Antti Valkonen

UAV-torjuntasovelluksen käyttöliittymä React Nativella

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

13.9.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Antti Valkonen UAV-torjuntasovelluksen käyttöliittymä React Nativella 29 sivua 13.9.2018
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	tieto- ja viestintätekniikka
Ammatillinen pääaine	Mobile Solutions
Ohjaajat	Lehtori Ilkka Kylmäniemi Sovelluskehittäjä Kalle Laukkanen

Insinöörityön tarkoituksena oli tavoitteena selvittää React Native -sovelluskirjaston vahvuuksia lähdeaineiston avulla ja soveltaa tätä tietoa uuden mobiilikäyttöliittymän tekemisessä yrityksen tuotteelle. Työ tehtiin projektina suomalaiselle dronentunnistusjärjestelmiä valmistavalle yritykselle. Uuden sovelluksen tarkoituksena oli parantaa ruohonjuuritason käyttäjän kokemusta käyttöliittymällä, joka on suunniteltu alusta alkaen matkapuhelimella tai tabletilla käytettäväksi.

Aikaisempi käyttöliittymä on toteutettu React-kirjastolla, ja siinä on ominaisuuksia ja asetuksia, jota loppukäyttäjä, esimerkiksi poliisi tai lennonvalvoja, ei koskaan tarvitse ulkona työskennellessään. Lisäksi sovelluksen käyttöönotto on hidasta niin sanotussa ”tositilanteessa”.

Projektin alussa perehdyttiin React Nativen vahvuuksiin, muun muassa VDOM, JSX, komponentit ja state. Projektin toteutuksessa ei ilmennyt suuria ongelmia, ja React Native -sovelluksen kehitys oli nopeaa ja mielekästä.

Insinöörityön lopputuloksena syntyi toimiva ja tavoitteet täyttävä React Native -sovellus, joka käyttää samaa taustajärjestelmää kuin olemassa oleva React-sovellus. Aikarajoitusten vuoksi sovellusta ei saatu vielä jakoon asiakkaille, ja sitä suoritettiin vain iPhone-simulaattorin avulla MacBook-tietokoneella.

Avainsanat	React, React Native, mobiilikehitys, JavaScript

Author Ttitle Number of Pages Date	Antti Valkonen Anti-UAV user-interface with React Native 29 pages 13 September 2018
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Mobile Solutions
Instructors	Ilkka Kylmäniemi, Lecturer Kalle Laukkanen, Software developer

This thesis was done as a project for a finnish anti-UAV (unmanned aerial vehicle) system manufacturer. The goal was to find out the strengths of the React Native-library through source material, and to apply this knowledge into making a new mobile application for the company's product. The intention of the application was to improve the user experience for the grassroots-level user, with a user interface designed to be used on a mobile device from the get-go.

The existing user interface was built using the React-library. It has features and settings the end user, for example air traffic manager or a police officer, never needs. In addition, getting the application to work in an emergency is slow.

At the start of the project the strengths of React Native were studied. Those include VDOM, JSX, components and state. Project application development proved to be fast and meaningful, and no real problems occurred.

As a result of this thesis a working application was developed that fulfills all goals. Due to time restrictions, the application hasn't been distributed to customers yet, and it was only run on an iPhone-simulator using a MacBook-computer.

Key words	React, React Native, mobile development, JavaScript

Sisällys

Lyhenteet

1 Johdanto 1

2 AIRFENCE-tuote 3

3 Taustatietoa sovelluskehityksestä 4

3.1 Reactin datamalli 4
3.2 JavaScript-ohjelmointikieli 5
3.3 React-sovelluskirjasto 7

4 React Native -sovellus AIRFENCE-tuotteelle 11

4.1 Projektin aloitus 12
4.2 Navigointi ja tilanhallinta sovelluksessa 14
4.3 Sisäänkirjautuminen sovellukseen 16
4.4 Mapbox GL -karttakirjasto 18
4.4.1 iOS-asennus 19
4.4.2 Android-asennus 20
4.4.3 JavaScript-asennus 21
4.5 Websocket-yhteys 24
4.6 Tyyli 25
4.7 Sovelluksen lopputulos 26

5 Yhteenveto 27

Lähteet 28

Lyhenteet

UAV	Unmanned Aerial Vehicle. Miehittämätön kulkuneuvo, yleensä lennokki tai kopteri.
SPA	Single Page Application. Web-sivu tai sovellus, joka lataa sisältönsä dynaamisesti eikä joka kerta lataa sivua uudestaan.
JSX	JavaScript XML. Lisäosa JavaScript-kielen syntaksiin. Yleisesti käytössä React-kirjaston kanssa.
DOM	Document Object Model. HTML- tai XML-koodista muodostuva kokonaisuus, joka esitetään käyttäjälle.
VDOM	Virtual Document Object Model. Ideaalinen kuvaus DOM-rajapinnasta.
MVC	Model, View ja Controller. Yleisesti käytetty arkkitehtuuri erottamaan websovelluksen toiminnalliset osat.
HTTP	Hypertext Transfer Protocol. Protokolla, jolla selaimet ja www-palvelimet hoitavat tiedonsiirron.
SDK	Software Development Kit. Työkalu, jolla voidaan kehittää esimerkiksi websovellus.
URL	Uniform Resource Locator. Tutummin osoite verkkosivulle.
ADS-B	Automatic dependent surveillance – broadcast. Siviililentokoneiden lähettämää julkista sijaintitietoa.
REST	Representational State Transfer. Arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen.

1 Johdanto

Miehittämätön lennokki tai kopteri on lentoalus ilman ihmispilottia, ja tästä tulee niiden yleisesti käytetty nimi UAV eli Unmanned Aerial Vehicle. Alukset ovat pääasiassa kaukoohjattuja. Alusten alkuperäinen tarkoitus oli toimia turvallisena vaihtoehtona miehitetyille sotilasaluksille. Alukset palvelevat yhä sotilaskäytössä, mutta myös yksityishenkilöt ja yritykset ovat ottaneet lennokit käyttöön.

Lennokkeja on eri kokoisia, mutta osat ovat pääasiassa samoja. Lennokit tehdään pitkälti samoista osista kuin älypuhelimet. Älypuhelimien kehityksen vuoksi myös lennokkien hinnat ovat tasoittuneet kuluttajille ja pienemmille yrityksille sopiviksi.

UAV-laitteiden markkina-arvo kasvaa tasaisesti kuluttaja-, kauppa- ja sotilassektorilla. Arvioiden mukaan UAV-laitteet saavuttavat 100 miljardin dollarin markkina-arvon vuosien 2016 ja 2020 välillä. (1.)

Trafi on kehittänyt Suomessa ilmailun säädösmuutoksia, jotka määrittelevät miehittämättömän ilma-aluksen ja lennokin eron. Ne voivat olla samannäköisiä, ja pelkästään käyttötarkoitus erottaa ne toisistaan. Lennokki on harrastekäyttöön tarkoitettu. Miehittämätön ilma-alus painaa enintään 25 kg ja on jotakin tarkoitusta varten, kuten mittaaminen. Trafin mukaan säädösten tavoitteena on turvata muita ilmassa liikkuvia ja maassa olevia ihmisiä. (2.)

Myös EU:lla on suunnitelmissa lisätä UAV-laitteiden sääntelyä, esimerkiksi SECOPS-projektin myötä, jossa myös tämän opinnäytetyön tilaaja on mukana. (3.)

Markkina-arvon huomattava kasvu, teknologian kehitys ja sääntelyn tiukentuminen kertovat siitä, että UAV-laitteilla voi olla suuri vaikutus yhteiskuntaan, niin hyvässä kuin pahassa. Tämän opinnäytetyön tilaaja pyrkii vastaamaan tähän kasvavaan ongelmaan, ja tämän opinnäytetyön tavoitteena on tukea yrityksen UAV-torjuntatuote AIRFENCEä uudella mobiilikäyttöliittymällä.

Pyrin opinnäytetyössä vastaamaan seuraaviin kysymyksiin:

- Miten React Native toimii?
- Miten tehdään mobiilisovellus React Nativella?

