de Expo SDK die dezelfde functionaliteit vervult.

4.1.3.6 I18nManager API

Met de I18nManager API kan de ontwikkelaar de directie van de layout aanpassen. Tekst is bijvoorbeeld uitgelijnd naar de rechterkant in de Arabische taal. De API bevat de properties isRTL en doLeftAndRightSwapInRTL om respectievelijk te controleren of de applicatie zich in RTL modus bevindt en of het de stijlen moet aanpassen naargelang de RTL modus. Daarnaast bevat het ook de methodes allowRTL, forceRTL, swapLeftAndRightInRTL en setPreferredLanguageRTL om de RTL modus te beheren.

4.1.4 Besluit

In essentie is React Native Web, vanaf de buitenkant gezien, gewoon een kopie van het React Native framework. Achter de schermen moet de code aangepast, uitgebreid of herschreven worden zodat de componenten en API's worden ondersteund op het web. Maar voor de ontwikkelaar, die afkomstig is van React Native, verandert er weinig tot niets qua applicatieontwikkeling met behulp van React Native Web.

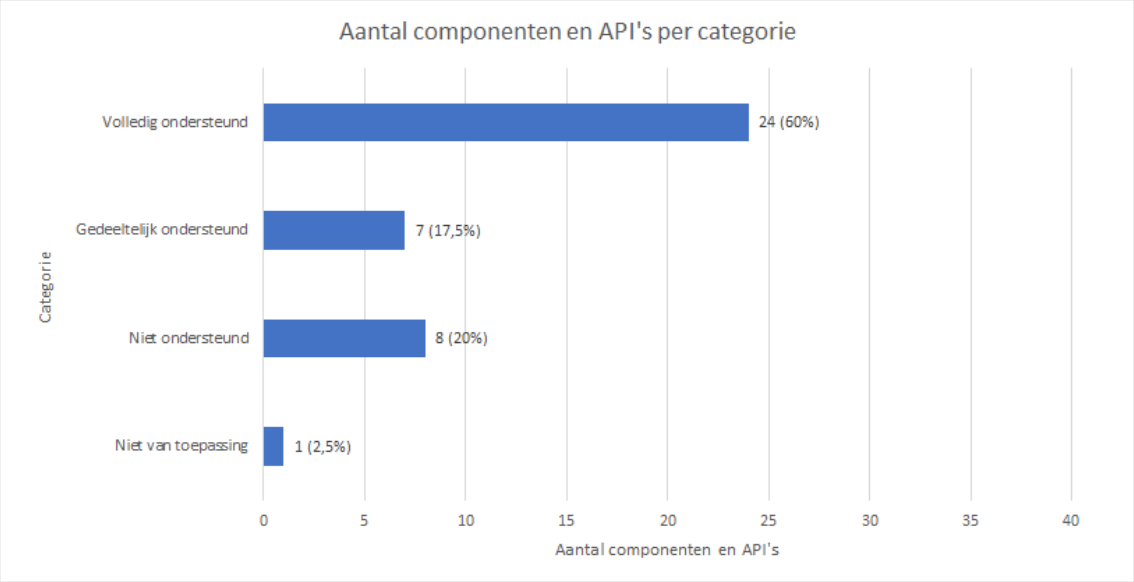
Als we in detail teruggaan op de gedeeltelijk ondersteunde componenten en API's, merken we dat deze in de meeste gevallen alle belangrijkste basisfunctionaliteiten bevatten. Het is vaak een gereduceerde versie van het originele React Native component of API. Zo mist het ScrollView component binnen React Native Web bijvoorbeeld de momentum functionaliteiten. Dit is niet noodzakelijk en maakt het component niet onbruikbaar.

Sommige componenten en API's ontbreken ook documentatie bij properties en methodes die wel degelijk functioneren en ondersteund worden. Andere componenten en API's binnen React Native Web bevatten zelfs helemaal geen documentatie. Het is belangrijk om de documentatie up-to-date en overeenstemmend met de library te houden zodat er geen verwarring of misinformatie ontstaat. Gelukkig kunnen we in de meeste gevallen beroep doen op de documentatie van de respectievelijke onderdelen van React Native.

We hebben onderzocht in welke mate de componenten en API's ondersteund worden binnen React Native Web. Het totale aantal, exclusief de Android en iOS specifieke componenten en API's, vormt een totaal van 40 onderdelen. In figuur 4.3 werden deze onderverdeeld in volgende categorieën:

- Volledig ondersteund: het is volledig ondersteund binnen React Native Web
- Gedeeltelijk ondersteund: het ontbreekt enkele functionaliteiten ten opzichte van React Native
- Niet ondersteund: het wordt niet ondersteund door React Native Web
- Niet van toepassing: het is niet van toepassing op het web

Uit figuur 4.3 kunnen we afleiden dat 60% van de componenten en API's van React Native volledig ondersteund worden door React Native Web. Meer dan de helft van de bouwstenen kan men dus volledig benutten op het web. Andere componenten en API's, namelijk 17,5%, worden gedeeltelijk ondersteund. Deze voorzien in de meeste gevallen alle belangrijkste basis functionaliteiten aan de ontwikkelaar. We kunnen dus zeggen dat 77,5% van de React Native componenten en API's bruikbaar zijn in React Native Web. Dit procent zal in de toekomst nog stijgen met de komst van niet ondersteunde onderdelen, die op dit moment 20% van het totaal vormen.



Figuur 4.3: Aantal componenten en API's per categorie en het percentage ten opzichte van het totale aantal. De categorieën zijn 'Volledig ondersteund', 'Gedeeltelijk ondersteund', 'Niet ondersteund' en 'Niet van toepassing'.

4.1.4.1 GitHub contribution

Nicolas Gallagher is de eigenaar van de GitHub repository van React Native Web. Een gezonde repository met open-source software bevat veel contributors. Men kan bijdrage leveren aan het project door middel van te helpen aan de implementatie van componenten, API's en features, problemen en bugs melden via issues en deze oplossen via pull requests. Als we echter kijken naar de Insights tab van de React Native Web repository, stellen we vast dat contributies van buitenstaande ontwikkelaars minimaal is. Met de 1200 commits van Gallagher tegenover de 10 commits van de contributor op de tweede plaats, voelt dit meer aan als een persoonlijke repository.

De repository bevat wel veel issues, grotendeels bestaande uit problemen en bugs, opgesteld door de community. Als het gaat om implementaties van features, is er verrassend weinig tot geen medewerking van contributors en staat Gallagher er dus alleen voor. Neem bijvoorbeeld de issues voor de implementatie van het Modal component en de Alert API (Gallagher, 2018a) (Gallagher, 2018b). Deze staan al sinds juli 2018 open zonder enige vooruitgang door zowel Gallagher als de community.

Algemeen is er nog veel werk met de implementatie van React Native componenten en API's, nieuwe features en het oplossen van problemen en bugs. Het zou de React Native Web repository goed doen indien deze wordt overgenomen door een professioneel team van ontwikkelaars, of collaborators in GitHub termen, binnen de GitHub organisatie Facebook zoals bij de repositories van React en React Native. Zo valt de last van het project niet op de schouders van één persoon en een handvol contributors, zodat de vooruitgang van het project versnelt en de kwaliteit verbeterd. Vanuit een business perspectief zou dit React Native Web veel aantrekkelijker maken aangezien het feit dat dit onderhouden wordt door één persoon, een groot risico vormt.

4.1.5 Expo SDK features

Expo voorziet een groot aantal features onder de Expo SDK die toegang geven tot bijvoorbeeld systeem- en native functionaliteiten, evenals functionaliteiten op hardware niveau. Een complete lijst kan men vinden op https://docs.expo.io/versions/v37.0.0. Een groot aantal van deze onderdelen biedt ook ondersteuning voor het web. In deze sectie bekijken we enkele features die vaak van toepassing zijn binnen applicaties en of deze al dan niet ondersteund worden op het web.

Merk op dat deze features enkel bruikbaar zijn binnen een Expo project. Deze kunnen niet benut worden in een alleenstaande React Native Web applicatie zoals we creëerde in sectie 4.1.1.1. Expo zorgt ervoor dat deze features stabiel functioneren op zowel native platformen als het web, zodat we als ontwikkelaar hierover geen zorgen hoeven te maken. Indien men toch gebruik maakt van een alleenstaande React Native of React Native Web applicatie, zonder toezicht van Expo, dan kan men opteren voor bijvoorbeeld het react-native-camera component van de organisatie React Native Community (Community, 2015a). Deze is gebaseerd op het camera component van Expo, maar in welke mate deze ondersteund wordt op het web ligt buiten de scope van dit onderzoek.

Let op dat sommige features vooraf geïnstalleerd zijn binnen een Expo project met managed workflow initialisatie en andere features extra stappen vereisen binnen een project met een bare workflow.

4.1.5.1 Camera

De camera is een belangrijk hardware feature dat de gebruiker toelaat om foto's en video's te maken. Applicaties waarin men bijvoorbeeld een profiel moet aanmaken dat een profielfoto bevat, kunnen gebruik maken van dit feature voor het nemen van een foto.

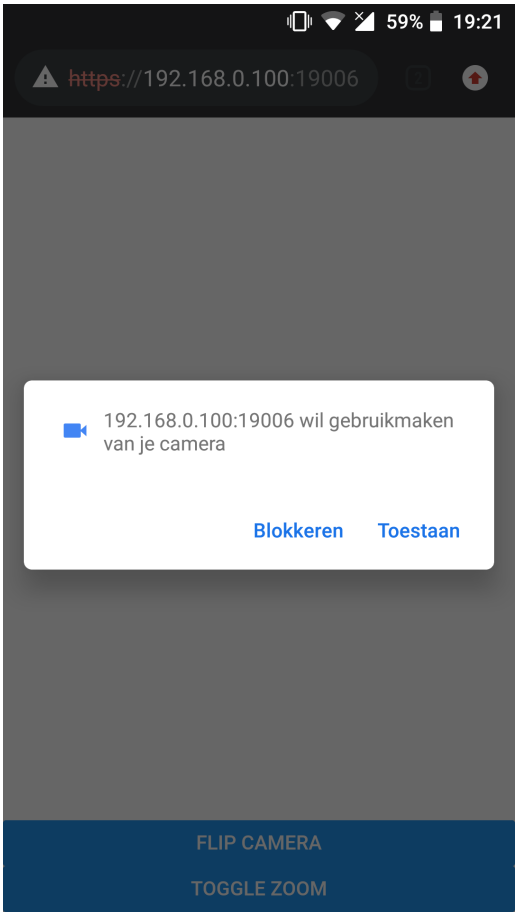
Expo voorziet toegang tot de camera van een apparaat via het expo-camera feature (Expo, 2020c). Om deze te installeren, voeren we volgend commando uit in de CLI van een Expo project.

```
1 expo install expo-camera
```

Vervolgens kunnen we een basis voorbeeld uitwerken dat gebruik maakt van het feature, evenals enkele functionaliteiten met behulp van de properties type en zoom. Op elk apparaat vraagt het besturingssysteem permissie aan de gebruiker om toegang te verlenen tot de camera (zie Figuur 4.4). Met behulp van de functie Camera.requestPermissionsAsync controleren we vervolgens of de applicatie deze permissies heeft ontvangen.

