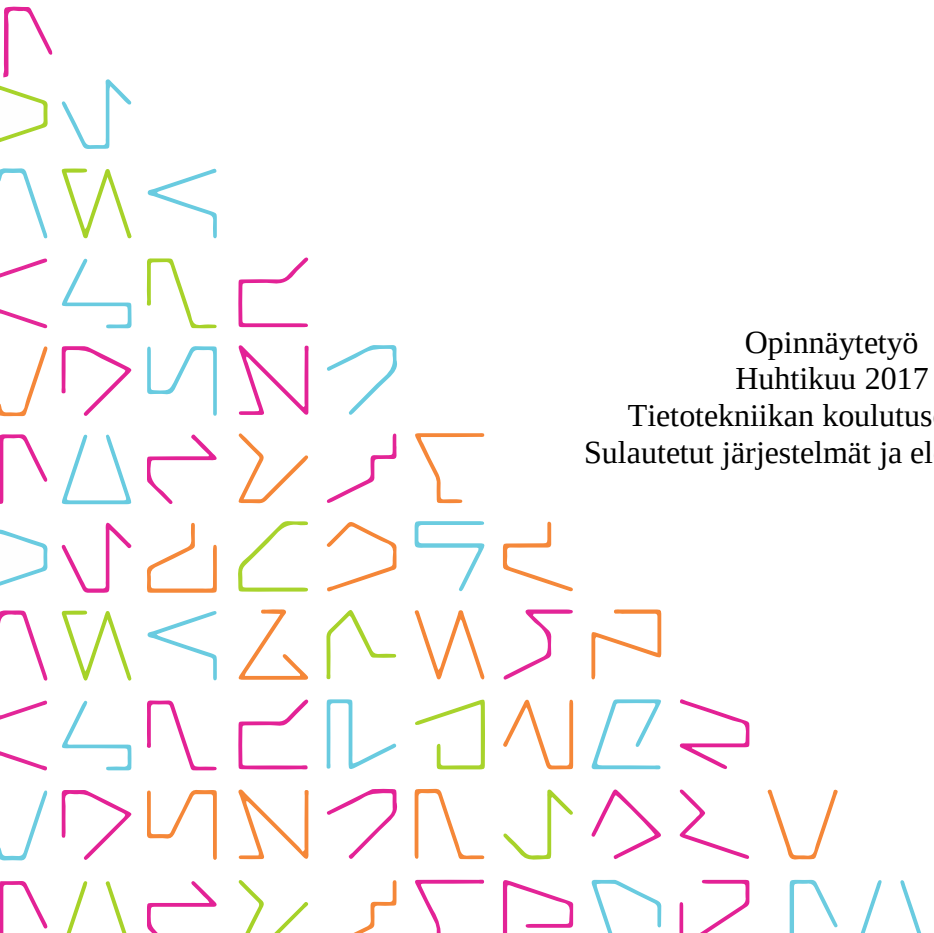
TAMPEREEN
AMMATTIKORKEAKOULU

# MOBIILIAPPLIKAATIO SÄHKÖRULLALAUDAN HALLINTAAN

Samuli Tamminen



Opinnäytetyö
Huhtikuu 2017
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

**TIIVISTELMÄ**

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

TAMMINEN, SAMULI:
Mobiiliapplikaatio sähkörullalaudan hallintaan

Opinnäytetyö 32 sivua
Huhtikuu 2017

---

Tämän opinnäytetyön tarkoituksena oli kehittää älypuhelimilla toimiva prototyyppisovellus, jolla voi kerätä ja visualisoida sähkörullalaudan sensoritietoja sekä säätää laudan asetuksia. Työn toimeksiantajana oli ELMEV-toiminimellä toimiva Simo Sihvonen, joka oli kehittänyt sähkörullalaudan. Sovelluksen tarkoituksena oli tuoda anturitietoja laudan kehittäjän saataville ja siten helpottaa kehitystyötä. Sovelluksella ei ohjata rullalautaa, vaan kerätään ja visualisoidaan erilaisia sensoritietoja.

Sovellus kehitettiin alustariippumattomaksi React Native -sovelluskehystä ja JavaScript-kieltä käyttäen. Rullalaudan moottorinohjainpiirin ja mobiilisovelluksen välisessä kommunikoinnissa käytettiin Bluetooth Low Energyä, joka löytyy useimmista älypuhelimista.

Mobiilisovelluksen ja rullalaudan välistä tiedonsiirtoa varten suunniteltiin viisitavuinen protokolla, jonka kehys koostuu tunnisteesta (mikä tieto on kyseessä), komennosta (luku vai kirjoitus), kahdesta datatavusta ja lopetusmerkistä. Kaikki Bluetooth-liikenne käyttää määriteltyä protokollaa. Protokollan ja sovelluksen testausta varten kehitettiin rullalaudan moottorinohjainpiiriä simuloiva testilaite Arduino-kehitysalustasta ja Bluetooth Low Energy -piiristä.

Sovellus suunniteltiin helppokäyttöiseksi, ja sen ulkoasuun otettiin vaikutteita erilaisista liikuntasovelluksista. Datan kerääminen rullalautailun aikana on helppoa, ja sovellus tallentaa sensoritietojen lisäksi myös kuljetun reitin. Omaa rullalautailusuoritusta ja rullalaudan toimintaa voi tarkastella tallennuksen jälkeen, jolloin sovellus näyttää viivakuvaajat kerätystä datasta. Kerätyn datan voi myös viedä sovelluksesta tietokoneelle taulukkomuodossa myöhempää analyysiä varten.

Lopputuloksena saatiin Android- ja iOS-alustoilla toimiva sovellus, jolla voidaan kerätä sensoridataa rullalaudalta ja sijaintitietoa mobiililaitteelta sekä visualisoida niitä laitteessa ja viedä jatkokäsittelyä varten taulukkomuodossa. React Native osoittautui toimivaksi ratkaisuksi monipuolisempaan sovellukseen, joka käyttää GPS-paikannusta ja Bluetoothia. Suurimmat ongelmat kehityksen aikana olivat sovelluksen taustatoiminta ja tavumuotoisen datan käsittely JavaScript-kielellä. Ongelmat ratkaistiin ja sovelluksen kaikki vaaditut ominaisuudet toteutettiin.

---

Asiasanat: react native, javascript, bluetooth, mobiilisovellus, sähkörullalauta

**ABSTRACT**

Tampere University of Applied Sciences
Degree programme in ICT Engineering
Embedded Systems and Electronics

TAMMINEN, SAMULI:
Mobile Application for Controlling Electric Skateboard

Bachelor's thesis 32 pages
April 2017

---

The purpose of this bachelor's thesis was to develop a prototype application that could collect and visualize sensor data from an electric skateboard and configure the board's settings. The assignment for the thesis was received from Simo Sihvonen, a sole trader with the business name ELMEV, who had developed his own electric skateboard. The purpose of the application was to bring sensor data more available to him in order to make the board's development easier. The application is not meant to control the skateboard, but to collect and visualize its data.

The mobile application was developed to be cross-platform by using JavaScript and the React Native framework. The communication between the board's motor controller and the application was carried out using Bluetooth Low Energy, which is found in most modern smartphones.

A five-byte-long protocol for the communication between the mobile application and the electric skateboard was designed. The protocol's first byte indicates the board data parameter identifier (which information the frame concerns). The second byte is a command byte telling whether we want to read or write the parameter. The third and fourth bytes contain the data and the fifth one is an ending byte. In order to test the protocol and the application, a testing device with Arduino development board and Bluetooth Low Energy chip was built.

The application was designed to be easy to use and the design got inspiration from existing sports tracking applications. Collecting sensor data while skateboarding is easy and the application also saves the route taken using GPS. One can inspect his or her skateboarding exercise and the skateboard's performance after the ride, as the application shows line charts of the collected data. The data can also be exported in a spreadsheet format in order to analyze it further on a computer.

The end result of the thesis was a working application running on both Android and iOS devices. The application can be used to collect sensor data from the electric skateboard and combine it with the location data from the mobile device. All the collected data can be visualized with graphs in the app and exported in a spreadsheet format. React Native proved to be a working solution for building such a versatile application which uses both GPS and Bluetooth. The biggest problems during the development were background mode and binary data handling with JavaScript. The problems were solved and all required features were implemented.

---

Key words: react native, javascript, bluetooth, mobile application, electric skateboard

**SISÄLLYS**

**LYHENTEET JA TERMIT**

| | |
|---|---|
| Arduino Mega 2560 | Arduino-tuoteperheen monipuolinen kehitysalusta, joka käyttää ATmega 2560 -mikrokontrolleria |
| BLE | Bluetooth Low Energy, Bluetooth-standardi, joka nimensä mukaisesti kuluttaa vähemmän energiaa kuin edeltäjänsä |
| GPS | Global Positioning System, maailmanlaajuinen satelliittipaikannusjärjestelmä |
| HTML5 | Hypertext Markup Language revision 5, web-sivujen kuvauskielen uusi määritelmä, joka lisäsi ominaisuuksia etenkin erilaisten mediaelementtien osalta |
| natiivi | alustalle alkuperäinen tai perinteinen tapa toteuttaa asia (tietotekniikassa) |
| sovelluskehys | framework, kokoelma ohjelmistoja ja rajapintoja, joiden pohjalta kehitetään oma sovellus |
| UART | Universal Asynchronous Receiver Transmitter, sarjaliikennepiiri asynkroniseen eli tahdistamattomaan sarjaliikenteeseen |

# 1 JOHDANTO

Tämän opinnäytetyön tarkoituksena oli luoda prototyyppisovellus, jolla voisi lukea sähkörullalaudan antureiden tietoja sekä säätää laudan asetuksia. Työn toimeksiantajana oli ELMEV-toiminimellä toimiva Simo Sihvonen, joka on kehittänyt sähkörullalaudan. Sovelluksen tarkoituksena oli tuoda anturitietoja laudan kehittäjän saataville helpottaen siten kehitystyötä.