Opinnäytetyön tavoitteena on luoda React Nativella mobiilisovellus tukemaan nykyistä AIRFENCE-tuotetta. Työssä keskitytään vain käyttöliittymään, sillä AIRFENCEllä on jo valmis taustajärjestelmä tietokantoineen.

Yhdessä työnantajani kanssa valitsimme React Nativen, koska tuotteen alkuperäinen käyttöliittymä on tehty React-kirjastolla. Olen myös tehnyt yhden opiskeluprojektin React Nativella. Olisimme voineet valita jonkin toisen teknologian, esimerkiksi Cordovan tai Xamarinin, mutta aikaisempien kokemuksieni ja olemassa olevan sovelluksen takia React Native tuntui luonnollisimmalta. React Nativen toiminta nyt ja tulevaisuudessa on myös varmaa, toisin kuin aikaisemmin mainittujen teknologioiden.

Alkuperäisestä React-käyttöliittymästä olisi mahdollista käyttää koodia pienillä muutoksilla, mutta usean henkilön kirjoittamassa vanhassa koodissa voi olla virheitä, jotka hidastavat kehitystä ja sovelluksen käyttöä, eikä sitä voisi kutsua omaksi opinnäytetyöksi. Näiden syiden vuoksi opinnäytetyön käytännön osa aloitetaan aivan alusta. Tämän lisäksi uuden mobiilikäyttöliittymän funktio on jossain määrin eri kuin alkuperäisen React-käyttöliittymän.

2 AIRFENCE-tuote

AIRFENCE on Sensofusion Oy:n ensimmäinen ja tällä hetkellä ainoa tuote. Yksinkertaistettuna Se on tietokone, johon on liitetty ohjelmistoradio. Se pystyy havaitsemaan, paikallistamaan ja seuraamaan lennokkeja automaattisesti. Tämän lisäksi AIRFENCE pystyy paikallistamaan lennokin ohjaajan reaaliajassa.

AIRFENCE-sensoria käyttävät muun muassa lentokentät, vankilat, sähkölaitokset, valtioiden virastot ja sotilastukikohdat estämään miehittämättömien lennokkien häiritsevän toiminnan. Sensofusion Oy:n käyttäjiin ja yhteistyökumppaneihin kuuluvat esimerkiksi Yhdysvaltain merijalkaväki, Yhdysvaltain ilmailuhallinto, NASA, NATO ja EU. (4.)

Vuonna 2017 Sensofusion julkaisi AIRFENCE-sensorin 5.0-version, joka sisälsi päivityksiä havaintonopeuteen ja havaintojen suuntaan. Uusina ominaisuuksina AIRFENCE kykenee ottamaan lennokin haltuun tai estämään sen toiminnan. Laitteen kotelo on tässä versiossa lujatekoinen ja testattu aidoissa sotilasolosuhteissa. Laitteen asennusta ja käyttöönottoa on myös helpotettu. (5.)

AIRFENCE-tuotteeseen kuuluu sensorin lisäksi suunta-antenni havaitsemaan UAV-lennokit, ja käyttöliittymä, jossa näkyy kaikkien AIRFENCE-sensorien data, jota käyttäjällä on oikeus nähdä. Nämä sensorit havaitsevat pääasiassa 2,4–5,8 GHz:n radiotaajuusalueella toimivia UAV-laitteita, mutta myös Wi-Fi:llä toimivat laitteet voidaan havaita. Tämän lisäksi käyttöliittymän taustajärjestelmästä saadaan geofence-tietoa, eli käyttäjän määrittelemä lentokieltoalue. UAV-laitteen saapuessa geofence-alueelle voidaan AIRFENCE-sensori asettaa tekemään ilmoitus käyttäjälle tai vastatoimi havaitulle UAV-laitteelle automaattisesti. Lisäominaisuutena sensori pystyy myös havaitsemaan siviililentokoneita, jotka lähettävät julkista ADS-B (Automatic dependent surveillance – broadcast) -sijaintitietoa.

Miehittämättömät lennokit ovat yleistyneet yhteiskunnassa nopeasti. Niistä on huomattavaa hyötyä muun muassa pelastustoimiin, valvontaan ja kuljetukseen. Lennokkien hyötytarkoitusten lisäksi niitä käytetään myös rikolliseen toimintaan. Tämä voi tarkoittaa esimerkiksi huumeiden tai matkapuhelinten salakuljettamista vankilaan tai sotilaskuljetuksen salakuvaamista. Sensofusion Oy kehittää AIRFENCE-tuotetta, jonka tarkoituksena on antaa viranomaisille keino estää rikollinen toiminta ja myös estää siviililennokeilla tehty tahaton häirintä, esimerkiksi kaupungin väkijoukossa. AIRFENCE-

tuotteeseen kuuluu fyysinen laite, joka havaitsee ja estää lennokit, sekä web-käyttöliittymä, jolla voidaan antaa komentoja laitteelle ja reagoida laitteesta saatuun dataan.

Käyttöliittymällä on useita viranomaiskäyttäjiä, ja se on myös Sensofusionin sisäisessä käytössä jatkuvasti. Viranomaisilta saatu palaute ja yrityksen omat testit viittaavat samaan asiaan: käyttöliittymä on hankala saada päälle ja toimintakuntoon tositilanteessa, esimerkiksi kun luvatonta UAV-laitteen lennättäjää yritetään etsiä.

3 Taustatietoa sovelluskehityksestä

3.1 Reactin datamalli

Perinteisissä WWW-sovelluksissa ohjelmakoodi ja käyttäjälle näkyvä HTML-koodi on sekoitettu keskenään. Pienissä käyttökohteissa tämä saattaa toimia, mutta vähänkin suuremmalla kuormalla se tekee ohjelmasta virhealttiin. Tämä johtuu siitä, että HTML-koodia generoidaan palvelimelta sitä mukaa kuin ohjelmaa suoritetaan.

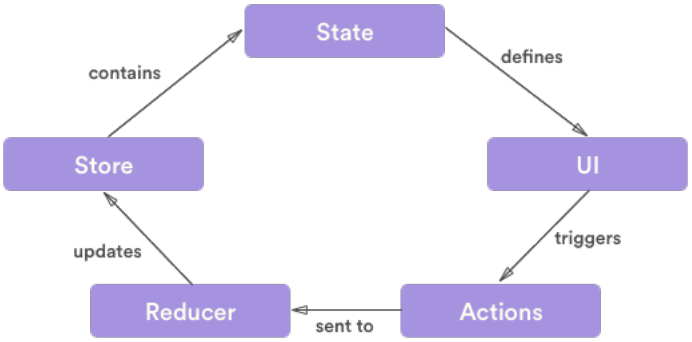
Pitkälle 2010-luvulle käytetty yleinen ratkaisu oli ohjelmakoodin ja HTML-koodin eristäminen. HTML-koodia tuotettaisiin tässä tapauksessa vain tarvittaessa ja muokataan DOM- tai muun rajapinnan kautta. Tämä ratkaisu vaatii tietokoneelta enemmän muistia, mutta selkeyttää ja helpottaa koodia ja sen suorittamista.

Tätä ratkaisua kutsuttiin MVC-malliksi (Model – View – Controller).

- Model eli malli hoitaa järjestelmän tiedon tallentamisen, ylläpidon ja käsittelyn eli vastaa tietokantaa.
- Controller eli käsittelijä vastaanottaa käyttäjältä tulevat komennot ja muuttaa mallia niiden mukaisesti.
- View eli näkymä määrittää käyttöliittymän ulkoasun eli sen mitä käyttäjä näkee web-sivulla tai sovelluksessa. Yksinkertaisimmillaan HTML-sivu, monimutkaisemmissa ratkaisuissa sisältää myös esimerkiksi JavaScript-koodia. (6.)

MVC-mallissa data voi kulkea kaksisuuntaisesti. Käyttäjä painaa nappia näkymässä, joka lähettää viestin käsittelijälle, joka vuorostaan päivittää mallin. Malli muuttaa jotakin arvoa ja palauttaa sen käsittelijälle, joka päivittää näkymän.