App.js

```
...
import { Camera } from 'expo-camera';
...

export default function App() {
    const [hasPermission, setHasPermission] = useState(false);
    const [cameraType, setCameraType] = useState(Camera.Constants.Type.back);
    const [cameraZoom, setCameraZoom] = useState(0);

    useEffect(() => {
        (async () => {
            const { status } = await Camera.requestPermissionsAsync();
            setHasPermission(status === 'granted');
        })();
    });

    if (hasPermission === false) {
        return <Text>No permission to access the camera</Text>;
    }

    function flipCamera() {
        setCameraType(cameraType === Camera.Constants.Type.back
            ? Camera.Constants.Type.front
            : Camera.Constants.Type.back
        );
    }

    function zoomCamera() {
        setCameraZoom(cameraZoom === 0 ? 1 : 0);
    }

    return (
        <View style={{flex: 1}}>
            <Camera style={{flex: 1}} type={cameraType} zoom={cameraZoom}></
                Camera>
            <Button title='Flip camera' onPress={() => flipCamera()}></Button>
            <Button title='Toggle zoom' onPress={() => zoomCamera()}></Button>
        </View>
    );
}
```



Figuur 4.4: Sommige Expo features maken gebruik van native functionaliteiten die permissie vereisen. Hier vraagt het besturingssysteem permissie aan de gebruiker zodat de mobiele webapplicatie toegang heeft tot de camera.

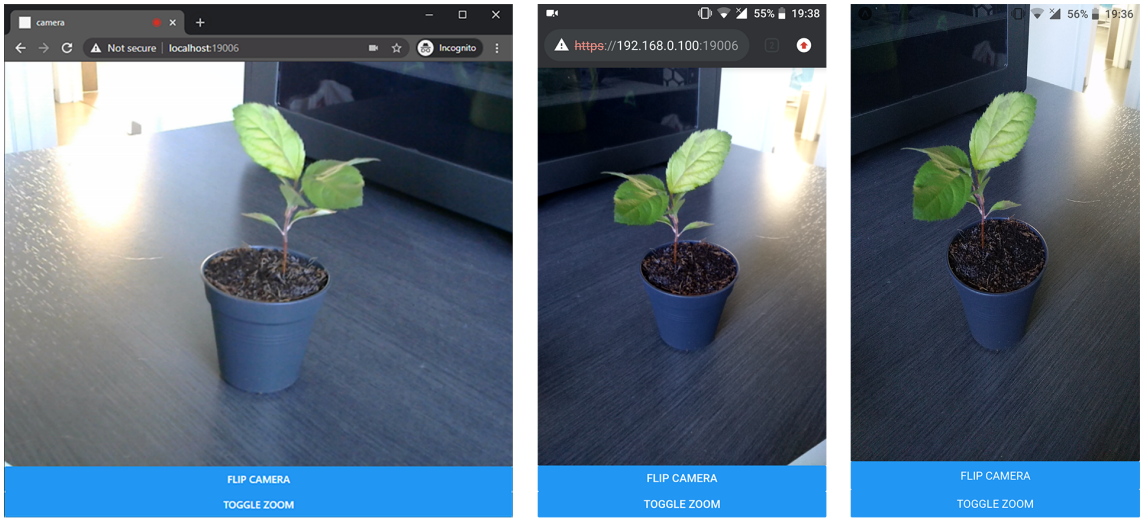
Na opstart van het Expo project en het scannen van de QR code via de Expo app op een native apparaat, zien we het voorbeeld in actie. De camera weergave met flip en zoom functionaliteit zijn operationeel. Wanneer we de applicatie echter openen in een webbrowser door middel van de W-toets in te drukken binnen de CLI van het Expo project, krijgen we een TypeError te zien: CameraManager.requestPermissionsAsync is not a function.

Het is onduidelijk of deze error een bug is, of het resultaat van een functie die niet ondersteund is op het web. In ieder geval kunnen we deze code voorlopig vermijden met behulp van de Platform module die controleert of de applicatie al dan niet op het web draait.

App.js

```
1  ...
2  if (Platform.OS !== 'web') {
3        const { status } = await Camera.requestPermissionsAsync();
4        setHasPermission(status === 'granted');
5  }
6  ...
7
8  ...
9  if (Platform.OS !== 'web' && hasPermission === false) {
10       return <Text>No permission to access the camera</Text>;
11 }
12 ...
```

Uiteindelijk is de camera ook operationeel in de webbrowser, in geval dat het apparaat over een interne of externe webcam of camera beschikt. Indien dit niet het geval is, krijgt de gebruiker een leeg scherm te zien. We kunnen met behulp van de functie Camera.isAvailableAsync controleren of het apparaat al dan niet over een camera beschikt. Merk ook op dat onze geïmplementeerde functies flipCamera en zoomCamera niet werken indien de camera niet over deze functionaliteiten beschikt op hardware niveau. In figuur 4.5 kunnen we concluderen dat expo-camera operationeel is op beide technologieën: het web (desktop webbrowser en mobile webbrowser) en de native platformen (Android).



Figuur 4.5: Expo applicatie dat gebruik maakt van het expo-camera feature in de desktop webbrowser (links), mobile webbrowser (midden) en Android app (rechts).

4.1.5.2 Push notifications

Push notifications zijn korte notificatie berichten die worden weergegeven op het start-scherm en in de statusbar van een mobiel apparaat. In geval van een desktop webapplicatie, worden deze weergegeven op de website zelf.

Expo heeft een ingebouwd push notifications feature, maar helaas wordt deze voorlopig niet ondersteund op het web (Expo, 2020h). Als alternatief kan men op eigen houtje push notifications implementeren in een webapplicatie met behulp van een service worker. Een service worker is een script dat, onafhankelijk van de webapplicatie, in de achtergrond van de webbrowser draait (Gaunt, 2019) en de basis vormt voor PWA's (Progressive Web App). Het opent de deur naar verschillende mogelijkheden zoals push notifications, achtergrond synchronisatie, offline werken, enz. De handleiding van Gaunt (2018) biedt inzicht in de implementatie van push notifications met behulp van een service worker binnen een webapplicatie. Anderzijds kan men ook kiezen voor een extern platform zoals Pushpad (AbstractBrain, 2015). Deze neemt het grootste deel van het werk op zich, zodat we als ontwikkelaar slechts een script hoeven toe te voegen aan de webapplicatie voor de integratie van push notifications. Via het platform van Pushpad registreren we een push notification service die notificatie berichten stuurt naar de applicatie. Zo'n platforms kunnen we echter in de meeste gevallen niet gratis benutten.

4.1.5.3 Location

Het Expo location feature laat de ontwikkelaar toe om geolocatie informatie op te vragen van het apparaat (Expo, 2020g). We kunnen deze installeren in een Expo project met behulp van onderstaand commando.

```
1 expo install expo-location
```

Vervolgens proberen we volgend voorbeeld uit dat de geolocatie opvraagt van het apparaat waarop de applicatie draait. Opnieuw vraagt het besturingssysteem of de webbrowser naar de permissie van de gebruiker om de locatie van het apparaat te kunnen gebruiken. Als resultaat zien we de coördinaten van de huidige locatie.

App.js

```
...
import * as Location from 'expo-location';
...

export default function App() {
    const [error, setError] = useState(null);
    const [location, setLocation] = useState(null);

    useEffect(() => {
        (async () => {
            let { status } = await Location.requestPermissionsAsync();

            if (status !== 'granted') {
                setError('Permission to access location was denied');
            }

            let location = await Location.getCurrentPositionAsync({});
            setLocation(location);
        })();
    });

    let locationInfo = 'Receiving location info..';

    if (error) {
        locationInfo = error;
    } else if (location) {
        locationInfo = JSON.stringify(location);
    }

    return (
        <Text>{locationInfo}</Text>
    );
}
```

4.1.5.4 Text-to-speech

Expo Speech is een TTS (Text-To-Speech) feature dat bijvoorbeeld een meerwaarde kan bieden voor mensen met hoor problemen door tekst te converteren naar spraak binnen de webapplicatie (Expo, 2020j). Welke engine dit TTS feature gebruikt of op welke engine het gebaseerd is, wordt niet vermeld in de documentatie. Het is dus onduidelijk in welke staat deze zich bevindt qua bereik en nauwkeurigheid van tekst naar spraak.

We kunnen het Text-to-speech feature van Expo installeren en een voorbeeld uitproberen met onderstaand commando en code.

```
expo install expo-speech
```

App.js

```
...
import * as Speech from 'expo-speech';
...

export default function App() {
    function speakText() {
        Speech.speak('Hello, this is a text to speech message!');
    }

    return (
        <Button title="Speak text" onPress={() => speakText()}></Button>
    );
}
```

Indien we Nederlands willen gebruiken als taal voor de TTS, dan voegen we een object met property language toe als tweede parameter aan de functie speak met de gewenste IETF BCP 47 code.

```
Speech.speak('Hallo, dit is een tekst naar spraak bericht!', {language: 'nl'});
```

4.1.5.5 Audio

Met behulp van het Audio feature van Expo kan men audio bestanden afspelen binnen de applicatie. Onderstaand commando installeert het feature in het Expo project en de voorbeeld code speelt, met een druk op de knop, een liedje af dat zich in de folder van het project bevindt.

```
expo install expo-av
```

App.js

```
...
import { Audio } from 'expo-av';
...

export default function App() {
    async function playSong() {
        const soundObject = new Audio.Sound();
        await soundObject.loadAsync(require('./song.mp3'));
        await soundObject.playAsync();
    }

    return (
        <Button title="Play song" onPress={() => playSong()}></Button>
    );
}
```

4.1.5.6 Overige Expo SDK features

Naast de besproken features, beschikt de SDK van Expo natuurlijk over nog meer onderdelen die het waard zijn om te melden. Zo verzorgt Expo AuthSession bijvoorbeeld de authenticatie van een applicatie (Expo, 2020a). Expo Device voorziet systeeminformatie over het fysieke apparaat (Expo, 2020d) en Expo Facebook levert integratie met Facebook zodat men bijvoorbeeld kan inloggen op de applicatie via een Facebook account (Expo, 2020e). Het Sensors feature geeft toegang tot de verschillende sensors van een fysiek apparaat (Expo, 2020i) en met Expo Video kan men video's afspelen binnen de applicatie (Expo, 2020k). Deze bieden allemaal ondersteuning voor het web.

Daarnaast bestaan er ook handige Expo features die helaas nog geen ondersteuning bieden voor het web waaronder Expo BarCodeScanner dat functionaliteit biedt voor het scannen van verschillende barcodes (Expo, 2020b) en Expo FileSystem dat toegang geeft tot het bestandssysteem van het apparaat (Expo, 2020f).

5. Proof-of-concept

5.1 Proof-of-concept

Het proof-of-concept is een vereenvoudigde basis applicatie ontwikkelen dat operationeel is op zowel mobile native platformen, als op het web via een webbrowser, gebruik makende van React Native componenten en API's die via React Native Web ook functioneel worden op het web.

Met een basis applicatie wordt bedoeld dat deze basis functionaliteiten bevat zoals navigatie tussen schermen, CRUD (Create, Read, Update en Delete) operaties op data en interacties met de gebruiker zoals knoppen en inputs. Met vereenvoudigd duiden we aan op het gebruik van lokale data die hard gecodeerd is. De applicatie staat dus niet in verbinding met een backend, databank of dergelijke.