Markkinoille on tullut viime vuosina useita sähkörullalautoja valmistavia toimijoita. Rullalaudat ovat valmistajasta ja käyttötarkoituksesta riippuen hieman erilaisia, mutta idea on kaikissa sama. Rullalaudan takatrukkiin (metallinen osa, johon on kiinnitetty pyörät, ja joka mahdollistaa rullalaudalla kääntymisen) on kiinnitetty sähkömoottori, joka pyörittää ketjun tai hihnan avulla rullalaudan pyörää. Moottorina voi olla myös napamoottori, jolloin se on pyörän keskiössä eikä voimansiirtohihnaa tarvita. Moottorin lisäksi laudassa pitää olla moottorinohjausjärjestelmä sekä akku. Näiden lisäksi tarvitaan yleensä kädessä pidettävä erillinen ohjain, jolla voidaan kiihdyttää ja jarruttaa.

Sähkörullalautamarkkinoilla on useita kaupallisia toimijoita, kuten Boosted Boards ja Evolve Skateboards. Molempien lautoihin on saatavilla mobiilisovellus, joka näyttää tietoja esimerkiksi laudan akun tilasta ja ajonopeudesta. (Evolve Skateboards 2015.) Boosted Boardsin sovellusta käytetään myös laudan ohjelmiston (firmware) päivittämiseen ja laudan ajomoodin valintaan. Erilaiset ajomoodit vaikuttavat mm. laudan kiihtyvyyteen ja maksiminopeuteen. (Boosted Boards 2015.)

Sähkörullalauta, jota varten tämä opinnäytetyö tehtiin, on jatkuvan kehityksen kohteena. Sen moottorinohjainta kehitettiin samaan aikaan tämän opinnäytetyön kanssa, jotta halutut toiminnot voitiin toteuttaa ja testata. Opinnäytetyönä ei siis ollut tarkoituksena tuottaa myytävää sovellusta, vaan toimiva prototyyppi.

Opinnäytetyön tarkoituksena oli myös selvittää React Native -sovelluskehyksen mahdollisuuksia monimutkaisemmassa Bluetooth- ja paikannusrajapintoja hyödyntävässä sovelluksessa. Alustariippumattomat mobiilisovelluskehykset mielletään yleensä käytännöllisiksi ainoastaan yksinkertaisia sovelluksia kehitettäessä.

# 2 SOVELLUKSEN SUUNNITTELU

## 2.1 Vaatimukset ja kehityksen lähtökohdat

Suunnittelun lähtökohtana oli kaksi vaatimusta: Android-yhteensopivuus sekä kommunikointi laudan kanssa Bluetoothilla. Laudan ohjaus (kiihdytys ja jarrutus) ei kuulunut mobiilisovelluksen vastuulle, vaan rullalautaa ohjattiin erillisellä ohjaimella, joka käytti erillistä radiopiiriä.

Sovelluksen tärkein toteutettava ominaisuus oli laudan sensoreilta ja ohjauslogiikalta saatavan tiedon lukeminen ja visualisointi. Laudan ohjausjärjestelmä lukee sensoreilta dataa, kuten akun lämpötila, moottorin pyörimisnopeus sekä pulssisuhde. Sensoridataan oli tarkoituksena yhdistää myös puhelimesta saatava sijaintitieto. Sovellusta voisi verrata erilaisiin liikuntasovelluksiin, joissa GPS-reitin lisäksi voi näkyä pyöräilijän syke ja kadenssi (poljinten pyörähdystaajuus). Opinnäytetyösovelluksen tapauksessa syke ja kadenssi korvautuvat moottorin virran voimakkuustiedolla, piirilevyn lämpötilalla ja moottorin pulssisuhteella.

## 2.2 React ja React Native

Mobiilisovelluksen tekniseen toteutukseen annettiin vapaat kädet. Natiivin Javalla tehtävän Android-kehityksen sijaan käytettiin alustariippumatonta React Native -sovelluskehystä.

React Native on Facebookin kehittämä avoimen lähdekoodin ohjelmistokehys. Se julkaistiin maaliskuussa 2015 iOS:lle (Occhino 2015) ja puoli vuotta myöhemmin, syyskuussa 2015 Androidille (Witte & Weitershausen 2015).

React Native antaa kehittäjien käyttää web-maailmasta tuttua JavaScript-kieltä ja React-ajatusmaailmaa. Kehys vetoaakin ehkä eniten web-kehittäjiin, joille Facebookin web-sovelluskehys, React, on entuudestaan tuttu. React Nativen mainoslause on "learn once, write anywhere" eli vapaasti suomennettuna "opettele kerran, käytä kaikkialla", millä vii-

tataan React-ajatusmallin opettelemiseen ja sen käyttämiseen web-, mobiili- tai jopa työpöytäsovellusten kehittämisessä (React Native 2016). React Native on kirjoitushetkellä kiinnostavin JavaScriptiä käyttävä mobiilisovelluskehys (Owens 2016).

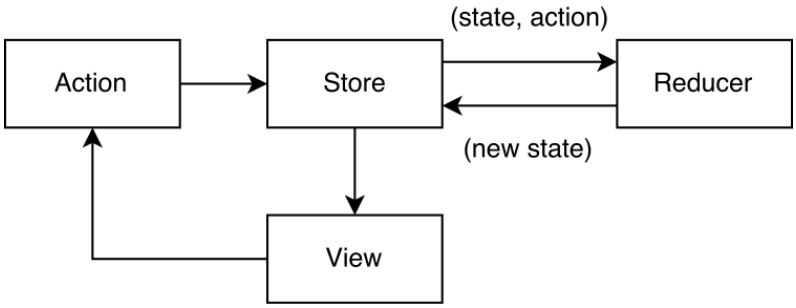
React Native perustuu Reactiin, joka on Facebookin kehittämä sovelluskirjasto käyttöliittymien ohjelmointiin. Se on kuvaileva ja komponenttipohjainen sovelluskehys, tarkoittaen sitä, että käyttöliittymä rakennetaan itsenäisistä osista, komponenteista, jotka päivittävät itsensä sovelluksen tilan mukaan. Deklaratiivinen käyttöliittymäohjelmointi helpottaa käyttöliittymän testausta, kun yksinkertaisten komponenttien sisältö määräytyy pelkästään niiden tilan mukaan. (React 2016).

### 2.2.1 Flux ja Redux

Facebook käyttää Reactin kanssa Flux-arkkitehtuuria, joka toimii hyvin yhteen Reactin kanssa. Flux-arkkitehtuurissa data liikkuu vain yhteen suuntaan. Flux koostuu neljästä osasta: Dispatcherista, Storeista, Actioneista ja Vieweistä. Dispatcher vastaanottaa actioneja ja lähettää (dispatch) sen kaikille Storeille. Store sisältää sovelluksen tilan ja Storeja voi olla yksi tai useampia. Storen sisältämää dataa voidaan muuttaa ainoastaan Actionin kautta. Kun Store muuttuu, se lähettää muutos-eventin, jota Viewit voivat kuunnella. Actionit muodostavat sovelluksen sisäisen rajapinnan, jonka kautta sovelluksen tilaa muutetaan. Action sisältää tyypin, esimerkiksi "poista-käyttäjä", ja mahdollisen datan, esimerkiksi "userId: 43". Actionin nimessä ei ole toteutuslogiikkaa, vaan pelkästään aikomus, mitä halutaan tehdä. Store tekee tarvittavat toimenpiteet käyttäjän poistamiseksi, kun se vastaanottaa *poista-käyttäjä*-actionin. View eli näkymä näyttää Storen sisältämää dataa ja päivittää itsensä huomatessaan Storen muutos-eventin. Viewissä voidaan käyttäjäsyötteen perusteella tehdä uusia Actioneita, jotka aiheuttavat tilan muutoksen ja näkymien päivittymisen. (Facebook 2017.)

Flux-arkkitehtuuri voidaan toteuttaa hieman eri tavoilla, joista yksi, ehkä suosituin on Redux. Kun Fluxissa datasäiliöitä eli storeja on useampia, Reduxissa niitä on vain yksi. Koko sovelluksen tila on yhdessä tavallisessa JavaScript-oliossa. Kun tilaa halutaan muuttaa (esimerkiksi kun vastaanotetaan uusi jännitearvo rullalaudalta), sovelluksessa lähetetään toiminto (dispatch action), joka kertoo mitä tapahtui (vastaanotettiin uusi arvo).

Tilan ja toimintojen lisäksi Reduxiin kuuluu *reducer*-funktioita, jotka ottavat parametrina nykyisen tilan (state) ja toiminnon (action) ja palauttavat toiminnon perusteella uuden tilan. React päivittää muuttuneet näkymät uuden tilan mukaisiksi automaattisesti. Datavirta Reduxissa on kuvattu kuvassa 1. (Abramov 2016.)




KUVA 1. Datavirta Reduxissa

Reducer-funktiot ovat puhtaita funktioita. Puhdas funktio on funktio, joka antaa samoilla parametreilla aina saman paluuarvon eikä aiheuta sivuvaikutuksia esimerkiksi muuttamalla (mutating) olemassa olevaa muuttujaa. Reducer-funktio ottaa parametrina vanhan tilaolion ja actionin, ja parametrina saamansa olion muuttamisen sijaan se palauttaa uuden tilaolion (Abramov 2016).

