Ohjelmistoratkaisu sähköajoneuvoon

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeen ammattikorkeakoulu, Tieto- ja viestintätekniikka

Riihimäki, kevät 2019

Henri Pirinen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka
Riihimäki

Tekijä Henri Pirinen **Vuosi** 2019

Työn nimi Ohjelmistoratkaisu sähköajoneuvoon

Työn ohjaaja/t Timo Karppinen

TIIVISTELMÄ

Opinnäytetyön tavoitteena on tuottaa ohjelmistoratkaisu opinnäytetyön tilaajan itserakentamaan sähköajoneuvoon. Ohjelmiston avulla on mahdollista hallita ajoneuvon akkuihin ja moottoriin liittyviä laitteita sekä tuottaa reaaliaikaista mittausdataa.

Ajoneuvoon tuotetaan rajapinta, joka mahdollistaa laitteiden kommunikoinnin keskenään. Rajapinnan kautta voidaan ohjata ja lukea laitteiden mittauksia Web-pohjaisen käyttöliittymän avulla. Julkiseen verkkoon tehdään toinen rajapinta, jolla on MQTT-yhteys ajoneuvon rajapintaan. Tämä rajapinta on yhteydessä tietokantaan, jonne kirjataan mittaustulokset. Sen kautta voidaan myös asettaa ajoneuvon lämmitin, ON/OFF tilaan.

Ajoneuvoon on valmistettu kolme erilaista Arduino-pohjaista korttia. Driver-kortin tehtävä on ohjata moottorin pyörityssuuntaa ja vakionopeudensäädintä, asettaa ajoneuvo lataustilaan ja ohjata lämmitintä. Controller-kortin tehtävä on välittää jännite- ja lämpötilamittaukset ajoneuvon rajapintaan ja asettaa akkukennoryhmät jännitteentasaustilaan. Thermocouple-kortin tehtävä on mitata ajoneuvon moottorin lämpötilaa ja lähettää ne ajoneuvon rajapintaan.

Opinnäytetyön tilaaja on Seppo Vihavainen.

Avainsanat Esineiden Internet, API, Node.js, React, MQTT, Arduino

Sivut 65 sivua, joista liitteitä 10 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication technology
Riihimäki

Author Henri Pirinen **Year** 2019

Subject Software solution for electric vehicle

Supervisors Timo Karppinen

ABSTRACT

The aim of the thesis was to produce a software solution for a self-made electric vehicle which is constructed by the commissioner of the thesis. The software solution makes it possible to control the batteries and engine related devices of the vehicle and it produces real-time measurement data.

An API (application programming interface) was created for the vehicle that allowed the devices to communicate with each other. The API can be used to control and read device measurements through a Web-based user interface. Another API was made to the public network, with a MQTT connection to the vehicle API. This API was connected to a database where the measurement results were recorded. It can also be used to set the vehicle's heater to an ON / OFF state.

Three different Arduino-based cards were produced for the vehicle. A Driver card to control engine rotation and cruise control, to turn the vehicle in to a charging mode and to control the heater. The Controller card that transmits voltage and temperature measurements to the vehicle’s API and turns the cell groups into a voltage balancing mode. The Thermocouple card measures the vehicle's engine temperature and transmits the measurements to the API.

The commissioner of the thesis is Seppo Vihavainen.

Keywords Internet of Things, API, Node.js, React, MQTT, Arduino

Pages 65 pages including appendices 10 pages

SISÄLLYS

1 JOHDANTO ... 1

2 LAITEMÄÄRITYKSET ... 2

2.1 Palvelimet ... 2
2.1.1 Laitteet ... 2
2.1.2 Käyttöjärjestelmät ... 2
2.2 Päätelaite ... 3
2.3 Mikro-ohjaimet (Mittaus- ja hallintalaitteet) ... 3
2.4 Verkkolaitteet ... 3
2.5 Vaihtosuuntaaja ... 4

3 OHJELMISTOMÄÄRITYKSET ... 4

3.1 Käyttöliittymä ... 4
3.2 Rajapinnat ... 5
3.2.1 Ohjelmiston ylläpito ... 6
3.3 Verkon rakenne ja toiminta ... 6

4 KEHITYSTYÖKALUT JA TEKNIIKAT ... 7

4.1 Arduino ... 7
4.2 NPM ... 8
4.3 Node.js ... 9
4.4 React.js ... 10
4.4.1 React.JS projektin luominen ja tärkeät komennot ... 10
4.4.2 Komponentit ja ominaisuudet ... 11
4.4.3 State ja elinkaari ... 11
4.4.4 VirtualDOM ... 14
4.4.5 JSX ... 15
4.5 Bash ... 16
4.6 Redis ... 19
4.6.1 Rediksen käyttö Nodessa ... 20
4.7 MQTT ... 20

5 TOTEUTUS ... 21

5.1 Versionhallinta ... 21
5.1.1 Git ... 21
5.1.2 GitHub ... 21
5.2 Lisensointi ... 22
5.3 Järjestelmäkuvaus ... 22
5.4 Paikallinen palvelin ... 23
5.4.1 Node.js palveluksi (Service) ... 23
5.4.2 Sarjaliikenne rajapinnan ja laitteiden välillä ... 24
5.4.3 WebSocket ... 26
5.4.4 Ohjelmiston päivittäminen ... 27
5.4.5 Vaihtosuuntaajan ohjauksen integrointi ... 28

5.4.6 Paikallinen WebSocket-rajapinta 29
5.5 Etäpalvelin 32
5.5.1 Etäpalvelimen tietokannat 32
5.5.1 Tunnistautuminen rajapintaan 33
5.5.2 Etäpalvelimen REST-rajapinta 34
5.6 TLS-sertifikaatti 35
5.7 MQTT 37
5.8 Käyttöliittymä 40
5.8.1 Komponentit 40
5.8.2 Rakenne 44
5.8.3 WebSocket 45
5.9 Controller-kortti 46
5.10 Driver-kortti 48
5.11 Thermocouple-kortti 49

6 YHTEENVETO JA KEHITETTÄVÄÄ 49

LÄHTEET 51

Liitteet

Liite 1 Käyttöliittymän ohjeet

1 JOHDANTO

Tämän opinnäytetyön tavoitteena on toteuttaa asiakkaan omarakenteiseen sähköautoon ohjelmistokokonaisuus, jonka avulla on mahdollista seurata ja ohjata ajoneuvon laitteiden tilaa.

Asiakas on valmistanut ajoneuvoon kolme erilaista Arduino-pohjaista korttia. Driver-kortti, jonka tehtävä on ohjata moottorin pyörityssuuntaa ja vakionopeudensäädintä, asettaa ajoneuvo lataustilaan ja ohjata lämmitintä. Controller-kortti, joka välittää jännite- ja lämpötilamittaukset rajapintaan ja asettaa kennoryhmät jännitteentasaustilaan. Thermocouple-kortti, joka mittaa ajoneuvon moottorin lämpötilaa. Ajoneuvoon tuotetaan rajapinta Node.js:llä, joka mahdollistaa laitteiden keskustelun keskenään. Rajapinnan kautta voidaan ohjata ja lukea laitteiden mittauksia, päivittää laitteiden ohjelmistot ja konfiguroida järjestelmän asetuksia. Ajoneuvoon on tuotettu React.js:llä käyttöliittymä, jolla voidaan hallita rajapintaa ja siihen liittyviä laitteita. Ajoneuvo on yhteydessä julkisessa verkossa olevaan etäpalvelimeen MQTT-yhteyden avulla. Etäpalvelin vastaanottaa ajoneuvosta mittaustuloksia ja varastoi ne tietokantaan. Etäpalvelimella on React.js:llä tuotettu käyttöliittymä, jonka kautta voidaan lukea mittaustuloksia tietokannasta ja asettaa ajoneuvon lämmitin ON/OFF-tilaan.

Opinnäytetyö koostuu suunnitteluosuudesta (kappaleet 2–3), jossa selostetaan mitä laitteita ja ohjelmistoja opinnäytteessä hyödynnetään ja miksi ne on valittu tähän opinnäytetyöhön. Kehitystyökaluista on kirjoitettu kappaleessa 5, jossa on kerrottu tarkemmin, mitä kullakin työkalulla tehdään ja miten niitä käytetään. Toteutusosuudessa (kappale 5) on selostettu miten opinnäytetyön eri ohjelmistokomponentit toimivat. Liitteessä 1 on esitetty käyttöliittymän käyttöohjeet.

Opinnäytetyössä kirjoitetut ohjelmat on julkaistu MIT-lisenssillä GitHubissa. Opinnäytetyössä kirjoitettujen ohjelmien kuvaukset ja dokumentaatiot perustuvat jokaisen ohjelman julkaisu versioon 1.0.0. Tähän opinnäytetyöhön liittyvät ohjelmavarastot löytyvät Henri Pirisen GitHubista (Pirinen, Henri Pirinen's repositories, 2019).

Ajoneuvon elektroniikan rakentaa opinnäytetyön tilaaja, Seppo Vihavainen. Ajoneuvo tulee tilaajan omaan käyttöön.

2 LAITEMÄÄRITYKSET

2.1 Palvelimet

2.1.1 Laitteet

Vaatimuksina ajoneuvon palvelinlaitteelle on pieni koko ja vähävirtaisuus. Järjestelmään valmistetut mittaus- ja hallintalaitteet yhdistetään palvelinlaitteeseen USB:llä, joten USB 2.0 portteja tarvitaan vähintään 4 kappaletta. Palvelinlaitteeksi valittiin Raspberry Pi 3 B+, koska se on helposti saatavilla ja sitä valmistetaan vuoteen 2023 saakka (raspberrypi.org), jonka ansiosta on helpompi saada tarvittaessa uusi laite rikkoutuneen tilalle. Raspberry Pi 3 B+ voidaan tulevaisuudessa korvata esimerkiksi Raspberry Pi Zero W-laitteella, mutta tällöin mittaus- ja hallintalaitteet pitää yhdistää GPIO-pinnien kautta. Koska mittaus- ja hallintalaitteiden sarjaliikenne toimii 0-5 voltin alueella ja Raspberryn GPIO-pinnit 0-3,3 voltin alueella, pitäisi välissä tehdä jännitteenmuunnos. Lisäksi mittaus- ja hallintalaitteille pitäisi tuoda käyttöjännite erilliseen pinniin. Mittaus- ja hallintalaitteet voidaan kytkeä Raspberry Pi 3 B+:ssa USB-portteihin, jotka toimivat 5 voltin alueella. Tällöin mittaus- ja hallintalaitteet saavat käyttöjännitteen USB:n kautta ja tiedonsiirto voidaan suorittaa USB-väylän kautta.

Etäpalvelin laitteeksi valittiin Digital Ocean palvelun Droplet-virtuaalikone. Virtuaalikoneeksi valittiin halvin vaihtoehto, jossa on 1GB keskusmuistia, 1 virtuaaliprosessori, 25GB SSD-asema ja tiedonsiirtoa maksimissaan 1TB/kuukausi. Virtuaalikone kustantaa asiakkaalle $5/kuukausi (Pricing on DigitalOcean - Cloud virtual machine & storage pricing, n.d). Tarvittaessa virtuaalikoneen SSD-asemaa, prosessorien ja keskusmuistin määrää voidaan kasvattaa lisämaksusta.

Kun opinnäytetyössä puhutaan palvelimesta, sillä viitataan tässä kappaleessa mainittuihin palvelinlaitteisiin, esimerkiksi Raspberryn.

2.1.2 Käyttöjärjestelmät

Ajoneuvon palvelinlaitteessa käytetään Debianiin perustuvaa Raspbian-käyttöjärjestelmää, joka on suunniteltu käytettäväksi Raspberry Pi:llä. Raspbianista on saatavilla kaksi eri versiota, desktop ja lite. Lite-versiossa ei ole graafista käyttöliittymää, eli sitä voidaan käyttää vain komentoliittymän (engl. Command line interface, CLI) kautta ja sen mukana ei tule esiasennettuja kehitystyökaluja (HawaiianPi, 2018). Työssä käytetään Lite-versiota, koska graafiselle käyttöliittymälle ei ole tarvetta. Etäpalvelimella käytetään Ubuntu 18.04.1 LTS (Long Term Support) käyttöjärjestelmää. Etäpalvelimen käyttöjärjestelmä toimii myös vain komentoliittymän kautta. Käyttöjärjestelmää voidaan tarvittaessa vaihtaa kummallakin laitteella, kunhan uusi käyttöjärjestelmä tukee tässä työssä käytettyjä ohjelmistoversioita.

2.2 Päätelaite

Ajoneuvon hallintaan tarkoitettu käyttöliittymä tehdään Web-pohjaiseksi. Tämä mahdollistaa minkä tahansa päätelaitteen käytön, kunhan sen selain tukee ECMAScript versiota 5. Jokainen nykyaikainen selain tukee ECMAScript version 5:ä (Kangax, n.d).

Päätelaitteena ajoneuvossa käytetään Android-taulutietokonetta. Käyttöjärjestelmällä ei ole merkitystä, kunhan se tukee jotakin modernia web-selainta.

2.3 Mikro-ohjaimet (Mittaus- ja hallintalaitteet)

Akkujen jännitteet ja lämpötilat mitataan PIC12F683 mikro-ohjaimella. Controller-kortilla mitatut arvot kerätään ATmega328P mikro-ohjaimelle, joka pakkaa kaikki mittausarvot yhteen JSON-datapakettiin ja välittää sen Raspberry Pi:llä toimivalle rajapinnalle USB-väylän kautta. ATmega328P ohjaa myös akkujen jännitteentasaajia, kun ajoneuvo on lataustilassa. Jännitteentasaajien ohjaamisen käytetään releitä, jotka kytketään päälle ja pois ATmega328P mikro-ohjaimen toimesta. Tieto siitä milloin jännitteentasaajat kytketään päälle tai pois, saadaan analysoimalla PIC-mikro-ohjaimelta vastaanotetusta datapaketista, tai komentona rajapinnasta. Controller-kortteja on tässä työssä käytössä kaksi kappaletta.

Driver-kortti käyttää ATmega328P mikro-ohjainta releiden tilan vaihteluun. Releillä ohjataan lämmittimen ja vaihtosuuntaajan tilaa. Vaihtosuuntaajan kautta ohjataan moottorinpyörimissuuntaa ja vakionopeudensäädintä. Driver-korttia ohjataan rajapinnan kautta.

Thermocouple-kortti käyttää Atmega328P mikro-ohjainta kahden termoparin mittaustulosten lukemiseen ja raportointiin. Termoparit on liitetty MAX31855-moduuliin, jonka kautta voidaan lukea mittaukset celsiusasteina. Mittaustulokset raportoidaan rajapintaan JSON-viestinä.

PIC-kortit ottavat käyttöjännitteen niistä akkukennoista, joihin ne on kytketty. Controller-, Driver-, ja Thermocouple-kortit ottavat käyttöjännitteen USB-johtojen kautta, jotka on kytketty Raspberry Pi:n USB-portteihin. Myös tiedonsiirto rajapintaan suoritetaan USB:n kautta.

Kun opinnäytetyössä puhutaan mittaus- ja hallintalaitteista, viitataan sillä kaikkiin tässä kappaleessa mainittuihin laitteisiin.

2.4 Verkkolaitteet

Ajoneuvossa käytetään TP Link M7300 4G modeemia. Laitteeseen saadaan virtaa Micro-USB-liittimen kautta ja siinä on sisäänrakennettu 2000mAh akku, joka kestää valmistajan mukaan noin 8 tuntia aktiivista käyttöä. Laitteessa on myös Micro SD-paikka, jossa voi käyttää maksimissaan 32GB:n Micro SD-korttia. (4G LTE Mobile Wi-Fi M7300, n.d)

Mobiililiittymäksi valittiin DNA Laitenetti, jossa on rajaton tiedonsiirto 128 kbit/s nopeudella (DNA Laitenetti, n.d). Liittymään saadaan käyttöön julkinen IP-osoite antamalla APN-osoitteeksi julkinen.dna.fi modeemin APN-asetuksissa (Mokkuloiden ja päätelaitteiden APN-asetukset, n.d).

2.5 Vaihtosuuntaaja

Vaihtosuuntaajana käytetään Johannes Hüebnerin kehittämää vaihtosuuntaaja rakennussarjaa (Hüebner, Inverter Kit Features, n.d).

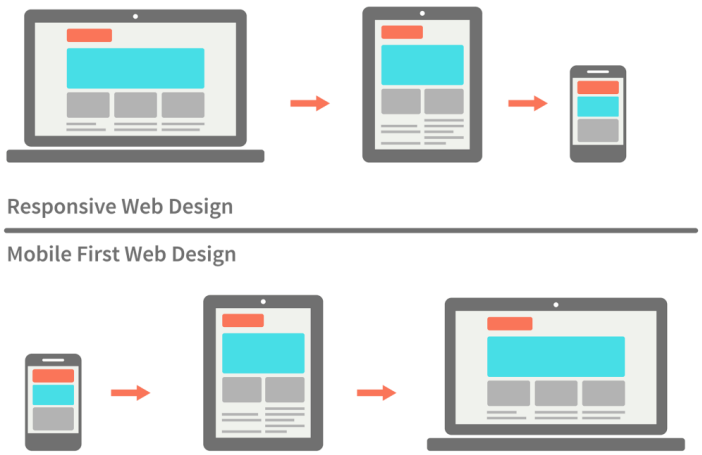
Vaihtosuuntaajassa on Web-pohjainen käyttöliittymä, josta voidaan asettaa ja seurata vaihtosuuntaajan arvoja sekä päivittää vaihtosuuntaajan ohjelmiston. Vaihtosuuntaajan käyttöliittymän toiminnallisuus integroidaan omaan käyttöliittymään, jotta käyttäjä ei tarvitse vaihdella näkymää kahden eri käyttöliittymän välillä.

3 OHJELMISTOMÄÄRITYKSET

3.1 Käyttöliittymä

Käyttöliittymä tehdään Web-pohjaiseksi, jotta sitä voidaan käyttää mahdollisimman monessa erityyppisessä laitteessa, joissa voi käyttää Web-selainta. Koska käyttöliittymä on tehty Web-pohjaiseksi, ei ole väliä käyttääkö asiakas Unix-, Linux- tai Windows-pohjaista käyttöjärjestelmää. Käyttöliittymä ei ole sidoksissa laitteen käyttöjärjestelmän versioon, toisin kuin natiiveissa ohjelmissa.

Koska ajoneuvossa käytetään päätelaitteena taulutietokonetta, suunnitellaan käyttöliittymä Mobile First periaatteella. Mobile First tarkoittaa sitä, että käyttöliittymä suunnitellaan siten että sitä käytetään pääasiassa mobiililaitteella, esimerkiksi älypuhelimella tai taulutietokoneella.



Kuva 1. Käyttöliittymä suunnitellaan Mobile First Design periaatteella. (Gonzalo, 2017)

Käyttöliittymän rakentamiseen käytetään React.js, Material-UI ja React-vis JavaScript-kirjastoja. React.js on kirjasto käyttöliittymän rakentamista varten. Material-UI on React.js-kirjasto, jonka avulla Web-pohjaisessa käyttöliittymässä voidaan käyttää Google Material Desing komponentteja. Käyttöliittymä saadaan Material-UI kirjaston avulla näyttämään natiivilta Android-sovellukselta ja se toimii alusta riippumattomasti. React-vis on Uberin kehittämä React.js-kirjasto datan visualisointia varten.

Datan siirto paikallisen rajapinnan ja käyttöliittymän välillä toteutetaan WebSocketilla. WebSocket-kirjastona käytetään Socket.iota. WebSocketin avulla käyttöliittymän ja rajapinnan välille saadaan jatkuva TCP-yhteys, jonka avulla rajapinta ja käyttöliittymä voivat kommunikoida keskenään reaaliajassa. Jos WebSocket yhteys katkeaa, osaa socket.io automaattisesti yhdistyä takaisin WebSocket-palvelimeen, kun se tulee takaisin saataville. Etärajapinnan ja etäkäyttöliittymän kommunikointi tehdään REST:llä (Representational State Transfer), jossa käyttöliittymä tekee pyyntöjä rajapintaan.

Ajoneuvon akkutilasta kerätty jännite- ja lämpötilamittaukset otetaan vastaan rajapinnasta WebSocketin kautta JSON-muodossa. Rajapinnasta saadut mittaustulokset käsitellään käyttöliittymässä. Mittaustulokset voidaan esittää reaaliajassa tai valitulta aikaväliltä graafisesti React-vis-kirjaston avulla.

Vaihtosuuntaajan käyttöliittymän ominaisuudet, eli datan lukeminen, arvojen asettaminen ja ohjelmiston päivittäminen integroidaan omaan käyttöliittymään, jotta käyttäjän ei tarvitse vaihdella näkymää kahden käyttöliittymän välillä.

Atmel-mikro-ohjainten ja rajapinnan ohjelmiston päivittäminen voidaan suorittaa käyttöliittymän kautta.

Käyttöliittymän kautta voidaan hakea myös paikalliset säätiedot, ja karttatietoja kolmannen osapuolen rajapinnan avulla.

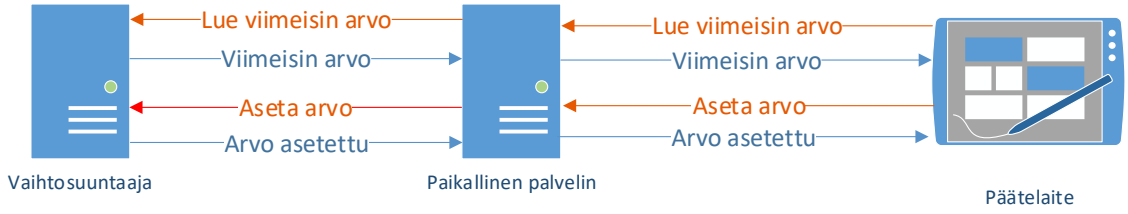
3.2 Rajapinnat

Palvelimille rakennetaan rajapinnat Node.js ympäristössä. Etärajapinnassa käytetään Express.js sovelluskehystä (engl. framework) ja paikallisessa rajapinnassa pääasiassa Socket.io WebSocket-kirjastoa. Express.js on suunniteltu Web-ohjelmien ja rajapintojen rakentamiseen. WebSocketin avulla rajapinnan ja käyttöliittymän välille muodostetaan jatkuva TCP-yhteys.

Paikallisen- ja etärajapinnan välinen M2M-yhteys (Machine-to-Machine) toteutetaan MQTT (Message Queuing Telemetry Transport) protokollalla. MQTT on publish-subscribe-tyyppinen viestintäprotokolla, joka toimii TCP/IP-tasolla (MQTT Frequently Asked Questions, n.d). MQTT sopii verkkoihin, joissa rajallinen verkon nopeus, tästä syystä se sopii erityisesti IoT-laitteisiin (MQTT Frequently Asked Questions, n.d).