Het proof-of-concept start vanaf een Expo applicatie, beschikbaar op https://github.com/JayvM/react-native-poc/tree/79d4030959dfdf6c84e2527ff1e0c405629d1b3d, die uitsluitend ontwikkeld is voor mobile native platformen. We proberen deze applicatie operationeel te maken in de webbrowser.

5.1.1 Het proof-of-concept verkennen

De applicatie van het proof-of-concept stelt een simpel forum voor waarin de gebruiker 'posts' kan beheren, 'comments' kan toevoegen aan posts en beide positieve of negatieve stemmen kan toewijzen.

De React Native applicatie bevat ten eerste een navigatie structuur om tussen de schermen te kunnen navigeren, een essentieel onderdeel van elke moderne mobiele applicatie. Deze is geïmplementeerd met behulp van de library React Navigation: een populaire keuze als het gaat om navigatie in React Native, welke een native uiterlijk en gevoel geeft aan de gebruiker (Natigation, 2017). De navigatie structuur bestaat uit een Drawer navigator die twee Stack navigators bevat. De eerste stack bevat een welkomscherm die wordt weergegeven bij opstart van de applicatie. De tweede stack bevat alle schermen voor het beheer van de posts.

Het beheer van de posts gebeurt via CRUD-operaties op lokale data. Een post bestaat uit een titel, tekst, extra informatie met de naam van de gebruiker die deze toegevoegd heeft en tenslotte de datum van creatie. De gebruiker kan deze post positieve of negatieve stemmen toewijzen via respectievelijk twee pijl icoontjes. Deze icoontjes zijn afkomstig van een externe library React Native Vector Icons (Arvidsson, 2015). Alle posts worden in een lijst weergegeven op de hoofdpagina, die aan de hand van een selectbox gesorteerd kan worden op aantal positieve stemmen of datum. Wanneer de gebruiker op een post drukt, wordt het details scherm getoond. Deze bevat alle gegevens van de post, evenals de comments. Een comment is een klein tekst bericht als reactie op een post dat ook positieve of negatieve stemmen kan krijgen. Via de knop rechts onderaan kan de gebruiker nieuwe comments toevoegen aan de post. Rechts bovenaan, in de navigatie balk, kan men navigeren naar het scherm om de gegevens van deze post te wijzigen. Terug op het hoofdscherm kan men posts verwijderen door hierop lang te drukken. Een modal komt tevoorschijn ter bevestiging om te verwijderen. Opnieuw rechts bovenaan in de navigatie balk, kan de gebruiker het scherm openen voor het toevoegen van een nieuwe post. Net zoals bij het wijzigen, wordt er een melding getoond indien niet alle velden zijn ingevuld. Als laatste onderdeel bevat het hoofdscherm een switch waarmee er kan gewisseld worden tussen een kleur en een afbeelding als achtergrond van de mobiele applicatie.

Als samenvatting voorzien we een opsomming van de React Native componenten en API's, evenals andere middelen van externe libraries, die de bouwstenen vormen van de applicatie en interacties met de gebruiker mogelijk maken.

React Native:

- ActivityIndicator
- Alert
- AppRegistry
- Button
- FlatList
- ImageBackground
- Modal
- Picker
- StyleSheet
- Switch
- Text
- TextInput
- ToastAndroid
- TouchableOpacity
- View

React Navigation:

- Drawer navigator
- Stack navigator

React Native Vector Icons:

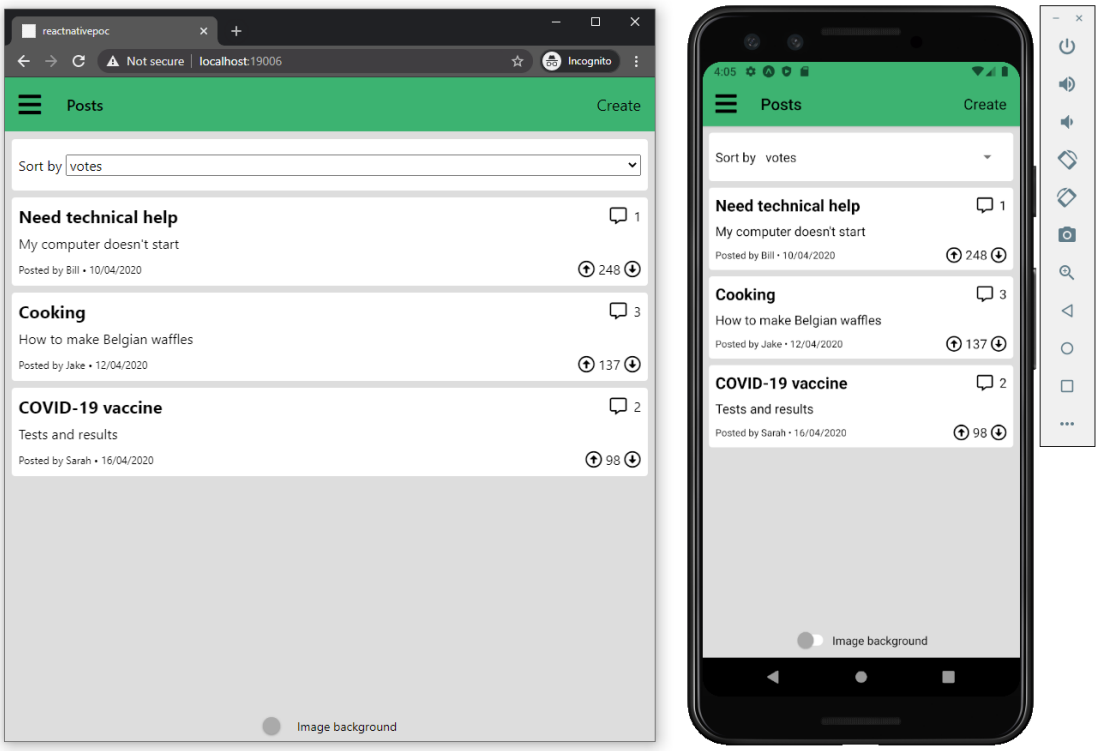
- Icon

5.1.2 Het proof-of-concept opstarten en verwerken

We beginnen het proof-of-concept met het opstarten van de Expo applicatie in een web-browser via het commando npm run web. Na initialisatie van Expo, Metro Bundler en Webpack, wordt de applicatie gecompileerd en start deze zonder problemen op in de browser. De navigation header en welkompagina met titel en afbeelding worden weergegeven zoals gewenst. Via het drawer menu kunnen we ook zonder problemen navigeren naar de pagina met alle posts. Opnieuw worden alle UI elementen op dezelfde manier weergegeven zoals in de React Native applicatie op mobiele apparaten (zie figuur 5.1). Een goede start voor React Native Web!

5.1.2.1 Picker component

Een eerste opmerking is de presentatie van het Picker component. Deze is getransformeerd naar een standaard HTML select element en ziet er dus minder strak en modern uit als de versie op Android en iOS. De functionaliteit, het sorteren van de lijst met posts, werkt wel zoals verwacht.



Figuur 5.1: Het scherm met het overzicht van alle posts binnen de applicatie van het proof-of-concept.

Merk op dat het Picker component deprecated is binnen React Native. Dit betekent dat het component niet meer wordt meegerekend in de verdere ontwikkeling van de library. Het is dus af te raden om dit component te gebruiken aangezien problemen en bugs niet meer worden opgelost door de ontwikkelaars en het onderdeel mogelijk wordt verwijderd in de toekomst. Hoe we dit aanpakken wordt uitgelegd in sectie 4.1.3.2 van hoofdstuk 4.

5.1.2.2 Alert API & Modal component

Wanneer we op een post klikken om door te gaan naar de pagina met gegevens en comments van deze post, merken we op dat het modal voor het toevoegen van een comment open en bloot op het scherm staat. In de React Native applicatie is deze verborgen en wordt het modal via een interactie gepresenteerd om een nieuwe comment toe te voegen aan de post. Het resultaat dat we nu te zien krijgen is het gevolg van de ontbrekende ondersteuning van het Modal component in React Native Web. In deze sectie overlopen we enkele oplossingen ter vervanging van het Modal component, evenals de Alert API van React Native, aangezien beide een vergelijkbare oplossing vereisen.

Alert is één van de React Native API's die React Native Web momenteel niet ondersteunt, zoals aangegeven in de 'Compatibility with React Native' sectie op de officiële GitHub pagina van de library. De GitHub issue voor de implementatie van deze API staat al sinds juli 2018 open zonder enig recent teken van leven. Ondertussen bevat het enkele door de community voorziene alternatieven. Maar in welke mate deze honderd procent getest en

stabiel zijn binnen de library, is geen garantie. In de tussentijd kan men dus hiervan gebruik maken of eigen oplossingen ontwikkelen in afwachting van de officiële ondersteuning van de API in React Native Web.

Merk op dat we toch de Alert API kunnen importeren in een React Native Web applicatie zonder dat er compilatiefouten of dergelijke optreden, op voorwaarde dat deze niet wordt uitgevoerd in de code. Via de Platform module kunnen we bepaalde code, componenten en API's al dan niet uitvoeren en weergeven, afhankelijk van het platform waarop de applicatie draait. Deze twee zaken maakt het mogelijk om broncode samen te stellen in één codebase die zowel in React Native als in React Native Web draait.

Een simpel voorbeeld is het gebruik van de Window.alert() API die standaard aanwezig is in JavaScript (JavaScript, 2020a). Via de Platform module van React Native kunnen we controleren op welk platform de applicatie draait en deze dan overeenkomstig afstemmen naar ofwel het gebruik van de React Native Alert API of de JavaScript Window.alert() API.

App.js

```
1 export default function App() {
2     if (Platform.OS === 'web') {
3         window.alert('This is an alert');
4     } else {
5         Alert.alert('This is an alert');
6     }
7
8     return (...);
9 }
```

De Alert API van React Native heeft wel wat meer in huis dan alleen een tekst bericht tonen. Zo kan men zelf de knoppen bepalen om weer te geven in het popup bericht: maximaal drie voor Android en onbeperkt voor iOS. JavaScript is hierin gelimiteerd met slechts een Window.confirm() API ter beschikking die een vaste 'OK' en 'Cancel' knop bevat (JavaScript, 2020b).

App.js

```
export default function App() {
    if (Platform.OS === 'web') {
        if (window.confirm('Are you sure?')) {
            console.log('OK pressed')
        } else {
            console.log('Cancel pressed')
        }
    } else {
        Alert.alert('Are you sure?', null, [
            { text: 'OK', onPress: () => console.log('OK pressed') },
            { text: 'Later', onPress: () => console.log('Later pressed') },
            { text: 'Cancel', onPress: () => console.log('Cancel pressed') }
        ]);
    }

    return (...);
}
```

Indien men een alert bericht met input wilt gebruiken, kan men opteren voor de Window.prompt() API van JavaScript, of het Modal component van React Native (JavaScript, 2020c) (Native, 2020b). Helaas wordt dit component voorlopig opnieuw niet ondersteund door React Native Web. Via een GitHub issue van juli 2018 laat Nicolas Gallagher weten dat er wordt gewerkt aan de implementatie, maar wanneer deze wordt uitgerold is onbekend. Men kan terug opteren voor alternatieven die ontwikkeld zijn door de community, maar wederom vormt dit een risico op gebied van testen en stabiliteit. Het is dus verstandiger om de implementatie van het Modal component in de officiële React Native Web library af te wachten.