Modernimpi tapa käsitellä dataa on yleisesti Reactinkin kanssa käytetty Redux. Reduxin perusidea on se, että sovelluksilla on yksittäinen kontti (store), johon dataa päivitetään kuvan 1 mukaisesti.



Kuva 1. Reduxin datamalli. (7.)

Reduxissa data kulkee vain yhteen suuntaan. Käyttöliittymä (UI) laukaisee toiminnan (action), joka kertoo päivittäjälle (reducer), millainen kontin (store) tulee olla. Kontti päivittää komponentin tilaa (state), joka määrittää, miltä käyttöliittymä näyttää. Tämän datamallin etuna esimerkiksi pelkän Reactin tilan käyttämiseen on se, että lapsikomponenteille (child) ei tarvitse jakaa props-objektia datan näyttämiseen tai muuttamiseen. Data kulkee yksinkertaisemmin ja vähemmillä virheillä, kun kaikki käyttöliittymän osat ovat yhteydessä Reduxin konttiin. (7.)

3.2 JavaScript-ohjelmointikieli

JavaScript on alun perin Netscapen kehittämä ohjelmointikieli, jonka tarkoituksena oli lisätä funktionaalisuutta staattisiin sivuihin. JavaScriptiä on perinteisesti suoritettu selaimella. JavaScript voi esimerkiksi kertoa ennen napin painallusta käyttäjälle, että hänen sähköpostiosoitteensa on väärin, eli ennen kuin mitään tietoa lähetetään palvelimelle.

Kuten palvelinpuolen kielillä, esimerkiksi PHP ja ASP, myös JavaScriptiä voi liittää suoraan HTML-sivulle. Tästä koodista näkyy vain sen prosessoima tulos, ja itse koodi on web-sivun lähdekoodissa. (8.)

JavaScript-kehitys on muuttunut huomattavasti sen alkuajoista. Silloin toimivan verkkosivun sai muutamalla rivillä HTML-koodia.

Esimerkkikoodista 1 nähdään, kuinka kaikki koodi tuodaan yhteen tiedostoon. Body-tunnisteiden sisällä on sivun näkyvä osa, ja index.js:ssä määritellään toiminnallisuus, esimerkiksi napin painallukset.

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>A surprise, to be sure, but a welcome one.</title>
  <script src="index.js"></script>
</head>
<body>
  <h1>Hello there.</h1>
  <h1>General Kenobi!</h1>
</body>
</html>
```

Esimerkkikoodi 1. Yksinkertainen HTML-sivu.

Tälle sivulle voisi tuoda myös kolmannen osapuolen kirjaston, esimerkiksi moment.js:n jossa on aikaan ja ajanhallintaan liittyviä funktioita. Kyseisen kirjaston liittäminen sivulle vaatisi sen ohjelmakoodin lataamista verkkosivun tiedostojen joukkoon, minkä jälkeen se lisättäisiin uudeksi script-tunnisteeksi HTML-koodin alkuun.

Ratkaisu on ihan toimiva niin kauan kunnes kirjasto saa päivityksen tai kirjaston koodista löytyy virhe. Päivitys tai korjaus toteutetaan siten, että sivun kehittäjä lataisi kirjaston päivitetyn ohjelmakoodin uudestaan sivun hakemistoon, olettaen että kirjasto edes päivitetään. (9.)

JavaScriptiä käytetään pääasiassa käyttöliittymien tekemiseen. Sovellusten taustajärjestelmien kehittämiseen voidaan käyttää esimerkiksi Node.js runtimeympäristöä, joka suorittaa JavaScript-koodia palvelimella. AIRFENCE-sovelluksen taustajärjestelmä käyttää juuri Node.js:ää tuomaan yhteen AIRFENCE-sensorien tuoman datan käyttöliittymään. (10.)

3.3 React-sovelluskirjasto

React on Facebookin kehittämä JavaScript-kirjasto, joka toimii aloituskohtana web-sovelluksen käyttöliittymän rakentamiselle. Reactin vahvuuksia ovat muun muassa helposti ymmärrettävä syntaksi (JSX), virtuaalinen DOM ja hyvin skaalautuva komponentteihin perustuva kehitysmalli. (11.)

DOM ja VDOM

DOM (Document Object Model) on HTML-koodia. HTML-elementeistä tulee niin sanottuja ”solmuja” DOM-rajapinnassa. Toisin sanoen, HTML on tekstiä ja DOM on muistiversio siitä.

HTML DOM tarjoaa kehittäjien käyttöön rajapinnan, jolla voi muokata DOM-solmuja. Se sisältää metodeja, kuten getElementById ja removeChild. Niillä kehittäjä pystyy muokkaamaan web-sivua haluamakseen, kuten esimerkkikoodissa 2 näytetään.

```
var item = document.getElementById("myLI");
item.parentNode.removeChild(item);
```

Esimerkkikoodi 2. DOM-rajapinnan muokkaamista koodilla. Document tarkoittaa DOM-juurisolmua, ja getElementById, parentNode ja removeChild ovat rajapinnan metodeja.

HTML DOM on aina rakenteeltaan puun kaltainen, eli aina on yksi juuri ja siitä lähtee niin sanottuja lapsisolmuja. Entisaikaan web-sivut olivat paljon yksinkertaisempia nykyisiin SPA-sovelluksiin (Single Page Application) verrattuna. Nykyaikaisessa SPA-sovelluksessa voi olla jopa tuhansia div-kontteja jakamassa sivuja eri osiin. Jokaisen tällaisen kontin sisällä voi olla nappeja, tekstikenttiä, valintaruutuja ynnä muuta, ja niiden tapahtumat pitää käsitellä yksi kerrallaan ja päivittää kontti tarvittaessa. Tämä ei ole pelkästään vaikeaa kehittää, se on myös epätehokasta. (12.)

VDOM (Virtual Document Object Model) on Reactin käyttämä DOM-rajapinnan ohjelmointikäsite. VDOM on ideaalinen versio tavallisesta DOM:sta, joka kertoo sovellukselle mitä DOM:n osia täytyy päivittää. Tästä kokonaisuudesta syntyy huomattavasti käyttäjäystävällisempi kokemus, koska web-sivut pystyvät päivittymään todella nopeasti. (13; 11.)

Komponentit ja props

Reactin perusta ovat komponentit. React-pohjainen web-sovellus tavallisesti koostuu näistä pienistä palasista. Komponenteilla on oma logiikkansa, ja ne päättävät HTML-koodin esityksestä itsenäisesti. Komponenteille annetaan yleensä dataa yhden objektin muodossa, jota kutsutaan nimellä props. Tämän objektin avulla komponentti näyttää käyttäjälle tietoa tai suorittaa halutun prosessin, esimerkiksi kutsun tietokantaan. Props tulee aina komponentin ulkopuolelta, eikä sitä pysty muokkaamaan.

```
render() {
return <h1>Hello, {this.props.name}</h1>;
      }
```

Esimerkkikoodi 3. Reactin render-funktio, jossa käytetään props-objektia.

Esimerkkikoodissa 3 on Reactin render-funktio, joka palauttaa tekstin ”Hello, ” ja käyttäjän nimen, joka on määritelty muualla. This.props.name voi tässä esimerkissä vaihtua dynaamisesti, esimerkiksi käyttäjätilin vaihtuessa web-palvelussa. (11.)

State ja lifecycle-konseptit

State tarkoittaa melkein samaa asiaa kuin props, mutta state on rajoitettu vain komponentteihin, eikä sitä voi lukea komponentin ulkopuolelta. State on objekti, jonka tarkoitus on seurata komponentin sisällä tapahtuvia prosesseja.

Esimerkkikoodi 4 on hyvä kuvaus sille, mitä varten state ja lifecycle on tehty. Tässä tapauksessa state seuraa kellonaikaa ja render-funktio päivittää sivua käyttäjälle.

Lifecycle tarkoittaa komponentin ”elämänkaarta” eli sitä, missä tilassa komponentti milloinkin on. Lifecycle-metodien käyttö on hyödyllistä lähes kaikissa React-komponenteissa. Lifecycle-metodeihin kuuluvat muun muassa componentDidMount() ja componentWillUnmount(). Näiden metodien sisällä oleva koodi suoritetaan riippuen komponentin tilasta. (11.)

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState({
      date: new Date()
    });
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

ReactDOM.render(
  <Clock />,
  document.getElementById('root')
);
```

Esimerkkikoodi 4. React-komponentti, jonka state-objekti seuraa kellonaikaa.