Paikallisen palvelimen rajapinnan tehtävä on vastaanottaa ja välittää viestejä käyttöliittymän, mittaus- ja hallintalaitteiden ja etärajapinnan välillä.

Vaihtosuuntaajassa REST-rajapinta, jonka kautta voidaan asettaa ja lukea vaihtosuuntaajan parametrien arvoja. Paikallinen rajapinta ottaa vaihtosuuntaaja komennot vastaan käyttöliittymästä WebSocketilla ja välittää komennon vaihtosuuntaajalle REST:llä. Kun vaihtosuuntaajan vastaa REST-pyyntöön, välitetään vastaus käyttöliittymään WebSocketin kautta. Kuvassa 2 on esitetty liikenne käyttöliittymän (Päätelaite) ja vaihtosuuntaajan välillä.



Kuva 2. Vaihtosuuntaajan arvojen lukeminen ja asettaminen.

Etäpalvelimella ylläpidetään PostgreSQL-relaatiotietokantaa, johon kerätään ajoneuvon akuista jännite- ja lämpötilamittauksia. Etäpalvelimelle tehdään REST-rajapinta, jonka kautta voidaan hakea mittaustuloksia päivämäärän mukaan. Etärajapintaan kirjaudutaan Web-käyttöliittymän kautta. Kun kirjautuminen etärajapintaan onnistuu, käyttäjälle lähetetään väliaikainen API-avain, jonka avulla voidaan tehdä pyyntöjä tietokantaan.

Kun opinnäytetyössä puhutaan rajapinnasta, viitataan sillä tässä kappaleessa kuvailtuun Node.js ohjelmaan. Jos puhutaan julkisessa verkossa sijaitsevasta kolmannen osapuolen rajapinnasta, mainitaan siitä kappaleessa erikseen.

3.2.1 Ohjelmiston ylläpito

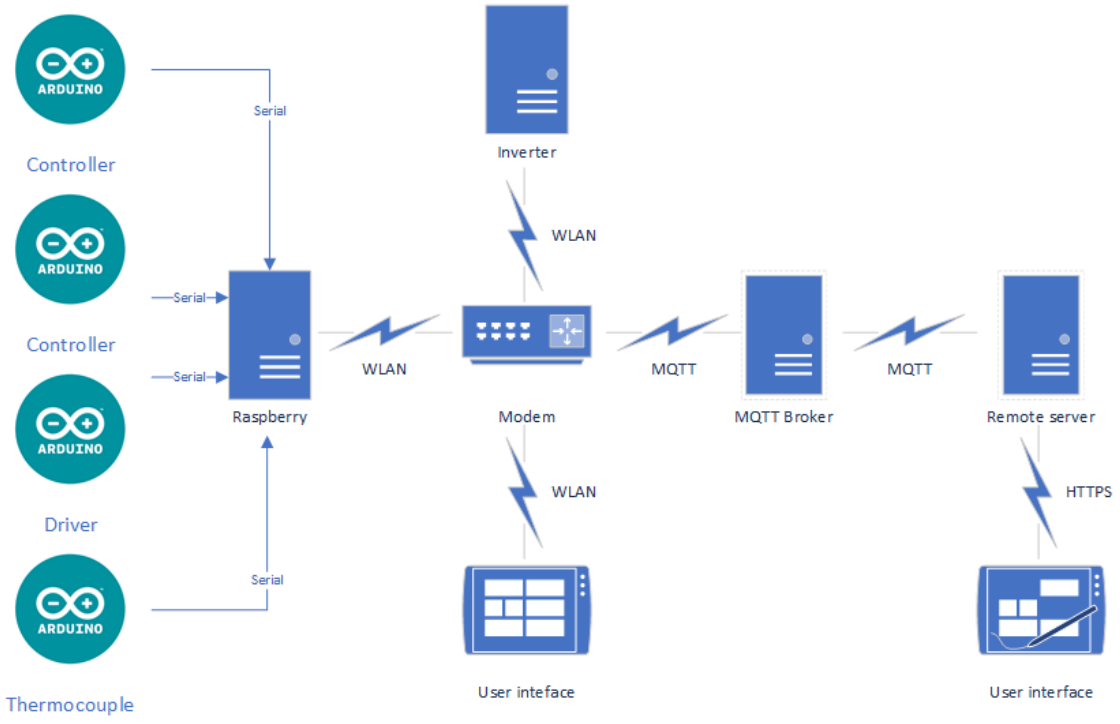
Driver-, Thermocouple- ja Controller-kortit sekä rajapinnan ohjelmat voidaan päivittää Web-käyttöliittymän kautta. Käyttöliittymästä voidaan valita yksi tai useampi laite päivitettäväksi. Rajapinta käynnistää Bash-skriptin, joka hakee päivitettävien ohjelmien uusimmat versiot GitHub-ohjelmavarastosta (engl. repository). Kun uusi versio Driver-, Thermocouple-, ja Controller-korttien ohjelmista on haettu, pitää ne koota (engl. compile). Rajapinnan koodille ei tarvitse tehdä toimenpiteitä sen jälkeen, kun se on ladattu. Kun kaikki päivitykset ovat valmiit, rajapinta käynnistetään uudelleen.

3.3 **Verkon rakenne ja toiminta**

Controller-, Driver-, ja Thermocouple-kortit on kytketty Raspberry Pi:n USB-portteihin. Vaihtosuuntaaja, päätelaite ja Raspberry ovat samassa langattomassa verkossa. Verkon langattomana reitittimenä käytetään TP-Link M7300 4G-modeemia. Mobiililiittymäksi on valittu sellainen liittymä, jossa on mahdollista käyttää julkista IP-osoitetta.

Yhteys käyttöliittymän ja paikallisen rajapinnan välillä toteutetaan WebSocketilla. Etäpalvelimen rajapinnan ja paikallisen palvelimen rajapinnan yhteys toteutetaan MQTT-protokollalla, joka on suojattu TLS:llä (Transport Security Layer). MQTT brokerina käytetään kolmannen osapuolen tarjoamaa palvelua.

Etäpalvelin käyttöliittymä on suojattu TLS:llä. Etäpalvelimen kautta päästään Web-pohjaiseen käyttöliittymään, josta voidaan lukea ajoneuvon akuston mittaustuloksia sekä asettaa etänä ajoneuvon lämmitin ON/OFF-tilaan. Etäyhteydessä käyttäjällä ei ole suoraa yhteyttä auton paikalliseen palvelimeen. Etäpalvelin ohjaa käyttäjän komennot paikalliselle palvelimelle. Etäpalvelin ei näytä autosta reaaliaikaista dataa ja sen kautta ei voi muuttaa vaihtosuuntaajan arvoja tai päivittää ajoneuvon ohjelmistoja. Kuvassa 3 on esitetty koko verkon rakenne.



Kuva 3. Verkon rakenne.

4 KEHITYSTYÖKALUT JA TEKNIIKAT

4.1 Arduino

Arduino on avoimeen lähdekoodin perustuva kehitysalusta elektroniikkaprojekteille. Arduinoon kuuluu ohjelmoitava mikro-ohjain piirilevyllä ja Arduino IDE (Integrated Development Enviroment) ohjelmisto, jolla voidaan kirjoittaa ja ohjelmoida Arduino mikro-ohjaimia. Arduino ei vaadi erillistä ohjelmointilaitetta, vaan se voidaan ohjelmoida suoraan tietokoneen USB-väylän kautta. (B_E_N, n.d)

Arduino kehitysalustoja on monia erilaisia ja niihin voi hankkia valmiita lisämoduuleita, joilla on mahdollista saada esimerkiksi WiFi-yhteys. Prosessorityypit vaihtelevat malleittain, yksinkertaisimmissa korteissa on AVR-arkkitehtuurin perustuvat prosessori ja monimutkaisimmissa alustoissa on ARM-arkkitehtuuriin perustuva prosessori. Tässä opinnäytetyössä käytetään Arduino Nano alustaa, jossa on Atmega328 AVR-mikro-ohjain (Arduino Nano, n.d). Kuvassa 4 on esitetty Arduino Nano kehitysalusta.

Kuva 4. Arduino Nano-kehitysalusta.

Arduino IDE:llä mikro-ohjain voidaan ohjelmoida C tai C++ kielellä. Jotkin C ja C++ ominaisuudet eivät kuitenkaan ole saatavilla. Kaikki C ja C++ toiminnot, jotka on tuettu avr-g++ kääntäjällä (engl. compiler), pitäisi toimia Arduinolla. Arduino IDE tekee kääntövaiheessa pieniä muutoksia ohjelmaan esimerkiksi lisäämällä #include osat ja funktioprototyypit, joten niitä ei ole pakko kirjoittaa ohjelmaan. (Arduino - FAQ, n.d)

4.2 NPM

NPM (Node Package Manager) on paketinhallintajärjestelmä JavaScript-ohjelmointikielelle. Työssä käytettävät JavaScript-moduulit asennetaan NPM:n kautta. NPM asennetaan oletuksen Node.js asennuksen yhteydessä. JavaScript-moduuleita voidaan selata osoitteessa https://www.npmjs.com/.

Kun käyttäjä on projekti kansion sisällä, annetaan komento **npm init.** Tämä komento kysyy tarvittavat tiedot **package.json**-tiedoston luomiseen, joka pitää sisällään mm. listan niistä NPM moduuleista, joita projektissa käytetään. Käytettävät moduulit voidaan asentaa kahdella tavalla. Kirjoittamalla moduulin nimi ja versio package.json-tiedostoon ja antamalla **npm install** komento, jolloin kaikki listatut paketit, joita ei vielä ole asennettu, asennetaan. Toinen vaihtoehto on antaa komento npm install <paketin nimi>, jolloin NPM lisää paketin automaattisesti package.json-tiedostoon ja asentaa paketin. Kuvassa 5 on esitetty Node.js projekti, jossa on käytössä express moduuli.

```
{
  "name": "npmesimerkki",
  "version": "1.0.0",
  "description": "NPM esimerkki",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Henri Pirinen",
  "license": "MIT",
  "dependencies": {
    "express": "^4.16.4"
  }
}
```

Kuva 5. Node.js projekti, jossa on käytössä express moduuli versio 4.16.4.

Moduuleita voidaan päivittää antamalla komento **npm update**. Tämä toimii, jos päivitettävään moduuliin ei ole tullut isoa päivitystä (engl. Major update). Esimerkiksi jos asennettu moduulin versio on 1.0.0, voidaan se päivittää versioon 1.x.x, mutta ei versioon 2.x.x. Iso päivitys voidaan asentaa poistamalla vanha versio ja sen jälkeen asentamalla uusin versio.

4.3 Node.js

Node.js on avoimeen lähdekoodiin perustuva palvelinohjelmisto. Se toimii asynkronisesti ja se on tapahtumapohjainen. Kun Node.js-ohjelma käynnistyy, se alustaa muuttujat ja funktiot, jonka jälkeen ohjelma aloittaa tapahtumien kuuntelun.

Node.js on suunniteltu skaalautuville verkkosovelluksille. Se toimii vain yhdellä prosessorin säikeellä, mutta siitä voidaan tehdä aliohjelmia muille prosessorin säikeille. (About Node.js, n.d) Node.js pohjautuu Googlen Chrome V8 JavaScript-moottoriin. Ohjelmointi kielenä Node.js ympäristössä käytetään JavaScriptiä, tai mitä tahansa muuta vaihtoehtoa, joka voidaan kääntää JavaScriptiksi, kuten TypeScript tai CoffeeScript. (Capan, n.d)

Node.js toimii Non-Blocking menetelmällä. Node voi esimerkiksi antaa lukuoperaation tiedostojärjestelmälle. Tämän jälkeen Node voi suorittaa muita operaatioita. Kun lukuoperaatio on valmis, Node saa vastauksen tiedostojärjestelmästä. Blocking menetelmällä operaatiot suoritetaan synkronisesti, Non-Blocking menetelmällä operaatiot suoritetaan asynkronisesti. (Blocking vs Non-Blocking, n.d.)

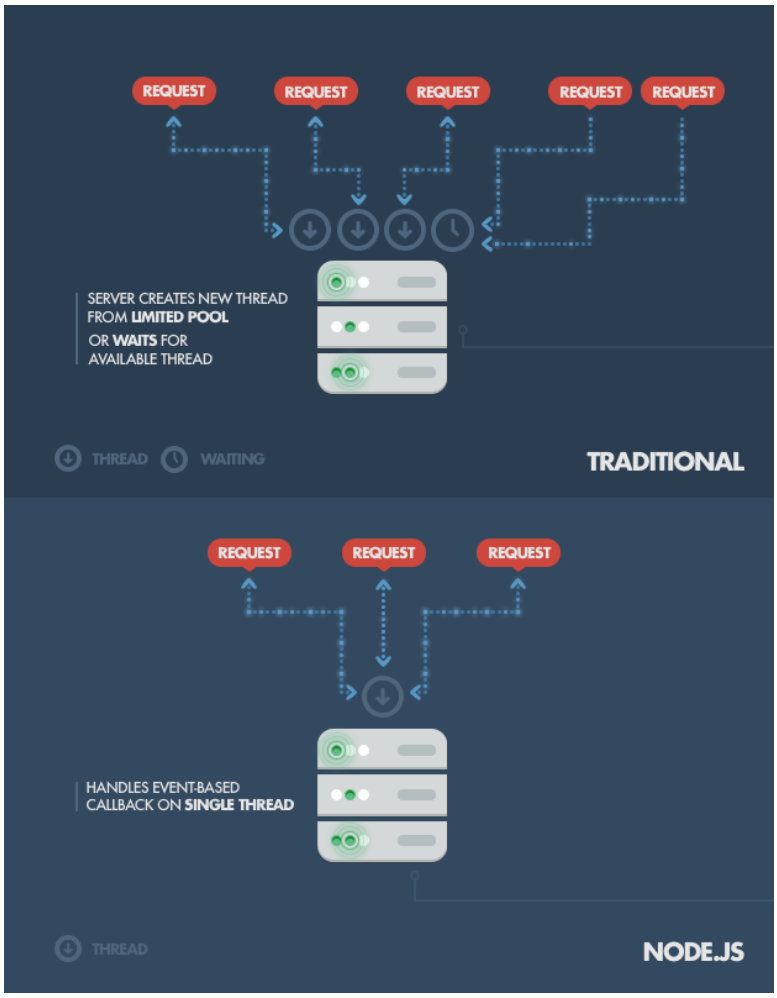
Node.js on erittäin hyvä ympäristö reaaliaikaisille Web-ohjelmille, jotka käyttävät esimerkiksi WebSockettia. WebSocketilla voidaan muodostaa kaksisuuntainen yhteys palvelimen ja käyttäjän välille, jossa data voi liikkua vapaasti molempiin suuntiin. Perinteisesti käyttäjä tekee pyynnön palvelimelle ja palvelin vastaa käyttäjälle. Node.js on suunniteltu käytettäväksi sellaisissa ympäristöissä, jossa pitää käsitellä suurta määrä yhteyksiä, mutta se ei ole sopiva sellaisiin ympäristöihin, joissa suoritetaan paljon prosessoritehoa vaativia operaatioita. (Capan, n.d)

Kuvassa 6 on esitetty Node.js palvelimen toiminta verrattuna perinteiseen palvelinmalliin esimerkiksi ASP:iin tai PHP:seen. PHP tai ASP vastaavat tiedostokyselyyn alla esitetyssä järjestyksessä:

1. Anna lukuoperaatio palvelin tiedostojärjestelmälle.
2. Odota kun palvelin avaa tiedoston ja lukee sen.
3. Palauttaa vastaus käyttäjälle.
4. Valmis suorittamaan seuraavan pyynnön.

Node.js vastaa samanlaiseen kyselyyn alla esitetyssä järjestyksessä:

1. Anna lukuoperaatio palvelimen tiedostojärjestelmälle.
2. Valmis suorittamaan seuraavan pyynnön.
3. Kun tiedostojärjestelmä on avannut ja lukenut tiedoston, palauta vastaus käyttäjälle.



Kuva 6. Node.js kyselyihin vastaaminen verrattuna perinteisen malliseen palvelinympäristöön, esimerkiksi ASP:iin tai PHP:seen. (Capan, n.d)

4.4 React.js

React.js on JavaScript-kirjasto, joka on suunniteltu Web-käyttöliittymien rakentamiseen. Reactia ylläpitää Facebook ja yhteisö, joka koostuu yrityksistä ja yksittäisistä kehittäjistä. React sopii Single-page applikaation (SPA) rakentamiseen. Monimutkaisissa applikaatioissa käytetään usein lisäkirjastoja, esimerkiksi tilan (engl. state) hallintaan tai reitittämiseen. (Suzdalnitski, 2018)

4.4.1 React.JS projektin luominen ja tärkeät komennot

Helppo tapa aloittaa React-projekti on asentaa create-react-app NPM-moduuli. Jotta Create React App toimii, pitää kehitysympäristössä olla asennettuna vähintään Node.js versio 6 ja npm versio 5.2. Create React App tekee projektirakenteen ja asentaa tarvittavat moduulit (Craete a New React App, n.d). Kuvassa 7 on esitetty React-applikaation rakennuskomento ja muut komennot, joita voidaan antaa React-projekti-kansion sisällä.

```
npx create-react-app webapp #Luo React app nimeltä webapp
npm start #Käynnistä react app
set HTTPS=true&&npm start #Käynnistä react app HTTPS:llä
npm run build #Rakenna tuotantoversio react app:stä
npm test #Käynnistä Jest, joka tarkistaa react appin
```

Kuva 7. Hyödyllisiä komentoja React kehitykseen.

Create React App käyttää Jest-sovelluskehystä applikaation testaamiseen. Jest toimii Nodella ja se tarkistaa applikaation toimivuuden aina kun tiedosto tallennetaan. Testit ajetaan Node ympäristössä, joten applikaatiota kannata välillä testata myös selaimella. (Running Tests, n.d)

4.4.2 Komponentit ja ominaisuudet

Komponentit ovat JavaScript-funktioita tai ECMAScript 6-luokkia, joita yleensä käytetään JSX-elementtien palauttamiseen. Ominaisuudet (engl. properties) ovat komponenttien argumentteja, Reactissa niistä käytetään lyhennettä props. Argumentteina voidaan antaa arvoja tai funktioita. Kuvassa 8 on esitetty kaksi tapaa luoda React-komponentti. Molemmat esimerkit toimivat samalla tavalla Reactissa, mutta luokalla saadaan käyttöön paikallinen state- ja elinkaarimetodit. (Components and Props, n.d)

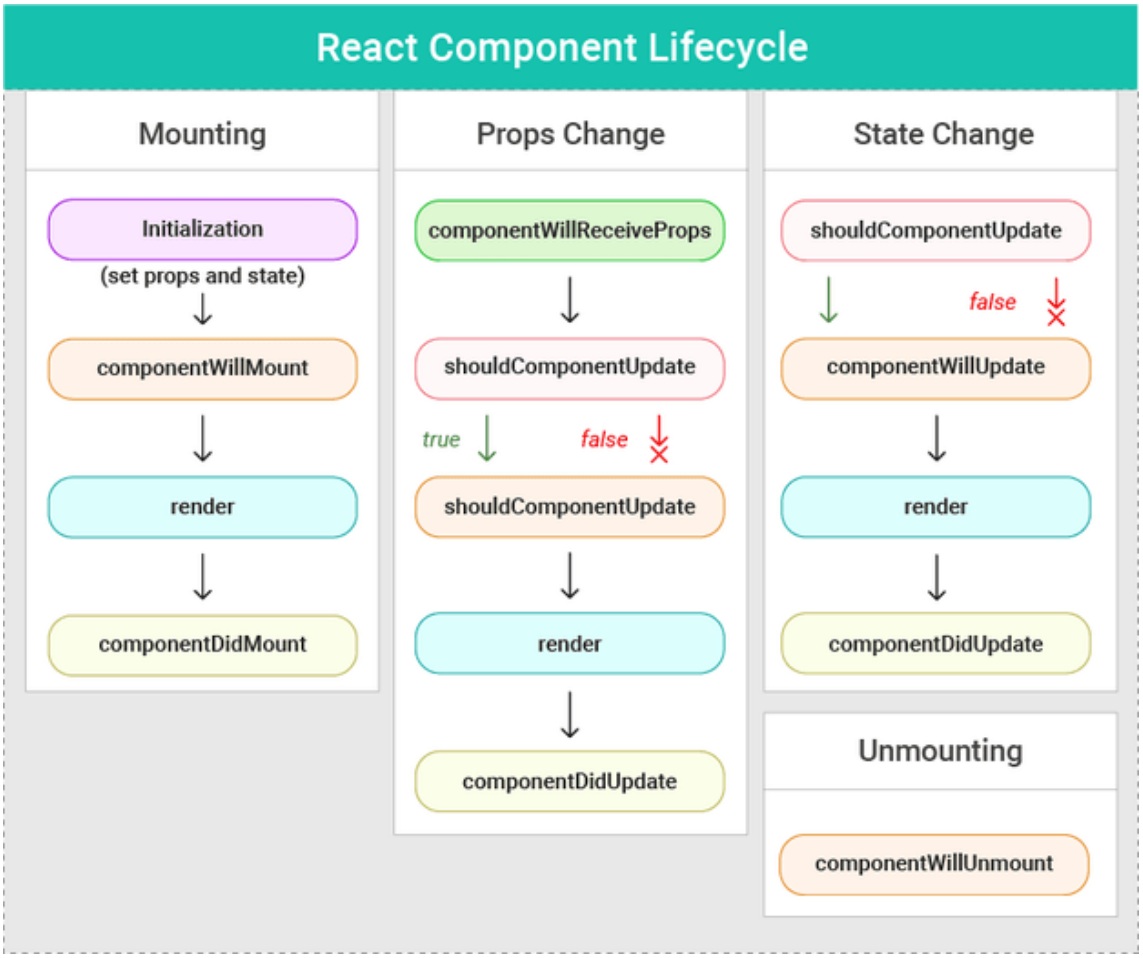
```
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
}

class Welcome extends React.Component {
    render() {
      return <h1>Hello, {this.props.name}</h1>;
    }
}
```

Kuva 8. React komponentti. (Components and Props, n.d)

4.4.3 State ja elinkaari

State on samankaltainen kuin props, mutta se on privaatti ja komponentti voi muokata sen tilaa. Käyttämällä JavaScript-luokkaa, saadaan React-komponenttiin state käytettäväksi. Samalla saadaan käyttöön elinkaarimetodit. (State and Lifecycle, n.d)

Elinkaarimetodeja kutsutaan, kun komponentti luodaan, siinä tapahtuu muutoksia propsissa tai statessa ja silloin kun komponenttia ei enää tarvita. Kuvassa 9 on esitetty Reactin elinkaarimetodit suoritusjärjestyksessä.



Kuva 9. React.js elinkaari metodit ja toiminta järjestys. (Kumar, 2018)

Kuvassa 10 on esitetty React-komponentti, joka tulostaa tekstin Hello World ja kellonajan, jota päivitetään joka sekunti. Kun komponenttia ei enää tarvita, komponentti suoritta unmounting-metodin, joka nolla sekuntilaskurin.