### 2.2.2 React Nativen ominaisuuksia

React Nativessa tulee mukana peruskomponentit, joilla sovellus rakennetaan. Näitä komponentteja ovat esimerkiksi *View*, *Text*, *ListView* ja *Button*. View on rakennekomponentti, jonka sisälle voi laittaa toisia komponentteja. Text on nimensä mukaan tekstikomponentti, jolla näytetään tekstiä. Useimmissa sovelluksissa on listamuotoista sisältöä, jota esitetään ListView-komponentilla. ListViewillä luodaan rullattava listanäkymä, jonka rivit voivat sisältää mitä tahansa. Toinen tapa luoda rullattavia näkymiä on käyttää ScrollView-komponenttia. Button luo kohdekäyttöjärjestelmästä riippuen oikean näköisen ja tutusti käyttäytyvän oletuspainikkeen. (React Native 2016.)

Komponenttien lisäksi React Native tarjoaa kehittäjälle erilaisia rajapintoja, kuten *Alert*, *Dimensions*, *Geolocation*, *CameraRoll*. Alertilla luodaan ponnahdusikkuna, Dimensions antaa laitteen näytön resoluution ja Geolocation tarjoaa paikannusrajapinnan. React Nativelle on lisäksi saatavilla tuhansia kolmannen osapuolen kirjastoja, jotka nopeuttavat

sovelluskehitystä (npm 2016). Komponentteja voidaan toteuttaa alustan natiivilla ohjelmointikielellä ja tarjota JavaScript-rajapinta sovelluksen käyttöön, jolloin kaikki laiterajapinnat ovat saatavilla. (React Native 2016.)

React Native poikkeaa useimmista muista JavaScriptiä käyttävistä mobiilisovelluskehyksistä sillä, että sen tuottama käyttöliittymä on kullekin mobiilialustalle natiivia. Toinen vaihtoehto alustariippumattomassa sovelluskehityksessä on sovellukset, joissa käyttöliittymä on toteutettu HTML5-tekniikalla ollen siten laiterajapintoja hyödyntävä nettisivu. Esimerkkinä tällaisesta lähestymistavasta on Apachen Cordova ja sen päälle rakennettu Ionic (Ionic 2016). Javascriptin lisäksi alustariippumattomia mobiilisovelluksia voi tehdä muun muassa C#-kielellä (Xamarin 2016).

Reactissa ja React Nativessa käyttöliittymä kuvataan JSX-kielellä, joka on XML-tyyppinen syntaksi käyttöliittymien kuvaamiseen (React 2016). Seuraava ohjelma tulostaa näytölle tekstin "Hello world!". JSX-tagi `<Text>` korvautuu alustasta riippuen natiivilla tekstielementillä ja StyleSheetissä määritellyillä ominaisuuksilla.

```
import React, { Component } from 'react';
import { AppRegistry, Text } from 'react-native';

class HelloWorldApp extends Component {
  render() {
    return (
      <Text style={styles.title}>Hello world!</Text>
    );
  }
}

const styles = StyleSheet.create({
  title: {
    fontSize: 19,
    color: '#66ff77',
  }
});

AppRegistry.registerComponent('HelloWorldApp', () => HelloWorldApp);
```

React Nativessa käytetään uusia JavaScriptin ominaisuuksia, jotka on määritelty EcmaScript 2015 -standardissa. Kyseisiä ominaisuuksia ovat mm. moduulien tuominen `import from` -käskyllä, luokan määrittely (`class` ja `extends`) sekä lyhyt syntaksi funktioiden määrittelyyn: `() => {}` (Ecma International 2015).

Käyttöliittymäkomponenttien tyylit määritellään luomalla StyleSheet-objekti ja antamalla kullekin komponentille omat tyylinsä. React Nativen StyleSheet-rajapinta muistuttaa hieman CSS-tyylejä web-maailmasta, esimerkiksi tekstin koon CSS:ssä määrittää `font-size` ja StyleSheetissä vastaava attribuutin nimi on `fontSize`. (React Native 2016.)

### 2.2.3 Sovelluksen jakelu

JavaScriptillä toteutetuissa mobiilisovelluksissa on eräs merkittävä hyvä puoli verrattuna natiivisovelluksiin, etenkin iOS-alustalla. Perinteisesti iOS-sovelluskehittäjät ovat lähettäneet sovelluksen testattavaksi Applelle, ennen kuin se voidaan julkaista sovelluskaupassa. Myös sovelluksen päivitykset täytyy hyväksyttää, ennen kuin ne pääsevät käyttäjille saakka. Nämä pakolliset testaukset saattavat viivästyttää sovelluksen julkaisua jopa usealla viikolla. React Nativessa ohjelmakoodia ajetaan JavaScriptCoressa, ja ohjelma voidaan ladata ja päivittää dynaamisesti ilman sovelluksen kääntämistä. Tämä ominaisuus mahdollistaa sovellusten etäpäivittämisen ilman, että päivitys pitää hyväksyttää Applella. Applen App Store sekä Googlen Play Store hyväksyvät tällaisen käytännön, kunhan sovelluksen alkuperäinen käyttötarkoitus säilyy. (Microsoft 2017.)

JavaScript-koodin etäpäivitykseen on saatavilla muutamia palveluita, kuten AppHub Deploy ja Microsoft CodePush. Molemmat ovat käytettävissä ilmaiseksi – AppHub on ilmainen rajoitetuilla ominaisuuksilla ja CodePush toistaiseksi ilmainen ilman rajoituksia. CodePush on kirjoitushetkellä selvästi suositumpi näistä vaihtoehdoista (NPMCompare 2017). Avoimen lähdekoodin projekteissa projektin aktiivisuudella ja käyttäjämäärällä on merkittävä rooli arvioitaessa ohjelmiston vakautta ja siinä esiintyviä ongelmia.

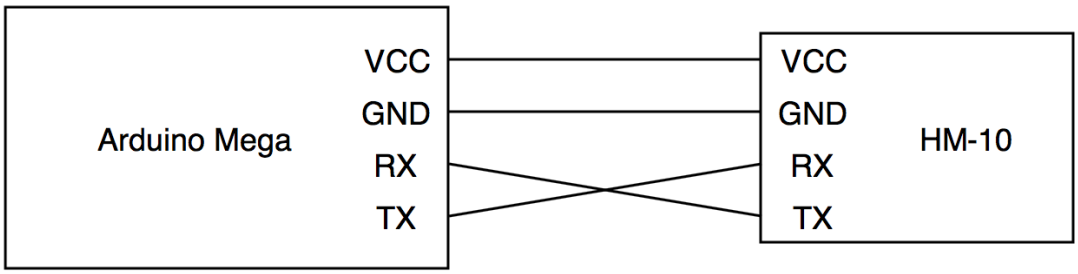
## 2.3 Bluetooth Low Energy

Kommunikointi laudan ja mobiilisovelluksen välillä hoidettiin Bluetooth Low Energy -tekniikalla (myöhemmin Bluetooth LE tai BLE). BLE on osa Bluetooth 4.0 -määritelmää ja sen ansiosta on voitu valmistaa pienempiä ja vähemmän virtaa kuluttavia laitteita kuin aiemmilla Bluetooth-versioilla. Nykyään useimmat puhelimet tukevat Bluetooth LE:tä (Bluetooth SIG 2016).

Bluetooth LE on tarkoitettu pienten datamäärien siirtoon, esimerkiksi juuri sensorikäyttöön. Se kuluttaa niin vähän virtaa, että laite voi toimia useita vuosia yhdellä nappiparistolla. Jos tiedonsiirto on jatkuvaa tai vaatii suurta bittinopeutta, on parempi käyttää Bluetoothin vanhempia versioita. (Heydon & Hunn 2016.)

## 2.4 Bluetooth-testilaite

Kehityksen avuksi rakennettiin testilaite Arduino Mega 2560 -kehitysalustasta ja HM-10-Bluetooth-breakout boardista. HM-10 sisältää Texas Instrumentsin CC2541-radiopiirin ja mm. regulaattorin, jonka avulla piiriä voi käyttää suoraan Arduinon 5 V jännitteellä. Bluetooth-piiriä käytettiin UART-sarjaliitynnän kautta kytkemällä se Arduinon RX- ja TX-pinneihin (kuva 2). Tämän jälkeen se oli käytettävissä ohjelmassa tavallisena sarjayhteytenä.




KUVA 2. Testilaitteen kytkentä

### 2.4.1 Laudan ja puhelimen välinen Bluetooth-protokolla

Mobiilisovelluksen ja laudan väliseen liikenteeseen kehitettiin seuraava tavumuotoinen protokolla. Yksinkertaisuuden vuoksi käytettiin vakiomittaista kehystä, jonka pituudeksi valittiin viisi tavua. Kehyksen ensimmäinen tavu määrittää parametrin, mitä laudalta halutaan lukea tai mitä halutaan muuttaa. Toinen tavu määrittää toiminnon eli halutaanko parametri lukea vai halutaanko siihen kirjoittaa. Kolmas ja neljäs tavu on varattu parametrin arvolle ja viides tavu on kehyksen loppumerkki (kuva 3).

| Parametri | R/W | Data 1 | Data 2 | Loppumerkki |
|---|---|---|---|---|

KUVA 3. Protokollakehyksen viisi tavua

Kullekin laudan parametrille määriteltiin ASCII-merkki, jota käytettiin kehyksen ensimmäisenä tavuna. Seuraavat parametrit ovat pelkästään lukemista varten eli niihin ei voi kirjoittaa:

- `T` Temperature eli lämpötila
- `V` Voltage eli akkujännite
- `H` Throttle eli kaasun asento
- `M` Motor Speed eli moottorin pyörimisnopeus
- `A` Actual Speed eli laudan todellinen nopeus
- `D` Duty Cycle eli moottorin pulssisuhde
- `C` Motor Current eli moottorin virta.

Seuraavat parametrit ovat konfiguroitavia eli niitä voidaan lukea ja ne voidaan asettaa uuteen arvoon:

- `d` Wheel Diameter eli laudan renkaan halkaisija
- `w` Wheel Gear Tooth Count eli renkaan hammaslukumäärä
- `m` Motor Gear Tooth Count eli moottorin hammaslukumäärä
- `k` Motor KV Value eli moottorin vääntöön liittyvä ominaisarvo.