In de tussentijd kunnen we eenvoudig zelf een op maat gemaakte modal ontwikkelen met eender welke inhoud, gebruik makende van React Native componenten die wél ondersteund worden door React Native Web. Het modal in de React Native applicatie van het proof-of-concept bevat een Text en TextInput component, evenals twee Button componenten. In een apart component CustomModal voorzien we de JSX code voor deze UI via de functie render. Met behulp van de style properties position: 'absolute' en zIndex: 999 kunnen we het modal boven de andere UI elementen van de applicatie weergeven. Het component App bevat een object modal in zijn state. Deze heeft een property visible met een boolean variabele dat wordt aangepast via interacties, afkomstig van het component Content, om de weergave van het component CustomModal te dirigeren via een conditie.

CustomModal.js

```
export default class CustomModal extends React.Component {
    constructor(props) {
        super(props);
        this.state = { inputValue: null };
    }

    render() {
        return (
            <View style={styles.modal}>
                <View style={styles.body}>
                    <Text>{this.props.message}</Text>
                    <TextInput style={styles.textInput} onChangeText={(value) =>
                        this.setState({ inputValue: value })}></TextInput>
                </View>
                <Button title='OK' onPress={() => this.props.onClose(this.state.
                    inputValue)}></Button>
                <Button title='Cancel' onPress={() => this.props.onClose()}></
                    Button>
            </View>
        );
    }
};

const styles = StyleSheet.create({
    ...
    modal: {
        width: '50%',
        height: '50%',
        position: 'absolute',
        top: '50%',
        left: '50%',
        zIndex: 999,
        elevation: 999,
        transform: [{ translateX: '-50%' }, { translateY: '-50%' }],
        backgroundColor: '#DDDDDD'
    },
    ...
});
```

App.js

```
export default class App extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            modal: {
                visible: false
            }
        };
    }

    openModal = (message) => {
        this.setState({
            modal: {
                visible: true,
                message: message
            }
        });
    };

    closeModal = (inputValue) => {
        this.setState({
            modal: {
            visible: false
        }
    });

    console.log(inputValue);
};

render() {
    return (
        <View style={styles.container}>
            {this.state.modal.visible && <CustomModal message={this.state.modal.
                message} onClose={this.closeModal}></CustomModal>}
            <Content openModal={this.openModal}></Content>
        </View>
        );
    }
};
```

Content.js

```
export default function Content(props) {
    return (
        <View>
            <Button title='Open modal' onPress={() => props.openModal('This is a
                modal message')}></Button>
        </View>
    );
}
```

Deze modal is perfect bruikbaar in een React Native applicatie, mits er één aanpassing gebeurt. De style property transform ondersteunt geen string waarde in React Native, maar vereist een numerieke waarde (Native, 2020d). We kunnen dus geen gebruik maken van percentages zoals in bovenstaande code, tenzij we een kleine omweg implementeren met behulp van de Dimensions API van React Native (Native, 2020a). Buiten de scope van het functionele component CustomModal voegen we een constante properties toe met een object als waarde. Deze bevat een property width en height die via de Dimensions API de breedte en hoogte van het scherm respectievelijk in twee verdeelt. Vervolgens kunnen we een style property transform toevoegen aan het style object van het modal die hiervan gebruik maakt om het modal in het midden van het scherm te positioneren.

CustomModal.js

```
const properties = {
    width: Dimensions.get('window').width / 2,
    height: Dimensions.get('window').height / 2
};

export default class CustomModal extends React.Component {
    ...
};

const styles = StyleSheet.create({
    ...
    modal: {
        width: properties.width,
        height: properties.height,
        ...
        transform: [{ translateX: -(properties.width / 2)}, { translateY: -(properties.
            height / 2)}],
        ...
    },
    ...
});
```

Zoals eerder vermeld, is deze zelfgemaakte modal bruikbaar in zowel React Native als in de webbrowser met behulp van React Native Web. Indien we toch het React Native Modal component willen benutten voor native platformen, moeten we enkele uitbreidingen toevoegen aan de JSX code.

Ten eerste wijzen we de JSX code van het zelfgemaakte modal toe aan een variabele modal. Daarna wordt het style object van het buitenste View component aangepast naargelang het platform waarop de applicatie draait. We maken hiervoor opnieuw gebruik van de Platform module. Daarna wordt het geheel binnen een extra View component geplaatst zodat het modal centraal op het scherm wordt weergegeven, door middel van het style object modalWrapper dat gebruik maakt van flex layout. Uiteindelijk wordt het geheel binnen een Modal component van React Native geplaatst.

CustomModal.js

```
export default class CustomModal extends React.Component {
    ...

    render() {
        let modal = (
            <View style={Platform.OS !== 'web' ? styles.modalMobile : styles.modal
                }>
                ...
            </View>
        );

        if (Platform.OS !== 'web') {
            modal = (
                <Modal animationType="slide" transparent={true} visible={true}>
                    <View style={styles.modalWrapper}>
                        {modal}
                    </View>
                </Modal>
            );
        }

        return modal;
    }
};

const styles = StyleSheet.create({
    modalWrapper: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center'
    },
        modalMobile: {
        width: properties.width,
        height: properties.height,
        backgroundColor: '#DDDDDD'
    },
    ...
});
```

Ondanks het feit dat de Alert API en het Modal component niet ondersteund worden in React Native Web, kunnen we voor de webapplicatie alternatieve oplossingen implementeren. De alert, confirm en prompt API's van JavaScript zijn hiervan een voorbeeld, maar op gebied van UI en functionaliteit zijn deze niet flexibel en zeer beperkt. Een andere oplossing is het ontwikkelen van een op maat gemaakte alert of modal met behulp van componenten en API's die wél ondersteund worden door React Native Web en de nodige style properties toewijzen om deze correct weer te geven. Met behulp van de Platform module kan deze benadering gebruikt worden in plaats van de Alert API en Modal component voor React Native Web, tot deze officieel ondersteund worden binnen de library.

Met deze bevindingen kunnen we de applicatie van het proof-of-concept aanpassen zodat deze niet meer rekent op de Alert API en het Modal component van React Native op het web. Hierdoor is de webapplicatie volledig functioneel en operationeel. Alle andere gebruikte componenten en API's binnen de applicatie van het proof-of-concept worden ondersteund door React Native Web en vereisen geen extra inspanning.

De aangepaste applicatie van het proof-of-concept is te vinden op https://github.com/JayvM/react-native-poc/tree/090e7e5413f17002d8838b82498c23c44cf81ddd.

De voorbeeldcode voor het zelfgemaakte Modal component kunnen we opnieuw raadplegen op https://github.com/JayvM/react-native-web-alert-modal.

5.1.2.3 React Navigation

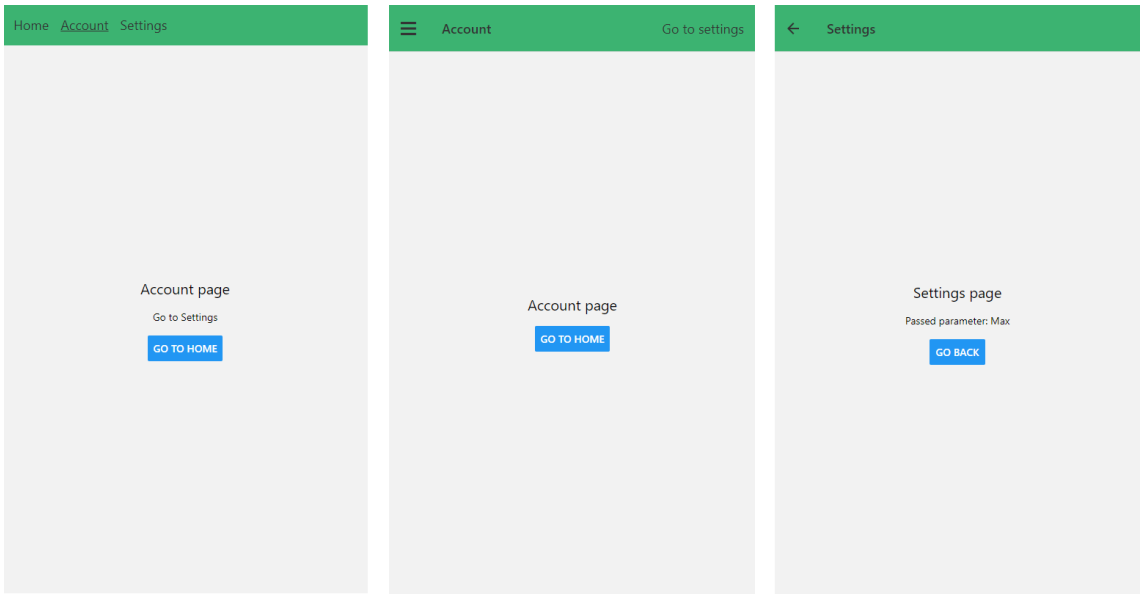
De Expo applicatie van het proof-of-concept maakt gebruik van de library React Navigation voor het navigeren tussen schermen. Sinds versie 5 biedt het, in samenwerking met React Native Web, ondersteuning voor navigatie in zowel React Native als in de webbrowser. Deze bevindt zich nog volop in ontwikkeling, dus bugs kunnen zich presenteren en wijzigingen aan features en API's kunnen plaatsvinden. Desondanks functioneert de navigatie verrassend vloeiend zonder enige compilatiefouten of andere problemen binnen het proof-of-concept.

Swipe gestures en animaties

Out-of-the-box is React Navigation operationeel in de webbrowser, maar niet zonder enkele minimale gevolgen. Swipe gestures is bijvoorbeeld een feature dat niet meer beschikbaar is op het web. Op een native platform kan men bijvoorbeeld met de vinger van de uiterst linkse kant van het scherm naar rechts vegen om het Drawer menu te openen of terug navigeren naar het vorige scherm bij gebruik van een Stack navigator. Deze laatste schakelt ook de animaties uit die plaatsvinden tijdens het navigeren tussen de schermen. We kunnen deze echter opnieuw inschakelen met behulp van het property animationEnabled dat via een object wordt toegewezen aan het property options van een scherm. Helaas treedt een visuele bug op waarbij de scrollbar van de webbrowser tevoorschijn komt tijdens de animatie. Hierdoor verschuift de gehele pagina een afstand, gelijk aan de breedte van de scrollbar, naar links.