Muodostin (engl. constructor) on ensimmäinen osa komponenttia, joka ajetaan ennen mitään elinkaarimetodia. Muodostin ottaa aina argumenttina propsin ja super(props) pitää olla aina ensimmäisenä määritetty. Super-metodilla saadaan this avainsana käyttöön muodostimen sisällä. Super pitää antaa, vaikka this avainsanaa ei tarvita, koska ECMAScript 6-luokka vaati sen käyttöä, jos luokka on peritty (extends) (Khatri, 2016). Muodostinta tarvitaan vain silloin, jos komponentille pitää antaa state tai jos metodeja pitää sitoa (engl. bind). Toisin kuin muualla komponentissa, muodostimessa komponentin tila annetaan **this.state = {nimi:arvo, ... }** komennolla. Kuvan 10 esimerkissä tilaksi annetaan päivämäärä.

Kuvassa 10 componentDidMount-metodi ajetaan heti, kun komponentti on renderöity. Tämän metodin sisällä on asetettu laskuri, joka suorittaa tick-funktion sekunnin välein, niin kauan kun komponentti on käytössä. Kun komponenttia ei enää tarvita, ajetaan componentWillUnmount-metodi, joka poistaa sekuntilaskurin.

React-komponentin ainoa pakollinen osa on Render-metodi. Render-metodin kautta voidaan palauttaa pääohjelmaan React-elementtejä (JSX), taulukkoja (engl. array), fragmentteja, portaaleita, tekstiä, numeroita, totuusarvomuuttuja (engl. boolean) tai tyhjä (engl. null). Jos shouldComponentUpdate-metodi palauttaa epätoden, render metodia ei suoriteta. (React.Component, n.d)

Kuvassa 10 komponentti renderöidään selaimeen ReactDOM.render-metodilla. Kuvan esimerkissä renderöidään Clock-komponentti applikaation juureen.

```
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')
  );
```

Kuva 10. React-komponentti ja sen rendaaminen applikaation juureen (State and Lifecycle, n.d).

4.4.4 VirtualDOM

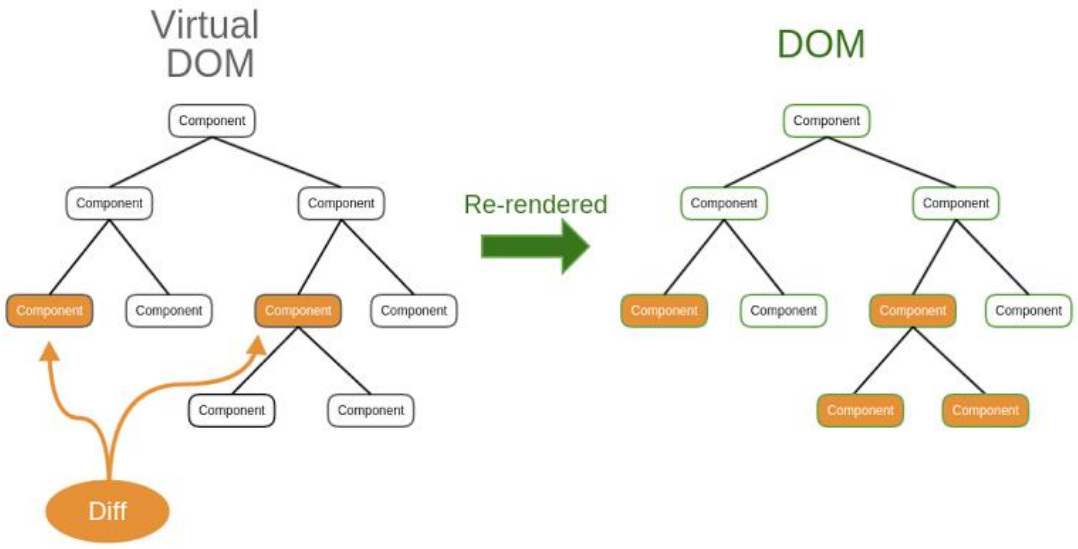
Verkkosivun DOMin (Document Object Model) päivittäminen on hidas prosessi koska siinä on monta vaihetta. Jos JavaScriptissä annetaan esimerkiksi seuraava komento:

```
document.getElementById('elementId').innerHTML = "Uusi teksti"
```

Selain suorittaa seuraavat prosessit:

1. Jäsentää HTML:ään.
2. Poistaa elementId:n alielementin.
3. Päivittää DOMin Uusi teksti arvolla.
4. Uudelleen laskee CSS:n pää- ja alielementille.
5. Päivittää jokaisen elementin koordinaatit selaimen ikkunassa.
6. Leikkaa (engl. traverse) render puun ja rasteroi selaimeen.

DOM:in päivittäminen vaati siis monta muutakin prosessia, kuin pelkästään DOM:n rakenteen päivittämisen. Yllä oleva prosessi suoritetaan joka kerta kun DOM:a pitää päivittää (Mishra, 2017). Tämä on huono silloin, kun sivua pitää päivittää jatkuvasti. Esimerkiksi jos sivulla on monta graafia, joita päivitetään reaaliajassa. React.js ei päivitä DOM:a suoraan, vaan sen sijaan se päivittää virtuaalista DOM:ia, joka on kopio selaimessa esitettävästä DOM:sta. Kuvassa 11 on esitetty Virtual DOM:n päivitysprosessi.



Kuva 11. Verkkosivun uudelleen renderöinti Reactissa. (virtual-dom-update.png, n.d)

Virtuaalinen DOM on JavaScript-olio kopio selaimessa esitetystä DOM:sta. React etsii VirtualDOMista eroavaisuudet esitettyyn DOM:iin Diffing Algoritmilla ja päivittää vain sen komponentin, josta on löydetty muutos sekä muutettavan komponentin alikomponentit. Tarvittaessa komponenttien turhaa uudelleen rendausta voidaan estää **shouldComponentUpdate**-elikaarimetodilla. Aina kun Reactissa kutsutaan **setState**-metodia, koko VirtualDOM rakennetaan uudestaan. Muuttuvien komponenttien etsintään käytetään BST (Breadth First Search) algoritmia. React pitää aina yllä kahta Virtual DOM:a, nykyistä, johon on tehty muutokset ja edellisellä päivitys kerralla ollutta VirtualDOM:a. (Mishra, 2017)

4.4.5 JSX

JSX (JavaScript eXtension) on React osa, jolla voidaan kirjoittaa JavaScriptiä, joka näyttää HTML:ltä. JSX elementille pitää antaa alku- ja lopetusmerkki esimerkiksi **<img></img>** tai **<img />**. Kuvassa 12 on esitetty React-komponentti, joka palauttaa tekstin Hello World <h1> elementissä.

```
class HelloWorld extends React.Component {
  render() {
    return (
      <h1 className='large'>Hello World</h1>
    );
  }
}
```

Kuva 12. React-komponentti, joka palauttaa tekstin h1 muodossa. (Lerner, n.d)

Kuvan 12 render funktio ei palauta HTML:ää vaan JSX:ää. JSX muutetaan JavaScriptiksi ajonaikana (engl. runtime). JSX on lyhyempi tapa kirjoittaa React.createElement() metodi. Koska JSX on JavaScriptiä, komponenteille ei voi antaa parametreiksi esimerkiksi class-avainta koska se on varattu nimi, vaan se pitää muodossa className. (Lerner, n.d). Kuvassa 13 esitetään kuvan 12 React-komponentti, kun JSX on muutettu JavaScriptiksi.

```
class HelloWorld extends React.Component {
  render() {
    return (
      React.createElement(
        'h1',
        {className: 'large'},
        'Hello World'
      )
    );
  }
}
```

Kuva 13. JSX-elementti on muutettu JavaScriptiksi. (Lerner, n.d)

JavaScript ominaisuuksia voi myös käyttää JSX:n sisällä. Jos JSX:n halutaan tehdä esimerkiksi conditional-renderiä eli JSX-elementtejä halutaan esittää komponentin sisällä vain tietyin ehdoin, pitää ehto lause sijoittaa kaarisulkujen sisään. Kuvassa 12 on esitetty React-komponentti, joka palauttaa tekstin toisen osan vain silloin jos props.name muuttujalle on annettu arvo, eli sen arvo ei ole määrittämätön.

```
class HelloYou extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello</h1>
        {this.props.name !== undefined ? (
          <h1>, {this.props.name}.</h1>
        ) : (
          null
        )
        }
      </div>
    );
  }
}

ReactDOM.render(
  <HelloYou name="Henri"/>,
  document.getElementById('root')
);
```

Kuva 14. React-komponentti, joka palauttaa aina Hello tekstin, mutta nimen vain silloin, jos se on annettu.

4.5 Bash

Tässä kappaleessa käydään läpi ne Bash-skriptauksen ominaisuudet, joita tässä opinnäytetyössä on hyödynnetty, jotta lukija osaa tulkita dokumentoitua skriptiä. Bash-skripti on tiedosto, joka sisältää joukon komentoja. Bash-skriptiin voidaan antaa täysin samat komennot, jotka normaalisti annettaisiin komentoriville. Bash-tiedostojen päätteenä käytetään .sh:ta ja tiedoston alussa pitää aina antaa shebang (#!) merkkijono. Kuvassa 15 on esitetty skripti, joka tulostaa tekstin Hello World. Kuvassa 16 on esitetty komento, jolla skripti voidaan suorittaa komentorivillä.

```
#!/bin/bash
# Hello World esimerkki hello.sh
echo Hello World!
```

Kuva 15. Esimerkki Bash-skripti nimeltä hello.sh.

```
user@bash: ./hello.sh
Hello World!
```

Kuva 16. Kuvan 15 skripti voidaan suorittaa komennolla ./hello.sh. Skripti tulostaa Hello World! tekstin.

Kuvassa 15 rivillä yksi on shebang (#!), sen perässä annetaan polku ohjelmaan, jolla skripti suoritetaan, tässä tapauksessa Bash. Shebang pitää aina olla rivillä yksi. #-merkin ja !-merkin sekä polun välissä ei saa olla välilyöntejä. Polkuna on suositeltavaa käyttää

absoluuttista polkua, jotta skripti voidaan suorittaa useammasta paikasta. Shebang voidaan myös jättää pois, jolloin skripti pitää suorittaa antamalla komento **bash hello.sh**. Rivillä kaksi on kommentti. Kaikki #-merkin takana olevat merkit jätetään huomiotta. Rivillä kolme annetaan echo-komento, joka tulostaa tekstin Hello World! komentoriville. Muotoilu on tärkeää Bash-skriptissä, etenkin välilyöntien osalta. Välilyönnin puute tai ylimääräinen välilyönti voi rikkoa skriptin.

Kun halutaan lukea muuttuja, asetetaan $-merkki muuttujan nimen eteen. Jos muuttuja halutaan asettaa, eteen ei laiteta merkkiä. Skriptille voidaan myös antaa argumentteja. Argumentit annetaan skriptitiedoston nimen jälkeen. Argumentit erotellaan välilyönnillä ja niihin päästään käsiksi $1 - $9 muuttujilla.

```
#!/bin/bash
muuttuja=$1
echo “Arvo: $muuttuja”
```

Kuva 17. Esimerkki skripti nimeltä muuttuja.sh, joka tulostaa argumenttina annetun arvon.

```
user@bash: ./muuttuja.sh 2
Arvo: 2
```

Kuva 18. Kuvan 6 skripti tulostaa annetun arvon.

Ehtolause on hyödyllinen, jos halutaan suorittaa vain tietty osa skriptistä. Ehtolauseen ehdossa on tärkeä sijoittaa ehto oikein muotoiltuna, muuten ehtolause ei toimi. Kuvassa 19 esitetään ehtolause joka testaa onko käyttäjän antama luku alle 10.

```
#!/bin/bash
If [ $1 -lt 10]
then
    echo ”Arvo $1 on pienempi kuin 10”
else
    echo “Arvo $1 on suurempi kuin 10”
fi
```

Kuva 19. Tulostaa tekstin, jos annettu arvo on pienempi kuin 10.

Operaattoreilla voidaan esimerkiksi vertailla arvoja keskenään tai tarkistaa minkälaiset oikeudet tiedostolle on annettu. Taulukossa 1 on esitetty usein käytettyjä Bash-operaattoreita.

Taulukko 1. Usein käytettyjä Bash-operaattoreita. (Chadwick, n.d)

Operator	Description
! EXPRESSION	The EXPRESSION is false.
-n STRING	The length of STRING is greater than zero.
-z STRING	The lengh of STRING is zero (ie it is empty).
STRING1 = STRING2	STRING1 is equal to STRING2
STRING1 != STRING2	STRING1 is not equal to STRING2
INTEGER1 -eq INTEGER2	INTEGER1 is numerically equal to INTEGER2
INTEGER1 -gt INTEGER2	INTEGER1 is numerically greater than INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is numerically less than INTEGER2
-d FILE	FILE exists and is a directory.
-e FILE	FILE exists.
-r FILE	FILE exists and the read permission is granted.
-s FILE	FILE exists and it's size is greater than zero (ie. it is not empty).
-w FILE	FILE exists and the write permission is granted.
-x FILE	FILE exists and the execute permission is granted.

Jos skriptissä on paljon erilaisia polkuja, voi olla järkevää käyttää case-lausetta pitkän else-if rakenteen sijaan. Tämä on toiminnaltaan samalainen kuin esimerkiksi switch C-kielessä. Kuvassa 20 on esitetty case-lauseke, joka tulostaa tekstin, kun käyttäjä antaa skriptille tekstijonon parametrinä.

```
#!/bin/bash
Case $1 in
    start)
        echo starting
        ;;
    stop)
        echo stopping
        ;;
    restart)
        echo restarting
        ;;
    *)
        echo unknown
        ;;
esac
```

Kuva 20. Bash case-lauseke.

Loop-ehtoa voidaan käyttää, jos halutaan suorittaa jokin tietty kohta skriptistä monta kertaa. Bashissa voidaan tehdä for-, while-, until- ja select-silmukoita. Selectiä voidaan käyttää menun tekoon. Kuvassa 21 on esitetty esimerkki for-silmukasta, joka ajetaan niin monta kertaa kuin skriptille on annettu argumentteja.

```
#!/bin/bash
for i
do
    echo $i
done
```

Kuva 21. Tulostaa jokaisen argumentin

4.6 Redis

Redis on avainpohjainen NoSQL-tietokanta. Avaimet voivat sisältää muun muassa tekstiä, listoja ja tiivisteitä (engl. hash). Redis tallentaa avaimet RAM-muistiin, joten tiedot häviävät, jos laite käynnistetään uudestaan, sammutetaan tai jos Percistance-asetus ei ole päällä. Redis tietokanta voi myös olla sijoitettuna toiselle palvelimelle.

Redis on hyödyllinen esimerkiksi silloin kuin Nodessa halutaan käyttää klustereita, eli ajetaan jokaisella prosessorin säikeellä omaa Node.js prosessia. Globaalit muuttujat eivät ole jaettu klustereiden kesken, mutta jokaisella klusterilla on oikeus lukea muuttuja arvot Rediksestä. Toinen käyttötarkoitus voi olla muistin hallintaan, kun halutaan siirtää muistin käyttöä Node.js-prosessista Redis-prosessiin.

Kuvassa 22, rivillä 1 avataan Redis-ohjelma. Rivillä 2 asetetaan muuttuja komennolla set, annetaan muuttujan nimi ja sen arvo. Rivillä kolme on viesti, joka kertoo, että arvo on asetettu. Rivillä 4 haetaan arvo antamalla get-komento ja muuttujan nimi, jonka jälkeen Redis tulostaa arvon seuraavalle riville.

```
redis-cli
redis> set muuttuja arvo
OK
redis> get muuttuja
”arvo”
```

Kuva 22. Arvon asettaminen ja lukeminen Redis-ohjelmalla

4.6.1 Rediksen käyttö Nodessa

Kun Redistä halutaan käyttää Noden kanssa, voidaan käyttää Redis NPM-moduulia. Kuvassa 23 on yksinkertainen esimerkki Rediksen käytöstä Node.js ympäristössä

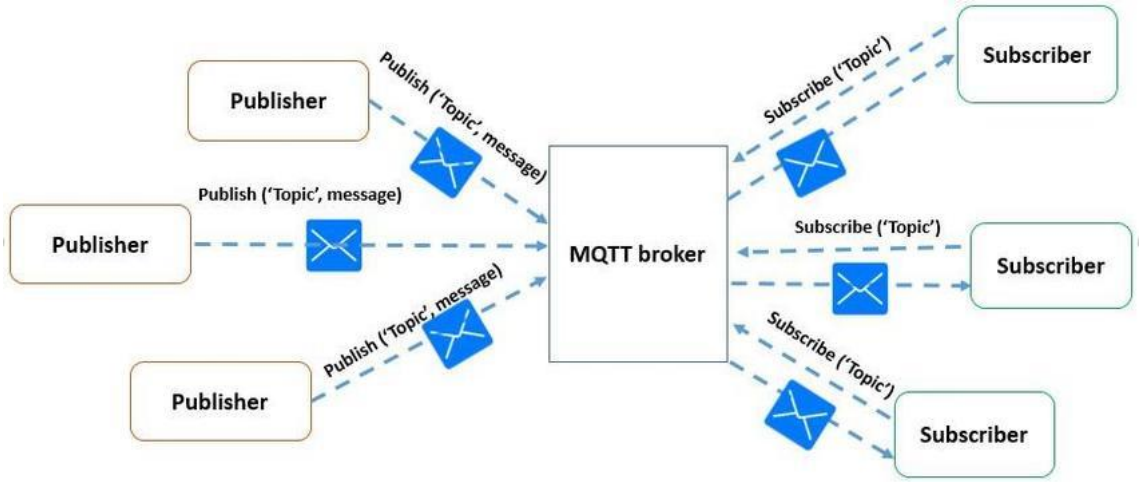
```
let redis = require('redis');
let client = redis.createClient();

client.on('connect', () => {
    console.log('Redis toiminnassa');
});

client.on('error', (error) => {
    console.log('Redis virhe: ' + error);
});

client.set('muuttuja', '10');
client.get('muuttuja', (error, result) => {
    if (error) {
        console.log(error);
        throw error;
    }
    console.log('Arvo: ' + result);
});
```

Kuva 23. Rediksen käyttö Node.js-ympäristössä

Rivillä yksi otetaan Redis käyttöön. Rivillä kaksi luodaan client, joka on yhteydessä Redis-tietokantaan. Funktiolle createClient voidaan myös antaa parametreinä IP-osoite ja portti, oletuksena ne on asetettu IP-osoitteeseen 127.0.0.1 ja porttiin 6379. Rivillä neljä tulostetaan viesti, jos Redikseen on saatu onnistuneesti yhteys. Rivillä kahdeksan tulostetaan virhesanoma, jos yhteyttä ei saatu muodostettua. Rivillä 12 asetetaan muuttuja nimellä muuttuja ja sille annetaan arvo 10. Rivillä 13 tulostetaan arvo, jos haussa on virhe, tulostetaan virhesanoma. Virheen tapahtuu esimerkiksi silloin kun yritetään hakea avainta, jota ei ole olemassa.

4.7 **MQTT**

MQTT (Message Queuing Telemetry Transport) on ISO-standardin mukainen publish-subscribe viestitysprotokolla ja toimii TCP/IP-protokollassa. MQTT on suunniteltu yhteyksiin, joissa on rajattu kaistanleveys. Toimiakseen se vaatii message broker-palvelimen, joka välittää viestit eteenpäin. MQTT:ssä jokaiselle yhteydelle määritetään QoS (Quality of Service) tasolla 0-2. Tasolla 0 vietin lähettäjä ei odota vastausta siitä onko viesti vastaanotettu, tasolla 1 viesti lähetetään niin monta kertaa, kunnes viesti on vastaanotettu ja tasolla 2 vastaanottaja ja lähettäjä tekevät kättelyn, jolloin lähetetään vain yksi viesti ja varmistetaan että se on vastaanotettu. Viestitys on jaettu aiheisiin (engl. topic), joihin asiakas (engl. client) voidaan liittää. Aiheita voi olla monia ja asiakas voivat kuulua yhteen tai useampaan aiheeseen. (Giant, n.d) Kuvassa 24 on esitetty MQTT-verkon toiminta.



Kuva 24. MQTT-verkko toimii Publish/Subscribe menetelmällä, jossa broker on viestin välittäjä. (Boulmakoul, 2016)

5 TOTEUTUS

5.1 Versionhallinta

5.1.1 Git

Projektin versiohallintaohjelmana käytetään komentorivi pohjaista Git-ohjelmaa. Gitin avulla projektista voidaan pitää helposti yllä montaa versiota ja tarvittaessa voidaan palata vanhaan ohjelma versioon. Gitin avulla projekti-tiedostot voidaan myös ladata suoraan ulkoiseen ohjelmavarastoon, josta ne ovat tarvittaessa helppo ladata muille työkoneille.

Gittiä käytetään myös järjestelmän ohjelmien päivittämiseen. Päivitys prosessista voi lukea tarkemmin luvussa 5.4.4 Ohjelmiston päivittäminen.

5.1.2 GitHub

Projektin ulkoisena ohjelmavarastona (engl. repository) käytetään GitHubia. GitHubissa säilytetään kaikki projektissa tarvittavat tiedostot, paitsi asetustiedostot, joissa säilytetään salasanoja, tunnuksia tai API-avaimia. GitHubissa voi myös helposti tarkastella koodiin tehtyjä muutoksia. GitHubin kanssa kommunikointiin tarvitaan Git-versiohallinta ohjelmaa.

GitHubia käytetään myös järjestelmän ohjelmien päivittämiseen. Päivitysprosessista voi lukea tarkemmin luvusta 5.4.4 Ohjelmiston päivittäminen.

5.2 Lisensointi

Kaikki tässä opinnäytetyössä tuotetut ohjelmat käyttävät MIT-lisenssiä. Pohjana käytetään Open Source Initiativen verkkosivulla esitettyä MIT-lisenssi pohjaa. Jokaisen ohjelmavaraston juureen on lisätty LICENSE-tekstitiedosto, joka sisältää alla esitetyn lisenssin:

> "MIT License
>
> Copyright (c) 2018 Henri Pirinen
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the Software), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED AS IS, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE."

5.3 Järjestelmäkuvaus

Kuvassa 3 on esitetty järjestelmä yhteystasolla. Kaikki modeemin taakse sijoitetut laitteet ovat osa auton sisäistä järjestelmää. Controller-, Driver- ja Thermocouple-kortit on liitetty Raspberry Pi:hin USB-portin kautta. Raspberryllä, päätelaitteella ja vaihtosuuntaajalla on WLAN-yhteys modeemiin. Modeemissa on 3G-liittymä, jolla saadaan muodostettua yhteys internetiin. Kaikki etä- ja paikallisen rajapinnan välinen liikenne kulkee MQTT-brokerin kautta.