JSX-liitännäinen

JSX on syntaksiliitännäinen JavaScriptille. JSX mahdollistaa sen, että kaikki käyttöliittymän logiikka on luonnollisesti yhtenäistä: miten eri tapahtumat hoidetaan, miten state vaihtuu ja miten dataa muokataan näytettäväksi. (11.)

JSX:llä kehittäjät voivat tehdä DOM-rajapintaa muistuttavia puurakenteita samassa tiedostossa missä JavaScript sijaitsee. Tämä rakenne kootaan JavaScript-koodiksi.

Esimerkkikoodin 5 sisältö käännetään suorituksen aikana esimerkkikoodin 6 mukaiseksi koodiksi.

```
var nav = (
    <ul id="nav">
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Clients</a></li>
      <li><a href="#">Contact Us</a></li>
    </ul>
);
```

Esimerkkikoodi 5. JSX-koodia, joka palauttaa navigointipalkin listan.

```
var nav = React.createElement(
   "ul",
   { id: "nav" },
   React.createElement(
      "li",
      null,
      React.createElement(
         "a",
         { href: "#" },
         "Home"
      )
   ),
   React.createElement(
      "li",
      null,
      React.createElement(
         "a",
         { href: "#" },
         "About"
      )
   ),
   React.createElement(
      "li",
      null,
      React.createElement(
         "a",
         { href: "#" },
         "Clients"
      )
   ),
   React.createElement(
      "li",
      null,
      React.createElement(
         "a",
         { href: "#" },
         "Contact Us"
      )
   )
);
```

Esimerkkikoodi 6. JSX-koodi, joka on muutettu JavaScriptiksi.

Esimerkkikoodeista 5 ja 6 nähdään, että JSX:ää voisi ajatella React.createElementin lyhentäjänä. JSX:n avulla JavaScriptistä ja HTML:stä saadaan helposti lähestyttävää ja kehitettävää koodia. JSX ei ole pakollinen React-kehityksessä, joskin sitä käytetään melkein aina. (14.)

4 React Native -sovellus AIRFENCE-tuotteelle

React Nativen kehitys on melkein sama kuin Reactin, mutta se käyttää nimensä mukaisesti natiivikomponentteja web-komponenttien sijaan. React Native -sovellus ei ole ”mobiili web-sovellus” tai ”hybridisovellus”, vaan täysin erottamaton sovelluksesta, joka on kirjoitettu Javalla tai Object-C:llä. React Native käyttää samoja käyttöliittymäelementtejä kuin tavalliset iOS- tai Android-sovellukset. Juuri tästä syystä Sensofusion Oy päätti toteuttaa mobiilikäyttöliittymän React Nativella, jotta kehittäjien ei tarvitse käyttää aikaa kahden erillisen sovelluksen kehittämiseen eri kielillä.

Opinnäytetyön osana kehitin uuden mobiilikäyttöliittymän AIRFENCE-tuotteelle. Käyttöliittymässä oli aluksi tarkoitus näkyä vain AIRFENCE-sensorit kartalla, sensorien UAV-laitehavainnot ja tietoa molemmista.

Sovelluksen toteutuksessa oli tarkoitus käyttää ainakin yhtä tiettyä kolmannen osapuolen kirjastoa, react-native-mapbox-gl. Se on React Native -toteutus alkuperäisessä AIRFENCE-käyttöliittymässä käytetystä Mapbox GL -karttakirjastosta. Ilman Mapboxia uuden käyttöliittymän toteutus olisi ollut hankalaa. Kaikki muut sovelluksessa käytetyt kirjastot olivat itse valitsemiani ja hyväksi todettuja.

Sovelluksella oli jo valmis taustajärjestelmä, ja insinöörityössä keskityttiin vain käyttöliittymään.

Kehitykseen käytin MacBook Pro-tietokonetta mac OS High Sierran versiolla 10.13.6. React Nativen kehittämiseen käytin mac OS-terminaalia ja Visual Studio Code-tekstieditoria.

4.1 Projektin aloitus

React Native voidaan asentaa tietokoneelle terminaalin kautta käyttämällä mac OS-koneiden paketinhallintaohjelma Homebrew'ta seuraavilla komennoilla:

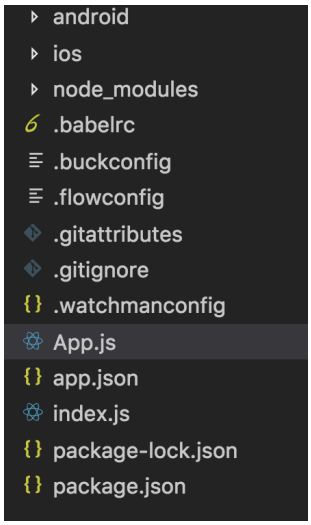
```
brew install node
brew install watchman
npm install -g react-native-cli
```

Node.js on asynkroninen eli muista prosesseista irrallinen JavaScript runtime -ympäristö, joka on rakennettu Google Chromen V8 JavaScript-moottorilla. Node suorittaa JavaScript-koodia selaimen ulkopuolella. Lisäksi käytin Noden paketinhallintajärjestelmä npm:ää (node packet manager).

React Native tarvitsee Nodea, jotta se pystyy suorittamaan projektin JavaScriptiä, ja watchman vahtii muutoksia koodissa. Kun Node on asennettu, voidaan käyttää npm:ää asentamaan React Native CLI globaalisti koko tietokoneelle parametrillä -g. Seuraavaksi voidaan luoda uusi projekti ja suorittaa se ensimmäisen kerran.

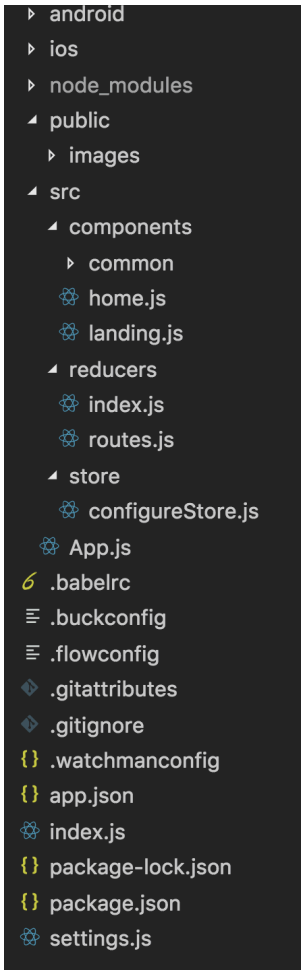
```
react-native init <projektin nimi>
```

React Nativen komento init luo sovelluksen hakemiston ja lisää sinne tarvittavat tiedostot ja alikansiot, kuten kuvassa 2 on esitetty.



Kuva 2. React Native -sovelluksen alkutaso react-native init -komennon jälkeen.

Kuvan 2 näyttämällä tavalla kaikki sovelluksen ohjelmakoodi sijaitsee juurikansiossa, joten ensi töiksi on määriteltävä sovelluksen rakenne selkeyttämään kehitystyötä nyt ja tulevaisuudessa, kun sovellusta on mahdollisesti kehittämässä useampi henkilö. Kuvassa 3 on esitetty, miltä sovelluksen lopullinen rakenne näyttää. Rakenteen määrittely vaihtelee kehittäjästä toiseen, mutta pääasiassa JavaScript-kehittäjänä tein sovelluksen rakenteen web-kehityksestä tuttuun tapaan.



Kuva 3. Sovelluksen lopullinen tiedostorakenne.