Toiselle tavulle määriteltiin kaksi mahdollista merkkiä:

- `R` Read eli luku
- `W` Write eli kirjoitus.

Sensoriparametrien lisäksi protokollaan määriteltiin jatkuvan lähetyksen tila (live mode, `L`), jolla rullalauta saatiin lähettämään jatkuvasti sensoritietoja sen sijaan, että niitä pyydettäisiin aina erikseen lukukehyksellä.

Kaksi datatavua tulkitaan yhtenä 16-bittisenä etumerkittömänä (eli positiivisena) kokonaislukuna siten, että kolmas tavu on luvun eniten merkitsevä osa ja neljäs tavu vähiten merkitsevä. Esimerkiksi jos tavujen sisältö olisi 0xAF ja 0x01, niin datatavujen sisältämä luku olisi 0xAF01 eli $44801_{10}$.

Koska datatavut tulkittiin etumerkittömänä kokonaislukuna, sovittiin desimaaliluvuille omat käytäntönsä. Esimerkiksi jos jännitearvo luetaan 0,1 V:n resoluutiolla, voidaan se

kertoa kymmenellä, jolloin siitä tulee kokonaisluku. Vastaanottopäässä tehdään vastaava käänteislaskutoimitus, jolloin saadaan alkuperäinen desimaaliluku.

Kehyksen lopetustavuksi määritettiin *Carriage Return* -merkki (`\r`, 0x0D). Taulukossa 1 on esimerkki kokonaisesta kehyksestä, jolla kirjoitetaan renkaan halkaisijaksi desimaaliluku 80.

TAULUKKO 1. Protokollan esimerkkikehys

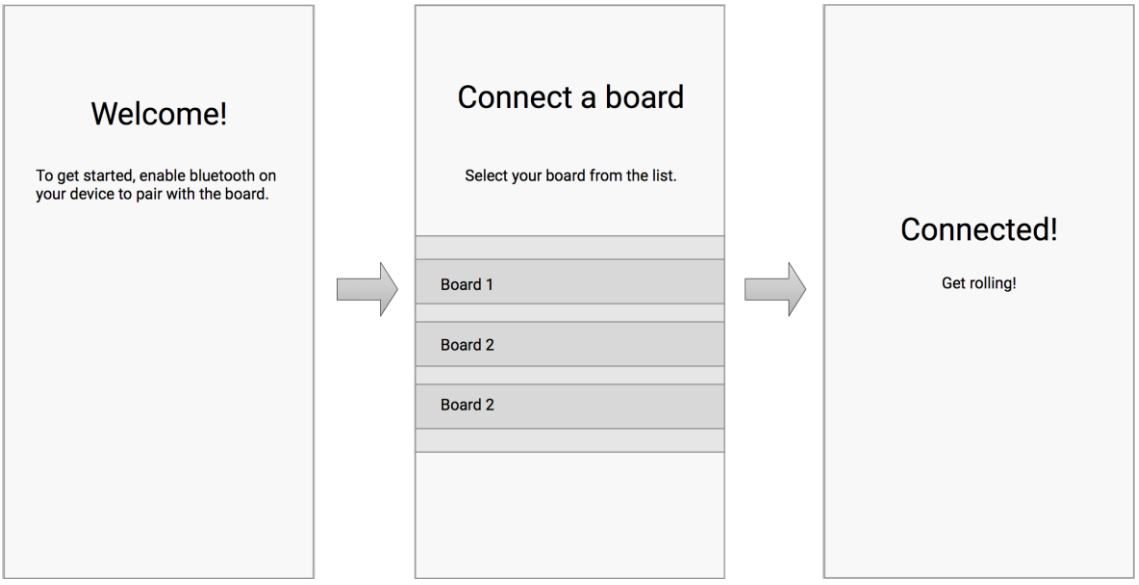
| Tavu | 16-kantainen | Binääri | Selitys |
|---|---|---|---|
| 1 | 0x64 | 0110 0100 | “d” eli renkaan halkaisija |
| 2 | 0x57 | 0101 0111 | “W” eli kirjoituskomento |
| 3 | 0x00 | 0000 0000 | Nollia pienen luvun alussa |
| 4 | 0x50 | 0101 0000 | Kymmenkantaisena luku 80 |
| 5 | 0x0D | 0000 1101 | Loppumerkki, Carriage Return |

Vakiomittainen kehys helpotti ohjelman kirjoittamista, sillä vastaanotetun kehyksen käsittelemiseen riittää yksinkertaisimmillaan pieni rengaspuskuri, johon vastaanotetut tavut tallennetaan. Kun vastaanotetaan kehyksen loppumerkki, voidaan päätellä koko kehyksen olevan vastaanotettu ja hakea kehyksen kaikki tavut rengaspuskurista.

Protokollaan ei toteutettu eheyden tarkistusta, sillä UART-piiri tekee yksinkertaisen pariteettibittiin perustuvan tarkistuksen laitteistotasolla (Basics of UART Communication 2016), jonka lisäksi BLE-standardiin kuuluu CRC-tarkistus (Heydon & Hunn 2016). Vaikka virheellisiä kehyksiä vastaanotettaisiin, se ei haittaisi mitään. Jos kehyksen ensimmäinen tai viimeinen tavu olisi viallinen, kehys hylättäisiin vastaanottopäässä. Jos datatavuissa olisi kuitenkin jokin virhe, se tallentuisi sellaisenaan, mutta häviäisi visualisoitaessa käytettävään suodatukseen.

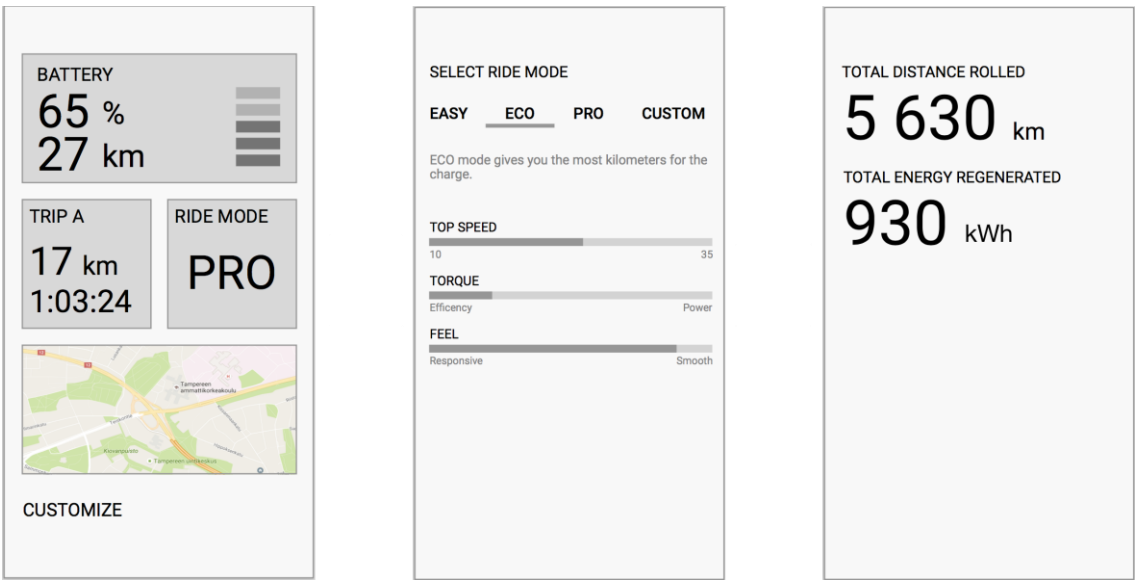
## 2.5 Käyttöliittymän suunnittelu

Sovelluksen käyttöliittymää varten pohdittiin, mitä käyttäjä haluaa sovelluksella saavuttaa. Tarpeita olivat a) tiedonsiirtoyhteyden avaaminen rullalaudan ja puhelinsovelluksen välille b) rullalaudan yleistietojen tarkastelu sekä säätö ja c) rullalaudan sensoridatan keräys ja visualisointi. Tiedonsiirtoyhteyden avaaminen puhelimen ja laudan välille on käyttöliittymältään varsin suoraviivainen prosessi. Jos sovellus ei ole yhteydessä rullalautaan, se huomauttaa siitä käyttäjälle, jos tämä yrittää esimerkiksi aloittaa uuden datankeräysjakson. Suunnittelun alussa piirrettiin viivapiirroksia, joista yhdistämisprosessi on kuvassa 4. Viivapiirrokset eivät edusta lopullista ulkoasua eikä käyttölogiikkaa. Kyseessä on ensimmäinen suunnitteluiteraatio, johon kerättiin ideoita niiden toteuttamiskelpoisuudesta riippumatta. Lopullinen design ja ominaisuudet muotoutuivat ohjelmistokehityksen yhteydessä.