Controller-kortit mittaavat kennojen jännitteet ja lämpötilat. Mittaustulokset siirretään paikalliselle palvelimelle, joka lähettää mittaustulokset päätelaitteelle, jos päätelaiteella on yhteys paikalliseen palvelimeen. Controller-kortit kytkevät automaattisesti jännitteen tasauksen pois tai päälle. Jännitteentasaus voidaan myös kytkeä pois ja päälle manuaalisesti käyttöliittymän kautta.

Driver-kortti on kytketty vaihtosuuntaajan ohjaukseen. Driver-kortin kautta moottori voidaan asettaa kolmeen eri tilaan, eteen, taakse ja vapaalle. Vakionopeudensäädin asetetaan myös Driver-kortin kautta. Driver-kortin tiloja muutetaan käyttöliittymän kautta. Driver-kortti kytkee myös latauksen ja lämmittimen päälle ja poistilaan.

Thermocouple-kortti mittaa moottorin lämpötilaa kahdesta pisteestä. Jos moottorin lämpötila ylittää maksimiarvon, moottori hätäsammutetaan. Moottori palaa takaisin käytettäväksi, kun lämpötila on pudonnut tarpeeksi monta asetetta maksilämpötilasta. Maksimilämpötila voidaan asettaa käyttöliittymän kautta.

Paikallisella palvelimella (Raspberry Pi) on WebSocket-rajapinta, joka suorittaa tiedonsiirron laitteiden ja käyttöliittymän välillä. Rajapinnan kautta voidaan myös päivittää mittaus- ja hallintalaitteiden, sekä rajapinnan ohjelmat.

Vaihtosuuntaaja (engl. inverter) ohjaa auton sähkömoottoria. Vaihtosuuntaaja on liitetty samaan WLAN-verkkoon kuin muutkin auton sisällä verkkoyhteydessä olevat laitteet. Vaihtosuuntaajan arvoja voidaan muuttaa REST-rajapinnan avulla. Vaihtosuuntaajaa ohjataan käyttöliittymän kautta.

MQTT-broker on kolmannen osapuolen tarjoama palvelu, joka hoitaa MQTT-verkon viestin välityksen paikallisen- ja etäpalvelimen välillä. MQTT-brokerin konfiguroinnista voi lukea kappaleesta 5.7.

Etäpalvelimelle on sijoitettu SQL-tietokanta, joka kerää kennojen jännite- ja lämpötila-arvot. Näitä arvoja voidaan tarkastella kirjautumalla etäpalvelimen käyttöliittymään. Palvelin ohjelmistona käytetään Node.js-ohjelmaa. Etäpalvelimen kautta voidaan myös asettaa auton lämmitin päälle- ja pois tilaan.

Käyttöliittymän kautta voidaan tarkkailla reaaliajassa akkukennojen lämpötilaa ja jännitettä, muuttaa moottorin suuntaa, asettaa vakionopeudensäätimen tila, käynnistää ja sammuttaa lämmittimen, manuaalisesti säätää kennojen jännitteentasoitusta ryhmätasolla, lukea järjestelmän lokia, selata karttaa, lukea paikallisen säätiedon ja viiden päivän ennusteen, päivittää laitteiden ohjelmat sekä konfiguroida palvelimen asetukset. Käyttöliittymä konfiguroidaan joko paikalliseksi tai etäkäyttöliittymäksi.

5.4 **Paikallinen palvelin**

5.4.1 Node.js palveluksi (Service)

Node.js-applikaatio halutaan palveluksi, koska sen pitää toimia taustaprosessina, käynnistyä laitteen yhteydessä automaattisesti ja uudelleen käynnistyä, jos se jostain syystä kaatuu. Tämä voidaan toteuttaa SystemDllä, joka on standardi työkalu Linux-jakeluissa.

Ensimmäisenä Node.js-applikaatiolle pitää antaa suoritus (engl. executable) oikeudet. Tämä onnistuu komennolla **chmod +x index.js**. Node.js-applikaation ensimmäiselle riville pitää lisätä teksti **#!/usr/bin/env node**, tämä kertoo käyttöjärjestelmälle, miten ohjelma ajetaan. Tässä tapauksessa kerrotaan, että ohjelma ajetaan Nodella.

Kuvassa 25 on esitetty tässä projektissa käytettävä SystemD-konfiguraatiotiedosto. Palvelu käynnistetään, kun palvelin laite on saanut verkkoyhteyden. Palvelu suoritetaan root käyttäjällä. Jos palvelu kaatuu, se käynnistetään aina uudestaan, kun kaatumisesta on kulunut 500 millisekuntia. Palvelu otetaan käyttöön antamalla komento **sudo systemctl enable regni-server.service**. Palvelulle voidaan antaa systemctl start, stop ja restart komennot. Palvelun virhelokia voidaan lukea antamalla komento **sudo journalctl -u regni-server.service**.

```
#/etc/systemd/system/regni-server.service
[Unit]
Description=Node.js Regni Server
After=network.target

[Service]
PIDFile=/tmp/regni-99.pid
User=root
Restart=always
RestartSec=500ms
StartLimitInterval=0
KillSignal=SIGQUIT
WorkingDirectory=/home/pi/Public/nodeServer/
ExecStart=/usr/bin/node /home/pi/Public/nodeServer/index.js

[Install]
WantedBy=multi-user.target
```

Kuva 25. Palvelun regni-server.service konfiguraatio

5.4.2 Sarjaliikenne rajapinnan ja laitteiden välillä

Hallinta- ja mittauslaitteet kommunikoivat rajapintaan USB-väylän kautta. Laitteiden nimet voidaan nähdä Linuxilla antamalla komento **ls /dev**. USB-laitteiden nimet voivat vaihdella, esimerkiksi nimimerkillä USB0 oleva laite voi seuraavalla käynnistyskerralla olla USB1. Tämä tuottaa ongelmia, jos ohjelma olettaa, että esimerkiksi Driver-kortti olisi aina USB0 nimellä. USB-laitteelle voidaan antaa staattinen nimi, jolloin se ilmaantuu aina samannimisenä laitteena käyttöjärjestelmälle. Kuvassa 26 esitetyllä komennolla voidaan lukea USB-laitteen muuttuja arvot. Komennon palauttamasta listasta tarvitaan arvot ATTRS{serial}, ATTRS{idVendor} ja ATTRS{idProduct}.

```
udevadm info -a -p  $(udevadm info -q path -n /dev/ttyUSB0)
```

Kuva 26. Komento, jolla saadaan USB-laitteet muuttujien arvot.

Kun tarvittavat muuttujat on löydetty, tehdään palvelimelle uusi udev sääntö USB-laitteelle. Sääntötiedosto tehdään kansio rakenteeseen /etc/udev/rules.d/, nimeksi annetaan esimerkiksi usb.rules. Kuvassa 27 on esitetty tässä työssä USB-laitteille annetut säännöt, joilla laitteille saadaan staattiset nimet. SYMLINK-arvoksi annetaan laitteelle

haluttu nimi. Uudet säännöt otetaan käyttöön antamalla komento **sudo udevadm trigger**.

```
#/etc/udev/rules.d/99-usb-serial.rules
SUBSYSTEM=="tty", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6001", ATTRS{se-
rial}=="AI03VPZX", SYMLINK+="controller.1"
SUBSYSTEM=="tty", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6001", ATTRS{se-
rial}=="AL00UGQ2", SYMLINK+="controller.2"
SUBSYSTEM=="tty", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6001", ATTRS{se-
rial}=="AL00UFYQ", SYMLINK+="driver.1"
SUBSYSTEM=="tty", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6001", ATTRS{se-
rial}=="AL00UHC7", SYMLINK+="thermo.1"
```

Kuva 27. USB-säännöt, joilla laitteet saavat aina samat nimet.

Node.js-ohjelmassa käytettään SerialPort ja Parser-Delimiter JavaScript-moduuleita. SerialPort-moduulilla voidaan lukea ja kirjoittaa sarjaliikenneporttiin. Parser-Delimiter-moduulilla helpotetaan sarjaportista tulevan liikenteen lukemista. Delimiterille voidaan antaa merkki, jolla se voi erotella sarjaliikenteestä tulevat viestit. Tässä ohjelmassa viestin erotusmerkkinä käytetään rivivaihtoa \n. Kun delimiter vastaanottaa \n-merkin, suoritetaan tapahtuma, jonka kautta voidaan lukea vastaanotettu viesti.

Kuvassa 28 on esitetty osa Controller-korttia ohjaavasta rajapinnasta. Kun rajapinta saa viestin, joka loppuu \n-merkkiin, tarkistetaan alkaako viesti $-merkillä. Jos viesti on $init, annetaan Controller-kortille alkuasetukset. Jos viesti on $!serialCharge, kirjoitetaan Driver-kortin sarjaliikenteeseen komento, joka pysäyttää ajoneuvon sarjalatauksen. Jos Controller-kortilta saatu viesti ei ala $-merkillä, viesti on tarkoitettu käyttöliittymään lokiviestinä tai dataviestinä.

```
import * as SerialPort from 'serialport';
import * as Delimiter from 'parser-delimiter';
import * as config from './serverCfg';
//...
const ctrl = new SerialPort(config.port.ctrlPort, { baudRate: 9600 });
//...
const driver = new SerialPort(config.port.driverPort, { baudRate: 9600 });
//...
const ctrl_input = ctrl.pipe(new Delimiter({ delimiter: `\n` }));
//...
ctrl_input.on(`data`, data => {
    let input = data.toString();
    if (input.charAt(0) === '$') {
        if (input.substring(0, 5) === '$init') {
            ctrl.write(`0,${config.limits.serialMax}`, (err) => {
                if (err) return console.log(err.message);
            });
        } else if (input.substring(0, 14) === '$!serialCharge') {
            driver.write('SC0', (err) => {
                if (err) return console.log(err.message);
            });
        }
    } else if (utilities.validateJSON(input)) {
        //...
    }
});
```

Kuva 28. Sarjaliikenteen lukeminen Node.js applikaatiolla.

5.4.3 WebSocket

WebSocketilla muodostetaan kaksisuuntainen yhteys rajapinnan ja käyttöliittymän välille. Kaikki tiedonsiirto paikallisen rajapinnan ja käyttöliittymän välillä suoritetaan WebSocketin avulla. Etärajapinnassa ei ole käytössä WebSocket rajapintaa, vaan kaikki kommunikaatio käyttöliittymän ja rajapinnan välillä suoritetaan REST:llä.

Rajapinnassa ja käyttöliittymässä käytetään Socket.io kirjastoa WebSocket-yhteyden luomiseen. WebSocketin käytöstä käyttöliittymässä voi lukea luvusta 5.8.3. Kuvassa 29 on esitetty paikallisen rajapinnan WebSocket, joka toimii HTTPS-yhteydellä. Paikallisella palvelimella käytetään itsekirjoitettua TLS-sertifikaattia.

```
import * as socket from 'socket.io';
import * as https from 'https';
import * as fs from 'fs';
//...
const sslOptions = {
  key: fs.readFileSync('regni-key.pem'),
  cert: fs.readFileSync('regni-cert.pem')
};
//...
const app = express();
const server = https.createServer(sslOptions, app).listen(443, () => {
  console.log('Serving webbapp on port 443');
});
//...
const io = socket(server);
```

Kuva 29. WebSocket HTTPS-yhteydellä.

Kun käyttäjä muodostaa WebSocket-yhteyden rajapintaan, haetaan ensimmäisenä vaihtosuuntaajalta parametrien arvot. Kun rajapinta on saanut arvot vaihtosuuntaajalta, lähetetään käyttäjälle kaikki palvelimen konfiguroitavat arvot, kuten MQTT-osoitteet, lämpötilarajat jne. Kun käyttäjä on liittynyt WebSocket-rajapintaan, rajapinta alkaa lähettämään kennoista uutta mittausdataa ja lokiviestejä aina kun niitä tulee saataville. Kokonaisen listan lähetettävistä arvoista löytyy luvusta 5.4.6 paikallisen palvelimen API-yhteenveto.

WebSocket-rajapinnan kautta voidaan antaa komentoja Controller-korteille, lähettää käskyjä vaihtosuuntaajalle ja antaa Unix-komentoja Raspberry Pi:lle. Myös palvelimen päivityskomennot ja uudelleen konfiguraatiokomennot annetaan WebSocket-rajapinnan kautta. Lista komennoista ja muotoilusta löytyy luvusta 5.4.6 paikallisen palvelimen API-yhteenveto.

5.4.4 Ohjelmiston päivittäminen

Ohjelmien uusin versio on saatavilla GitHub-ohjelmavarastossa. Uusin versio saadaan antamalla Gitille komento **git pull origin master**. Käyttöliittymästä voidaan valita, mitkä laiteet halutaan päivittää. Palvelimella on Bash-skripti, joka suorittaa tarvittavat komennot. Skriptille annetaan parametreinä ne laitteet, jotka halutaan päivittää. Arduino-korttien päivittämiseen käytetään arduino-mk ohjelmaa, jonka avulla voidaan tehdä Makefile Arduino-ohjelmille. Makefilessä pitää antaa vähintään kuvassa 30 esitetyt tiedot.

```
ARDUINO_DIR = /usr/share/arduino
ARDUINO_PORT = /dev/thermo.1
USER_LIB_PATH = /home/pi/Public/vehicle-thermo/lib
BOARD_TAG = nano328
include /usr/share/arduino/Arduino.mk
```

Kuva 30. Makefile-tiedosto, jossa annetaan tarvittavat tiedot koodiin laukselle.

Kuvassa 31 on esitetty osa softwareUpdate.sh Bash-skriptistä (Pirinen, softwareUpdate.sh, 2018), jolla saadaan uusi koodi ladattua Driver-kortille. Ensimmäisellä rivillä tulostetaan teksti, joka kertoo seuraavan päivitettävän laitteen olevan Driver-kortti. Rivillä kaksi siirrytään vehicle-driver/driver/ kansioon, jossa sijaitsee Git-ohjelmavarasto. Kolmannella rivillä annetaan git-komento, joka lataa uusimman version GitHubista origin master haarasta (engl. branch). Koodi ajetaan kortille antamalla **make upload** komento. Tämä komento vaatii Makefile-tiedoston, jossa annettu vähintään kuvassa 30 annetut tiedot. Komento make upload suorittaa ennen koodin latausta **make compile** komennon, joka tekee build-kansion. Kun koodi on ajettu Arduinolle, voidaan build-kansio poistaa.

```
echo "### DRIVER UPDATE ###"
cd /home/pi/Public/vehicle-driver/driver/
sudo -u pi git pull origin master
make upload
rm -rf build-nano328/
```

Kuva 31. Komennot, jotka palvelin suorittaa, kun Driver-kortti päivitetään.

Käyttöliittymä päivitetään rajapinnan yhteydessä, joten sillä ei ole omaa päivityskategoriaa. Päivitys käynnistetään palvelimella, kun WebSockettiin saadaan viesti nimellä update. Parametreinä viestissä annetaan päivitettävät laitteet. Laitteet erotellaan välilyönnillä. Kuvassa 32 on esitetty päivitysskriptin käynnistys palvelimella.

```
socket.on(`update`, (command) => {
  exec(`sudo bash /home/pi/Public/nodeServer/softwareUpdate.sh
${command.target}`, function (err, stdout, stderr) {
    if (err) console.log(stderr);
  })
});
```

Kuva 32. Päivitys skriptin käynnistys palvelimella

Liitteessä 1 käyttöliittymän ohjeet, on esitetty laiteiden päivittäminen käyttöliittymän kautta.

5.4.5 Vaihtosuuntaajan ohjauksen integrointi

Vaihtosuuntaaja on liitetty samaan verkkoon paikallisen palvelimen kanssa. Modeemista ei pysty antamaan laitteille staattisia IP-osoitteita, joten vaihtosuuntaajan osoite pitää etsiä verkosta MAC-osoitteen perusteella. Tähän prosessiin löytyy valmis arpscan NPM-moduuli (goliatone, 2019). Arpscan etsii kaikki IP-osoitteet verkosta ja tekee niistä listan. Kun haku on valmis, etsitään listasta vaihtosuuntaajan MAC-osoitetta vastaava IP-osoite. Kuvassa 33 on esitetty arpScanner-moduulin käyttö. Arpscan suoritetaan rajapinnan käynnistyksen yhteydessä.

```
const options = { command: 'arp-scan', interface: 'wlan0', sudo: true };
arpScanner(onResult, options); //Scan network.
function onResult(err, data) {
    if (err) throw err;
    for (let i = 0; i < data.length; i++) { //Search scanner results
        if (data[i].mac === '5C:CF:7F:8E:26:00') inverterIpAdress = data[i].ip;
    }
}
```

Kuva 33. arpScanner moduulilla voidaan etsiä laitteen IP-osoite verkosta MAC-osoitteen perusteella.

Vaihtosuuntaajalla on oma REST-rajapinta, jonka kautta voidaan hakea ja asettaa parametrien arvoja. Vaihtosuuntaajan parametritaulukko löytyy vaihtosuuntaajan dokumentaatiosta (Hüebner, Inverter Parameters, 2019).

Kun vaihtosuuntaajan IP-osoite on tiedossa, voidaan sille tehdä GET-pyyntöjä. Esimerkiksi lähettämällä http://IP-OSOITE/cmd?cmd=json pyynnön, vaihtosuuntaaja palauttaa kaikkien parametrien arvot JSON-muodossa.

Parametri arvoja voidaan hakea vaihtosuuntaajalta tekemällä REST-pyyntö vaihtosuuntaajalle. Alla on esitetty REST-pyyntö, joka asettaa ampmin parametrille arvon 50.

```
http://IP-OSOITE/cmd?cmd=set ampmin 50
```

Parametrin arvo voidaan hakea antamalla alle esitetty REST-pyyntö.

```
http://IP-OSOITE/cmd?cmd=get ampmin
```

Koska järjestelmän kaikki sisäinen liikenne rajapinnan ja käyttöliittymä väillä halutaan toimivan WebSocketilla, on omaan rajapintaan tehty WebSocket-tapahtuma, joka lähettää REST-pyynnön vaihtosuuntaajalle. Tästä on myös se hyöty, että vaihtosuuntaajan tapahtumien lokittaminen on helpompaa. Kuvassa 34 tehdään käyttöliittymän kautta asetus pyyntö vaihtosuuntaajalle oman rajapinnan kautta.

```
webSocket.emit('command', { //Pyyntö omaan rajapintaan
    command: 'set ampmin 50',
    handle: 'client',
    target: 'inverter'
});
```

Kuva 34. WebSocket-komento, jolla asetetaan vaihtosuuntaajan ampmin parametrille arvo 50.

Kuvassa 35 on esitetty osa rajapinnan command-switch-rakenteesta, jossa rajapinta tekee REST-pyynnön vaihtosuuntaajan rajapintaan. Kun vaihtosuuntaaja vastaa takaisin, muutetaan vastaus JSON-muotoon ja lähetetään se käyttöliittymään WebSocket kanavalla inverterResponse.

```
socket.on(`command`, (data) => {//Palvelin välittää
    switch (data.target) {
        //...
        case "inverter":
            fetch(`http://${inverterIpAdress}/cmd?cmd=${data.command}`)
                .then((res) => res.json())
                .then((result) => {
                    socket.emit(`inverterResponse`, {
                        message: JSON.stringify(result),
                        handle: `Server`
                    });
                })
            break;
        //...
    };
});
```

Kuva 35. Palvelin välittää pyynnön vaihtosuuntaajalle.

5.4.6 Paikallinen WebSocket-rajapinta

Paikallisella palvelimella kommunikointi käyttäjän ja järjestelmän välillä suoritetaan WebSocket-rajapinnalla. WebSocket mahdollistaa kaksisuuntaisen jatkuvan liikenteen rajapinnan ja käyttäjän välillä. Kun laitteilta tulee viestejä, voidaan ne välittää suoraan käyttöliittymään ilman, että käyttöliittymän tarvitsisi tehdä pyyntöä rajapintaan.

Nimellä **dataset** vastaanotetaan kennojen jännite- ja lämpötilamittaukset. Mittaukset tulevat ryhmäkohtaisesti ja välittömästi kun kaikki ryhmän kennot on mitattu. Kuvassa 36 on esitetty dataset JSON-viesti.

```
{"Group":0,"type":"data","voltage":[0, ... ,7],"temperature":[0, ... ,7]}
```

Kuva 36. dataset JSON-viesti.

Nimellä **systemLog** vastaanotetaan lokitietoja rajapinnasta sekä mittaus- ja hallintalaitteilta. Lokiviesteissä ei voi valita vain yhden laitteen lokia vaan sen kautta tulee jokaisen rajapintaan kytketyn laitteen lokiviestit. Laitteet voi erotella toisistaan origin-tagi mukaan. Importance-tagissa on merkattu lokiviestin tärkeysaste. Vikatapahtumat on merkattu High-tasolla, tilamuutokset Medium-tasolla ja muut tapahtumat Low-tasolla. Kuvassa 37 on esitetty systemLog JSON-viesti Controller-kortilta.

```
{"origin":"Controller","type":"log","msg":"Message","importance":"High"}
```

Kuva 37. systemLog JSON-viesti Controller-kortilta.

Nimellä systemState voidaan vastaan ottaa parametri muutokset. Kuvassa 38 on esitetty systemState JSON-viesti. Nimi kentässä on kerrottu parametrin nimi ja value-kentässä parametrin arvo.

```
{"origin":"Dev","type":"param","name":"pName","value":"00","importance":"Low"}
```

Kuva 38. systemState JSON-viesti.

Nimellä **systemParam** voidaan vastaan ottaa rajapintaan ja vaihtosuuntaajaan asetetut parametriarvot. Kuvassa 39 on esitetty systemParam-viesti, joka voidaan ottaa vastaan, kun käyttöliittymä on muodostanut yhteyden WebSocket-rajapintaan. Taulukossa 2 on esitetty kuvaukset systemParam JSON-viestissä käytettävistä parametreistä.

```
{
    "weatherAPI": "key",
    "mapAPI": "key",
    "remoteAddress": "mqtts://address.com",
    "controller_1": "/dev/controller.1",
    "controller_2": "/dev/controller.2",
    "driverPort": "/dev/driver.1",
    "driverState": "0000",
    "remoteUpdateInterval": 5,
    "groupChargeStatus": [0,0,0,0,0,0,0,0,0,0],
    "thermoDevice": "/dev/thermo.1",
    "temperatureLimit": "110",
    "voltageLimit": "3.1",
    "isCharging": false,
    "inverterValues": null,
    "mqttUName": "name",
    "mqttPWord": "password"
}
```

Kuva 39. systemParam JSON-viesti.