Kun sovellus on luotu, se voidaan ajaa ensimmäisen kerran komennolla ”react-native run-ios” tai ”react-native run-android”. Näillä komennoilla React Native kääntää projektin JavaScript-koodin Objective-C:ksi iOS:lle tai Javaksi Androidille. Jotta opinnäytetyön kehitys ei olisi vienyt liikaa aikaa, käytettiin vain run-ios-komentoa. Molempien komentojen käyttö olisi vaatinut Android SDK:n (Software Development Kit) asentamista ja jokaisen

koodimuutoksen testausta sekä Androidilla että iOS:llä. Komento suorittaa JavaScript-tiedostot paikallisesti ja käynnistää emulaattorin. Emulaattorista yleensä kytketään päälle ainakin live reload, jotta kehitystyö on nopeampaa. Tekstieditorin koodi pitää sen jälkeen vain tallentaa, ja emulaattori suorittaa muutokset välittömästi näkyville. (15.)

4.2 Navigointi ja tilanhallinta sovelluksessa

AIRFENCE-sovelluksessa navigointi ei ole niin suuressa roolissa kuin muissa sovelluksissa. AIRFENCE-sovelluksen idea on tarjota käyttäjälle helppoa, nopeaa ja kattavaa tietoa UAV-havainnoista. Näiden syiden vuoksi sovelluksessa on vain kaksi näkymää, sisäänkirjautumis- ja karttanäkymä. Helppokäyttöisyyttä auttaa myös se, että sovellus tietää, jos käyttäjä on aikaisemmin kirjautunut sisään, eli käytännössä sisäänkirjautumisen REST-kutsun (Representational state transfer) vastauksena tullut merkkijono tallennetaan sovelluksen muistiin. Näin käyttäjän ei tarvitse joka kerta sovellusta avatessaan kirjoittaa sähköpostiosoitetta ja salasanaa.

Vaikka navigaatio ja tilanhallinta eivät ole suuressa roolissa, hyvään sovelluskehitykseen kuuluu valmistautuminen tulevaan, joten navigointia ja tilanhallintaa varten käytetään react-navigation, react-redux, ja react-native-router-flux npm-kirjastoja.

Esimerkkikoodista 7 nähdään, että routes.js määrittelee navigointitoiminnot, jotka se saa react-native-router-flux-kirjastolta. Reducer/index.js yhdistää edellä mainitun routes.js-tiedoston muiden redux-tiedostojen kanssa, jos niitä olisi. Tähän mennessä sovelluksessa reduxille ei ole löytynyt oikeaa käyttöä, sillä tiloja muutetaan tällä hetkellä vain yhden komponentin sisällä hyvin vähän. Viimeiseksi configureStore.js määrittelee redux loggerin näyttämään tiloja ja toimintoja storen sisällä.

```
// store/configureStore.js

import { createStore, applyMiddleware, compose } from 'redux';
import reducers from '../reducers/index';
import { createLogger } from 'redux-logger';

const loggerMiddleware = createLogger({ predicate: (getState, action) =>
__DEV__ })

export default function configureStore () {
  const enhancer = compose(
    applyMiddleware(
      loggerMiddleware
    )
  )
```

```
  const store = createStore(reducers, enhancer)

  if (module.hot) {
    module.hot.accept(() => {
      const nextRootReducer = require('../reducers/index').default
      store.replaceReducer(nextRootReducer)
    })
  }
  return store
}

// reducers/routes.js

import { ActionConst } from 'react-native-router-flux';

const initialState = {
  scene: {},
};

export default function reducer(state = initialState, action = {}) {
  switch (action.type) {
    case ActionConst.FOCUS:
    console.log(action);
      return {
        ...state,
        scene: action.scene,
      };


    default:
      return state;
  }
}

// reducers/index.js

import { combineReducers } from 'redux';
import routes from './routes';
// ... other reducers

export default combineReducers({
  routes,
  // ... other reducers
});
```

Esimerkkikoodi 7. Navigoinnin ja reduxin määrittely React Native -projektissa (15).

Navigointi voidaan seuraavaksi toteuttaa App.js-tiedostoon, kun RouterWithRedux on määritelty. Sovelluksen käynnistyksessä juurikansiossa sijaitseva index.js on sovelluksen alkukohta. Se on määritelty ohjaamaan käyttäjä App.js-tiedostoon, joka puolestaan näyttää käyttäjälle joko karttakomponentin (home) tai sisäänkirjautumisikkunan (landing) esimerkkikoodin 8 mukaisesti.

```
      <Provider store={store}>
        <RouterWithRedux>
          <Scene key="root" navigationBarStyle={styles.navigationBarStyle}>
            <Scene key="landing" component={Landing} title="AIRFENCE" ini-
tial={true} />
            <Scene
              key="rootTabBar"
              tabs={true}
              tabBarStyle={{display: 'none'}}>
              <Scene key="home" component={Home} title="AIRFENCE" icon={TabI-
con} initial />

            </Scene>
          </Scene>
        </RouterWithRedux>
      </Provider>
```

Esimerkkikoodi 8. Sovelluksen App.js-tiedoston render-metodi. RouterWithRedux-elementtiin määritellään näkymät eli ”scenet”.

Tulevaisuudessa jatkokehityksessä saattaa tulla tilanteita, jolloin lisänäkymiä tarvitaan, ja ne on helppo lisätä sovellukseen tämän prosessin jälkeen. (16.)

4.3 Sisäänkirjautuminen sovellukseen

Sovelluksen todennus, eli tutummin sisäänkirjautuminen, toteutetaan Rest-kutsulla taustajärjestelmään. Tämä tapahtuu superagent-nimisellä npm-kirjastolla. Se asennetaan komennolla ’npm install superagent’, minkä jälkeen se lisätään landing.js-tiedoston alkuun esimerkkikoodin 9 näyttämällä tavalla.

```
const request = require('superagent');
```

Esimerkkikoodi 9. Vakiona määritelty request, joka hakee superagent-koodin node-moduuleista.

Todennukseen käytetään Rest-kutsua, joka ottaa vastaan käyttäjän sähköpostiosoitteen ja salasanan. Mikäli edellä mainittu yhdistelmä on oikein, palvelin palauttaa todennusavaimen takaisin sovellukseen tallennettavaksi. Todennusavaimella sovellus pystyy avaamaan WebSocket-yhteyden palvelimelle. Rest-kutsun jälkeen tietojen tallentamiseen käytetään storeInfo-funktiota esimerkkikoodien 10 ja 11 tavalla, jossa asynkronisesti tallennetaan kaikki tarvittava tieto palvelimelta.

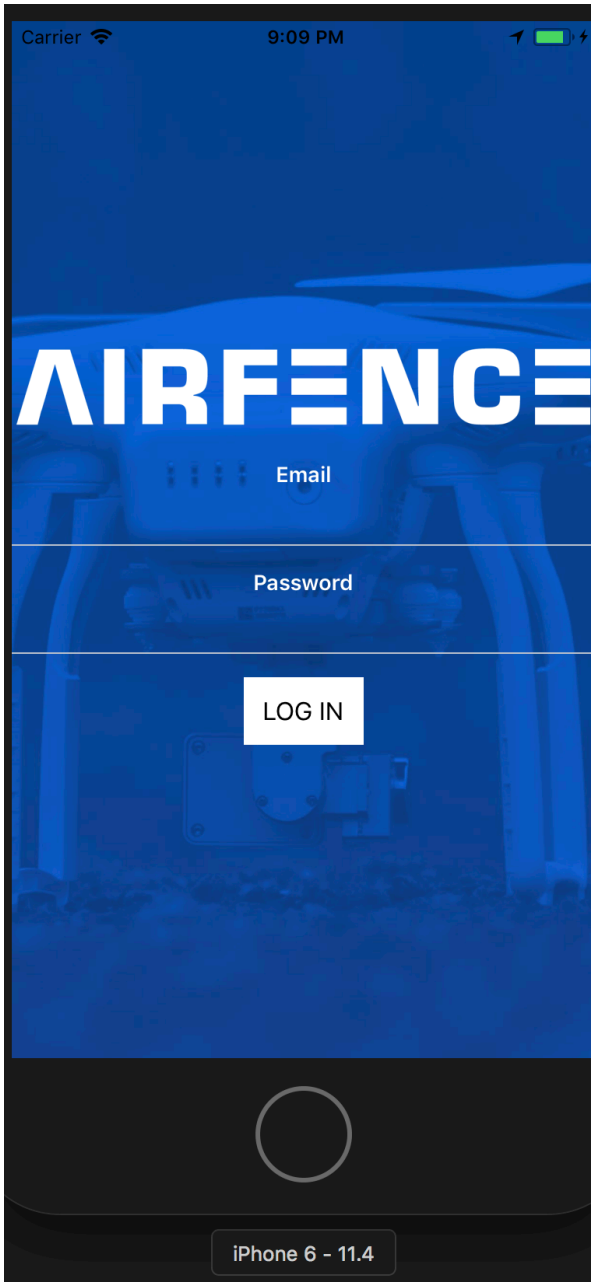
```
request.post('https://testurl.com/auth')
  .set('Content-Type', 'application/json')
  .set('Accept', 'application/json')
  .send({
    "email": email,
    "password": password,
  })
  .end((err, res) => {
    if (err || !res.ok) {
      this.setState({
        error: res,
        busy: false,
      });
    } else {
      this.storeInfo('airfence_token_id', res.body.token);
      Actions.home();
    }
  });
```