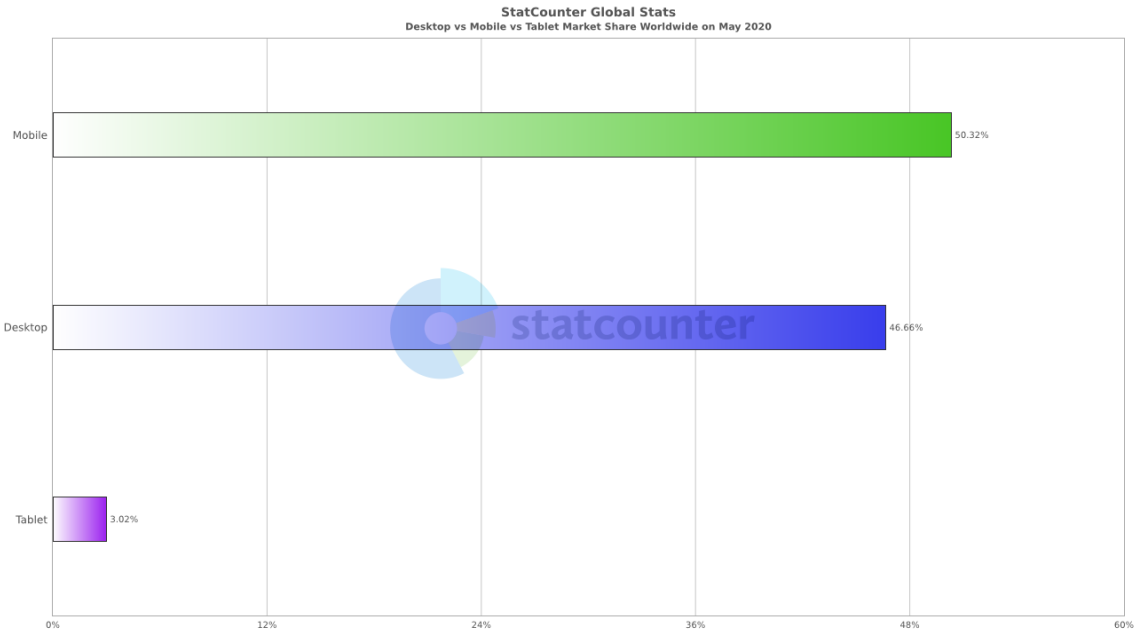
Navigatie UI voor het web

Het valt meteen op dat de navigatie UI niet op zijn plaats voelt binnen een webbrowser. Het drawer menu en de stack navigatie is kenmerkend voor mobiele apparaten wegens de gebruiksvriendelijkheid en overzichtelijkheid op kleine schermen. Deze manier van navigatie, zowel in werking als uiterlijk, voelt vreemd op het web. De gebruiker verwacht eerder een navigatie bar, vaak horizontaal aan de top of verticaal aan de linkerkant van de applicatie, waarin alle links en sublinks naar de verschillende pagina's te vinden zijn. Zie figuur 5.2.

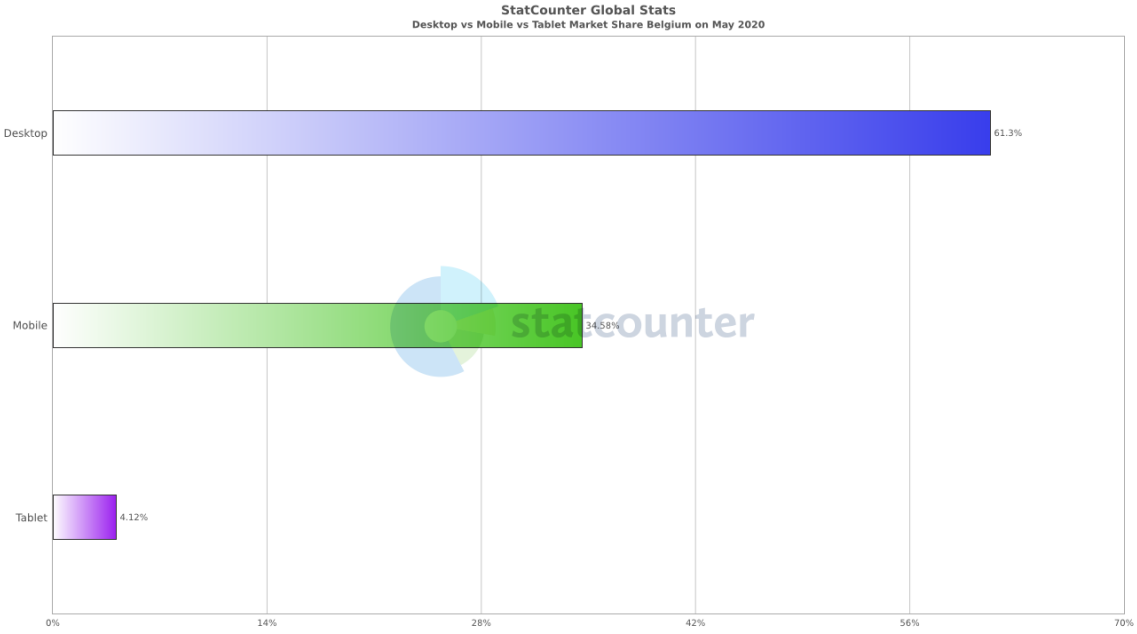


Figuur 5.2: De verschillen in UI tussen een web navigatie (links) en een mobile native navigatie (midden en rechts).

Anders bekeken heeft dit ook een positieve kant. Een dergelijk navigatie bar van een webapplicatie is vaak niet geoptimaliseerd voor mobiele apparaten. Daarbovenop toont data van GlobalStats aan dat een groot marktaandeel op gebied van websitebezoeken in handen is van mobiele apparaten (StatCounter, 2020a). Op figuur 5.3 stellen we vast dat dit wereldwijd 50.32% bedraagt. Specifiek voor België, bezitten mobiele apparaten 34.58% of ongeveer 1/3 van het marktaandeel, zoals aangegeven in figuur 5.4. Deze data is afkomstig van 2 miljoen wereldwijde websites die elke maand miljarden pagina bezoeken registreren. We kunnen hieruit dus besluiten dat een mobiele navigatie een welkom gegeven is binnen een webapplicatie voor een groot deel van de gebruikers.



Figuur 5.3: Procent marktaandeel in wereldwijde websitebezoeken voor desktop computers, mobiele apparaten en tablets in mei 2020 (StatCounter, 2020c).



Figuur 5.4: Procent marktaandeel in Belgische websitebezoeken voor desktop computers, mobiele apparaten en tablets in mei 2020 (StatCounter, 2020b).

React Navigation biedt geen opties om de navigatie UI aan te passen indien men toch bijvoorbeeld een navigatie bar verkiest dat kenmerkend is voor webapplicaties. Met de beschikbare hulpmiddelen, componenten en API's van React Native en React Navigation, kunnen we echter zelf een navigatie bar ontwikkelen. Het zou handig zijn indien we de navigatie dynamisch kunnen veranderen naargelang het apparaat waarop de webapplicatie draait. Voor tablets, laptops en desktop computers voorzien we de web gespecifieerde navigatie en kleine apparaten zoals smartphones benutten de mobiele navigatie. Er bestaan verschillende manieren om dit te bereiken. Een eerste voorbeeld is het gebruik van de user agent dat allerlei informatie bevat over de webbrowser, het besturingssysteem, het apparaat, enz. (Docs, 2020). Detect Mobile Browsers is een website dat voor verschillende programmeertalen en ontwikkelingsmiddelen scripts voorziet. Deze detecteren of de applicatie al dan niet op een mobiel apparaat draait, gebruik makende van de user agent van de webbrowser (Smith, 2014). Een andere toepassing is de repository React Native Device Info van de organisatie React Native Community dat allerlei informatie aanbiedt over het apparaat waarop de mobiele- of webapplicatie draait zoals naam, merk, fabrikant, API level, besturingssysteem, enz. (Community, 2015b).

Zelf een navigatie bar ontwikkelen voor het web

React Navigation bestaat uit één NavigationContainer met daarin één of meerdere navigators zoals Drawer, Stack of Tab. Binnen een navigator voorziet men alle schermen waartussen genavigeerd kan worden. In volgend voorbeeld maken we gebruik van meerdere navigators die genesteld zijn: een Drawer navigator met twee schermen die elk een Stack navigator bevatten en op hun beurt ook één tot twee schermen bezitten. Om plaats te maken voor onze toekomstige zelfgemaakte navigatie bar, voegen we het property headerShown met waarde false aan het object toe, dat via het property options aan elk scherm binnen een Stack navigator wordt meegegeven.

De applicatie van het proof-of-concept maakt ook gebruik van genestelde navigators, bestaande uit een Drawer navigator met daarin meerdere Stack navigators. Volgende bevindingen zijn dus representatief voor het proof-of-concept indien men de navigatie wil uitbreiden met een versie voor het web.

App.js

```
1  ...
2  const Drawer = createDrawerNavigator();
3  const Stack = createStackNavigator();
4  ...
5
6  export default function App() {
7      return (
8          <NavigationContainer >
9              <Drawer.Navigator>
10                 <Drawer.Screen name="rootHome" component={HomeStack}></
                       Drawer.Screen>
11                 <Drawer.Screen name="rootAccount" component={AccountStack
                       }></Drawer.Screen>
12             </Drawer.Navigator>
13         </NavigationContainer>
14         );
15     }
16
17 const options = {
18     headerShown: false
19 };
20
21 function HomeStack() {
22     return (
23         <Stack.Navigator>
24             <Stack.Screen name='home' component={Home} options={options}></
                   Stack.Screen>
25         </Stack.Navigator>
26     );
27 }
28
29 function AccountStack() {
30     return (
31         <Stack.Navigator>
32             <Stack.Screen name='account' component={Account} options={options
                   }></Stack.Screen>
33             <Stack.Screen name='settings' component={Settings} options={options
                   }></Stack.Screen>
34         </Stack.Navigator>
35     );
36 }
```

De traditionele implementatie om te navigeren tussen schermen in React Navigation, is het gebruik van de functie navigation.navigate. De naam van een scherm, dat zich binnen de NavigationContainer bevindt, kan worden meegegeven aan de functie om hiernaar te navigeren. We kunnen dus simpelweg een component maken dat dienst doet als navigatie bar en gebruik maakt van deze functie voor de navigatie functionaliteit. Het zelfgemaakte navigatie bar component NavigationBar bestaat uit enkele Text componenten die weblinks nabootsen en omringd zijn door een View component dat dienst doet als container. Tenslotte voorzien we styling, zodat de container op een navigatiebar lijkt.

NavigationBar.js

```
1  export default function NavigationBar(props) {
2      return (
3          <View style={styles.container}>
4              <Text style={[styles.link, props.activeLink === 'home' ? styles.active :
                   null]} onPress={() => props.navigation.navigate('rootHome', {screen:
                    'home'})}>Home</Text>
5              <Text style={[styles.link, props.activeLink === 'account' ? styles.active :
                   null]} onPress={() => props.navigation.navigate('rootAccount', {
                   screen: 'account'})}>Account</Text>
6              <Text style={[styles.link, props.activeLink === 'settings' ? styles.active :
                   null]} onPress={() => props.navigation.navigate('rootAccount', {
                   screen: 'settings', params: {name: 'Max'}})}>Settings</Text>
7          </View>
8      );
9  }
10
11 const styles = StyleSheet.create({
12     container: {
13         padding: 16,
14         flexDirection: 'row'
15     },
16     link: {
17         marginRight: 16,
18         fontSize: 18
19     },
20     active: {
21         textDecorationLine: 'underline'
22     }
23 });
```

De Stack navigator toont de naam van het huidige scherm als titel in de header. Nu we niet meer gebruik maken van deze UI, voorzien we een style object active bij het actieve scherm. Deze zorgt ervoor dat de weblink in de navigatie bar onderlijnd is, om de gebruiker duidelijk te maken op welke pagina men zich bevindt. Deze wordt geactiveerd door middel van het property activeLink, dat naast het property navigation wordt doorgegeven aan het component NavigationBar.

```
1 <NavigationBar navigation={props.navigation} activeLink='home'></NavigationBar>
```

Het component NavigationBar moeten we aan elke component toevoegen dat binnen een Screen component wordt gebruikt. We kunnen deze niet eenmalig aan de functie render van het component App toevoegen, aangezien het object navigation met functie navigate alleen toegankelijk is vanuit componenten die benut worden binnen een scherm zoals Home.js. Het component NavigationBar wordt, samen met een View component dat dienst doet als container voor de inhoud van de pagina, omringd door een tweede View component dat beide omvat. De twee Views krijgen een style property flex met waarde 1 om de volledige beschikbare ruimte van de webbrowser te benutten.