Taulukko 2. Kuvassa 39 annettujen parametrien kuvaukset.

weatherAPI	API avain OpenWeatherMap rajapintaan
mapAPI	API avain Google Maps rajapintaan
remoteAddress	MQTT-broker osoite
controller_1	Laitteen portin nimi
controller_2	Laitteen portin nimi
driverPort	Laitteen portin nimi
driverState	Driver-kortin pinnien tila. Järjestyksessä eteen, taakse, vakionopeus, Webasto Jos eteen ja taakse ovat 0, ajoneuvo on neutraali tilassa.
remoteUpdateInterval	Aika (min) jonka välein mittauksen lähetetään remote palvelimelle.
groupChargeStatus	1 = Jännitteentasaus aktiivinen 0 = Jännitteentasaus ei aktiivinen
thermoDevice	Laitteen portin nimi
temperatureLimit	Moottorin maksimi lämpötila

voltageLimit	Sarjalatauksen raja-arvo
isCharging	Onko ajoneuvo latauksessa?
inverterValues	Vaihtosuuntaajan parametrien arvot JSON-merkkijonona. Jos vaihtosuunta ei ole päällä, arvo on tyhjä (null).
mqttUName	MQTT käyttäjänimi
mqttPWord	MQTT salasana

Nimellä **inverterResponse** voidaan ottaa vastaan viestejä vaihtosuuntaajalta. Tämän kanavan kautta otetaan vastaan kaikki takaisin tuleva liikenne vaihtosuuntaajalta. Pyynnöt ja vastaukset vaihtosuuntaajalle reititetään WebSocket-rajapinnan kautta. Vaihtosuuntaajan komennoista ja voi lukea tarkemmin luvusta 5.4.5 vaihtosuuntaajan ohjauksen integrointi.

Nimellä **command** annetaan komentoja eri laitteille. Komento annetaan commad-kenttään WebSocket-viestissä ja kohde laite annetaan target-kenttään. Driver-kortille annettavaan komentoon pitää antaa type-kenttään instant arvo, jos komento halutaan suorittaa välittömästi. Muut laitteet suorittavat komennot välittömästi. Laitteiden komennoista voi lukea kappaleista 5.9 Controller, 5.10 Driver ja 5.11 Thermocouple.

Nimellä **reconfigure** voidaan uudelleen konfiguroida palvelimenasetukset. Reconfigure-viestissä pitää antaa parametreinä kaikki palvelimen konfiguroitavat arvot. Kuvassa 40 lähetetään reconfigure-komento käyttöliittymän kautta rajapintaan.

```
this.socket.emit('reconfigure', { //Send reconfigure command to server
    command: command,
    weather: this.state.weatherAPI,
    map: this.state.mapAPI,
    address: this.state.remoteServerAddress,
    controller1port: this.state.controller1port,
    controller2port: this.state.controller2port,
    driverPort: this.state.driver1port,
    thermoDevice: this.state.thermoDevice,
    temperatureLimit: this.state.temperatureLimit,
    voltageLimit: this.state.voltageLimit,
    interval: this.state.remoteUpdateInterval,
    mqttUName: this.state.mqttUName,
    mqttPWord: this.state.mqttPWord,
    handle: 'client',
    target: 'server'
  });
}
```

Kuva 40. Reconfigure komento käyttöliittymän kautta.

Nimellä **update** päivitetään Driver-, Thermocouple- ja Controller-kortit sekä rajapinta. Viestissä pitää antaa target-kenttään päivitettävät laitteet välilyönnillä erotettuna. Jos halutaan päivittää esimerkiksi Controller ja rajapinta, annetaan target kenttään tekstijono: controller server.

Nimellä thermalWarning vastaanotetaan lämpövaroitus-viesti Thermocouple-kortilta. Kuvassa 41 on esitetty hälytys viesti.

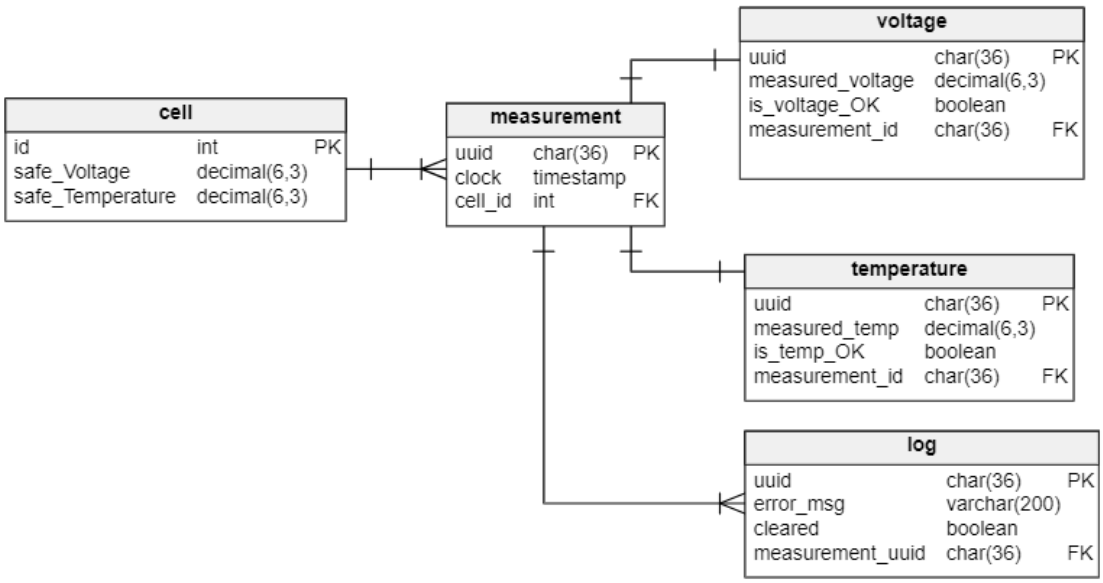
```
{"origin":"Thermocouple","type":"log","msg":"High temperature"}
```

Kuva 41. Thermocouple hälyytysviesti.

5.5 Etäpalvelin

5.5.1 Etäpalvelimen tietokannat

Etäpalvelimella on yksi tietokanta kerätyille mittaustuloksille ja yksi tietokanta käyttäjätunnuksille, joilla kirjaudutaan etäpalvelimen käyttöliittymään. Tietokantaohjelmistona käytetään PostgreSQL-relaatiotietokantaa. Tietokannat on suunniteltu vertabelo.com verkkosivulla, jossa voi graafisen käyttöliittymän avulla suunnitella tietokannan rakenteen ja ladata valmiin tietokannan luomislausekkeet. Kuvassa 42 on esitetty vertabelossa suunniteltu tietokanta mittausdatan säilyttämiseen. Jokaisella akkukennolla on tietokantaan annettu jännite- ja lämpötilaraja, mutta näitä tietoja ei lopullisessa työssä hyödynnetä. Myöskään lokitaulua ei lopullisessa työssä hyödynnetä. Tauluihin on jätetty ylimääräiset kentät, jotta tulevissa versioissa on mahdollista hyödyntää edellä mainittuja tietoja. Kennoilla voi olla monta mittausta tulosta (1:*) ja jokaisella mittauksella on aina yksi jännite- ja lämpötila-arvo (1:1), mutta lokiviestejä voi olla monia (1:*).



Kuva 42. Mittausdatan tietokanta

Etäpalvelimen pääohjelman riveillä 73-106 (Pirinen, vehicle-remote-server/index.js, 2019) on esitetty MQTT-tapahtuma, jossa etäpalvelimen rajapinta ottaa vastaan viestin ajoneuvon rajapinnasta ja kirjaa tulokset tietokantaan. MQTT-viestin vastaanotto on esitetty kuvassa 50. Riveillä 79-100 jokaisesta group listan olion temperature ja voltage listan elementistä luodaan SQL-lausekkeet, jotka liitetään toisiinsa. Rivillä 102 mittaustulokset asetetaan tietokantaan. Rajapinnassa käytetään node-postgres (pg) NPM-moduulia, joka toimii asiakkaana (engl. client) PostgreSQL tietokantaan.

Etäpalvelimen pääohjelman riveillä 148-207 (Pirinen, vehicle-remote-server/index.js, 2019) on esitetty getData REST-tapahtuman käsittely etäpalvelimen rajapinnalla. Rivillä 149 etsitään käyttäjätunnus Redis-tietokannasta ja rivillä 150 testataan vastaako REST-pyynnössä annettu tunnus ja avain Redis-tietokannan arvoja. Riveillä 151-177 rakennetaan SQL-lauseke, joka hakee kaikkien kennojen mittaustulokset käyttäjän antamien päivämäärien väliltä. Kun tulokset on saatu tietokannasta, muodostetaan niistä JSON-viesti riveillä 180-197. Kun viesti on muotoiltu oikein, lähetetään se käyttöliittymään.

users		
id	int	PK
email	text	N
password	text	N

Users-tietokantaan riittää tässä applikaatiossa sähköpostiosoite, jota käytetään kirjautumistunnuksena ja salasana. Salasanat on salattu PostgreSQL:n sisäänrakennetulla pgcrypt-moduulilla, jolloin salasanat eivät ole selkolukuisia. Käyttäjän tunnistuksesta voi lukea luvusta 5.5.1. Kuvassa 43 on esitetty users-tietokanta, joka on suunniteltu vertabelossa.

Kuva 43. User-tietokanta

5.5.1 Tunnistautuminen rajapintaan

Etäpalvelimen Web-pohjaiseen käyttöliittymään kirjaudutaan sähköpostilla ja salasanalla. Tunnukset on tallennettu palvelimella olevaan PostgreSQL-tietokantaan. Salasanojen salaukseen käytetään pgcrypto-moduulia. Suolan (engl. salt) luomiseen käytetään gen_salt-funktiota ja algoritmina käytetään blowfishtä.

Jotta käyttäjä pystyy tekemään pyyntöjä etäpalvelimen rajapintaa, tarvitsee käyttäjä authToken-avaimen, jolla voidaan tunnistautua pyyntöjen yhteydessä. Tämä avain saadaan tekemällä REST-pyyntö /auth etäpalvelimen rajapintaan. Parametreiksi annetaan email ja password. Rajapinta tarkistaa vastaatko tunnus tietokannassa olevaa tunnusta. Jos tunnus on oikein, rajapinta lisää käyttäjän Redis-tietokantaan. Redis-avaimena käytetään käyttäjän antamaa sähköpostiosoitetta ja arvoksi annetaan crypto-moduulilla luotu heksatekstijono. Jos kirjautuminen onnistui, käyttäjälle palautetaan authToken. Avainta authToken tarvitaan, jos käyttäjä haluaa tehdä muita REST-pyyntöjä etäpalvelimen rajapintaan. Kuvassa 44 on esitetty käyttäjätunnuksen tarkistus ja istunnon luominen etäpalvelimen rajapinnassa.

```
app.post('/auth', function (req, res) {
    let query = `SELECT EXISTS(
        SELECT true
        FROM users
        WHERE email = $1
        AND password = crypt($2, password)
        );`;

    auth.query(query, [`${req.body.email}`, `${req.body.password}`],
    (err, response) => {
        if (response.rows[0].exists) {
            let authToken = crypto.randomBytes(20).toString('hex');
            redisClient.set(req.body.email, authToken);
            redisClient.expire(req.body.email, 1800);

            res.cookie('session_key', authToken, { maxAge: 1800000 }).json(
                { "success": true, 'key': authToken }
            );
        } else {
            res.json({ 'success': false });
        };
    });
});
```

Kuva 44. Käyttäjä istunnon luominen etäpalvelimen rajapinnassa.

5.5.2 Etäpalvelimen REST-rajapinta

Etäpalvelimen rajapinnan ja etäkäyttöliittymän välinen liikenne suoritetaan REST-rajapinnalla. Käyttäjä tekee REST-pyynnön rajapintaan ja rajapinta vastaa pyyntöön. REST-rajapinnassa käyttöliittymän ja palvelimen välillä ei siis ole jatkuvaa yhteyttä.

Antamalla pelkästään palvelimen osoitteen https://osoite.net/, rajapinta palauttaa Web-käyttöliittymän.

Pyyntö https://osoite.net/auth tarkistaa ja kirjaa käyttäjän järjestelmään. Tämä pyyntö pitää tehdä POST-muodossa ja parametreina annetaan sähköpostiosoite email-kenttään ja salasana password-kenttään. Jos käyttäjän antamat tunnukset vastaavat tunnuksia tietokannassa, käyttäjälle palautetaan eväste ja JSON-viesti, jossa on avain, jonka avulla voidaan tehdä muita pyyntöjä rajapintaan. Jos tunnukset eivät vastaa, rajapinta palauttaa JSON-viestin, joka kertoo kirjautumisen epäonnistuneen. Tunnuksien tarkistus rajapinnassa on esitetty kuvassa 44.

Pyyntö https://osoite.net/getData palauttaa mittausdatan valitulta aikaväliltä. Jotta tämä pyyntö voidaan tehdä, käyttäjän pitää olla kirjautuneena palvelimella. Pyyntö tehdään POST-muodossa ja sille pitää antaa parametrina sähköpostiosoite user-kenttään, kirjautumisen yhteydessä saatu avain key-kenttään, mittaustulosten aloituspäivämäärä sDate-kenttään ja lopetuspäivänmäärä eDate-kenttään. Jos user- ja key-kenttä eivät vastaan palvelimen tietoja, pyyntöön ei vastata.

Pyynnöllä https://osoite.net/webasto annetaan komento, jolla voidaan käynnistää ja sammuttaa ajoneuvon lämmitin etänä. Jotta tämä pyyntö voidaan tehdä, käyttäjän pitää olla kirjautuneena palvelimella. Pyyntö tehdään POST-muodossa ja sille pitää antaa command-kenttään joko komento $!webasto, joka sammuttaa lämmittimen tai komento $webasto, joka käynnistää lämmittimen. Kenttään key annetaan kirjautumisen yhteydessä saatu avain ja user-kenttään sähköpostiosoite. Rajapinta lähettää webasto komennon MQTT-protokollan kautta paikallisen palvelimen rajapintaan, joka välittää komennon Driver-kortille.

5.6 TLS-sertifikaatti

Etäpalvelimen rajapinnan ja käyttöliittymä välinen yhteys on suojattu HTTPS:llä. Jotta HTTPS-yhteys on mahdollinen, tarvitaan siihen TLS-sertifikaatti. Let's Encrypt palvelu tarjoaa ilmaisen sertifikaatin, joka on voimassa 90 päivää. Tämän jälkeen sertifikaatti pitää uusia. Sertifikaatin uusiminen on myös ilmaista. Jotta Let's Encryptiä voidaan käyttää, pitää käyttäjän omistaa verkkotunnus (engl. domain), joka on ohjattu sille palvelimelle, jolle sertifikaatti halutaan käyttöön. Tässä kappaleessa on esitetty sertifikaatin asennus prosessi.

Palvelimella pitää olla asennettua certbot-ohjelma. Kun certbot on asennettu, annetaan kuvassa 45 esitetty komento, joka aloittaa sertifikaatin hankinta prosessin. Certbot pyytää verkkotunnukset ilman protokolla osaa. Jos verkkotunnuksia on useita, erotellaan ne pilkulla tai välilyönnillä.

```
root:~# certbot certonly --manual
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Please enter in your domain name(s) (comma and/or space separated)  (Enter
to'cancel):
```

Kuva 45. certbot pyytää domain nimiä. (Mellul, 2018)

Kun verkkotunnukset on annettu, certbot kysyy, onko IP-lokitus sallittua. Tähän vastataan kyllä. Tämän jälkeen palvelin ohjelman juureen pitää tehdä .well-known kansio ja sen sisään acme-challenge-kansio. Acme-challenge-kansion sisään tehdään a-string-tiedosto, jonka sisään kopioidaan certbotin antaman a-challenge-tekstijono. Kuvassa 46 on esitetty a-challenge, tekstijono on piilotettu turvallisuus syistä.

```
-------------------------------------------------------------------------------
Make sure your web server displays the following content at
http://yourdomain.com/.well-known/acme-challenge/a-string before continuing:

a-challenge

[You don't care about what's next]
-------------------------------------------------------------------------------
Press Enter to Continue
```

Kuva 46. Certbot antaa a-challenge tekstijonon. Turvallisuus syistä teksti on piilotettu tässä kuvassa. (Mellul, 2018)

Ennen kuin sertifikaatti prosessia jatketaan pitää varmistaa, että palvelin on päällä ja sieltä pystytään lataamaan a-string-tiedosto. Tämä voidaan testata antamalla selaimeen kuvassa 46 esitetty URL-osoite. Jos selain lataa tiedoston, voidaan sertifikaatti prosessia jatkaa. Kuvassa 47 sertifikaatti prosessi on suoritettu onnistuneesti loppuun.

```
Press Enter to Continue
Waiting for verification...
Cleaning up challenges
Generating key (2048 bits): /etc/letsencrypt/keys/0002_key-certbot.pem
Creating CSR: /etc/letsencrypt/csr/0002_csr-certbot.pem

IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at
   /etc/letsencrypt/live/yourdomain.com/fullchain.pem. Your cert
   will expire on 2018-07-21. To obtain a new or tweaked version of
   this certificate in the future, simply run certbot again. To
   non-interactively renew *all* of your certificates, run "certbot
   renew"
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le
```

Kuva 47. Sertifikaatin hankittu onnistuneesti. (Mellul, 2018)

Sertifikaatti-tiedostot löytyvät /etc/letsencrypt/live/chl650.net/ kansiosta. Täältä pitää löytyä kolme tiedostoa; privkey.pem, cert.pem ja chain.pem. Nämä tiedostot pitää antaa palvelimen koodissa, jos HTTPS halutaan käyttöön. Kuvassa 48 on esitetty osa rajapinnan ohjelmasta, jolla saadaan sertifikaatti käyttöön.

```
const https = require("https");
const fs = require("fs");
//...
const options = {
  key: fs.readFileSync("/etc/letsencrypt/live/chl650.net/privkey.pem"),
  cert: fs.readFileSync("/etc/letsencrypt/live/chl650.net/cert.pem"),
  ca: fs.readFileSync("/etc/letsencrypt/live/chl650.net/chain.pem")
};
//...
const app = express();
const server = https.createServer(options, app).listen(443, () => {
  console.log('Server started');
});
//...
```

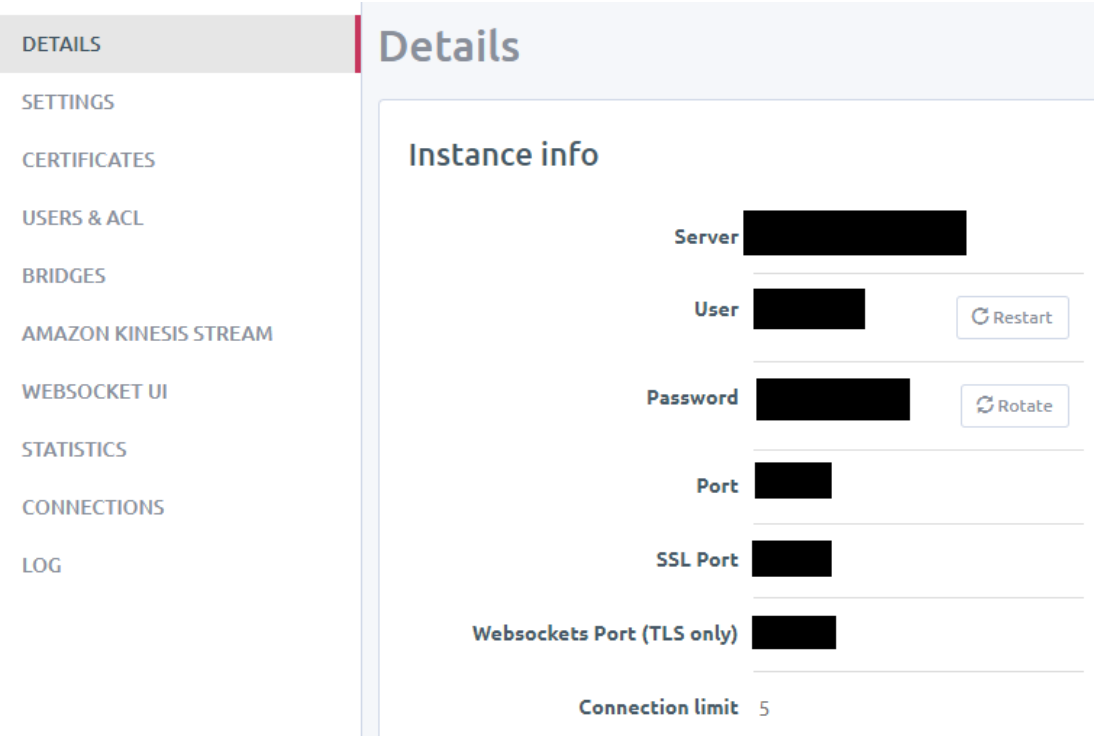
Kuva 48. Palvelin, joka kuuntelee porttia 443 ja on TLS-suojattu.

Sertifikaatti pitää päivittää vähintään 90 päivän välein. Tähän tarkoitukseen on tehty cert-renew.sh Bash-skripti (Pirinen, cert-renew.sh, 2019), joka suoritetaan crontabilla joka kuukauden ensimmäisenä päivänä. Crontabilla voidaan antaa Linuxille ajoitettuja tehtäviä. Skripti pysäyttää rajapinnan, suorittaa certbot renew -standalone komennon, joka hakee uuden sertifikaatin ja tämän jälkeen käynnistää rajapinnan uudestaan.