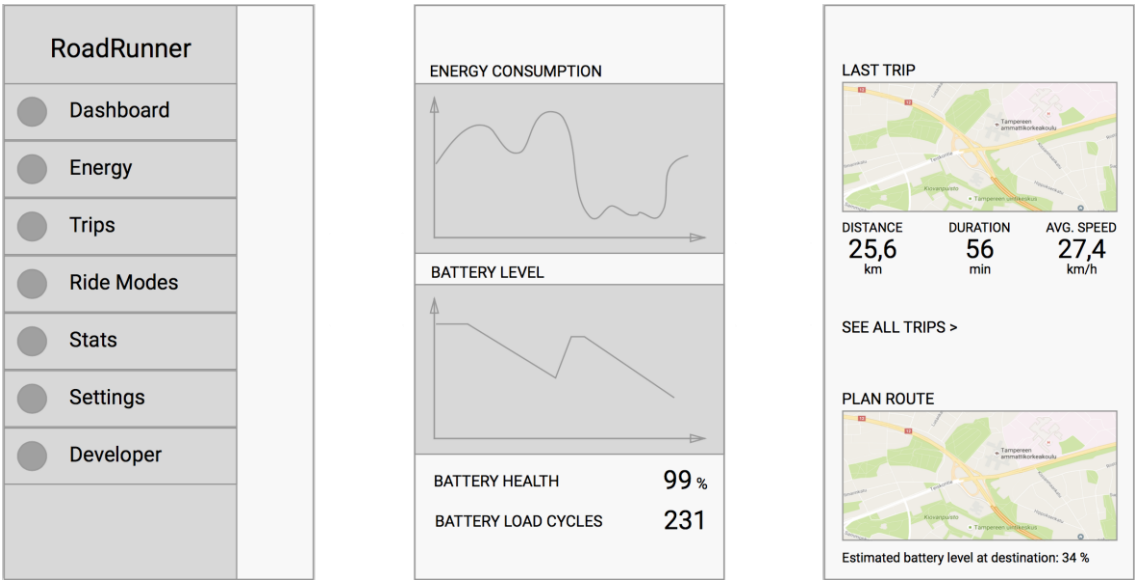

KUVA 4. Tiedonsiirtoyhteyden avaaminen rullalaudan ja sovelluksen välille

Tiedonsiirtoyhteyden avaamisen jälkeen käyttäjä näkee laudan yleistietoja ja tilastoja sekä pääsee kustomoimaan laudan käyttäytymistä erilaisilla ajomoodeilla (kuva 5).




KUVA 5. Yleiskatsaus, ajomoodin kustomointi ja tilastot

Kuten kuvasta 6 nähdään, sovelluksen navigaatio piirrettiin aluksi vetovalikkopohjaiseksi (drawer), sillä pois näytöltä olevaan valikkoon on helppo lisätä näkymiä sen suuremmin suunnittelematta. Valikkorakenne yksinkertaistui kuitenkin välilehdillä toimivaksi toteutuksen yhteydessä. Laudalta saatavaa dataa visualisoidaan erilaisilla viivakuvaajilla. Kuvaan on piirretty Tesla-sähköautoista tutut historiakuvaajat energian kulutuksesta ja akun varauksesta. Ajatuksena oli näyttää viimeaikaiseen käyttöön perustuva ennuste akun kestosta.




KUVA 6. Vetovalikko, kuvaajat ja reittinäkymä

## 2.6 Mobiilisovelluksen energiankulutus

Mobiililaitteiden yksi suurimmista ongelmista on akun kesto. Laitteen käytöllä on luonnollisesti suuri merkitys akun kestoaikaan. Sovelluksen kuluttamaan energiaan vaikuttaa mm. prosessorin käyttöaste, taustatoiminta, paikannus ja Bluetoothin käyttö. Prosessorin käyttöasteeseen voi vaikuttaa suunnittelemalla tehokas ohjelmisto, jossa ei tehdä ylimääräistä laskentaa. Paikannuksen kuluttamaa energiaa voi vähentää pienentämällä paikannuksen tarkkuutta sekä kasvattamalla sijaintipyyntöjen väliaikaa. Bluetoothin kuluttamaa energiaa voi vähentää niputtamalla siirrettävän datan ja lähettämällä useita datayksikköjä kerralla. Bluetooth Low Energyn energiankulutus lepotilassa on huomattavasti pienempi kuin aktiivisena. (Apple 2016.)

Sovelluksen taustatoiminta tulisi olla mahdollisimman vähäistä. Huonosti toteutettu sovellus voi pysyä käynnissä taustalla ja kuluttaa turhaan energiaa. Taustatoimintaa käytetään esimerkiksi musiikin toistamiseen tai pitkäkestoisiin verkkolatauksiin. Opinnäytetyön tapauksessa sovelluksella on tarve käyttää taustalla paikannusta ja Bluetooth-yhteyttä, kun ajon tallennus on käynnissä. Kun ajon tallennus ei ole meneillään, sovelluksen tulisi käyttää mahdollisimman vähän resursseja.

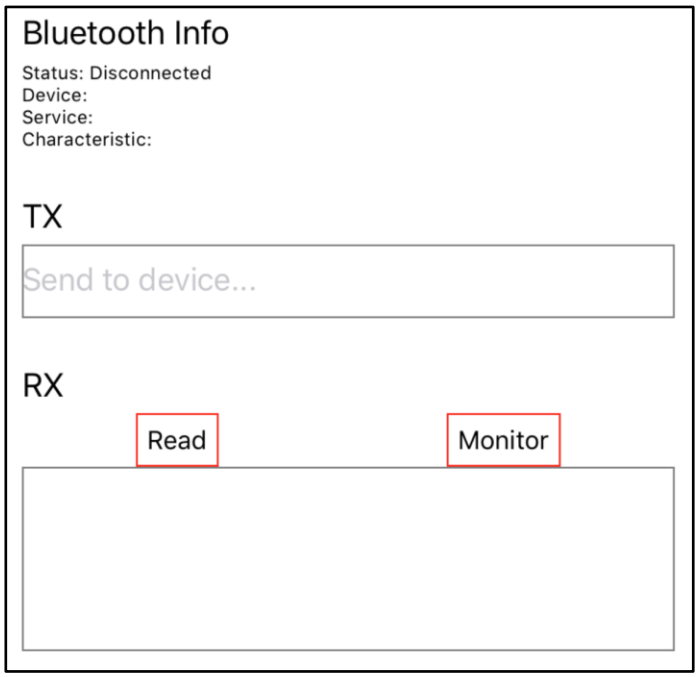
# 3 SOVELLUKSEN TOTEUTUS

## 3.1 React Native ja BLE

React Native ei sisällä oletuksena minkäänlaista Bluetooth-rajapintaa. BLE-kirjastoksi valittiin varsovalaisen ohjelmistoyritys Polidean kehittämä avoimen lähdekoodin kirjasto *react-native-ble-plx*, koska se oli hyvin dokumentoitu ja aktiivisen kehityksen kohteena. Polidea on kehittänyt myös kirjaston pohjalla olevat natiivit kirjastot iOS:lle ja Androidille. (Polidea 2017.)

### 3.1.1 Sarjaliikennemonitori

Sovellukseen toteutettiin aluksi yksinkertainen sarjaliikennemonitori, jonka avulla testattiin BLE-kirjaston toimivuutta. Kuvan 7 yläosassa ovat Bluetooth-yhteyden tiedot. TX-kenttään kirjoitettava teksti lähetetään Bluetoothilla ja vastaanotetut tavut näkyvät RX-kentässä. RX-kentän yläpuolella ovat painikkeet Bluetooth-laitteen lukemiselle. Käytännössä Read-toimintoa ei tarvittu, sillä yhteyden avaamisen jälkeen sovellus asettui kuuntelemaan (monitor) laitetta, jolloin se vastaanotti uudet tavut automaattisesti.




KUVA 7. Sarjaliikennemonitorin käyttöliittymä

### 3.1.2 JavaScript ja Base64

React Nativen ja natiivin koodin välillä siirtyvä data voi olla jotain JavaScriptin perustietotyypeistä (mm. String, Number, Array), joten lähetettävä data muunnetaan Base64-muotoon ennen BLE-kirjastolle lähetystä. Vastaavasti kirjaston kautta vastaanotetut tavut ovat Base64-koodattuja. (Polidea 2017.) Base64 on koodaustapa, jolla voidaan esittää binääridataa ASCII-merkistöllä (käytössä merkit A-Z, a-z, 0-9, +, / ja =) (Josefsson 2006).

JavaScript on heikosti tyypitetty ohjelmointikieli ja siinä on vain yksi tietotyyppi luvuille: Number. Number on aina 64-bittinen double-tarkkuuksinen desimaaliluku (Ecma International 2015, 4.3.20) eikä se siis sovi sellaisenaan protokollassa määriteltyyn 16-bittiseen etumerkittömään kokonaislukuun.

Binääridatan käsittely JavaScriptissä on mahdollista Node.js:n Buffer-rajapinnalla (Node.js Foundation 2017), joka on toteutettu myös selaimille ja React Nativelle omana JavaScript-moduulina (Aboukhadijeh 2016).

Buffer-rajapintaa käyttämällä voidaan luoda UInt16-tyyppinen luku ja koodata se Base64-merkkijonoksi seuraavasti.

```
// Koodattava luku
const fifty = 50;

// Alustetaan 2 tavun mittainen Buffer-olio
const buffer = Buffer.alloc(2);

// Kirjoitetaan 16-bittinen etumerkitön kokonaisluku Bufferiin
buffer.writeUInt16BE(fifty);

// Luodaan Base64-merkkijono Bufferista
const base64string = buffer.toString('base64');

// Luodaan uusi Buffer Base64-merkkijonosta
const buffer2 = new Buffer(base64string, 'base64');

// Luetaan 16-bittinen etumerkitön kokonaisluku
const fiftyAgain = buffer2.readUInt16BE();

// fifty == fiftyAgain == 50
```

## 3.2 Testilaitteen ohjelmisto

Testilaitteelle kirjoitettiin aluksi yksinkertainen ohjelma, joka vastaanottaa tavuja tietokoneelta USB-sarjayhteydellä ja lähettää ne Bluetoothilla puhelimeen. Vastaavasti Bluetoothilla vastaanotetut tavut välitetään tietokoneelle. Näin todettiin siirtotien toimivuus ja React Nativen sekä valitun Bluetooth-kirjaston käyttökelpoisuus Bluetoothia hyödyntävissä sovelluksissa.