Home.js

```
1  export default function Home(props) {
2      return (
3          <View style={styles.container}>
4              <NavigationBar navigation={props.navigation} activeLink='home'></
                   NavigationBar>
5              <View style={styles.content}>
6                  <Text>Home page</Text>
7              </View>
8          </View>
9          );
10     }
11
12 const styles = StyleSheet.create({
13     container: {
14         flex: 1
15     },
16     content: {
17         flex: 1
18     }
19 });
```

URL bar

Een andere essentieel onderdeel van navigatie binnen een webapplicatie is de URL bar. Het biedt de gebruiker een overzicht van de applicatie structuur en de huidige webpagina waarop men zich bevind, net zoals de titel in de header bar van React Navigation. Met de huidige implementatie blijft de URL onveranderd tijdens het navigeren tussen schermen. Ook hierover heeft React Navigation nagedacht tijdens de ondersteuning van het web. Met behulp van het property linking van de NavigationContainer en Link components kunnen we navigeren aan de hand van paden in plaats van schermnamen en wordt de URL bar overeenkomstig aangepast.

De NavigationContainer accepteert een property linking met als waarde een object dat binnenkomende links behandelt. Dit object moet een property config bevatten, dat op zijn beurt een object als waarde heeft die dienst doet als configuratie van de schermen en hun paden binnen de link structuur. In dit voorbeeld maken we gebruik van genestelde navigators. Ook de configuratie structuur moet deze nesteling aantonen. De twee schermen binnen de Drawer navigator rootHome en rootAccount bevinden zich op het hoogste niveau van de link structuur en worden dus als eerste in de vorm van properties toegevoegd aan config. Merk op dat de namen van deze properties identiek moeten zijn aan de namen van de schermen. Het scherm rootAccount bevat op zijn beurt een Stack navigator met daarin twee schermen account en settings. Aangezien we met een geneste structuur te maken hebben, voegen we een property screens toe aan het object van rootAccount, die op zijn beurt een object bevat met daarin de properties van de twee schermen. De waarde van deze twee properties is dit keer geen object, maar een string waarde die het pad naar het component definieert. Let op dat dit pad niet de navigatie structuur bepaalt. Dit wordt samengesteld door de boomstructuur waarin de properties en objecten zich bevinden binnen het object van het property config. De string waardes bepalen slechts de weergave van het pad van het component in de URL bar. Het component home is dus te vinden met het pad /home en via /account/settings kunnen we navigeren naar het component settings.

App.js

```
1  const linkStructure = {
2      config: {
3          rootHome: {
4              screens: {
5                  home: 'home'
6              }
7          },
8          rootAccount: {
9              screens: {
10                 account: 'account',
11                 settings: 'account/settings'
12             }
13         }
14     }
15 };
```

Vervolgens voegen we het configuratie object linkStructure toe aan het property linking van de NavigationContainer. Als we nu navigeren tussen de schermen, dan stellen we vast dat de URL bar aangepast wordt naar de string waardes van het huidige scherm die we definieerden in de link structuur. We kunnen ook manueel via de URL surfen naar een bepaald scherm. Indien we https://localhost:19006/account ingeven in de URL bar, dan herlaadt de webbrowser naar de pagina waarin het component account zich bevindt.

App.js

```
1 export default function App() {
2     return (
3         <NavigationContainer linking={linkStructure}>
4             ...
5         </NavigationContainer>
6     );
7 }
```

Merk op dat de URL bar zich niet altijd correct aanpast. Wanneer we navigeren naar een andere pagina, past deze zich correct aan met het nieuwe component, maar het pad in de URL blijft mogelijk onveranderd. Zoals vermeld in de documentatie is de ondersteuning van React Navigation in de webbrowser nog volop in ontwikkeling. Dus bugs zoals deze kunnen zich voorlopig voordoen.

Web gedrag

Een andere klein, maar belangrijk onderdeel bij navigatie op het web, is bijvoorbeeld het vermogen om te klikken op een link met de rechter muisknop en deze te openen in een nieuwe tab of venster van de webbrowser. Hier treedt het Link component op in het verhaal. Het biedt op zich dezelfde functionaliteit aan als de functie navigation.navigate, behalve dat het web gedrag behoudt binnen React Navigation. Om deze reden beveelt de library het Link component aan over de traditionele functie navigation.navigate. We vervangen dus de Text componenten door Link componenten. De navigation.navigate functionaliteit wordt vervolgens omgewisseld met het property to van een Link. Deze verwacht een absoluut pad naar het gewenste component, zoals gedefinieerd in het link structuur object. Als resultaat hebben we opnieuw het web gedrag zoals we vertrouwelijk zijn bij traditionele web navigaties.

NavigationBar.js

```
1 export default function NavigationBar(props) {
2     return (
3         <View style={styles.container}>
4             <Link style={[styles.link, props.activeLink === 'home' ? styles.active :
                  null]} to='/home'>Home</Link>
5             <Link style={[styles.link, props.activeLink === 'account' ? styles.active :
                  null]} to='/account'>Account</Link>
6             <Link style={[styles.link, props.activeLink === 'settings' ? styles.active :
                  null]} to='/account/settings?name=Max'>Settings</Link>
7         </View>
8     );
9 }
```

Alle code omtrent de bevindingen van navigatie is te vinden op https://github.com/JayvM/react-native-web-navigation.

Parameters

Merk op dat we via het object met sleutel params, binnen het object dat wordt meegegeven aan navigation.navigatie, query parameters kunnen definiëren aan het te navigeren scherm. We kunnen ook zo'n parameters toevoegen aan het pad dat wordt meegegeven met het property to van een Link component, zoals aangetoond in het derde Link component van bovenstaande code. Indien we echter willen gebruik maken van URI parameters, moeten we deze definiëren in de link structuur van het property linking van de NavigationContainer. In onderstaand voorbeeld voegen we een URI parameter id toe aan het pad van het settings scherm. URI parameters worden gedefinieerd met een voorgaand : karakter. Stel dat we vervolgens naar de URL http://localhost:19002/account/settings/12 navigeren via de URL bar, dan kunnen we deze parameter opvragen in het scherm settings met behulp van de code props.route.params.id.

App.js

```
1 ...
2 rootAccount: {
3     screens: {
4         account: 'account',
5         settings: 'account/settings/:id'
6     }
7 }
8 ...
```

Tekortkomingen

React Navigation voorziet een goede basis voor navigatie op het web dat bloeit vanuit mobiele navigatie bedoelt voor native platformen. Desondanks is deze basis niet voldoende voor heel wat webapplicaties. Het is bijvoorbeeld niet mogelijk om een 404 pagina te implementeren in geval dat er genavigeerd wordt naar een URL die niet overeenkomt met één van de gedefinieerde paden uit de link structuur. Stel dat we in ons voorbeeld navigeren naar https://localhost:19006/account/test, dan valt de navigatie terug op het eerste hogere scherm dat gedefinieerd werd in de link structuur dat wel bestaat. In dit geval dus het scherm account.

Alternatieven

React Navigation is niet de enige library die navigatie verzorgt binnen een React Native project en uitbreiding aanbiedt voor het web. React Router is bijvoorbeeld ook een kandidaat (Training, 2020). Deze heeft onder andere wel ondersteuning voor een 404 pagina bij een niet gedefinieerd pad in de URL. Beide libraries bieden dus hoogstwaarschijnlijk voor- en nadelen ten opzichte van elkaar, maar onderzoek hiernaar ligt buiten de scope van deze studie en kan mogelijk in een toekomstige studie onderzocht worden.

6. Conclusie

In deze studie hebben we onderzocht in welke mate men een applicatie kan ontwikkelen voor twee aparte technologieën: mobile native platformen en het web, met behulp van React Native Web binnen een Expo project.

Uit het onderzoek naar de ondersteunde React Native componenten en API's binnen React Native Web kunnen we besluiten dat React Native Web alle belangrijkste basisonderdelen ondersteunt voor de ontwikkeling van applicaties. Zoals aangetoond, kunnen we niet ondersteunde componenten en API's makkelijk vervangen door eigen implementaties op basis van bouwstenen die wel ondersteund worden. Dit zorgt er echter voor dat de ontwikkelaar extra inspanning moet leveren zodat de applicatie operationeel is in de webbrowser.

Dat een applicatie kan ontwikkeld worden voor zowel mobile native platformen als het web via één codebase, wordt bevestigd aan de hand van het proof-of-concept. Een cross-technology applicatieontwikkeling middel zoals React Native Web zorgt dus wel degelijk voor een 'write once, render anywhere' ervaring, zoals verwacht.

Vervolgens komen we bij de vraag of ontwikkelaars zoals Endare BVBA, die onder andere digitale producten leveren in de vorm van webapplicaties en mobile native applicaties, tijd en geld kunnen besparen bij het gebruik van React Native Web binnen een Expo project. In deze studie voorzagen we een exploratief onderzoek naar de mogelijkheden, limitaties en knelpunten van deze library. Uit het onderzoek kunnen we dus veronderstellen dat ontwikkelaars waarschijnlijk tijd en geld besparen bij het gebruik van React Native Web binnen een Expo project. Dit is slechts een theoretische veronderstelling. In een toekomstige studie kunnen we beide ontwikkelingsprocessen in kaart brengen en vergelijken op gebied van tijd en inspanning: de traditionele manier waarin een webapplicatie en een mobile

native applicatie onafhankelijk van elkaar worden ontwikkeld met behulp van verschillende ontwikkelingsmethodes én het ontwikkelingsproces met behulp van React Native Web en Expo.

In welke mate React Native Web kan gebruikt worden, hangt af van de business use case waarvoor de applicatie wordt ingeschakeld. Als de ontwikkelaar zich beperkt tot de set van ondersteunde componenten en API's van React Native Web, en indien nodig, extra inspanning levert om niet ondersteunde onderdelen te vervangen met externe alternatieven of eigen implementaties, dan is het mogelijk om een werkelijke 'write once, render anywhere' ervaring te beleven. Deze eigen implementaties zorgen er echter voor dat de code uit veel herhalende condities bestaat, met behulp van de Platform module, waarin er onderscheid wordt gemaakt tussen mobile native platformen en het web. Gelukkig kunnen we gebruik maken van specifieke bestand-extensies om code, specifiek voor een bepaalde technologie, op te splitsen in aparte bestanden en zo herhalende condities te verminderen, evenals overzichtelijkheid van het project te behouden. Ook voor navigatie moet de ontwikkelaar extra inspanning leveren indien men geen gebruik wil maken van een mobile native typerende navigatie op het web. Gebruik makende van de ondersteunde componenten en API's, is React Native dus volledig operationeel op het web zonder nood aan drastische wijziginen.