Esimerkkikoodi 10. AIRFENCE-sovelluksen todennusfunktion sisältö. Palvelimelle lähetetään sähköpostiosoite ja salasana, ja vastauksessa saadaan merkkijono (token) sekä tietoa käyttäjätilistä. Tietojen tallennuksen jälkeen siirrytään karttanäkymään (home).

```
async storeInfo(key, item) {
  try {
      var jsonOfItem = await AsyncStorage.setItem(key, item);
      return jsonOfItem;
  } catch (error) {
    console.log(error.message);
  }
}
```

Esimerkkikoodi 11. Asynkroninen funktio storeInfo, jossa kirjautumisen jälkeen tallennetaan palvelimelta saatu tieto.

Kuvassa 4 on viimeisin versio sisäänkirjautumisikkunasta. Käyttäjän painaessa LOG IN -nappia hänet ohjataan karttasivulle, kun sovellus on suorittanut sisäänkirjautumiskutsun esimerkkikoodissa 8 esitellyllä tavalla. Muita näkymiä sovelluksessa ei ole.



Kuva 4. Viimeistelty kirjautumisikkuna. Ensimmäinen asia, jonka käyttäjä näkee sovelluksessa.

4.4 Mapbox GL -karttakirjasto

Koko AIRFENCE-sovelluksen ideana on karttanäkymä, jossa näytetään sekä AIRFENCE-sensorit että niiden havaitsemat UAV-lennokit. Alkuperäisessä käyttöliittymässä käytetty Mapbox GL ei sellaisenaan käy tähän sovellukseen, mutta onneksi Mapboxin kehitystiimi on tehnyt React Nativelle oman version nimeltä react-native-mapbox-gl. Mapboxin asentaminen ei ole aivan yksinkertaista, koska se tietysti käyttää

natiivikomponentteja React Nativella. Vaikka tämän sovelluksen pääkohteena ei ole Android-kehitys, pitää mapbox silti asentaa molemmille alustoille, jotta React Native voi rakentaa JavaScript-tiedostoista toimivan sovelluksen.

Seuraavissa alaluvuissa asennetut natiivikomponentit tulevat käyttöön siinä vaiheessa, kun käyttäjä on kirjautunut sisään ja karttanäkymä piirretään näytölle. Käytännössä JavaScript-koodi suoritetaan kääntäjällä, joka tekee siitä natiivikoodia eli Object-C:tä tai Javaa. Tavallisessa React-web-sovelluksessa tätä kohtaa ei suoritettaisi, joten seuraavien kappaleiden asennukset ovat se osa, jossa mennään alkuperäisestä AIRFENCE-käyttöliittymästä askel pidemmälle.

4.4.1 iOS-asennus

iOS asennukseen käytettiin paketinhallintaohjelma CocoaPodsia joka on yleisesti käytössä Swift ja Objective-C -projekteissa ja tässä tapauksessa myös React Native -projektissa.

Esimerkkikoodissa 12 esitetyllä tavalla lisätään tarvittavat natiivikomponentit iOS-projektiin. Seuraavaksi pitää ajaa komento ’pod install’ terminaalissa AIRFENCE-sovelluksen ios-kansiossa, ja CocoaPods hoitaa natiivikomponenttien asennuksen.

```
# Flexbox Layout Manager Used By React Natve
  pod 'yoga', :path => '../node_modules/react-native/ReactCom-
mon/yoga/Yoga.podspec'

  # React Native
  pod 'React', path: '../node_modules/react-native', subspecs: [
    # Comment out any unneeded subspecs to reduce bundle size.
    'Core',
    'DevSupport',
    'RCTActionSheet',
    'RCTAnimation',
    'RCTBlob',
    'RCTCameraRoll',
    'RCTGeolocation',
    'RCTImage',
    'RCTNetwork',
    'RCTPushNotification',
    'RCTSettings',
    'RCTTest',
    'RCTText',
    'RCTVibration',
    'RCTWebSocket',
    'RCTLinkingIOS'
  ]

  # Mapbox
```

```
  pod 'react-native-mapbox-gl', :path => '../node_modules/@mapbox/react-na-
tive-mapbox-gl'
```

Esimerkkikoodi 12. CocoaPodsin Podfile-konfiguraatio Mapbox GL -liitännäiselle (13).

4.4.2 Android-asennus

Jotta Mapboxin saa toimimaan Androidissa, pitää muokata gradle-asetuksia ja MainApplication.javaa sopivaksi.

```
allprojects {
    repositories {
        jcenter()
        maven { url "$rootDir/../node_modules/react-native/android" }
        maven { url "https://jitpack.io" }
        maven { url "https://maven.google.com" }
    }
}
```

Esimerkkikoodi 13. Project:build.gradle-muutokset Android-projektiin.

```
dependencies {
    compile project(':mapbox-react-native-mapbox-gl')
}
```

Esimerkkikoodi 14. App:build.gradle-muutokset Android-projektiin.

Esimerkkikoodissa 13 määrittellään paketinhallintaohjelma Android-projektille, ja koodissa 14 kerrotaan ohjelmalle, että projektiin halutaan mapbox-gl-liitännäinen.

Viimeinen asia, jonka build.gradle vaatii, on Androidin vähimmäisversion muuttaminen versioon 26, kuten esimerkkikoodissa 15 tehdään.

```
android {
    compileSdkVersion 26
    buildToolsVersion "26.0.1"

    defaultConfig {
        applicationId "com.airfencenative"
        minSdkVersion 16
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        ndk {
            abiFilters "armeabi-v7a", "x86"
        }
    }
```

Esimerkkikoodi 15. Build.gradle-asetus, jolla määritellään, mikä Androidin version tulee vähintään olla, jotta sovelluksen voi suorittaa.

React Native todennäköisesti ei tarjoa tukea Androidin vanhemmille versioille, joten sovelluskirjaston kehitystiimin on vedettävä raja versioon 26.

Viimeiseksi lisätään Mapbox GL:n vaatima natiivikomponentti Androidin MainApplication.java-tiedostoon esimerkkikoodin 16 mukaisesti.

```
import com.mapbox.rctmgl.RCTMGLPackage;

import java.util.Arrays;
import java.util.List;

public class MainApplication extends Application implements ReactApplication {

  private final ReactNativeHost mReactNativeHost = new ReactNativeHost(this) {
    @Override
    public boolean getUseDeveloperSupport() {
      return BuildConfig.DEBUG;
    }
```

Esimerkkikoodi 16. Mapbox GL -liitännäisen vaatima natiivikomponentti RCTMGLPackage MainApplication.java -tiedostossa.