5.7 MQTT

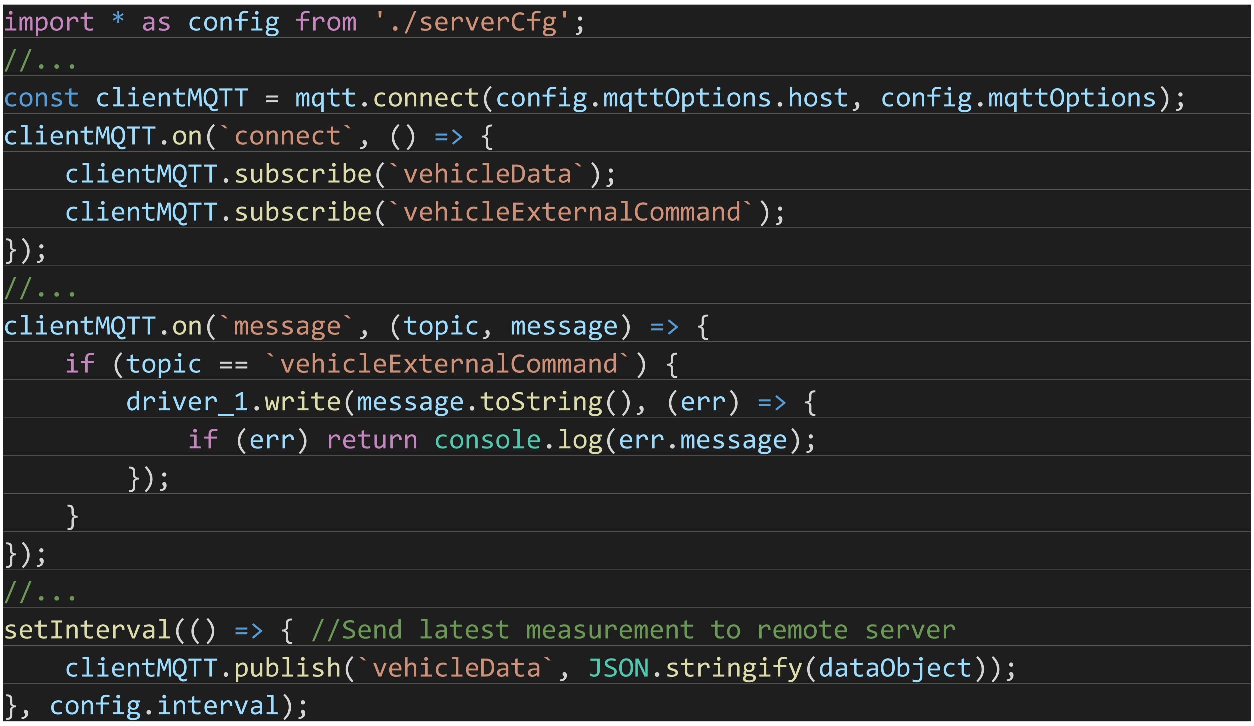
Rajapintojen välinen kommunikointi toteutetaan MQTT-protokollalla. Etä- ja paikallisella rajapinnalla käytetään mqtt.js-moduulia Node.js koodissa. MQTT-brokerina käytetään CloudMQTT palveluntarjoajaa. CloudMQTT:tä voi käyttää ilmaiseksi Cute Cat-paketilla. Paketin rajoitukset ovat 10Kbit/s ja 5 saman aikaista käyttäjää / sääntöä / yhteyttä (Plans | CloudMQTT, n.d). Yhden lähetettävän datapaketin koko alle yksi kilotavu, joten tämä on riittävä.

Kuvassa 49 on esitetty CloudMQTT-hallintapaneeli. Details-välilehdeltä löytyy MQTT-brokerin osoite ja portit. USERS & ACL-välilehdellä voidaan tehdä MQTT-tunnukset ja ACL-säännöt, joilla annetaan kirjoitus ja lukuoikeudet. CONNECTIONS-välilehdellä nähdään kaikki MQTT-brokeriin yhdistyneet laitteet ja niiden IP-osoitteet. LOG-välilehdellä nähdään loki, josta voidaan esimerkiksi nähdä minkä takia laite ei pystynyt yhdistymään MQTT-brokeriin.



Kuva 49. CloudMQTT hallintapaneeli.

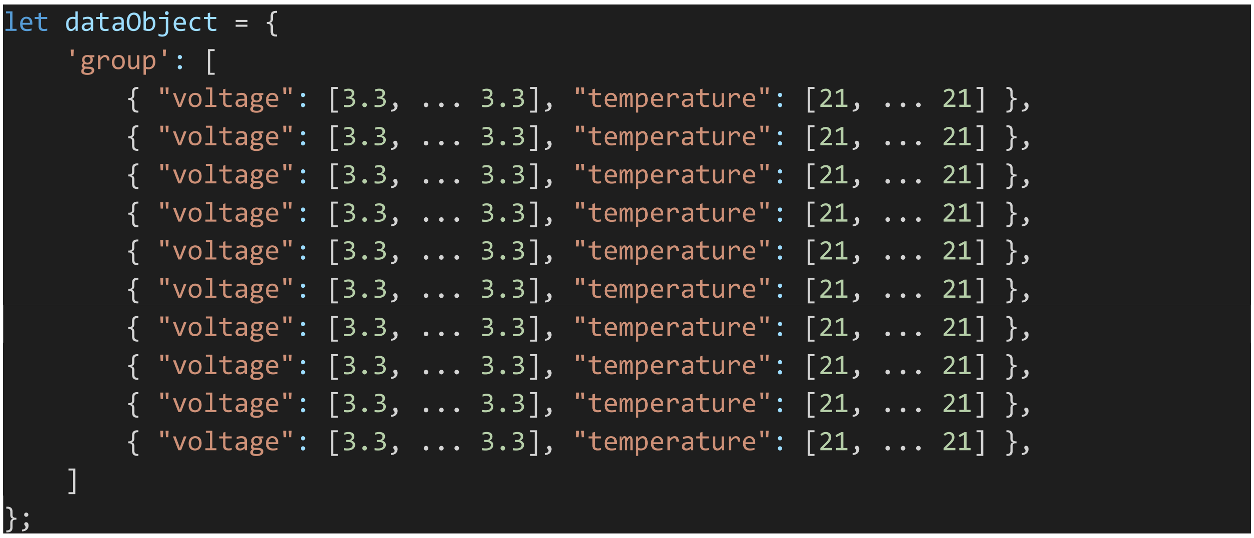
Kuvassa 50 on esitetty MQTT:n käyttö Node.js palvelimella. MQTT-asetukset tuodaan erillisestä serverCfg-tiedostosta. MQTT-yhteys muodostetaan rivillä 3. Rivillä 4 liitytään kanaviin vehicleData ja vehicleExternalCommand, kun yhteys MQTT-brokeriin on muodostettu. Kun palvelin vastaan ottaa MQTT-viestin ja sen kanava on vehicleExternalCommand, kirjoitetaan viesti Driver-kortin USB-väylään. Viimeisellä rivillä on esitetty viestin lähettäminen vehicleData-kanavaan. MQTT-viesti lähetetään sillä aikavälillä, joka on config-tiedostoon määritetty.



```
import * as config from './serverCfg';
//...
const clientMQTT = mqtt.connect(config.mqttOptions.host, config.mqttOptions);
clientMQTT.on(`connect`, () => {
    clientMQTT.subscribe(`vehicleData`);
    clientMQTT.subscribe(`vehicleExternalCommand`);
});
//...
clientMQTT.on(`message`, (topic, message) => {
    if (topic == `vehicleExternalCommand`) {
        driver_1.write(message.toString(), (err) => {
            if (err) return console.log(err.message);
        });
    }
});
//...
setInterval(() => { //Send latest measurement to remote server
    clientMQTT.publish(`vehicleData`, JSON.stringify(dataObject));
}, config.interval);
```

Kuva 50. MQTT:n käyttö Node.js palvelimella.

Kuvassa 51 on esitetty dataObjectin muotoilu. Rakenne dataObject on JSON-muodossa, ja se sisältää group listan. Jokaisessa listan indeksissä on olio (engl. object), joka sisältää listat voltage ja temperature. Yksi olio groupin sisällä vastaa yhtä akkukennoryhmää ja se sisältää viimeisimmät mittaustulokset akkukennoryhmästä.



```
let dataObject = {
    'group': [
        { "voltage": [3.3, ... 3.3], "temperature": [21, ... 21] },
        { "voltage": [3.3, ... 3.3], "temperature": [21, ... 21] },
        { "voltage": [3.3, ... 3.3], "temperature": [21, ... 21] },
        { "voltage": [3.3, ... 3.3], "temperature": [21, ... 21] },
        { "voltage": [3.3, ... 3.3], "temperature": [21, ... 21] },
        { "voltage": [3.3, ... 3.3], "temperature": [21, ... 21] },
        { "voltage": [3.3, ... 3.3], "temperature": [21, ... 21] },
        { "voltage": [3.3, ... 3.3], "temperature": [21, ... 21] },
        { "voltage": [3.3, ... 3.3], "temperature": [21, ... 21] },
        { "voltage": [3.3, ... 3.3], "temperature": [21, ... 21] },
    ]
};
```

Kuva 51. dataObject-olio, joka sisältää viimeisimmät mittaustulokset.

Kuvassa 52 on esitetty MQTT-asetukset, joita käytetään molemmilla palvelimilla. Tunnukset ja osoitteet on peitetty *-merkillä.

```
mqttOptions:{
    port: *****,
    host: '*********',
    clientId: 'mqttjs_' + Math.random().toString(16).substr(2, 8),
    username: '*******',
    password: '***********',
    keepalive: 60,
    reconnectPeriod: 1000,
    protocolId: 'MQTT',
    protocolVersion: 4,
    clean: true,
}
```

Kuva 52. MQTT-asetukset etäpalvelimella.

5.8 Käyttöliittymä

5.8.1 Komponentit

Käyttöliittymä koostuu useasta React-komponentista. React-komponentit ovat JavaScript-luokkia, jotka palauttavat JSX-koodia. Tässä luvussa on lueteltu jokainen työssä tehty React-komponentti ja niiden tehtävä käyttöliittymässä. Käyttöliittämän komponenttien elementit on tehty Material UI-kirjastolla. UI-kirjasto helpottaa työn tekoa siltä osin, että elementtien CSS-on tehty valmiiksi. Omaa CSS:ää joutuu tekemään sen verran että komponentit on sijoitettu oikealla tavalla käyttöliittymän.

Index.js (Pirinen, vehicle-ui/index.js, 2019) tiedosto on wrapper, jonka ainoa tehtävä on renderöidä React-komponentit RealDOMiin.

App (Pirinen, vehicle-ui/app.js, 2019) on käyttöliittymän tärkein komponentti. Sen statessa pidetään kaikki rajapinnasta saatava tieto ajoneuvosta sekä tiedot, joita pitää antaa alikomponenteille propseina. App-komponentin state-arvot ovat olemassa niin kauan kuin käyttöliittymä on avattuna selaimessa. App-komponentin alikomponentit menettävät paikallisen state-tilansa, kun ne irrotetaan (unmount). Esimerkiksi Main-komponentti ja sen paikalliset tilat eivät ole olemassa, kun käyttäjällä on valittuna Data-välilehti käyttöliittymästä. Jos alikomponenttien pitää vaihtaa App-komponentista state-arvoa, annetaan alikomponentille propsina updareParentState-funktio, jonka avulla on mahdollista vaihtaa mitä tahansa state-arvoa App-komponentista. App-komponentti hoitaa myös käyttöliittymän välilehtien vaihtelemisen switch-ehdon avulla. App-komponenttiin on tehty funktioita, joita käytetään muissa komponenteissa. Jotta näitä funktioita voidaan käyttää muissa komponenteissa, pitää ne sitoa (engl. bind) React-komponentin muodostimessa. Kun funktio on sidottu, voidaan se antaa käytettäväksi komponentille propsina. Kuvassa 53 on esitetty queryDB-funktion sitominen. queryDB-funktiota käytetään, kun käyttöliittymä on asetettu remote-tilaan. Sillä tehdään REST-pyyntö etäpalvelimen rajapintaan, joka pyytää mittausdataa käyttäjän valitsemalta aikavälilta, TimeLine-komponentissa.

```
import React, { Component } from 'react';
import Timeline from './components/timeline';
//...
class App extends Component {
    constructor(props) {
        super(props)
        //...
        this.queryDB = this.queryDB.bind(this); //Used @ timeline component
    }
    //...
    queryDB = (sDate, eDate) => {
        fetch(`https://${window.location.hostname}/getData`, {
          method: 'POST',
          credentials: 'same-origin',
          headers: { "Content-Type": "application/x-www-form-urlencoded" },
          body:
`sDate=${sDate}&eDate=${eDate}&user=${this.state.loggedInAs}&key=${this.state.sec
urityToken}`
        })
        //...
    }
    //...
    render(){
        return (
            //...
            <Timeline queryDB={this.queryDB} />
            //...
        )
    }
}
```

Kuva 53. Funktion sitominen (engl. bind) muodostimessa.

MapContainer- ja MapTab-komponentteja käytetään Google-mapsin esittämiseen. Komponentissa MapTab (Pirinen, vehicle-ui/mapTab.js, 2019), käytetään google-maps-react-moduulia, jonka kautta saadaan GoogleApiWrapper. Wrapper vaati toimiakseen aktiivisen GoogleMaps api-avaimen, jonka voi saada Google-kehittäjä konsolin kautta. Jotta kartta toimii, pitää laitteella olla internet-yhteys. Komponentissa MapContainer (Pirinen, vehicle-ui/MapContainer.js, 2019), annetaan GoogleMaps asetukset ja palautetaan renderille itse kartta. GoogleMapsia ei voi käyttää JavaScript-API:lla offline-tilassa, joten se korvataan mahdollisesti jollain muulla kartta API:lla, jota on mahdollista käyttää offline-tilassa, tulevissa käyttöliittymä versioissa.

DrawerList-komponentti (Pirinen, vehicle-ui/controlDrawerList.js, 2019) tekee välilehtilistan käyttöliittymän vasempaan laitaan. Oletusarvoisesti DrawerList-komponentti on esillä koko ajan, mutta jos laitteen näyttö on tarpeeksi kapea, DrawerList piilotetaan ja sen saa tällöin auki painamalla käyttöliittymän vasemmassa kulmassa olevaa painiketta. DrawerList-komponentissa, on painikkeet, joilla voidaan navigoida käyttöliittymässä eri

alueille. Alimpina painikkeina on palvelimen sammutusnappula ja rajapinnan uudelleen-käynnistysnappula.

DataLine-komponentti (Pirinen, vehicle-ui/dataLine.js, 2019) tekee viivadiagrammin sille annetuista datapisteistä. Diagrammilla annettava data pitää koostua listasta, joka sisältää objekteja, joissa on X- ja Y-arvot. X-arvoksi annetaan mittaus aika ja Y-arvoksi annetaan mitattutulos. Viivadiagrammi on tehty react-vis-kirjastolla. Jokaiselle kennoryhmälle on oma viiva graafissa. Yhdessä graafissa on esitetty 8 kennon mittaustulokset. Graafeja voidaan piilottaa ja niiden mittauspistehistorian pituutta voidaan säätää asetusvalikosta, jonne päästään käyttöliittymän vasemmassa reunassa olevan valikon kautta. Graafin alla on legend-osio, josta näkee mille kennolle graafin viiva kuuluu. Legend osion alla on nappula, josta voidaan manuaaliesti asettaa ryhmän jännitteentasaus, ON/OFF-tilaan. Nappula on aktiivinen vain silloin kun ajoneuvo on lataustilassa. DataLine-komponentilla voidaan esittää jännite- ja lämpötilamittaukset. Lämpötila voidaan esittää 0-150 celsiusasteen välillä ja kennojännite esitetään 2-4 voltin välillä. Jos komponentti esittää lämpötilamittauksia, sillä ei ole jännitteentasaus asetusnappulaa.

TimeLine-komponentin (Pirinen, vehicle-ui/timeline.js, 2019) kautta voidaan valita ajanjakso, jolta halutaan tarkastella kennojen jännitettä ja lämpötilaa. TimeLine-komponentti on saatavilla silloin kun käyttöliittymä on konfiguroitu remote-tilaan. Kun TimeLine-komponentista on valittu ajanjakso, käyttöliittymä tekee REST-pyynnön palvelimelle, joka hakee tietokannasta mittaustulokset.

GraphContainer-komponentti (Pirinen, vehicle-ui/graphContainer.js, 2019) palauttaa graafit sen mukaan onko ControllerDrawer-komponentista valittu Voltage (jännite) vai Temperature (lämpötila). GraphContainer tekee niin monta DataLine-komponenttia, kuin sille on sen paikallisessa state-arvossa määritetty. Jos käyttäjä on valinnut jännitteen, GraphContainer palauttaa jokaisen kennoryhmän DataLine-komponentin, ellei asetuksissa ole toisin määritetty. Jos käyttäjä valitsee lämpötilan, palautetaan Heatmap-komponentti ja DataLine-komponentit. DataLine-komponenteilla on esitetty Thermocouple-kortin mittaukset ja jokaisen kennon lämpötilamittaukset ryhmäkohtaisesti omissa komponenteissaan.

Heatmap-komponentti (Pirinen, vehicle-ui/heatmap.js, 2019) esittää lämpökartan ajoneuvon akkutilasta. Lämpökartta rakennetaan samasta datasta kuin viivadiagrammi. Lämpökartan avulla nähdään, jos kennojen lämpötila vaikuttaa vieressä oleviin kennoihin. Lämpötila on esitetty rgb värillä (rgb(0,0,255) – rgb(255,0,0)), jossa matalat lämpötilat näkyvät sinisenä ja korkea lämpötila punaisessa. Lämpökartan lämpötilarajoja voidaan muuttaa asetuksista ja värikartta skaalautuu asetusten mukaan. Heatmap-komponentti ei ole saatavilla, jos käyttöliittymä on konfiguroitu remote-tilaan.

InverterTab-, InverterCommands-, ja InverterManagmentTable-komponentit ovat vaihtosuuntaajan hallintaa varten. InverterTab (Pirinen, vehicle-ui/inverter.js, 2019) on wrapper-komponentti, joka palauttaa InverterCommands-, ja InverterManagment-Table-komponentit. InverterCommands-komponentin (Pirinen, vehicle-ui/inverterCommands.js, 2019) kautta annetaan komentoja, joilla voi käynnistää vaihtosuuntaaja manuaalitilassa, pysäyttää vaihtosuuntaaja, tallentaa vaihtosuuntaajan parametrit sen omaan Flash-muistiin, ladata parametriasetukset Flash-muistista, palauttaa

sen oletusasetukset ja lähettää komentoja tekstimuodossa. InverterManagmentTable-komponentissa (Pirinen, vehicle-ui/dataTable.js, 2019) on listassa kaikki vaihtosuuntaajan parametrit. Jos parametria painaa, listasta aukeaa konfiguraatio ikkuna, jossa on kuvaus parametrista, sen minimi, maksimi ja oletus arvo sekä kenttä, josta voidaan muokata sen arvoa. Vaihtosuuntaaja komponentit eivät ole saatavilla, jos käyttöliittymä on konfiguroitu remote-tilaan.

LogTab-komponentti (Pirinen, vehicle-ui/log.js, 2019) esittää kaikki lokiviestit laitteilta ja rajapinnasta. Lokiviestissä on esitetty lokin kellonaika, viesti tekstimuodossa ja viestin tärkeys kolmitasoisella asteikolla. Loki voidaan tyhjentää ja viestejä voidaan suodattaa tärkeystason perusteella. LogTab-komponentti ei ole saatavilla, jos käyttöliittymä on konfiguroitu remote-tilaan.

MainMenu-komponentin (Pirinen, vehicle-ui/main.js, 2019) kautta voidaan asettaa moottorin pyörimissuunta sekä vakionopeudensäätimen ja lämmittimen tilat. Kun jotakin tilaa vaihdetaan painamalla sitä osoittavaa kuvaketta MainMenu-komponentista, suoritetaan App-komponentista setDriverState-funktio, joka lähettää WebSocketilla rajapintaan komennon, jolla asetetaan Driver-kortin tila. Jos käyttäjä haluaa vaihtaa moottorin pyöritä tilaa, pitää käyttäjän painaa varmistukseksi fyysistä painiketta, joka on kytketty Driver-korttiin, näin vältytään siltä, että moottorin suuntaa ei vaihdeta vahingossa. Lämmitin ja vakionopeudensäädin asetetaan päälle välittömästi, kun painiketta on painettu käyttöliittymästä. Jos käyttöliittymä on konfiguroitu remote-tilaan, ainoastaan lämmitin voidaan asettaa MainMenu-komponentin kautta.

InterfaceSettings-, GraphConfiguration- ja ApiSettings-komponentit muodostavat käyttöliittymän asetusvalikon. InterfaceSettings-komponentti (Pirinen, vehicle-ui/settings.js, 2019) on wrapper, joka palauttaa GraphConfiguration- ja ApiSettings-komponentit. GraphConfiguration-komponentin (Pirinen, vehicle-ui/graphConfig.js, 2019) kautta voidaan piilottaa viivadiagrammeja ryhmäkohtaisesti, asettaa niiden päivitys nopeus reaaliaikaiseksi, minuutin tai 5 minuutin välein päivittyväksi. Myös lämpökartan raja-arvot voidaan asettaa GraphConfiguration-komponentin kautta. ApiSettings-komponentissa (Pirinen, vehicle-ui/apiSettings.js, 2019) voidaan muuttaa rajapinnan asetuksia, kuten Controller-kortin jänniterajoja, api-avaimia ja MQTT-asetuksia. ApiSettings-komponentti saa asetukset propseina ylemmän tason komponenteilta. Rajapinnan aktiivisena olevat asetukset tulevat App-komponentille siinä vaiheessa, kun WebSocket-yhteys on muodostettu. Uudet asetukset hyväksytään painamalla Reconfigure nappulaa, joka käyttää handleSystemCommand-funktiota App-komponentista. Tämä funktio lähettää uudet asetukset rajapintaan ja käynnistää restart.sh skriptin, joka päivittää konfiguraatiotiedoston ja uudelleen käynnistää palvelimen. ApiSettings-komponentti ei ole saatavilla, jos käyttöliittymä on konfiguroitu remote-tilaan.

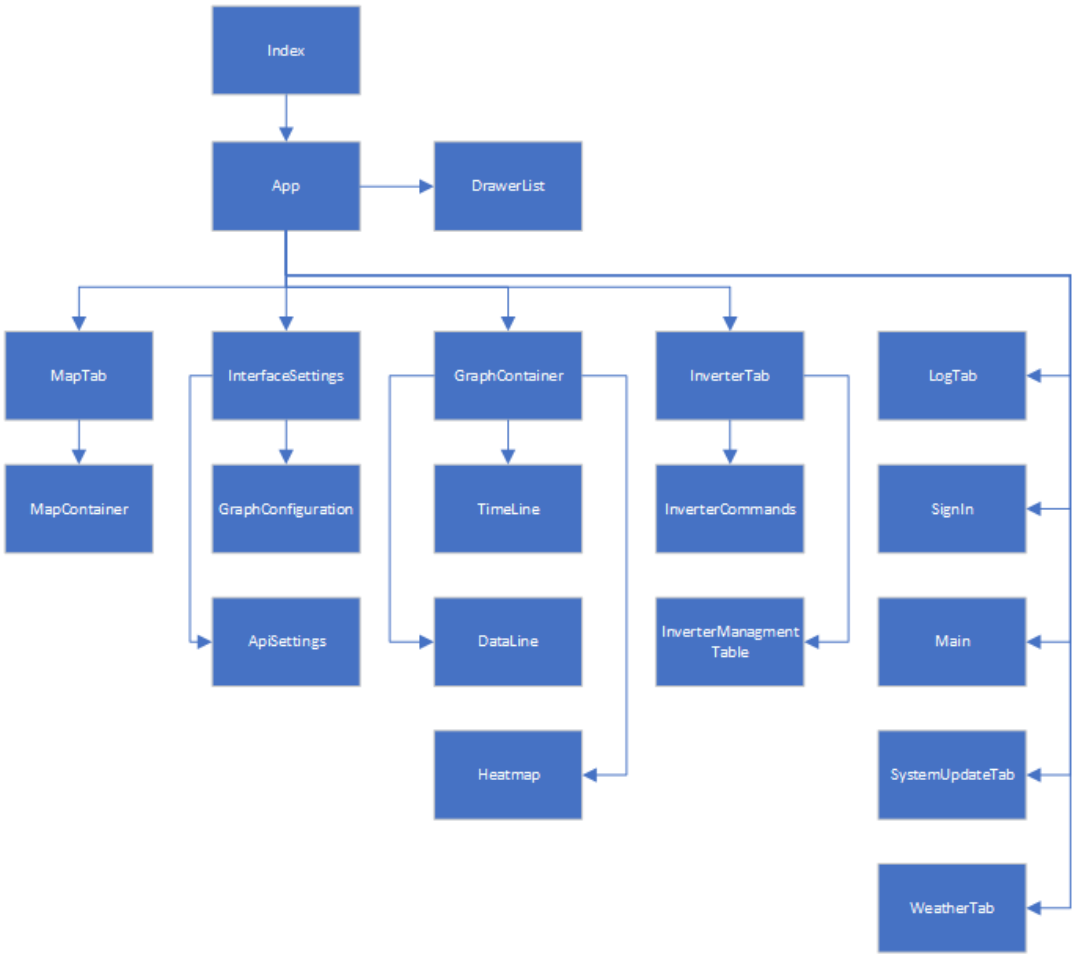
SignIn-komponentti (Pirinen, vehicle-ui/signIn.js, 2019) esitetään, kun käyttöliittymään pitää kirjautua sisään. Sitä käytetään vain, jos käyttöliittymä on konfiguroitu remote-tilaan. Komponentille annetaan tunnuksena sähköpostiosoite ja salasana. Käyttöliittymä lähettää REST-pyynnön palvelimelle, joka tarkistaa vastaavatko annetut tunnukset järjestelmän tietokannassa olevia tietoja. Tunnistautumisesta voi lukea tarkemmin luvuista tunnistautuminen rajapintaan 5.5.1 ja etäpalvelimen REST-rajapinta 5.5.2.