Tämän jälkeen toteutettiin protokolla testilaitteelle. Rullalaudan sensoriarvoja simuloitiin määrittämällä parametrille arvo ja summaamalla siihen pientä kohinaa. Testilaite tuki myös konfiguroitavia parametreja, joista esimerkkinä renkaan halkaisijan määrittäminen. Jos laite vastaanotti kehyksen `[Write | Renkaan halkaisija | 75 mm | EOF]`, se tallensi renkaan halkaisijaksi vastaanotetun arvon 75.

Protokollamäärittelyn mukaisesti laitteen sai myös jatkuvan lähetyksen tilaan, jolloin se lähetti kaikki data-arvot puhelimelle määrätyin aikavälein. Tämä oli tarpeellista mobiilisovelluksen rajoitetun taustatoiminnan vuoksi. Datan keräys puhelimen näytön ollessa kiinni ei onnistunut pyytämällä tietoja puhelimella, sillä puhelimen lukitseminen aiheutti sovelluksen siirtymisen lepotilaan ja pyyntöjen loppumisen. Datan keräys taustalla onnistui monitoroimalla, jolloin käyttöjärjestelmä herätti sovelluksen käsittelemään vastaanotetun kehyksen.

### 3.2.1 Tavujen vastaanotto ja lähetys testilaitteessa

Testilaitteen ohjelmassa on rengaspuskuri vastaanotetuille tavuille. Kun tavu vastaanotettiin, tarkistettiin, oliko se kehyksen loppumerkki. Jos ei, tallennettiin se vastaanottopuskuriin. Jos kyseessä oli kehyksen loppumerkki, luettiin vastaanottopuskurista neljä viimeisintä tavua ja tulkittiin kehyksen tarkoitus.

Kehysten lähetys tapahtui seuraavasti. Muuttuja *data* on 16-bittinen kokonaisluku, joka jaetaan kahteen 8-bittiseen osaan (*hi* ja *lo*) tavupohjaista protokollaa varten. Muuttuja *txBuffer* on lähetyspuskuri, jonka sisältö lähetetään lopulta Bluetooth-yhteydellä.

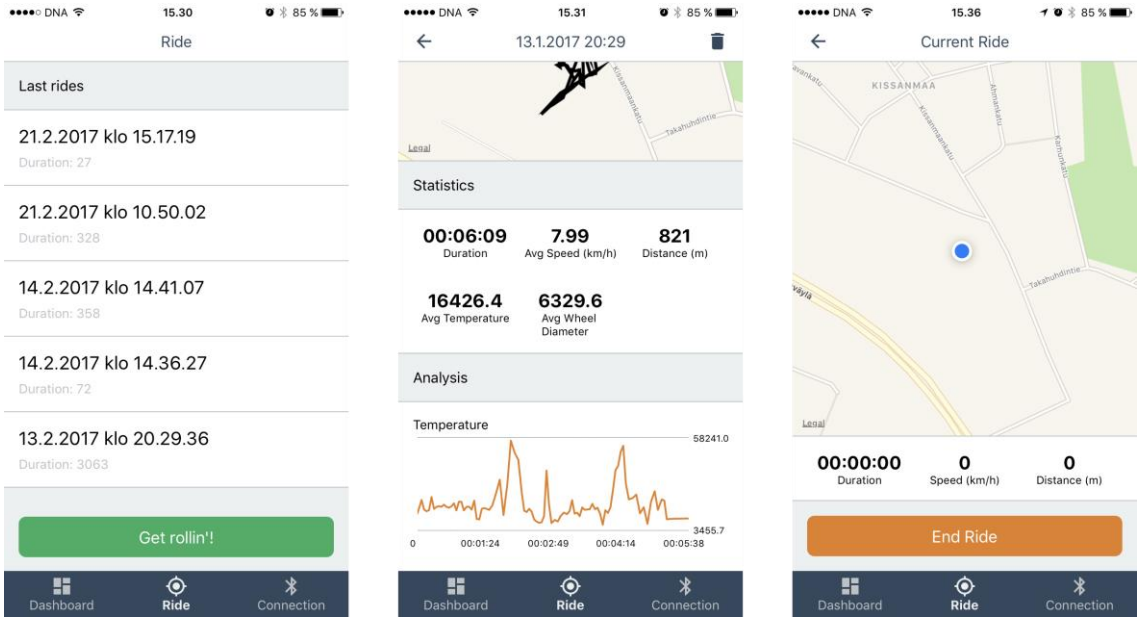
```
uint8_t hi = ((data >> 8) & 0xff);
uint8_t lo = ((data >> 0) & 0xff);

txBuffer[0] = parameterId;
txBuffer[1] = WRITE;
txBuffer[2] = hi;
txBuffer[3] = lo;
txBuffer[4] = END_OF_FRAME;

Serial1.write(txBuffer, txBufferSize);
```

## 3.3 Käyttöliittymän toteutus

### 3.3.1 Navigaatiorakenne

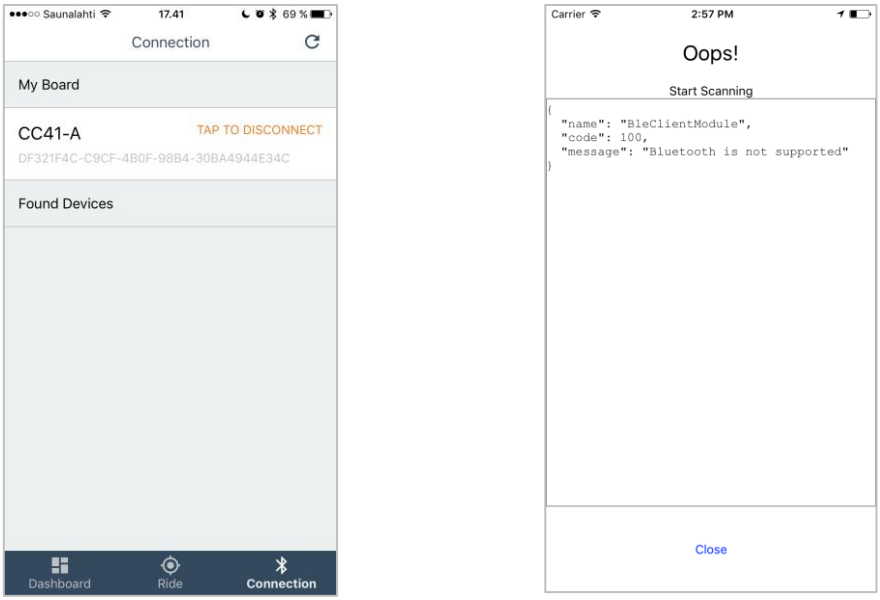
Kuten aiemmassa käyttöliittymän suunnittelua koskevassa luvussa mainittiin, sovelluksessa käytettiin hampurilaiskuvakkeen ja sivuvalikon sijaan alhaalla sijaitsevaa navigointiriviä. Kyseinen käyttöliittymäelementti on ollut iOS-järjestelmissä alusta asti suositeltava tapa toteuttaa navigointi päätason näkymien välillä (Apple 2017, Tab Bars). Androidin tyyliohjeeseen alanavigointi tuli maaliskuussa 2016 (Google 2017).

Sovellukseen tehtiin kolme päänäkymää: *dashboard*, *ride* ja *connection*. Dashboard on yleisnäkymä, jossa tulee olemaan laudan tietoja ja tilastoja. Dashboard-näkymää ei kuitenkaan toteutettu opinnäytetyön puitteissa. Ride-välilehdeltä (kuvassa 8 vasemmalla) voi aloittaa ajon tallennuksen (kuvassa oikealla) sekä tarkastella aiemmin tallennettuja ajoja (kuvassa keskellä). Tallennetun ajon tarkastelunäkymässä ovat kuljettu reitti kartalla, tilastot sekä viivakuvaajat sensoreilta. Ajon voi poistaa painamalla oikeasta yläkulmasta roskakorin kuvakkeesta, jolloin ohjelma vielä varmistaa poistamisaikeen. Näkymästä voi myös jakaa ajon taulukkomuodossa. Ajon tallennusnäkymässä ovat oma sijainti ja kuljettu reitti kartalla sekä jatkuvasti päivittyvät tilastot ajosta.




KUVA 8. Ride-näkymä ja sen alinäkymät

Connection-välilehdellä (kuva 9) voi skannata lähellä olevia Bluetooth-laitteita ja käynnistää tiedonsiirtoyhteyden rullalautaan sekä katkaista olemassa olevan yhteyden. Laitteiden skannaus käynnistetään oikeassa yläkulmassa olevasta kuvakkeesta, jonka jälkeen lähistöllä olevat Bluetooth-laitteet ilmestyvät listaan Found Devices -otsikon alle. Päänäkymien lisäksi luotiin muun sovelluksen päälle tuleva modal-näkymä virheilmoituksia varten. Virhetilanteessa näkymä avautuu ja virheen tiedot esitetään käyttäjälle. Kuvassa 9 on iOS-simulaattorissa esiintyvä virheilmoitus, joka kertoo, ettei Bluetooth ole käytettävissä.




KUVA 9. Connection-näkymä ja virheilmoitus

### 3.3.2 Viivakuvaajien piirtäminen

React Nativelle ei ollut saatavilla sopivia, kehityshetkellä ylläpidettyjä ja toimivia, kirjastoja viivakuvaajan piirtämistä varten, joten viivakuvaajat piti piirtää itse. React Nativessa on vektorigrafiikan piirtämistä varten React Native ART -kirjasto, johon ei kirjoitushetkellä ollut virallista dokumentaatiota. ART-rajapinnan käyttöä oli kuitenkin esitelty muutamissa blogikirjoituksissa yhdessä D3-JavaScript-kirjaston kanssa.