React Native Web en Expo bevatten echter geen eerste stabiele release. Deze bevinden zich nog steeds in ontwikkelingsfase, evenals externe middelen zoals React Navigation. Bugs en problemen kunnen zich dus mogelijk presenteren, zoals vastgesteld werd bij de navigatie van React Navigation. Het ontwikkelen van applicaties met behulp van deze middelen en vervolgens in productie brengen, vormt dus een risico.

Er bevinden zich ook enkele gebreken in de documentatie van React Native Web. Sommige componenten en API's missen documentatie voor properties en methodes en andere onderdelen bevatten zelfs helemaal geen documentatie. Het zou de library ook goed doen indien deze wordt overgenomen door een professioneel team van ontwikkelaars, zodat de vooruitgang en kwaliteit van de software verbeterd en de last niet op één persoon en een paar bijdragers valt. Vooral vanuit het business perspectief, vormt dit een groot risico.

Tenslotte heeft React Native Web toch hetgeen gerealiseerd dat men niet voor mogelijk hield: het samenvoegen van twee aparte en verschillende werelden, namelijk mobile native platformen en het web. De library maakt het mogelijk om met behulp van één codebase, meerdere applicaties te ontwikkelen voor beide technologieën. Het biedt in theorie voordeel in tijd en kosten. Maar wegens de onstabiliteit van de ondersteuning is het dus nog afwachten vooraleer deze middelen kunnen gebruikt worden in de praktijk. Uiteindelijk kunnen we besluiten dat cross-technology een oplossing is voor het probleem van de ontwikkeling van verschillende applicaties voor twee verschillende technologieën.

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

Vandaag de dag wordt er voor bijna elke applicatie twee types ontwikkeld: een webapplicatie voor in de browser en één of meerdere mobile native applicaties voor platformen zoals Android, iOS, Windows, enz. Beide technologieën bieden voor- en nadelen ten opzichte van elkaar: een mobile native applicatie levert bijvoorbeeld meer controle en betere performantie, maar is niet zo universeel als een webapplicatie. Omdat men alle voordelen wilt benutten en het grootste bereik van eindgebruikers wilt garanderen, is het belangrijk om een applicatie voor beide technologieën beschikbaar te stellen. Als nadelig gevolg neemt het ontwikkelingsproces en het onderhoud aanzienelijk meer tijd, kosten en inspanning in beslag.

Op gebied van mobiele applicaties moet men ook beslissingen nemen. Android en iOS zijn de twee populairste platformen, maar beide hebben een verschillende programmeertaal en ontwikkelingsproces. Bouwt men een applicatie voor Android, iOS of beide? Hiervoor komt hybrid applicatieontwikkeling als oplossing naar voren. Zogenaamde cross-platform frameworks zoals Xamarin (de Icaza e.a., 2011), React Native (Walke & Facebook, 2015), Ionic (Sperry e.a., 2012), enz. maken gebruik van één codebase voor het renderen en onderhouden van één mobiele applicatie op meerdere native platformen.

Toch lossen hybrid applicatieontwikkeling of cross-
platform frameworks niet het totaalplaatje op. Ontwikkelaars zitten nog steeds met het

probleem dat men twee types applicaties in verschillende technologieën moet ontwikkelen en onderhouden: mobile native applicaties via hybrid applicatieontwikkeling én webapplicatieontwikkeling. De laatste jaren zijn enkele frameworks boven water gekomen die dit probleem oplossen. Voorbeelden zijn Flutter (Google, 2017) en React Native Web (Gallagher, 2015). Deze hebben geen officiële benaming zoals 'cross-platform', dus verwijzen we in deze studie naar 'cross-technology' als omvattende term voor deze frameworks en dit type applicatieontwikkeling.

Endare BVBA (Goossens, 2012) is een bedrijf dat gespecialiseerd is in het ontwikkelen van digitale producten zoals websites en mobiele applicaties. Kunnen Endare en andere ontwikkelaars voordeel halen uit een cross-technology framework dat zowel web- als hybrid applicatieontwikkeling samenvoegt? Op het eerste zicht zou er minder tijd, kosten en inspanning nodig zijn voor de ontwikkeling en het onderhoud van een applicatie. Maar hoe precies zit zo'n framework in elkaar en zit er een addertje onder het gras?

Endare maakt onder andere gebruik van frameworks zoals React (Walke & Facebook, 2013) voor webapplicatieontwikkeling en React Native voor hybrid applicatieontwikkeling. Daarom doen we in deze studie een exploratief onderzoek naar React Native Web: een cross-technology framework dat een React Native applicatie tot leven brengt in de webbrowser.

A.1.1 Onderzoeksvraag

We onderzoeken in welke mate een applicatie kan ontwikkeld worden als mobile native applicatie voor native platformen én als webapplicatie voor in de browser met behulp van het framework React Native Web.

A.1.2 Onderliggende onderzoeksvragen

Naast de onderzoeksvraag willen we ook weten hoeveel extra inspanning er nodig is om de applicatie volledig operationeel te maken op beide technologieën of met andere woorden, heeft de React Native applicatie extra hulp nodig om te functioneren in de browser? Zijn er hierbij drastische wijzigingen nodig in de code? Worden er onderdelen van de applicatie beïnvloed?

We bekijken ook algemeen tot op welke hoogte cross-technology een oplossing is voor het probleem van twee aparte ontwikkelingstechnologieën: web- en hybrid applicatieontwikkeling, en vergelijken deze met bepaalde business use cases.

A.2 Stand van zaken

Kosten en tijd zijn belangrijke factoren voor de ontwikkeling van een applicatie. Hybrid applicatieontwikkeling lost het probleem van meerdere verschillende native platformen op, aangetoond in 'Real Challenges in Mobile App Development (Joorabchi e.a., 2013), en

reduceert de kosten en tijd voor de ontwikkeling van een mobiele applicatie zoals bewezen in 'An Introduction to Hybrid Platform Mobile Application Development' (Khandeparkar e.a., 2015).

Hybrid applicatieontwikkeling of cross-platform frameworks zoals Xamarin, React Native en Ionic worden hierdoor als waardige keuze beschouwd door ontwikkelaars voor het bouwen van een applicatie in de afgelopen jaren. Voor React Native zijn bijvoorbeeld verscheidene artikels zoals 'React Native for Android: How we built the first cross-platform React Native app' (Facebook, 2015) te vinden op het internet en allerlei handleidingen, documentatie en API's beschikbaar op de officiële website (Facebook, 2020). Talloze studies hebben ook onderzoek gedaan naar de performantie en bruikbaarheid van cross-platform frameworks waaronder React Native door Hansson en Vidhall (2016).

De term 'write once, render anywhere' of een sterke variant wordt vaak gebruikt binnen de wereld van hybrid of cross-platform applicatieontwikkeling. Hiermee wordt eerder 'render on every native platform' bedoeld dan 'render everywhere'. Als we verder kijken naar cross-technology is er verrassend weinig materiaal te vinden op het internet aangezien deze wereld nog volop moet openbloeien. Het framework Flutter staat al in de kijker maar beschikt over weinig tot geen studies en onderzoeken. De ondersteuning voor webapplicatieontwikkeling in Flutter bevindt zich dan ook slechts in beta fase zoals beschreven in de documentatie op de officiële website (Flutter, 2020b). React Native Web loopt in de voetsporen van Flutter, maar ook met zeer weinig tot geen beschikbaar materiaal op het internet. Artikels zoals 'React Native Web: Write Once Render Anywhere' (Singh, 2018) en de presentatie van Peggy Rayzis op ReactNext 2017 (Rayzis, 2017) geven een introductie tot het framework met minimale uitleg, voorbeelden en code.

Algemeen kunnen we uit deze korte literatuurstudie besluiten dat cross-technology frameworks nog in haar kinderschoentjes staat. Er is haast geen onderzoek gedaan naar zo'n frameworks qua exploratie, bruikbaarheid, performantie, enz.

A.3 Methodologie

In eerste instantie bestaat het onderzoek uit een introductie over de huidige stand van zaken binnen web en hybrid applicatieontwikkeling, het probleem van twee aparte ontwikkelingstechnologieën en cross-technology applicatieontwikkeling als oplossing. Hiernaast wordt de werking van cross-technology ook beknopt beschreven en geïllustreerd.

In de vorm van een exploratief onderzoek gaan we verder in op cross-technology applicatieontwikkeling met behulp van het framework React Native Web. We verkennen de vier grote fases van het ontwikkelingsproces:

- Development (ontwikkelen)
- Testing (testen)
- Deployment (uitrollen)
- Maintenance (onderhouden)

A.3.1 Development

We verdiepen ons in de werking en mogelijkheden van het framework aan de hand van code en uitgewerkte voorbeelden. In dit onderzoek maken we hiervoor gebruik van Visual Studio Code (Microsoft, 2015) als ontwikkelingsomgeving. React Native Web maakt gebruik van React Native components en API's, maar welke originele bouwstenen kunnen we precies gebruiken en in welke mate worden deze ondersteund? We onderzoeken de limitaties van het framework in components en API's van React Native, alsook specifieke hardware features van het mobiele apparaat. Enkele voorbeelden:

- Camera
- Microfoon
- Luidsprekers
- Push notifications
- Touch gestures

A.3.2 Testing

Testing is een belangrijke fase in het ontwikkelingsproces. Het spoort onder andere problemen en bugs op in de code en garandeert de kwaliteit van het product. We onderzoeken hoe we een applicatie, ontwikkeld met behulp van React Native Web, kunnen testen. Beschikt het framework over eigen methodes voor het product te testen of maakt men gebruik van externe manieren?

A.3.3 Deployment

In deze fase bekijken we het deployment proces. Hoe stellen we de applicatie beschikbaar op alle technologieën? Gebeurt dit automatisch door React Native Web? We komen te weten of er extra inspanning nodig is om de applicatie volledig operationeel te maken op beide technologieën en of er invloed is op bepaalde components of API's.

A.3.4 Maintenance

Als laatste fase onderzoeken we het onderhoud van een cross-technology applicatie. Kan men gemakkelijk aanpassingen doorvoeren, nieuwe features toevoegen of verwijderen, en hebben updates van native platformen impact op de applicatie?

A.3.5 Proof-of-concept

Aan de hand van het exploratief onderzoek en de uitgewerkte voorbeelden maken we een basis applicatie als proof-of-concept om de onderzoeksvraag te beantwoorden: in welke mate kan een applicatie ontwikkeld worden voor zowel mobile native platformen als voor de webbrowser en hoeveel extra inspanning is er nodig om deze volledig operationeel te

maken op beide technologieën. Uiteindelijk kunnen we ook cross-technology applicatieontwikkeling vergelijken met web- en hybrid applicatieontwikkeling als twee aparte ontwikkelingsprocessen aan de hand van het proof-of-concept. Hieruit vallen de voor- en nadelen af te leiden en kunnen we concluderen of ontwikkelaars al dan niet voordeel kunnen halen uit het framework React Native Web in bepaalde business use cases.

A.4 Verwachte resultaten

Uit het onderzoek verwachten we dat het framework een React Native applicatie, zonder al te veel extra inspanning en aanpassingen in de code, vlot kan converteren naar een webapplicatie voor in de browser. Aangezien het framework zich nog steeds in een ontwikkelingsfase bevindt, zullen bugs en problemen hoogstwaarschijnlijk voorkomen. Qua limitaties verwachten we dat het framework de meeste componenten en API's van React Native ter beschikking stelt voor de ontwikkelaar waaronder zeker de belangrijkste. Als men spreekt over features op hardware niveau, verwachten we dat React Native Web hierin zeer gelimiteerd is.