SystemUpdateTab-komponentin (Pirinen, vehicle-ui/systemUpdate.js, 2019) kautta voidaan päivittää Controller-, Thermocouple-, ja Driver-kortit sekä rajapinta. Laitteita voidaan päivittää useamman kerralla tai yksi kerrallaan. Komponentti näyttää milloin on viimeksi suoritettu päivitys prosessi. Silloin kun päivitys prosessi on käynnissä, komponentti näyttää animoitua latausmerkkiä, kunnes päivitys on valmis. Käyttöliittymä ladataan uudelleen, kun päivitykset on asennettu.

WeatherTab-komponentti (Pirinen, vehicle-ui/weather.js, 2019) esittää paikallisen sään ja viiden päivän ennusteen viivadiagrammilla. Säätiedot haetaan OpenWeatherMap-rajapinnasta. Ajoneuvon sijainti haetaan selaimen GPS-ominaisuudella. Paikallisessa säässä ilmoitetaan lämpötila, sään kuvaus (esim. sateen voimakkuus), ilmankosteus ja tuulen voimakkuus. Komponentti esittää myös säätilasta SVG-animaation, joka vastaa säätilaa. SVG-animaatiot on haettu amcharts.com sivustolta (Free animated SVG weather icons, n.d). Animaatiot on lisensoitu Creative Commons Attribution 4.0 lisenssillä.

5.8.2 Rakenne

Käyttöliittymä rakentuu luvussa 5.8.1 esitetyistä komponenteista. Komponentit ovat uudelleenkäytettäviä, esimerkiksi jokaiselle kennoryhmälle on oma DataLine-komponentti, jossa esitetään yksittäisen ryhmän omat jännite- tai lämpötilamittaukset. Kuvassa 54 on esitetty käyttöliittymän rakenne komponentti tasolla. DrawerList komponentti renderöidään aina, jos käyttöliittymä ei ole remote-tilassa. Jos käyttöliittymä on remote-tilassa, ainoa komponentti, joka renderöidään, on SignIn-komponentti, kunnes käyttäjä on onnistuneesti kirjautunut järjestelmään. App-komponentin alla olevat komponentit renderöidään sen perusteella mikä ikkuna on valittuna DrawerList-komponentista. Esimerkiksi MapTab- ja InterfaceSettings-komponentteja ei koskaan renderöidä samaan aikaan. MapTab-, ApiSettings-, Heatmap-, InverterTab-, LogTab- ja WeatherTab -komponentit eivät ole käytettävissä remote-tilassa. TimeLine- ja SignIn -komponentit ovat käytössä vain remote-tilassa. Liitteessä 1 on esitetty käyttöliittymän ohjeet, josta myös näkee miltä käyttöliittymä näyttää visuaalisesti.



Kuva 54. Käyttöliittymän rakenne komponenttitasolla.

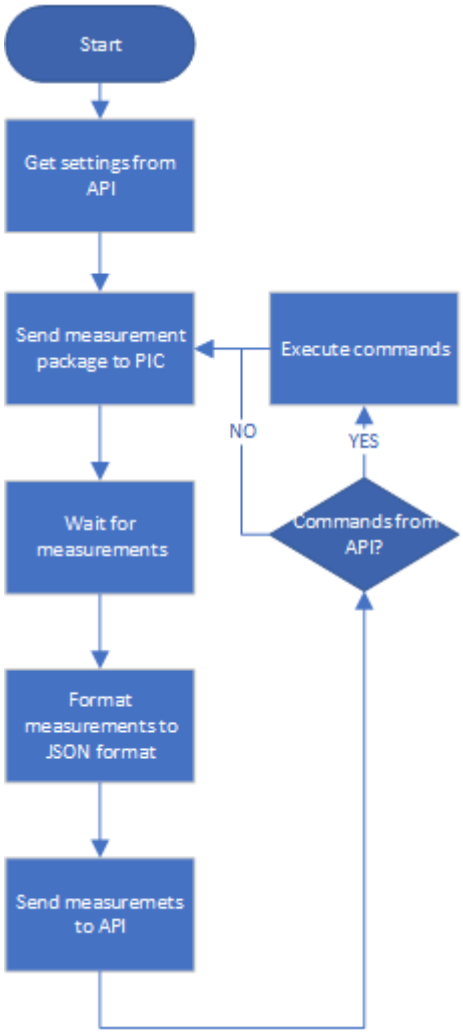
5.8.3 WebSocket

Tiedonsiirto käyttöliittymän ja rajapinnan välillä toteutetaan Socket.io-kirjastolla. Socket.io pystyy automaattisesti yhdistymään takaisin palvelimeen, jos yhteys katkeaa ja sillä voidaan käyttää nimiavaruuksia ja huoneita, joka helpottavat viestien jaottelua käyttöliittymässä ja rajapinnassa. Socket.io voidaan ottaa käyttöön käyttöliittymässä socket.io-client-moduulilla. Kuvassa 55 on esitetty WebSocketin käyttö käyttöliittymässä. Jotta kirjastoa voidaan käyttää, tarvitaan socket.io-client-moduulia. WebSocket-palvelin on samalla osoitteessa kuin käyttöliittymä, joten osoite saadaan Window-objektista. WebSocket-viestejä voidaan vastaanottaa on-metodilla ja antamalla aiheeksi (engl. topic) haluttu kanava. Kuvassa 55 vastaanotetaan viestejä kanavalla systemParam, jonka kautta saadaan rajapinnan ja vaihtosuuntaajan asetukset, kun käyttöliittymä on renderöity selaimeen. Kaikki WebSocket-kuuntelutapahtumat on sijoitettu React-metodiin componentDidMount, jolloin viestien kuuntelu alkaa vasta silloin kun käyttöliittymä on kokonaan renderöity ensimmäisen kerran. App-komponenttia ei irroteta (engl. unmount) koskaan, joten WebSocket-yhteydet ovat niin kauan saatavilla, kun käyttöliittymä on aktiivisena selaimessa.

```
import openSocket from 'socket.io-client';
//...
this.socket = openSocket(`https://${window.location.hostname}`);
//...
this.socket.on('systemParam', (data) => {
    let _message = JSON.parse(data.message.toString());
    //...
});
//...
handleSystemCommand = (command) => {
    this.socket.emit('reconfigure', {
        command: command,
        //...
    });
};
```

Kuva 55. Socket.io-kirjaston käyttöönotto käyttöliittymässä.

5.9 Controller-kortti

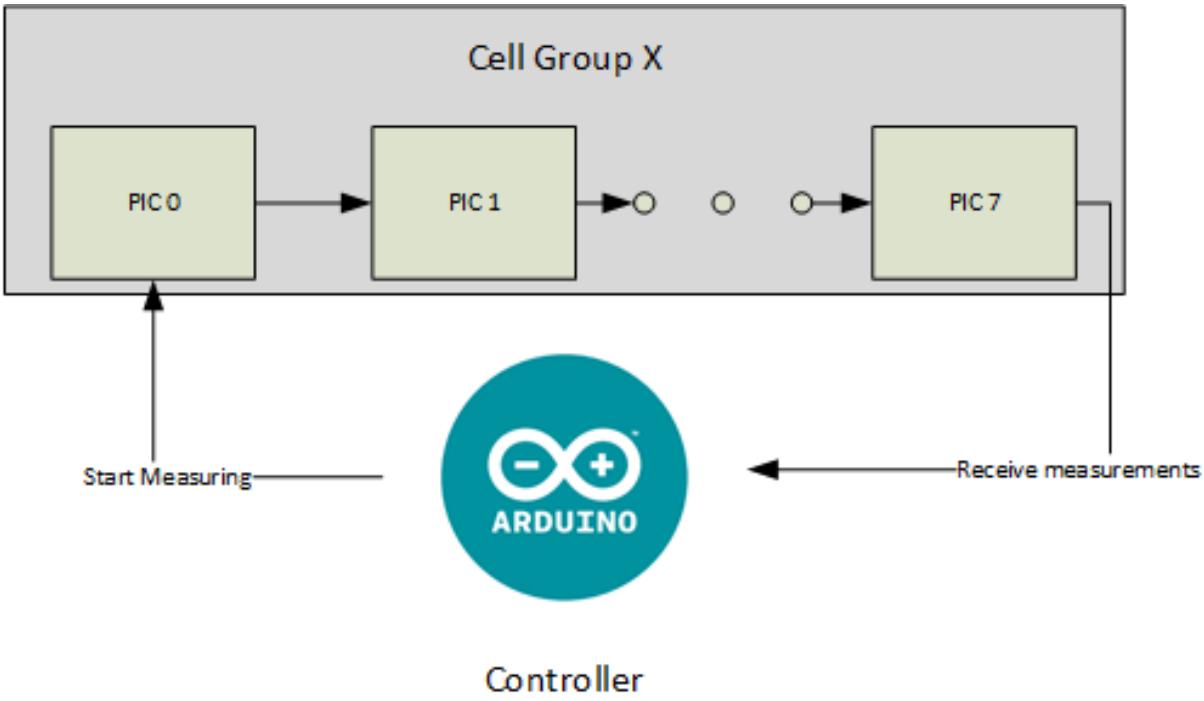


Controller-kortti antaa PIC-korteille komennot jännitteen ja lämpötilojen mittaukseen ja muotoilee mittaus tulokset JSON-muotoon. Mittaus tulokset lähetetään USB-väylän kautta rajapinnalle. Vikatilanteet tai tilamuutokset lähetetään myös JSON-muodossa rajapinnalle. Kun ajoneuvo on latauksessa, Controller-kortti asettaa kennojen jännitteentasaustilat ON/OFF-tilaan.

Kuvassa 56 on esitetty Controller-kortin koodin suoritus. Ohjelman alussa Controller-kortti pyytää rajapinnalta jännitteen maksimi latausrajan ja Controller-korttiin kytketyn ensimmäisen akkukennoryhmän numeron. Jos rajapinta ei vastaa kyselyyn, käytetään oletusarvoja. Pääohjelmassa lähetetään PIC-kortille mittauspyyntö, jonka jälkeen ohjelma odottaa viisi sekuntia vastausta. Yksi mittauskierros kestää noin 3 sekuntia. Mitattavia arvoja ovat lämpötilat ja jännitteet.

Kuvassa 57 on esitetty mittaustapahtuma. Controller lähettää tyhjän mittauspaketin ensimmäiselle PIC-kortille. PIC-kortti lisää omat mittaukset pakettiin ja lähettää tämän paketin seuraavalle PIC-kortille. Ryhmän viimeinen PIC-kortti lähettää valmiin paketin Controller-kortille.

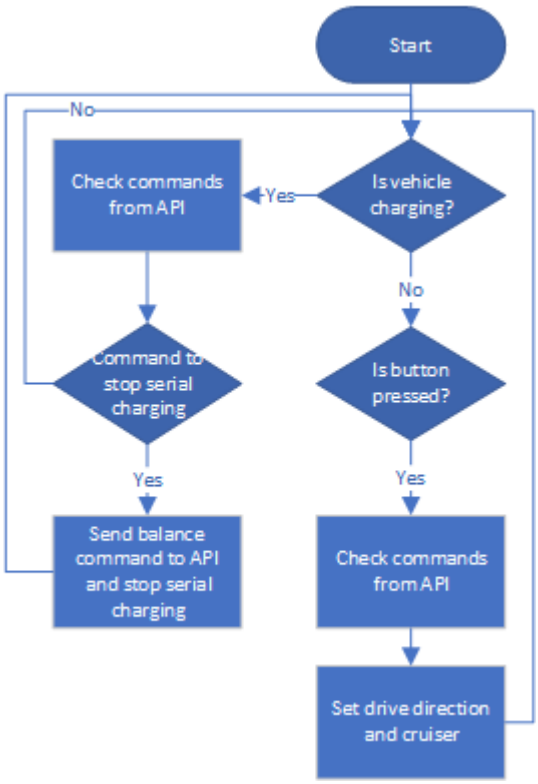
Kuva 56. Controller-kortin codeflow.



Kuva 57. Kennoryhmän mittaus. PIC-kortit on kytketty sarjaan.

Kun Controller-kortti vastaanottaa mittaustulokset viimeiseltä PIC-kortilta, muotoillaan tulokset JSON-muotoon. Mittaustulokset lähetetään rajapintaan. Viimeiseksi ohjelma tarkistaa onko rajapinnasta saatu komentoja. Controller-kortille voidaan antaa komentoja, jotka kytkevät jännitteentasausta ohjaavien releiden tilaa. Jos ajoneuvo on latausrajassa, Controller-kortti seuraa onko jonkin kennon jännite kasvanut yli latausrajan. Kun tämä tapahtuu, Controller-kortti lähettää viestin rajapintaan, joka pyytää Driver-korttia lopettamaan sarjalatauksen. Kun Controller-kortti on saanut vastauksen rajapinnasta, että sarjalataus on lopetettu, Controller-kortti asettaa kennot jännitteentasaustilaan. Edellä kuvaillut toiminnot toistetaan jokaiselle kennoryhmälle. Yhdessä ryhmässä on 8 akkukennoa ja PIC-mittauskorttia.

5.10 Driver-kortti



Driver-kortti asettaa ajoneuvon laitteita ON/OFF-tilaan. Kuvassa 58 on esitetty Driver-kortin koodin toiminta.

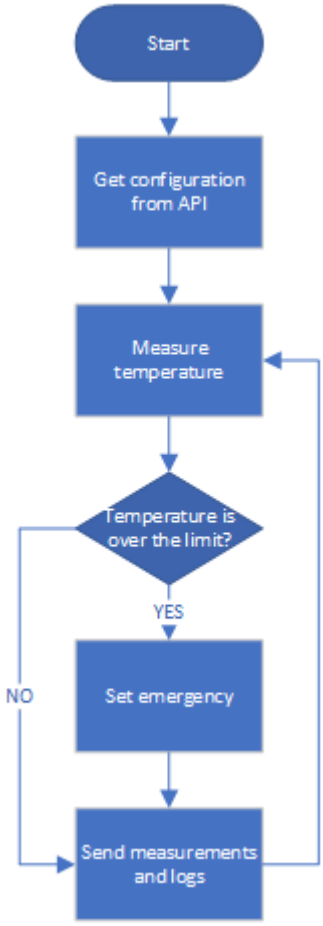
Driver-kortin kautta voidaan asettaa moottori pyörintäsuunta tai asettaa se vapaalle. Ajoneuvon moottorin suunta asetetaan käyttöliittymän kautta. Kun haluttu suunta on valittu, pitää painaa varmistuskytkintä, jonka jälkeen Driver-kortti tekee pyynnön asetetuista arvoista rajapintaan. Kytkin on laitettu varotoimenpiteeksi, jotta käyttöliittymän kautta ei voi vahingossa vaihtaa ajoneuvon moottorin pyörintäsuuntaa.

Ajoneuvon lämmitin voidaan asettaa ON/OFF-tilaan. Lämmittimen tilaa voidaan vaihtaa ajoneuvon käyttöliittymän kautta, tai etäpalvelimen kautta. Lämmittimen kytkentään ei tarvitse painaa varmistuskytkintä.

Kuva 58. Driver-kortin codeflow.

Driver-kortti myös seuraa onko ajoneuvo lataustilassa. Kun ajoneuvo on latauksessa ja Controller-kortti havaitsee, että jännite on saavuttanut sarjalatausrajan, voidaan aloittaa siirtyminen jännitteentasaustilaan. Controller-kortti lähettää tästä viestin rajapintaan ja rajapintaa välittää viestin Driver-kortille, joka kytkee sarjalatauspinnin HIGH-tilaan. Tämän jälkeen Driver-kortti lähettää viestin rajapintaan, joka kertoo sarjalatauksen päättyneen ja rajapinta välittää tämän viestin Controller-kortille. Kun Controller-kortti on vastaanottanut varmistusviestin, kennojen jännitteen tasaaminen voidaan aloittaa.

5.11 Thermocouple-kortti



Thermocouple-kortti mittaa ja raportoi moottorin lämpötilat rajapintaan. Lämpötilan mittaukseen käytetään kahta MAX31855-termoparimoduulia. Lämpötila mitataan moottorista, kahdesta eri pisteestä. Moduulin kanssa kommunikointiin käytetään Adafruit_MAX31855-kirjastoa.

Kuvassa 59 on esitetty Thermocouple-kortin toiminta. Ohjelman alussa Thermocouple-kortti pyytää rajapinnasta lämpötilaraja-arvon. Jos rajapinta ei vastaa pyyntöön käytetään oletuslämpötila raja-arvona 125 celsius astetta. Kun raja-arvo on asetettu, siirrytään pääohjelmaan. Pääohjelman alussa MAX31855-moduuleilta luetaan lämpötila celsiusasteina. Seuraavaksi lämpötilaa verrataan raja-arvoon, jos raja-arvo ylitetään, asetetaan Arduinosta pinni 5 HIGH-tilaan, joka kytkee moottorin hätäsammutuksen. Kun lämpötila putoaa 5 celsiusastetta alle raja-arvon, hälytys kytketään pois päältä asettamalla pinni 5 LOW-tilaan. Thermocouple-kortti lähettää mittaus arvot ja lokitiedot rajapintaan viiden sekunnin välein.

Kuva 59. Thermocouple-kortin codeflow

6 YHTEENVETO JA KEHITETTÄVÄÄ

Opinnäytetyön tärkeimpinä tavoitteina oli tuottaa ajoneuvoon järjestelmä, jonka avulla on mahdollista hallita ja lukea sähköajoneuvon akkukennojen tilaa sekä hallita moottorin ja vaihtosuuntaajan tilaa. Ajoneuvon elektroniikkaa oli suunniteltu ja toteutettu opinnäytetilaajan puolesta osittain siinä vaiheessa, kun opinnäytetyö aloitettiin. Työn alussa opinnäytteen tilaaja oli valmistanut prototyypin akkujen jännitteen ja lämpötilan mittauslaitteista PIC-mikro-ohjaimilla ja Arduinolla (controller). Elektroniset mittaus- ja hallintalaitteet rakennettiin Arduino-kehitysalustojen ympärille. Mittaus ja hallintalaitteita tuotettiin kolme erilaista, joilla jokaisella on oma tehtävä järjestelmässä. Raspberry Pi-laitetta käytetään palvelinlaitteena paikalliselle rajapinnalle, joka toteutettiin Node.js:llä. Paikallisen rajapinnan tehtävänä on mahdollistaa liikenne käyttöliittymän ja mittaus- ja hallintalaitteiden välillä, mittaustulosten raportointi julkisessa verkossa olevaan toiseen rajapintaan MQTT-yhteydellä sekä ohjelmien päivittäminen. Julkisessa verkossa oleva rajapinta kerää mittausdataa ja tallentaa ne PostgreSQL-tietokantaan sekä

mahdollistaa ajoneuvon lämmittimen tilan asettamisen etänä. Etäpalvelimella ja paikallisella palvelimella käytetään React.js:llä toteutettua käyttöliittymää. Käyttöliittymässä käytetään Material UI-kirjastoa, jonka avulla käyttöliittymässä voidaan käyttää Material Desing-tyypin elementtejä. Käyttöliittymän tärkeimpänä tehtävänä on mahdollistaa omien laitteiden ja Johannes Hübnerin vaihtosuuntaajan tilan ohjaamisen ja lukemisen. Käyttöliittymän kautta voidaan myös päivittää latteiden ohjelmat ja konfiguroida rajapintaa. Lisäksi käyttöliittymässä on mahdollista lukea Google Maps-karttaa ja paikallista säätietoa.

Opinnäytetyössä onnistuttiin tuottamaan järjestelmä asiakkaan tarpeiden mukaan, ja sitä on helppo päivittää tarvittaessa. Opinnäytetyötä on mahdollista parantaa monella osa-alueella tulevaisuudessa. Esimerkiksi käyttöliittymä voidaan uudelleen rakentaa Qt-kehitysympäristössä, jolloin käyttöliittymä olisi natiivi ja se voitaisiin kääntää helposti monelle eri käyttöjärjestelmälle. Raspberry Pi 3 B+ voidaan korvata esimerkiksi Raspberry Pi Zero W-laitteella, joka olisi kooltaan ja virran kulutukselta pienempi, mutta tällöin sarjaliikenne pitäisi siirtää GPIO-pinneihin. Ajoneuvosta voitaisiin myös kerätä enemmän dataa tietokantaan, esimerkiksi paikkatietoja.