Aluksi luotiin skaalausfunktiot x- ja y-akseleille. X-akselilla on aika, joten käytettiin D3:n scaleTime-funktiota. Y-akselilla on sensorin arvo, joka skaalattiin lineaarisesti. Skaalausfunktiot ottavat parametrina arvon lukuväliltä *domain* ja skaalaa sen välille *range*. Kun rangeksi asetetaan `[0, chartWidth]`, on paluuarvo tarkka pisteen sijainti kuvaajassa.

```
const scaleX = d3.scale.scaleTime()
  .domain([new Date(minX), new Date(maxX)])
  .range([0, width]);

const scaleY = d3.scale.scaleLinear()
  .domain([minY, maxY]).nice()
  .range([height, 0]);
```

Tämän jälkeen luotiin skaalausfunktioiden avulla kuvaajan murtoviivaa kuvaava objekti.

```
const lineShape = d3.shape.line()
  .x(d => scaleX(d.timestamp))
  .y(d => scaleY(d.value));

const path = lineShape(data);
```

Lopulta voitiin piirtää ART-kirjastolla viivakuvaajan murtoviiva. Shape-komponentille annettiin d-propsina aiemmin luotu murtoviivan polku.

```
<Surface
  width={chartWidth}
  height={chartHeight}
>
  <Group x={0} y={0}>
    <Shape
      d={path}
      stroke={Color.carrot}
      strokeWidth={2}
    />
  </Group>
</Surface>
```

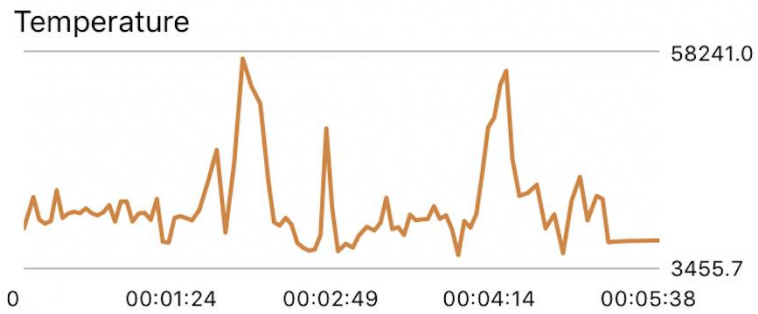
Aika-akselin arvojen sijoitusta varten jaettiin ajon aikaväli neljään osaan ja skaalattiin ajanhetki akselille.

```
const ticks = [0, 1, 2, 3, 4]
  .map(n => {
    const tickTime = n * (maxX - minX) / 4;
    return {
      x: scaleX(new Date(minX + tickTime)),
      time: tickTime,
    };
  }),
```

Tekstien piirtämiseen käytettiin tavallista Text-komponenttia, joka asemoitiin sopivalle etäisyydelle vasemmasta reunasta.

```
<View style={styles.axisX}>
  {
    this.state.ticks.map(tick => {
      const tickWidth = chartWidth / 6;
      const tickStyles = {
        position: 'absolute',
        textAlign: 'center',
        width: tickWidth,
        left: tick.x - tickWidth / 2,
      };
      return (
        <Text key={tick.time} style={tickStyles}>
          {tick.time}
        </Text>
      );
    })
  }
</View>
```

Y-akseli oli yksinkertaisempi, koska se sisältää vain minimi- ja maksimiarvot, eikä väliarvoja. Näin piirrettiin kuvan 10 mukainen kuvaaja.




KUVA 10. Viivakuvaaja

### 3.3.3 Käyttöliittymien eroja iOS- ja Android-järjestelmissä

Android- ja iOS-käyttöjärjestelmissä on erilaisia ohjeistuksia käyttöliittymästä. Esimerkiksi näytön yläreunassa olevan navigaatiopalkin teksti on Androidilla vasemmassa reunassa ja iOS:lla keskitettynä. Alustakohtaisia tyylimäärityksiä voi tehdä React Nativessa muutamalla eri tavalla. Pieniin eroavaisuuksiin voi käyttää Platform-rajapintaa, esimerkiksi seuraavalla tavalla. Koodilla määritellään navigaatiopalkin otsikon tyyli. Jos alustana on iOS, keskitetään teksti ja asetetaan fonttikoko 17. Vastaavasti Androidin ollessa kyseessä teksti asemoidaan vasempaan reunaan ja fonttikoko on hieman suurempi.

```
title: {
  ...Platform.select({
    ios: {
      textAlign: 'center',
      fontSize: 17,
    },
    android: {
      textAlign: 'left',
      fontSize: 20,
    },
  }),
  color: Color.wetAsphalt,
  flex: 2,
},
```

Jos eroavaisuuksia on paljon, on selkeämpää tehdä komponentista molemmille alustoille oma tiedostonsa. Esimerkiksi jos projektissa ovat tiedostot *NavigationBar.android.js* ja *NavigationBar.ios.js*, voidaan komponenttia käyttää lisäämällä se komennolla `import './NavigationBar';` ilman *android*- tai *ios*-päätettä tiedostonimessä. React Native osaa ottaa oikean tiedoston käyttöön oikealle alustalle.

## 3.4 GPS-jäljen tallennus

React Nativessa on *Geolocation*-paikannusrajapinta, jonka kautta kehittäjä voi pyytää käyttäjän sijaintia ohjelmassa. Rajapinta palauttaa sijainnista mm. koordinaatit (latitude ja longitude), korkeuden (altitude) sekä aikaleiman (timestamp). Sijainnin tallennuksen lisäksi kuljettu reitti haluttiin esittää kartalla. React Nativen mukana tullut MapView-rajapinta on vanhentunut, ja dokumentaatiossa suositellaan käyttämään Airbnb:n kehittämää monipuolista *react-native-maps*-kirjastoa. (React Native 2016.)

Kartta ja murtoviiva (polyline) tallennetuista koordinaateista voidaan luoda yksinkertaisimmillaan seuraavasti. Seuraava JSX-kuvaus luo iOS-käyttöjärjestelmillä natiivin kartan käyttäen Applen MapKitiä ja vastaavasti Androidilla käyttäen Google Maps -rajapintaa (Airbnb 2017).

```
<MapView>
  <MapView.Polyline
    coordinates={myCoordinates}
  />
</MapView>
```

GPS-laitteelta saatua sijaintitietoa ei yleensä hyödynnetä sellaisenaan sen epätarkkuuden vuoksi. Kuvassa 11 on Geolocation-rajapinnalta tullut sijaintidata murtoviivalla esitettynä. Viiva mutkittelee paljon, vaikka kävelyreitti oli suoraviivainen. Liikennevaloissa paikallaan oltaessa GPS-sijainti pyörii todellisen sijainnin ympärillä aiheuttaen suttua.

GPS-sijainnin epätarkkuus johtuu mm. GPS-satelliitin kiertoratapoikkeamista, kellovirheistä, maapallon ionosfääristä ja troposfääristä sekä vastaanotinlaitteen virheistä. Näiden lisäksi usein merkittävimmän virheen aiheuttaa ympäristön esteet ja siitä johtuva monitieeteneminen.

Suodattamaton sijaintitieto aiheuttaa suttuisen reittiviivan lisäksi ongelmia reitin tilastotietojen laskennassa. Jos reitin pituuden laskee pisteestä pisteeseen, aiheuttaa epätarkkuus paljon ylimääräistä pituutta. Siitä johtuen myös reitin keskinopeuden arvo menee pieleen.

Opinnäytetyössä tilannetta parannettiin käyttämällä yksinkertaista suodatusta. Geolocation-rajapinta antaa sijaintitiedossa koordinaattien lisäksi tarkkuuslukeman, joka kertoo todellisen sijainnin osuvan jonnekin tarkkuuslukeman säteisen ympyrän sisäpuolelle. Suodatus koostuu kahdesta vaiheesta, joista ensimmäinen on sijaintien suodatus tarkkuuden perusteella. Jos saadun sijaintipisteen tarkkuusarvo on huonompi kuin 15 m esimerkiksi oltaessa rakennuksen sisällä tai korkeiden rakennusten välissä, sijaintipistettä ei oteta huomioon. Toinen suodatusvaihe on seuraava: Otetaan sijainti ja tallennetaan se. Sen jälkeen saadut sijainnit tallennetaan vain, jos niiden poikkeama edellisestä tallennetusta sijainnista on enemmän kuin kaksi kertaa sijainnin tarkkuuslukema. Näin vältetään ylimääräiset sijaintipisteet lähellä toisiaan.