4.4.3 JavaScript-asennus

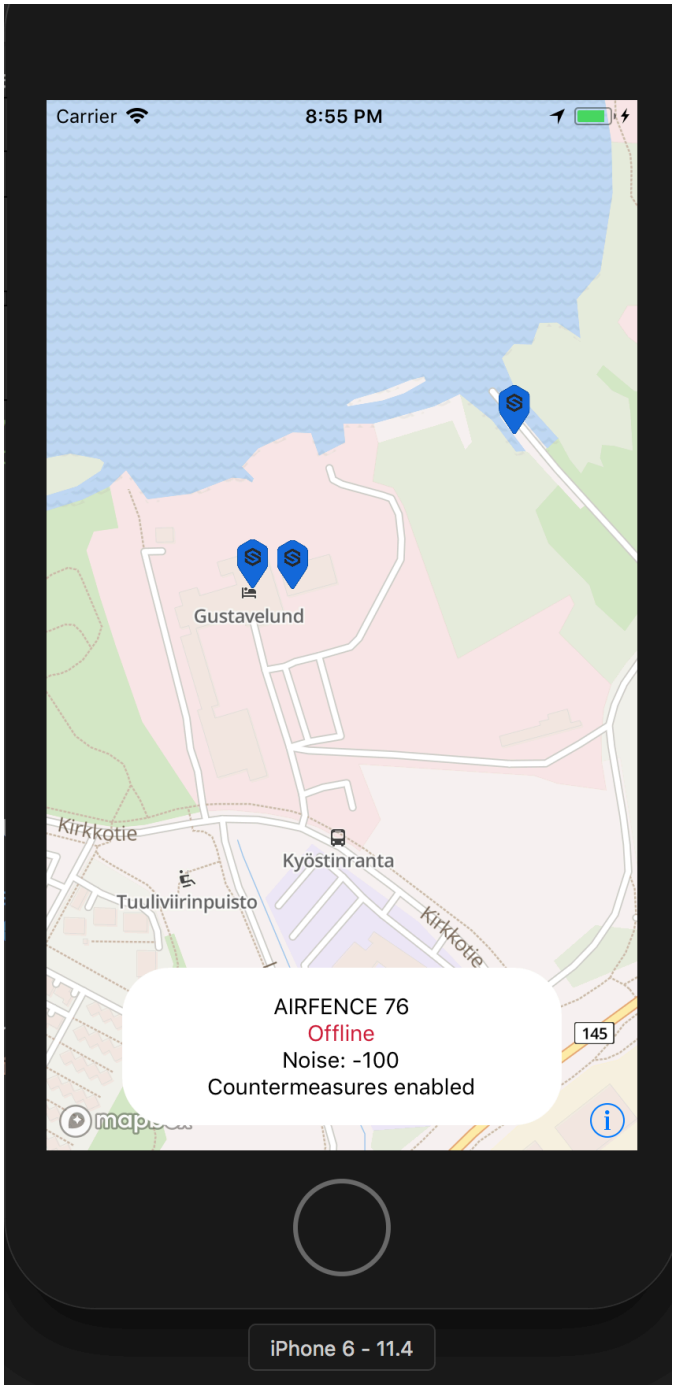
Kun kaikki natiivikomponentit on asennettu, voidaan lisätä Mapbox itse React Native -sovellukseen terminaalissa komennolla ’npm install –save react-native-mapbox-gl’. –save-parametri kertoo npm:lle, että liitännäinen pitää tallentaa package.jsoniin node_modules-kansion lisäksi.

Kartta on määritelty home.js-tiedostossa esimerkkikoodissa 17 esitellyllä tavalla. Kartta käyttää Sensofusion Oy:n lisensoiman kartan JSON-tiedostoa, joka haetaan tiedoston import-osiossa. Asetukset zoomLevel ja centerCoordinate, eli kartan aloituspaikka ja zoomaustaso, on määritelty sovelluksen juurikansion settings.js-tiedostossa. Niiden lisäksi karttaan on lisätty MapboxGL.ShapeSource- ja SymbolLayer-elementit näyttämään AIRFENCE-sensorit ja niiden havaitsemat UAV-laitteet. Näitä elementtejä klikkaamalla avautuu pieni ikkuna, joka esittää tietoa valitusta laitteesta tai dronesta, esimerkiksi UAV-laitteen malli, milloin se on havaittu ja mikä laite sen havaitsi. (17.)

```
<View style={{flex: 1}}>
        <MapboxGL.MapView
        onPress={this.onMapPress}
          userTrackingMode={3}
          showUserLocation={true}
          zoomLevel={settings.default_zoom}
          styleURL={style}
          style={{flex: 1}}
          centerCoordinate={[settings.default_lon, settings.default_lat]}>

          <MapboxGL.ShapeSource id="marker-airfence" onPress={this.onSource-
LayerPress} shape={this.airfences}>
          <MapboxGL.SymbolLayer id="marker-style-airfence" style={{ iconImage:
airfenceMarker, iconSize: 0.1 }} />
          </MapboxGL.ShapeSource>

          <MapboxGL.ShapeSource id="marker-drone" onPress={this.onSourceLayer-
Press} shape={this.drones}>
          <MapboxGL.SymbolLayer id="marker-style-drone" style={{ iconImage:
droneMarker, iconSize: 0.1 }} />
          </MapboxGL.ShapeSource>

          </MapboxGL.MapView>
          {this.renderBubble()}
          </View>
```

Esimerkkikoodi 17. Src/components/home.js render-funktio.

Kuvasta 5 voidaan havaita järven rannalle asennetut AIRFENCE-laitteistot. Tiettyä laitetta painamalla avautuu tietoa laitteesta, tässä tapauksessa laitteen numero, tieto onko se päällä vai ei, radiotaajuusmelun määrä sekä pystyykö sillä tekemään vastatoimia UAV-laitteille. Vastatoimia ei ole vielä kehitetty tähän sovellukseen aikarajoitusten vuoksi.



Kuva 5. Kuvakaappaus käyttöliittymästä. AIRFENCE-laitteistoa järven rannalla. (Xcode emulaattori.)

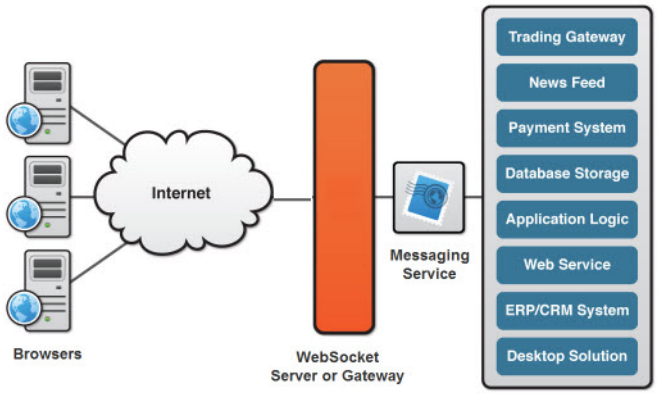
4.5 Websocket-yhteys

HTML5 Websocketilla web-sivut pystyvät käymään jatkuvaa molemminpuolista kommunikointia etäpalvelimen kanssa. Websocket-yhteys luo huomattavasti vähemmän liikennettä ja viivettä verkossa verrattuna perinteisiin verkkoratkaisuihin, joiden pitää kysyä palvelimelta tietoa.

Websocket tukee myös yhteyksiä palomuurien tai välityspalvelimien läpi vaivatta. Käytännössä välityspalvelimelle lähetetään HTTP CONNECT -viesti, joka pyytää välityspalvelinta avaamaan TCP/IP-yhteyden tiettyyn isäntäpalvelimeen ja porttiin.

Koska Websocket rasittaa palvelimia vähemmän, se myös tukee enemmän yhtäaikaisia yhteyksiä, toisin sanottuna suurempi määrä käyttäjiä pystyy olemaan yhteydessä websivulle samaan aikaan. (18.)

AIRFENCE-palvelin toimii siten, että yksittäiset AIRFENCE-sensorit ovat yhteydessä palvelimeen, joka tarjoilee dataa WebSocketilla käyttöliittymään, joka ei eroa paljoa kuvasta 6. WebSocketin lisäksi käytössä on yksinkertainen REST-rajapinta, jolla pystytään aktivoimaan esimerkiksi vastatoimet UAV-laitteille.