A.5 Verwachte conclusies

Het onderzoek geeft ons een inzicht in de wereld van cross-technology applicatieontwikkeling door de ogen van het framework React Native Web. We maken kennis met de verschillende fases van het ontwikkelingsproces voor applicaties en onderzoeken de mogelijkheden en limitaties van het framework. Uiteindelijk concluderen we ook de voor- en nadelen en business use cases waarin React Native Web kan gebruikt worden.

Uit de resultaten verwachten we dat React Native Web een sterk framework is en zeker aandacht verdient binnen cross-technology applicatieontwikkeling. Het biedt de meeste en belangrijkste componenten en API's van React Native aan voor de ontwikkeling van applicaties die zowel op web als mobile native platformen functioneren. Op gebied van hardware features zoals camera en push notifications staat het framework aanzienlijk achter.

Uit het onderzoek kunnen we aantonen dat React Native Web een waardige keuze is voor de ontwikkeling van eenvoudige applicaties die niet overdreven complex zijn. Hiermee worden applicaties getypeerd die bestaan uit basis componenten zonder het gebruik van gecompliceerde animaties, 3D graphics, hardware features, enz. Voorbeelden van basis componenten:

- Text en images
- Textinputs, buttons, radiobuttons, checkboxes, switches, dropdownlists
- Views en scrollviews
- Menu en navigation

Voorbeelden van eenvoudige, niet complexe applicaties:

- Chat applicatie: berichten versturen
- Nieuws applicatie: nieuwsberichten bekijken
- Shopping applicatie: producten bekijken en bestelling plaatsen
- Forum applicatie: berichten bekijken en plaatsen

Bedrijven en ontwikkelaars zoals Endare BVBA besparen veel tijd, kosten en inspanning voor de ontwikkeling van applicaties wanneer men gebruik maakt van cross-technology applicatieontwikkeling frameworks zoals React Native Web.

Bibliografie

AbstractBrain. (2015). Pushpad. Verkregen van https://pushpad.xyz/

Ancheta, W. (2020, maart 24). React Native Web vs. Flutter web. Verkregen van https://blog.logrocket.com/react-native-web-vs-flutter-web/

Arvidsson, J. (2015). React Native Vector Icons. Verkregen van https://github.com/oblador/react-native-vector-icons

Cheever, C. & Ide, J. (2013). Expo. Verkregen van https://expo.io/

Community, R. N. (2015a). React Native Camera. Verkregen van https://react-native-community.github.io/react-native-camera/

Community, R. N. (2015b). *React Native Device Info*. Verkregen van https://github.com/react-native-community/react-native-device-info

Community, R. N. (2019a). *React Native CheckBox*. Verkregen van https://github.com/react-native-community/react-native-checkbox

Community, R. N. (2019b). *React Native Picker*. Verkregen van https://github.com/react-native-community/react-native-picker

Community, R. N. (2019c). *React Native ProgressBarAndroid*. Verkregen van https://github.com/react-native-community/progress-bar-android

Community, R. N. (2019d). *React Native ProgressView*. Verkregen van https://github.com/react-native-community/progress-view

Dalmasso, I., Datta, S. K., Bonnet, C. & Nikaein, N. (2013). *Survey, Comparison and Evaluation of Cross Platform Mobile Application Development Tools* (Mobile Communication Department, EURECOMSophia Antipolis, France). Verkregen van http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.849.9061&rep=rep1&type=pdf

de Icaza, M., Friedman, N. & Microsoft. (2011). Xamarin. Verkregen van https://dotnet.microsoft.com/apps/xamarin

Docs, M. W. (2020). *Browser detection using the user agent*. Verkregen van https://developer.mozilla.org/en-US/docs/Web/HTTP/Browser_detection_using_the_user_agent

Duplessis, J. (2017, februari 14). Using Native Driver for Animated. Verkregen van https://reactnative.dev/blog/2017/02/14/using-native-driver-for-animated

Expo. (2020a). *Expo AuthSession*. Verkregen van https://docs.expo.io/versions/v37.0.0/sdk/auth-session/

Expo. (2020b). *Expo BarCodeScanner*. Verkregen van https://docs.expo.io/versions/v37.0.0/sdk/bar-code-scanner/

Expo. (2020c). *Expo Camera*. Verkregen van https://docs.expo.io/versions/v37.0.0/sdk/camera/

Expo. (2020d). *Expo Device*. Verkregen van https://docs.expo.io/versions/v37.0.0/sdk/device/

Expo. (2020e). *Expo Facebook*. Verkregen van https://docs.expo.io/versions/v37.0.0/sdk/facebook/

Expo. (2020f). *Expo FileSystem*. Verkregen van https://docs.expo.io/versions/v37.0.0/sdk/filesystem/

Expo. (2020g). *Expo Location*. Verkregen van https://docs.expo.io/versions/v37.0.0/sdk/location/

Expo. (2020h). *Expo Notifications*. Verkregen van https://docs.expo.io/versions/v37.0.0/sdk/notifications/

Expo. (2020i). *Expo Sensors*. Verkregen van https://docs.expo.io/versions/v37.0.0/sdk/sensors/

Expo. (2020j). *Expo Speech*. Verkregen van https://docs.expo.io/versions/v37.0.0/sdk/speech/

Expo. (2020k). *Expo Video*. Verkregen van https://docs.expo.io/versions/v37.0.0/sdk/video/

Expo. (2020l). *Introduction to Expo*. Verkregen van https://docs.expo.io/

Facebook. (2015). React Native for Android: How we built the first cross-platform React Native app. Verkregen van https://engineering.fb.com/developer-tools/react-native-for-android-how-we-built-the-first-cross-platform-react-native-app/

Facebook. (2016a). Create React App. Verkregen van https://create-react-app.dev/

Facebook. (2016b). Metro. Verkregen van https://facebook.github.io/metro/

Facebook. (2020). *React Native Documentation*. Verkregen van https://reactnative.dev/docs/getting-started

Flutter. (2020a). *Building a web application with Flutter*. Verkregen van https://flutter.dev/docs/get-started/web

Flutter. (2020b). *Web support for Flutter*. Verkregen van https://flutter.dev/web

Gallagher, N. (2015). React Native Web. Verkregen van https://github.com/necolas/react-native-web

Gallagher, N. (2017). Twitter Lite, React Native, and Progressive Web Apps. Verkregen van https://www.youtube.com/watch?v=tFFn39lLO-U

Gallagher, N. (2018a, juli 7). React Native Web Issue #1020 - Modal: implementation. Verkregen van https://github.com/necolas/react-native-web/issues/1020

Gallagher, N. (2018b, juli 7). React Native Web Issue #1026 - Alert: implementation. Verkregen van https://github.com/necolas/react-native-web/issues/1026

Gaunt, M. (2018). *Adding Push Notifications to a Web App*. Verkregen van https://developers.google.com/web/fundamentals/codelabs/push-notifications

Gaunt, M. (2019). *Service Workers: an Introduction*. Verkregen van https://developers.google.com/web/fundamentals/primers/service-workers

Google. (2017). Flutter. Verkregen van https://flutter.dev/

Goossens, S. (2012). Endare BVBA. Verkregen van https://www.endare.com/

Hansson, N. & Vidhall, T. (2016). *Effects on performance and usability for cross-platform application development using React Native* (Linköpings universitet, Linköping, Sweden). Verkregen van https://liu.diva-portal.org/smash/get/diva2:946127/FULLTEXT01.pdf

JavaScript. (2020a). *JavaScript Window.alert() API*. Verkregen van https://developer.mozilla.org/en-US/docs/Web/API/Window/alert

JavaScript. (2020b). *JavaScript Window.confirm() API*. Verkregen van https://developer.mozilla.org/en-US/docs/Web/API/Window/confirm

JavaScript. (2020c). *JavaScript Window.prompt() API*. Verkregen van https://developer.mozilla.org/en-US/docs/Web/API/Window/prompt

Joorabchi, M. E., Mesbah, A. & Kruchten, P. (2013). *Real Challenges in Mobile App Development* (University of British Columbia, Vancouver, BC, Canada). Verkregen van http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.724.2463&rep=rep1&type=pdf

Khandeparkar, A., Gupta, R. & Sindhya, B. (2015). *An Introduction to Hybrid Platform Mobile Application Development* (Mukesh Patel School of Technology, Management en Engineering, Mumbai, India). Verkregen van https://research.ijcaonline.org/volume118/number15/pxc3903463.pdf

Koppers, T., Larkin, S., Ewald, J., Vepsäläinen, J. & Kluskens, K. (2012). Webpack. Verkregen van https://webpack.js.org/

Marquez, J. (2019, april 10). The state of React Native Web in 2019. Verkregen van https://blog.logrocket.com/the-state-of-react-native-web-in-2019-6ab67ac5c51e/

McKenzie, S. (2014). Babel. Verkregen van https://babeljs.io/

Microsoft. (2015). Visual Studio Code. Verkregen van https://code.visualstudio.com/

Natigation, R. (2017). React Navigation. Verkregen van https://reactnavigation.org/

Native, R. (2020a). *React Native Dimensions API*. Verkregen van https://reactnative.dev/docs/dimensions

Native, R. (2020b). *React Native Modal component*. Verkregen van https://reactnative.dev/docs/modal

Native, R. (2020c). *React Native Platform module*. Verkregen van https://reactnative.dev/docs/platform-specific-code

Native, R. (2020d). *React Native Transforms API*. Verkregen van https://reactnative.dev/docs/transforms

NodeJS. (2011, augustus 26). What is npm? Verkregen van https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/

Nwamba, C. (2019, december 12). Build Mobile-Friendly Web Apps with React Native Web. Verkregen van https://www.digitalocean.com/community/tutorials/build-mobile-friendly-web-apps-with-react-native-web

Rayzis, P. (2017). Write Once, Render Anywhere - ReactNext 2017. Verkregen van https://www.youtube.com/watch?v=HLWM2uhv2wI

Rehman, Z. (2019, mei 17). Flutter For Web: A Complete Guide to Create Run a Web Application. Verkregen van https://itnext.io/flutter-for-web-c75011a41956

Schlueter, I. Z., Turner, R. & Marchán, K. (2010). npm. Verkregen van https://www.npmjs.com/

Singh, V. (2018, november 19). React Native Web: Write Once Render Anywhere. Verkregen van https://hashedin.com/blog/react-native-webview-react-native-webview-npm/

Smith, C. (2014). Detect Mobile Browsers. Verkregen van http://detectmobilebrowsers.com/

Sperry, B., Lynch, M. & Co., D. (2012). Ionic. Verkregen van https://ionicframework.com/

StatCounter. (2020a). GlobalStats. Verkregen van https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#monthly-202005-202005-bar

StatCounter. (2020b, mei 1). *Mobile Operating System Market Share Belgium*. Verkregen van https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#monthly-202005-202005-bar

StatCounter. (2020c, mei 1). *Mobile Operating System Market Share Worldwide*. Verkregen van https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/belgium/#monthly-202005-202005-bar

Training, R. (2020). React Router. Verkregen van https://reacttraining.com/react-router/

Walke, J. & Facebook. (2013). React. Verkregen van https://reactjs.org/

Walke, J. & Facebook. (2015). React Native. Verkregen van https://reactnative.dev/