KUVA 11. Puhelimen antama raaka sijaintidata kartalla

### 3.5 Tietojen vienti myöhempää tarkastelua varten

Yksi sovelluksen tarkoituksista oli kerätyn datan analysointi ja sitä varten toteutettiin datan vienti taulukkomuodossa erillisiä analysointityökaluja varten. Tiedostomuodoksi valittiin yksinkertaisin CSV-muoto (comma separated values), joka tarkoittaa pilkuilla erotettuja arvoja. Esimerkki viedystä CSV-tiedostosta (osa siitä) on alla. Ensimmäisellä rivillä ovat otsikot ja seuraavilla riveillä *timestamp*-sarakkeen ajanhetkellä saadut muut arvot.

```
timestamp,     latitude,          longitude,         altitude
1487672833300, 61.49929454340078, 23.80640041641419, 121.518946137921
1487672834400, 61.49929454340078, 23.80640041641419, 121.518946137921
1487672838400, 61.49930048311586, 23.80643916647799, 110.075066999064
1487672841200, 61.49938101269366, 23.80643131627714, 121.351475682721
1487672842200, 61.49940076298765, 23.80644049991235, 121.781408047156
1487672844000, 61.49936051794328, 23.80650113911132, 122.928232356368
```

### 3.6 Sovelluksen testaus

Testaus on tärkeä osa sovelluskehitystä myös opinnäyteprojektissa. Projektissa käytettiin Jest-testauskirjastoa, joka on Facebookin kehittämä, Reactin kanssa hyvin yhteensopiva sekä helposti käyttöönotettava JavaScript-testauskirjasto. Jestillä testattaessa käytetään muistakin testikirjastoista tuttuja *describe*- ja *it*-lohkoja esimerkiksi seuraavasti. Kyseessä on osa lukuja pyöristävän apufunktion testeistä.

```
describe('msToHHMMSS', () => {
  it('handles zero', () => {
    expect(msToHHMMSS(0)).toBe('00:00:00');
  });
  it('handles hours', () => {
    expect(msToHHMMSS(9000000)).toBe('02:30:00');
    expect(msToHHMMSS(1000 * 60 * 60 * 4)).toBe('04:00:00');
  });
});
```

Jestissä on perinteisen, käsin kirjoitettavan, `expect(x).toBe(x)`-syntaksin lisäksi joissain tapauksissa helpompi *Snapshot*-testaustapa. Sitä käytettäessä voidaan testata monimutkaisia objekteja ilman vastaavien, monimutkaisten, objektien käsin kirjoittamista. Jest tekee ensimmäisellä kerralla testattavasta objektista snapshotin erilliseen tiedostoon, jonka jälkeisillä kerroilla se vertailee saatua objektia edelliseen snapshottiin. Jos tulos ei vastaa tallennettua snapshottia, Jest näyttää, mitkä rivit eroavat ja miten. Tämä mahdollistaa monimutkaisten objektien, esimerkiksi Redux-storen testaamisen hyvin pienellä määrällä ylimääräistä koodia:

```
describe('Board Reducer', () => {
  it('returns the initial state', () => {
    const initial = reducer(undefined, {});
    expect(initial).toMatchSnapshot();
  });

  it('turns on live mode', () => {
    const next = reducer(initialState, actions.liveMode(true));
    expect(next).toMatchSnapshot();
  });
});
```

Automaattiset testit auttavat huomaamaan vahingossa tehtyjä muutoksia, jotka voivat aiheuttaa odottamattomia ongelmia muualla koodissa.

# 4 YHTEENVETO

Sovelluksen kehityksessä oli joitakin epävarmuuksia liittyen React Native -ympäristöön. Tausta-ajo, Bluetooth, GPS-sijainti ja bittitason operointi olivat kaikki uusia ja kokeilemattomia asioita projektia aloitettaessa. React Native osoittautui kuitenkin toimivaksi ratkaisuksi monipuolisen alustariippumattoman mobiilisovelluksen kehitykseen. Samassa ajassa osaava Android-kehittäjä olisi voinut kehittää myös natiivin Android-sovelluksen, mutta React Nativea käyttäen sovellus saatiin toimimaan myös iOS-järjestelmissä.

Suurin yksittäinen huomio kehityksessä oli sovelluksen tausta-ajo. Mobiilikäyttöjärjestelmät, varsinkin iOS, pyrkivät minimoimaan sovelluksen energiankulutusta mm. rajoittamalla sen toimintaa taustalla tai laitteen ollessa lukittuna. React Native -sovelluksen suoritus pysäytetään, kun laite menee lepotilaan. Käyttöjärjestelmä voi kuitenkin herättää sovelluksen käsittelemään sille tullut viesti, jos sovellus on asetettu kyseisen viestin kuuntelutilaan. Näin sovelluksella on hetki aikaa suorittaa tarvittavat toimenpiteet, jonka jälkeen suoritus taas pysäytetään. Tätä toiminnallisuutta hyödynnettiin GPS-sijainnin ja Bluetooth-yhteyden taustatoiminnassa.

JavaScript on heikosti tyypitetty kieli, jonka ainut lukuja kuvaava tietotyyppi on 32-bittinen double-tarkkuuksinen desimaaliluku. Tämä aiheuttaa lisää työtä, jos sovelluksessa halutaan käsitellä lukuja tarkemmin, bittitasolla. Ongelma ratkaistiin Buffer-kirjastolla, jonka avulla lukuja muutettiin JavaScriptin Number-tyypistä tavuiksi ja takaisin.

React Native -sovelluksessa ohjelmakoodia ajetaan käyttöjärjestelmän JavaScriptCoressa, jonka toiminnassa on pieniä käyttöjärjestelmä- ja versiokohtaisia eroja. Tämän lisäksi React Native on uusi alusta, jota parhaillaan kehitetään nopeasti. Siksi sen ominaisuudet eivät välttämättä ole täysin viimeisteltyjä ja virheettömiä. Kolmannen muuttujan tuo kolmansien osapuolten kirjastot, joiden laatua täytyy arvioida aina kriittisesti. Opinnäytteen kehityksen aikana suurin osa toiminnoista toimi molemmilla alustoilla ilman eri toimenpiteitä. Jotkin rajapinnat olivat kuitenkin erilaisia eri alustoilla sekä joidenkin kirjastojen pohjalla oleva natiivikoodi ei välttämättä ole virheetöntä. Näistä johtuen sovellus välillä kaatui Androidilla, vaikka sama koodi toimi oikein iOS:lla.

Sovelluksen tärkeimmät ominaisuudet toteutettiin opinnäytetyön puitteissa. Sovelluksella voi skannata BLE-laitteita ja yhdistää niihin, siirtää tietoa määriteltyä protokollaa noudattaen sekä visualisoida saatua dataa. Myös GPS-sijaintiin perustuvan reitin keräys ja kartalle piirtäminen sekä tietojen vienti taulukkomuodossa toteutettiin.

Sovelluksen kehitystä jatketaan yhdessä tilaajan kanssa, ja kesän tullessa päästään keräämään oikeaa dataa laudalta. Kehityskohteita ovat mm. laudan ajoprofiilin kustomointi ja tiedon visualisoinnin parantaminen. Tulevaisuudessa sovellusta voidaan kehittää ajankohtaisten tarpeiden mukaan, jopa myytäväksi sovellukseksi. Viimeistään laajemman käyttäjäjoukon tapauksessa sovelluksen alustariippumattomuudesta on hyötyä laajemman käyttäjäjoukon muodossa.

**LÄHTEET**


Aboukhadijeh, Feross 2016: *buffer-repository Githubissa*. Luettu 29.12.2016.
https://github.com/feross/buffer

Abramov, Dan: *Redux*. Luettu 30.12.2016.
http://redux.js.org/

Airbnb 2017: *React-native-maps* v0.12.4. Luettu 9.2.2017.
https://github.com/airbnb/react-native-maps

Apple 2016: *Energy Efficiency Guide for iOS Apps. Fundamental Concepts*. Luettu 7.2.2017.
https://developer.apple.com/library/content/documentation/Performance/Conceptual/EnergyGuide-iOS/FundamentalConcepts.html#//apple_ref/doc/uid/TP40015243-CH4-SW1

Apple 2017: *iOS Human Interface Guidelines*. Luettu 6.3.2017.
https://developer.apple.com/ios/human-interface-guidelines/

Basics of UART Communication 2016. Circuit Basics. Luettu 28.2.2017.
http://www.circuitbasics.com/basics-uart-communication/

Bluetooth SIG 2016: *Bluetooth Low Energy*. Luettu 21.11.2016.
https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/low-energy

Boosted Boards 2015: *Boosted App v1.0 Launches*. Luettu 21.11.2016.
https://boostedboards.com/boosted-app/

Ecma International 2015: *ECMAScript 2015 Language Specification.* Luettu 9.2.2017.
http://www.ecma-international.org/ecma-262/6.0

Evolve Skateboards 2015: *Evolve Skateboards*. Luettu 21.11.2016.
https://itunes.apple.com/au/app/evolve-skateboards/id941480011?mt=8&ign-mpt=uo%3D4

Facebook 2017: Flux. Luettu 8.3.2017.
https://facebook.github.io/flux/

Google 2017: *Material Design*. Luettu 6.3.2017.
https://material.io/guidelines/components/bottom-navigation.html

Heydon, R., Hunn, N. 2016: *Bluetooth low energy*. Luettu 21.11.2016.
https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=227336

Ionic 2016: *Ionic-sovelluskehys*. Luettu 20.12.2016.
http://ionicframework.com/

Josefsson, S. 2006: *RFC 4648-standardiehdotus*. Luettu 29.12.2016.
https://tools.ietf.org/html/rfc4648#section-4

Microsoft 2017: *CodePush.* Luettu 24.2.2017.
https://microsoft.github.io/code-push/

Node.js Foundation 2017: *Buffer*. Luettu 24.2.2017.
https://nodejs.org/dist/v7.6.0/docs/api/buffer.html

npm 2016: *Hakutulossivu hakusanalle "react native"*. Luettu 29.12.2016.
https://www.npmjs.com/search?q=react%20native&page=2&ranking=optimal

NPMCompare 2017: *Comparing apphub vs. react-native-code-push.* Luettu 24.2.2017.
https://npmcompare.com/compare/apphub,react-native-code-push

Occhino, Tom 2015: *React Native: Bringing modern web techniques to mobile.* Luettu 20.12.2016.
https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/

Owens Josh 2016: *Mobile Frameworks*. Luettu 20.12.2016.
http://stateofjs.com/2016/mobile/

Polidea 2017: *React Native BLE PLX*. Luettu 31.1.2017.
https://github.com/Polidea/react-native-ble-plx

React 2016. Facebook. Luettu 20.12.2016.
https://facebook.github.io/react/

React Native 2016. Facebook. Luettu 20.12.2016.
https://facebook.github.io/react-native/

Witte, D., Weitershausen, P. 2015: *React Native for Android: How we built the first cross-platform React Native app.* Luettu 20.12.2016.
https://code.facebook.com/posts/1189117404435352

Xamarin 2016: *Xamarin-sovelluskehys*. Luettu 20.12.2016.
https://www.xamarin.com/