Kuva 6. Websocket-arkkitehtuuri selitettynä. Selaimilla (browsers) on käytössä suora websocket-yhteys erilaisiin web-palveluihin.

WebSocket-yhteys avataan try-catch-lausekkeen sisällä esimerkkikoodin 18 mukaisesti. Sille annetaan merkkijono (token) joka tallennettiin sisäänkirjautumisen yhteydessä ja lisätään se URL-vakioon (Uniform Resource Locator). URL:n avulla WebSocket voidaan avata, ja sen omilla funktioilla määritellään, mitä tapahtuu, kun esimerkiksi viesti saapuu palvelimelta. Esimerkiksi this.websocket.onmessagen tapahtuessa tallennetaan saapuvat sensori- ja UAV-laitetiedot.

```
initWebSocket = (org, token) => {

    try {
      const url = settings.db_ws + '/ws/api/geojson?token=' + token; //'&in-
terval=' + this.interval+orgString;

      this.websocket = new WebSocket(url);
      this.websocket.onopen = (e) => { this.onOpenWS(e); };
      this.websocket.onclose = (e) => { this.onCloseWS(e); };
      this.websocket.onmessage = (e) => { this.onMessageWS(e); };
      this.websocket.onerror = (e) => { this.onCloseWS(e); };
    } catch (exception) {
        console.log(exception);
      this.setState({
        error: 'WebSockets are not supported. Please upgrade your browser.'
      });
    }
  }
```

Esimerkkikoodi 18. WebSocket-yhteyden avaaminen sovelluksessa.

4.6 Tyyli

Tyylien käyttäminen ei ole sovelluksen kehityslistalla kärjessä, sillä koko sovelluksen pääidea on näyttää karttasivulla tietoa. Sisäänkirjautumissivu on silti lähes samankaltainen minkä tahansa web-sivun kanssa, joten siihen päätettiin käyttää react-native-elements-kirjastoa kuvan 7 näyttämällä tavalla. FormInput on tekstikenttä, johon syötetään tietoa, ja FormLabel sen otsikko. Sovelluksessa on myös käytetty FormValidationMessage-komponenttia ilmoittamaan, jos esimerkiksi salasana tai käyttäjätunnus on väärin.

```
import { FormLabel, FormInput, FormValidationMessage } from 'react-native-ele-
ments'

<FormLabel>Name</FormLabel>
<FormInput onChangeText={someFunction}/>
<FormValidationMessage>Error message</FormValidationMessage>
```

Kuva 7. React Native Elementsin määrittely ja sen komponentteja.

4.7 Sovelluksen lopputulos

Sovelluksen kehitys sujui hyvin. Aiempaa JavaScript- ja React-kokemusta omaavana en odottanut suuria ongelmia, eikä niitä todellisuudessa koskaan ilmennyt. Sovellus täyttää kaikki ne vaatimukset, jotka sille asetettiin alussa, eli sovellus pystyy antamaan käyttäjälle tietoa ilmassa lentävistä UAV-laitteista, joihin hän mahdollisesti haluaa reagoida.

Sovelluksessa käytettiin hyväksi niitä Reactin vahvuuksia, joita työn alussa tutkittiin.

Aikarajoitusten vuoksi sovellusta ei tämän opinnäytetyön aikana julkaistu Applen App Storessa eikä Android Play -kaupassa, ja sitä suoritettiin vain Xcode-simulaattorilla.

Tulevaisuuden suunnitelmissa sovellukselle on tiedossa usean kehittäjän tekemää jatkokehitystä. Sovellukseen olisi hyvä muun muassa saada käyttäjän rekisteröintimahdollisuus ja UAV-vastatoimet.

Jatkokehityksessä on hyvä pitää mielessä, että sovelluksesta ei saa tulla mobiilikäyttäjälle hankalaa käyttää. Sovelluksen on tarkoitus pysyä niin sanotusti ”lite”-versiona AIR-FENCE-käyttöliittymästä, ja alkuperäisessä React-pohjaisessa web-palvelussa on kaikki mahdolliset ominaisuudet.

5 Yhteenveto

React Native on varteenotettava kehitysalusta kenelle tahansa, joka haluaa päästä natiivisovellusten piiriin pelkällä JavaScript-kokemuksella. Yhden kehittäjän tekemällä työllä voidaan sovellus julkaista sekä Android Play -kaupassa että Applen App Storessa. Vaihtoehtona olisi ollut Java- ja Swift-kehittäjien palkkaaminen tekemään kahta erillistä sovellusta puhtaasti natiivikoodilla.

Insinöörityössä tutustuttiin React Nativen vahvuuksiin ja toteutettiin sovellus tämän tiedon pohjalta. Tuloksena on käyttövalmis AIRFENCE-sovellus Androidille tai iOS-käyttöjärjestelmälle. Tarkoitus oli saada sovellus lähes kaikille puhelinkäyttöjärjestelmille yhden koodikannan avulla. Työlle asetetut tavoitteet saavutettiin, ja sovellus on valmis julkaistavaksi.

Lähteet

1 Commercial Drones Are Revolutionizing Business Operations. Verkkoaineisto. Toptal. https://www.toptal.com/finance/market-research-analysts/drone-market. Luettu 15.4.2018.

2 Käyttötarkoitus määrittää lennokin ja miehittämättömän ilma-aluksen eron. Verkkoaineisto. Trafi. https://www.trafi.fi/trafi/ajankohtaista/2903/kayttotarkoitus_maarittaa_le. Luettu 12.5.2018.

3 SECOPS. Verkkoaineisto. Community Research and Development Information Service. https://cordis.europa.eu/project/rcn/210103_en.html. Luettu 12.5.2018.

4 Sensofusion. Verkkosivu. https://sensofusion.com. Luettu 15.4.2018

5 Sensofusion launches AIRFENCE 5.0. Verkkoaineisto. Shephard media. https://www.shephardmedia.com/news/uv-online/sensofusion-launches-airfence-50-counter-uas-techn/. Luettu 15.4.2018.

6 Model-view-controller (MVC) -arkkitehtuuri. Verkkoaineisto. Jyväskylän yliopiston informaatioteknologian tiedekunta. http://appro.mit.jyu.fi/tiea2080/luennot/mvc/. Luettu 12.5.2018.

7 Thinking in Redux (when all you've known is MVC). Verkkoaineisto. Hackernoon. https://hackernoon.com/thinking-in-redux-when-all-youve-known-is-mvc-c78a74d35133. Luettu 23.8.2018.

8 JavaScript definition. Verkkoaineisto. TechTerms. https://techterms.com/definition/javascript. Luettu 11.8.2018.

9 Modern JavaScript Explained For Dinosaurs. Verkkoaineisto. Jang, Peter. https://medium.com/the-node-js-collection/modern-javascript-explained-for-dinosaurs-f695e9747b70. Luettu 12.06.2018.

10 About Node.js. Verkkoaineisto. NodeJS. https://nodejs.org/en/about/. Luettu 23.8.2018.

11 React.js documentation. Verkkoaineisto. ReactJS. https://reactjs.org/docs. Luettu 11.8.2018.

12 What is the DOM? Verkkoaineisto. Mozilla. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. Luettu 12.8.2018.

13 The difference between Virtual DOM and DOM. Verkkoaineisto. React Kungfu. http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/. Luettu 16.8.2018.

14 What is JSX? Verkkoaineisto. React Enlightenment. https://www.reactenlightenment.com/react-jsx/5.1.html. Luettu 22.8.2018.

15 Getting Started – React Native. Verkkoaineisto. Facebook. https://facebook.github.io/react-native/docs/getting-started.html. Luettu 15.4.2018.

16 React-Native router flux with Redux. Verkkoaineisto. Medium. https://medium.com/gumtreelab/react-native-router-flux-with-redux-9966dd4041b6. Luettu 15.8.2018.

17 React Native Mapbox GL Installation. Verkkoaineisto. Github. https://github.com/mapbox/react-native-mapbox-gl/blob/master/ios/install.md. Luettu 16.8.2018.

18 About HTML5 WebSocket. Verkkoaineisto. WebSocket. https://www.websocket.org/aboutwebsocket.html. Luettu 17.8.2018.