LÄHTEET

4G LTE Mobile Wi-Fi M7300. (n.d). Haettu 6.3.2019 osoitteesta https://www.tp-link.com/en/products/details/cat-4692_M7300.html

About Node.js. (n.d). Haettu 6.3.2019 osoitteesta https://nodejs.org/en/about/

Arduino - FAQ. (n.d). Haettu 6.3.2019 osoitteesta https://www.arduino.cc/en/main/FAQ#toc13

Arduino Nano. (n.d). Haettu 6.3.2019 osoitteesta https://store.arduino.cc/arduino-nano

B_E_N. (n.d). *What is and Arduino?* Haettu 6.3.2019 osoitteesta https://learn.sparkfun.com/tutorials/what-is-an-arduino/all

Blocking vs Non-Blocking. (n.d). Haettu 6.3.2019 osoitteesta https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/

Boulmakoul, A. (12.2016). *Smart Data Collection Based on IoT Protocols*. Haettu 6.3.2019 osoitteesta https://www.researchgate.net/figure/Publish-Subscribe-structure-MQTT-network-contains-an-MQTT-broker-which-mediates_fig1_320865914

Capan, T. (n.d). *Why The Hell Would I Use Node.js A Case-by-Case Tutorial*. Haettu 6.3.2019 osoitteesta https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

Chadwick, R. (n.d). *If statements - Bash Scripting Tutorial*. Haettu 6.3.2019 osoitteesta https://ryanstutorials.net/bash-scripting-tutorial/bash-if-statements.php#test

Components and Props. (n.d). Haettu 6.3.2019 osoitteesta https://reactjs.org/docs/components-and-props.html

Craete a New React App. (n.d). Haettu 6.3.2019 osoitteesta https://reactjs.org/docs/create-a-new-react-app.html

DNA Laitenetti. (n.d). Haettu 6.3.2019 osoitteesta https://kauppa4.dna.fi/c/DNA-Laitenetti/p/QDS00001

Free animated SVG weather icons. (n.d). Haettu 6.3.2019 osoitteesta https://www.amcharts.com/free-animated-svg-weather-icons/

Giant, A. T. (n.d). *Introduction to MQTT*. Haettu 6.3.2019 osoitteesta https://learn.sparkfun.com/tutorials/introduction-to-mqtt/all

goliatone. (3.3.2019). *arpscan*. Haettu 6.3.2019 osoitteesta https://www.npmjs.com/package/arpscan

Gonzalo, F. (1.3.2017). *Understanding the difference between mobile-first, adaptive and responsive design*. Haettu 6.3.2019 osoitteesta http://fredericgonzalo.com/en/2017/03/01/understanding-the-difference-between mobile-first-adaptive-and-responsive-design/

HawaiianPi. (1.2.2018). *Difference in raspbian stretch with desktop and raspbian stretch lite*. Haettu 6.3.2019 osoitteesta https://www.raspberrypi.org/forums/viewtopic.php?t=204204

Hüebner, J. (n.d). *Inverter Parameters*. Haettu 6.3.2019 osoitteesta http://johanneshuebner.com/quickcms/index.html%3Fen_parameters,24.html

Hüebner, J. (n.d). *Inverter Kit Features*. Haettu 6.3.2019 osoitteesta http://johanneshuebner.com/quickcms/index.html%3Fen_features,8.html

Kangax, w. &. (n.d). *ECMAScript compatibly tabke*. Haettu 6.3.2019 osoitteesta http://kangax.github.io/compat-table/es5/

Khatri, S. (5.11.2016). *What does calling super() in a React constructor do?* Haettu 6.3.2019 osoitteesta stackoverflow.com: https://stackoverflow.com/questions/40433463/what-does-calling-super-in-a-react constructor-do

Kumar, A. (9.5.2018). *React Component Lifecycle*. Haettu 6.3.2019 osoitteesta http://blogs.innovationm.com/react-component-lifecycle/

Lerner, A. (n.d). *Fullstack React: What is JSX?* Haettu 6.3.2019 osoitteesta https://www.fullstackreact.com/30-days-of-react/day-2/

Mellul, D. (22.4.2018). *Node + Express + LetsEncrypt : Generate a free SSL certificate and run an HTTPS server in 5 minutes or less*. Haettu 6.3.2019 osoitteesta https://itnext.io/node-express-letsencrypt-generate-a-free-ssl-certificate-and-run-an https-server-in-5-minutes-a730fbe528ca

Mishra, R. (7.5.2017). *Virtual DOM in ReactJS*. Haettu 6.3.2019 osoitteesta https://hackernoon.com/virtual-dom-in-reactjs-43a3fdb1d130

Mokkuloiden ja päätelaitteiden APN-asetukset. (n.d). Haettu 6.3.2019 osoitteesta https://www.dna.fi/documents/753910/853450/DNA_Mokkuloiden_ja_paatelaitteid n_APN-asetukset.pdf/70d0801a-27c1-a63c-67b5-5698ddefbbde

MQTT Frequently Asked Questions. (n.d.). Haettu 6.3.2019 osoitteesta http://mqtt.org/faq

Pirinen, H. (1.12.2018). *softwareUpdate.sh*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-server/blob/master/softwareUpdate.sh

Pirinen, H. (3.3.2019). *cert-renew.sh*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-server-remote/blob/master/cert-renew.sh

Pirinen, H. (3.3.2019). *Henri Pirinen's repositories*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen?tab=repositories

Pirinen, H. (3.3.2019). *vehicle-remote-server/index.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-server-remote/blob/a8fb6637b44a887724144e98d2ed3419733ae032/index.js#L73

Pirinen, H. (4.3.2019). *vehicle-ui/apiSettings.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/apiSettings.js

Pirinen, H. (4.3.2019). *vehicle-ui/app.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/App.js

Pirinen, H. (4.3.2019). *vehicle-ui/controlDrawerList.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/controlDrawerList.js

Pirinen, H. (4.3.2019). *vehicle-ui/dataLine.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/dataLine.js

Pirinen, H. (4.3.2019). *vehicle-ui/dataTable.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/dataTable.js

Pirinen, H. (4.3.2019). *vehicle-ui/graphConfig.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/graphConfig.js

Pirinen, H. (4.3.2019). *vehicle-ui/graphContainer.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/graphContainer.js

Pirinen, H. (4.3.2019). *vehicle-ui/heatmap.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/heatmap.js

Pirinen, H. (3.3.2019). *vehicle-ui/index.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/master/src/index.js

Pirinen, H. (4.3.2019). *vehicle-ui/inverter.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/inverter.js

Pirinen, H. (4.3.2019). *vehicle-ui/inverterCommands.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/inverterCommand.js

Pirinen, H. (4.3.2019). *vehicle-ui/log.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/log.js

Pirinen, H. (4.3.2019). *vehicle-ui/main.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/main.js

Pirinen, H. (4.3.2019). *vehicle-ui/MapContainer.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/MapContainer.js

Pirinen, H. (4.3.2019). *vehicle-ui/mapTab.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/mapTab.js

Pirinen, H. (4.3.2019). *vehicle-ui/settings.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/settings.js

Pirinen, H. (4.3.2019). *vehicle-ui/signIn.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/signIn.js

Pirinen, H. (4.3.2019). *vehicle-ui/systemUpdate.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/systemUpdate.js

Pirinen, H. (4.4.2019). *vehicle-ui/timeline.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/timeline.js

Pirinen, H. (4.3.2019). *vehicle-ui/weather.js*. Haettu 6.3.2019 osoitteesta https://github.com/HenriPirinen/vehicle-ui/blob/855466850c042d3612543f82e795daa3872f9b6c/src/components/weather.js

Plans | CloudMQTT. (n.d). Haettu 6.3.2019 osoitteesta: https://www.cloudmqtt.com/plans.html

Pricing on DigitalOcean - Cloud virtual machine & storage pricing. (n.d). Haettu 6.3.2019 osoitteesta
https://www.digitalocean.com/pricing/

raspberrypi.org. (n.d). Haettu 6.3.2019 osoitteesta
https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

React.Component. (n.d). Haettu 6.3.2019 osoitteesta
https://reactjs.org/docs/react-component.html

Running Tests. (n.d). Haettu 6.3.2019 osoitteesta
https://facebook.github.io/create-react-app/docs/running-tests

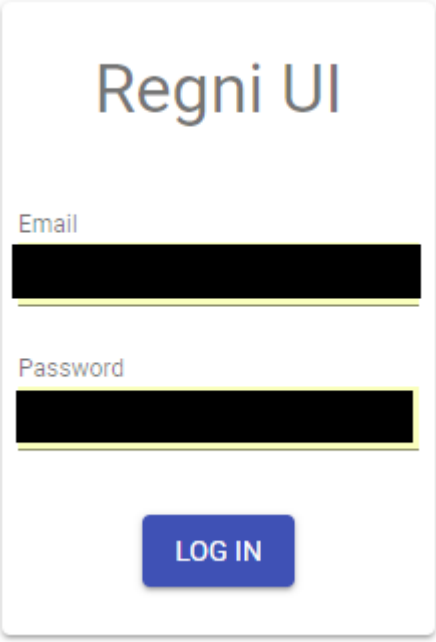
State and Lifecycle. (n.d). Haettu 6.3.2019 osoitteesta
https://reactjs.org/docs/state-and-lifecycle.html

State and Lifecycle. (n.d). Haettu 6.3.2019 osoitteesta
https://reactjs.org/docs/state-and-lifecycle.html

Suzdalnitski, I. (4.9.2018). *React.js: a better introduction to the most powerful UI library ever created.* Haettu 6.3.2019 osoitteesta
https://hackernoon.com/react-js-a-better-introduction-to-the-most-powerful-ui library-ever-created-ecd96e8f4621

virtual-dom-update.png. (n.d). Haettu 6.3.2019 osoitteesta
https://s3.amazonaws.com/media-p.slid.es/uploads/63938/images/2379617/ virtual-dom-update.png

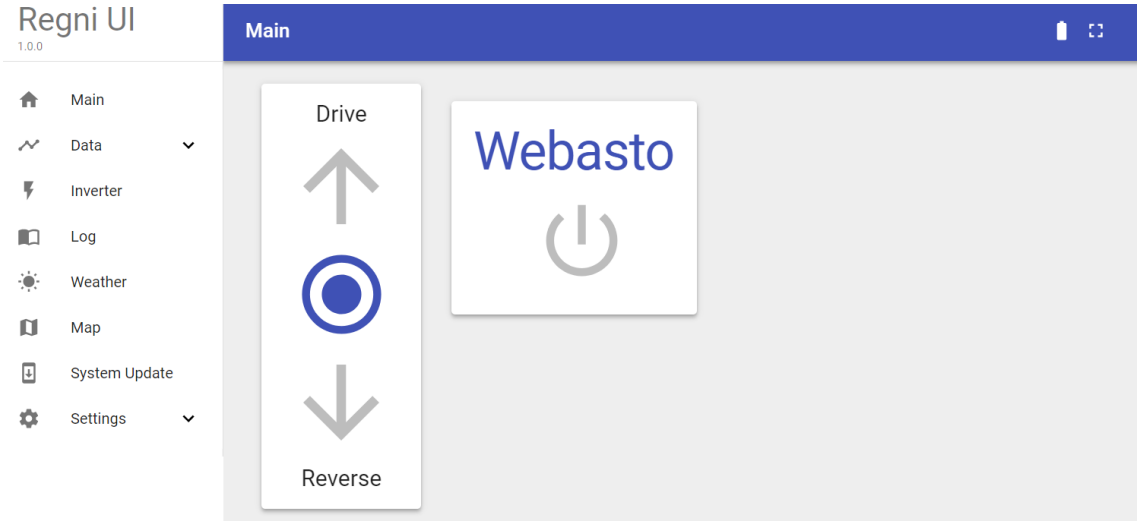
Liite 1



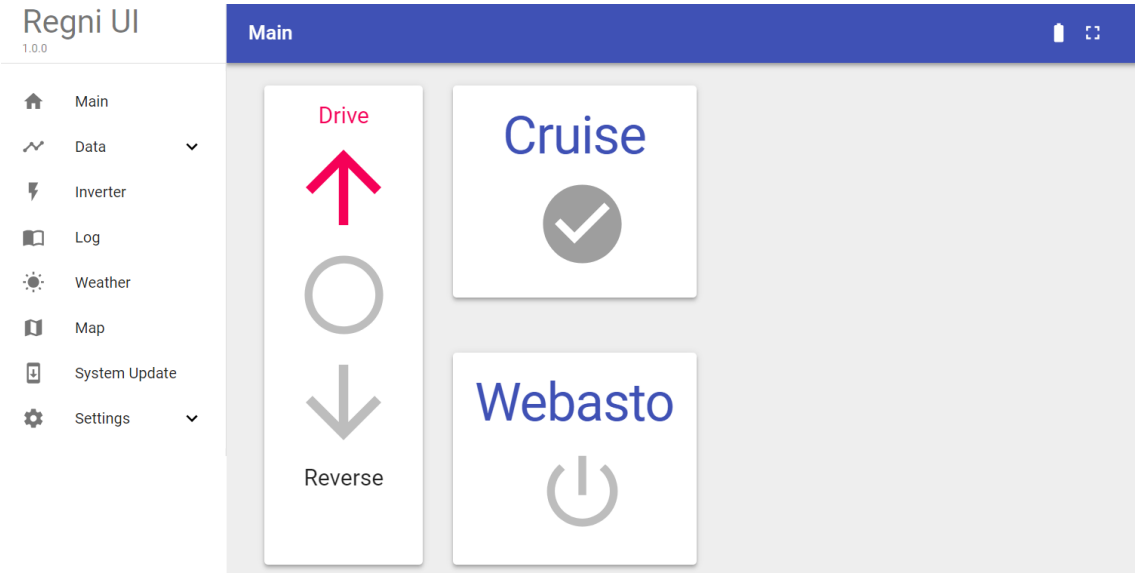
KÄYTTÖLIITTYMÄN OHJEET

Paikalliseen käyttöliittymään päästään selaimella antamalla osoitekenttään https://192.168.0.10.

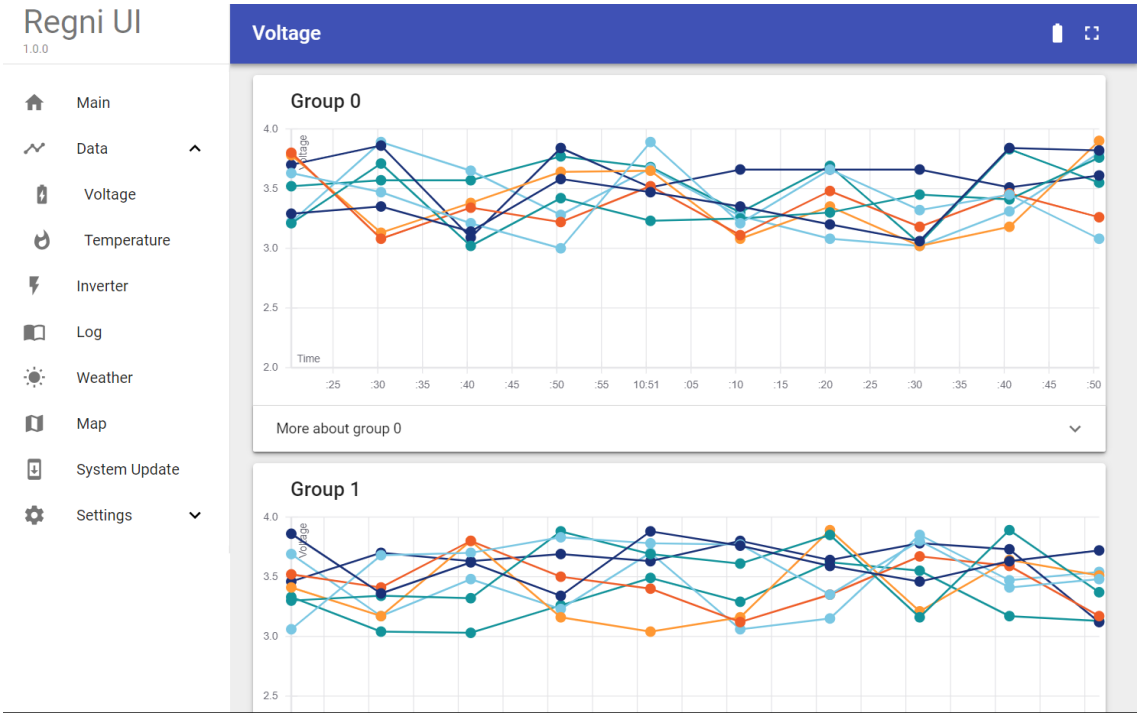
Etäkäyttöliittymään päästään antamalla selaimen osoitekenttään https://chl650.net



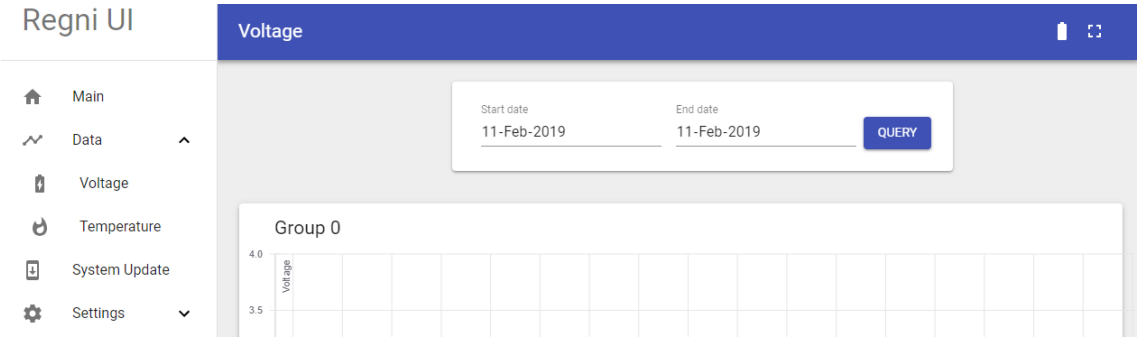
Käyttöliittymä aukeaa Main-välilehdelle, jossa voidaan asettaa moottorin pyörintäsuunta, lämmittimen(webasto)- ja vakionopeudensäätimentila. Etäkäyttöliittymässä Main-välilehdellä on vain lämmitinasetus. Moottori voidaan asettaa neutraaliin tilaan painamalla ympyräikonia. Eteenpäin päästään painamalla nuolta, joka osoittaa ylöspäin. Taaksepäin päästään painamalla nuolta, joka osoittaa alaspäin. Jos ajoneuvon moottori on neutraalissa tilassa, vakionopeudensäätimen painike on piilotettu.



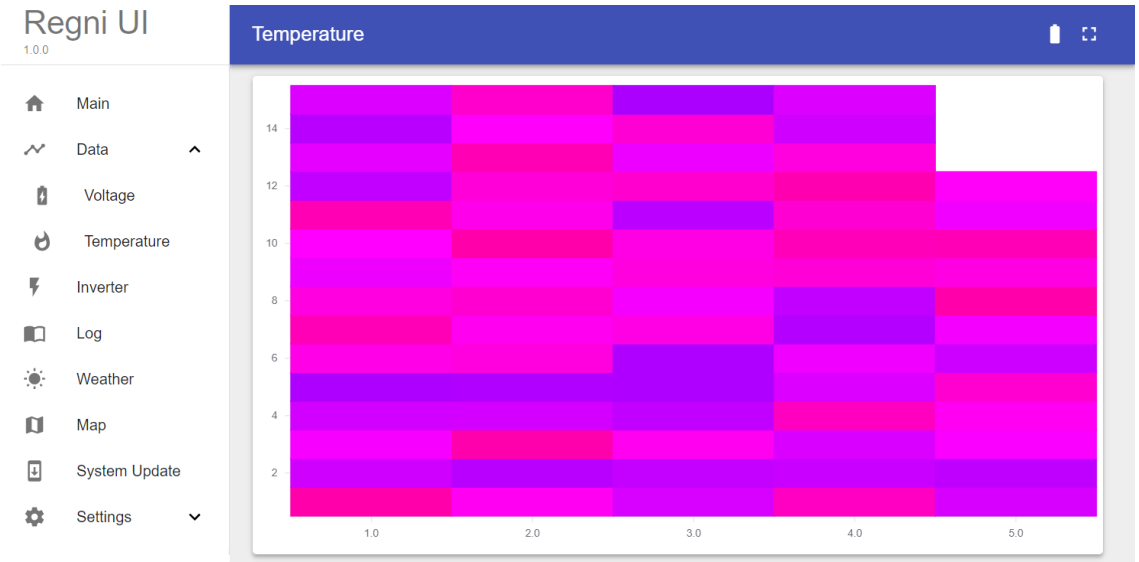
Kun ikonin väri on punainen, tila ei ole vielä aktiivinen. Jotta muutos aktivoidaan, paina fyysistä Execute nappulaa.



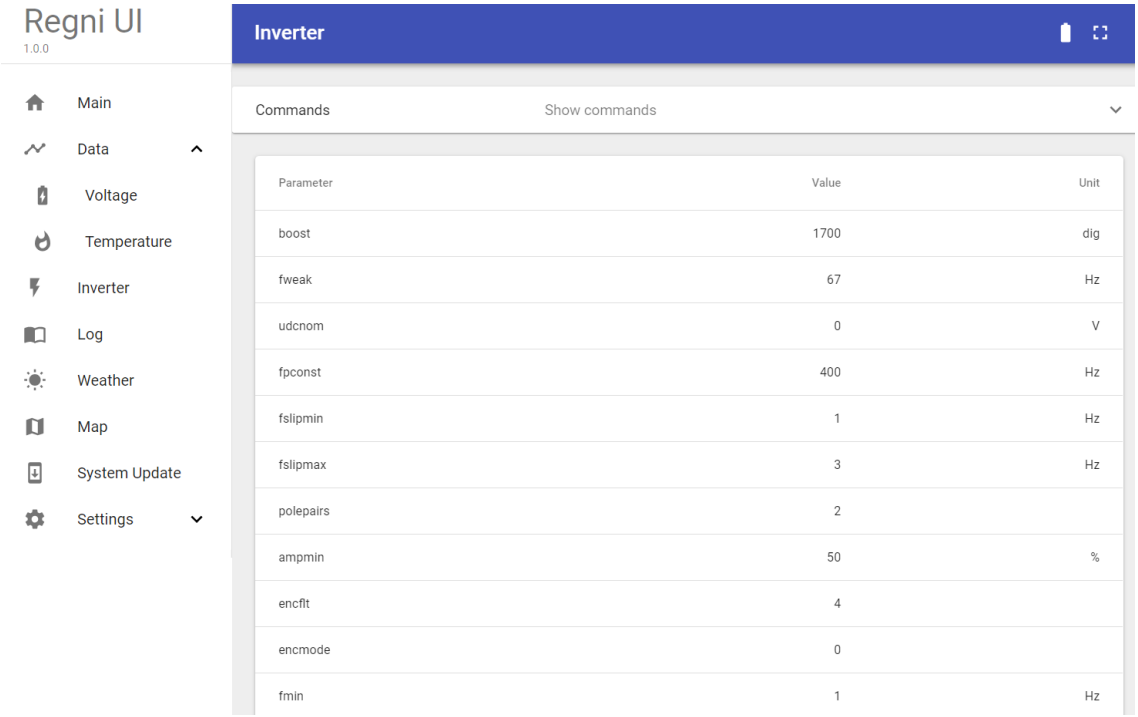
Ajoneuvon akkukennojen jännite- ja lämpötilamittaustuloksia voidaan tarkastella Data-välilehdellä. Voltage-välilehdeltä löytyy jokaisen kennoryhmän jännitemittaukset. Viemällä kursorin mittauspisteen pallon päälle, käyttöliittymä esittää mittaustuloksen numeraalisesti. Viivadiagrammin päivitysnopeutta ja mittauspisteiden määrää voidaan muuttaa Settings-valikosta.



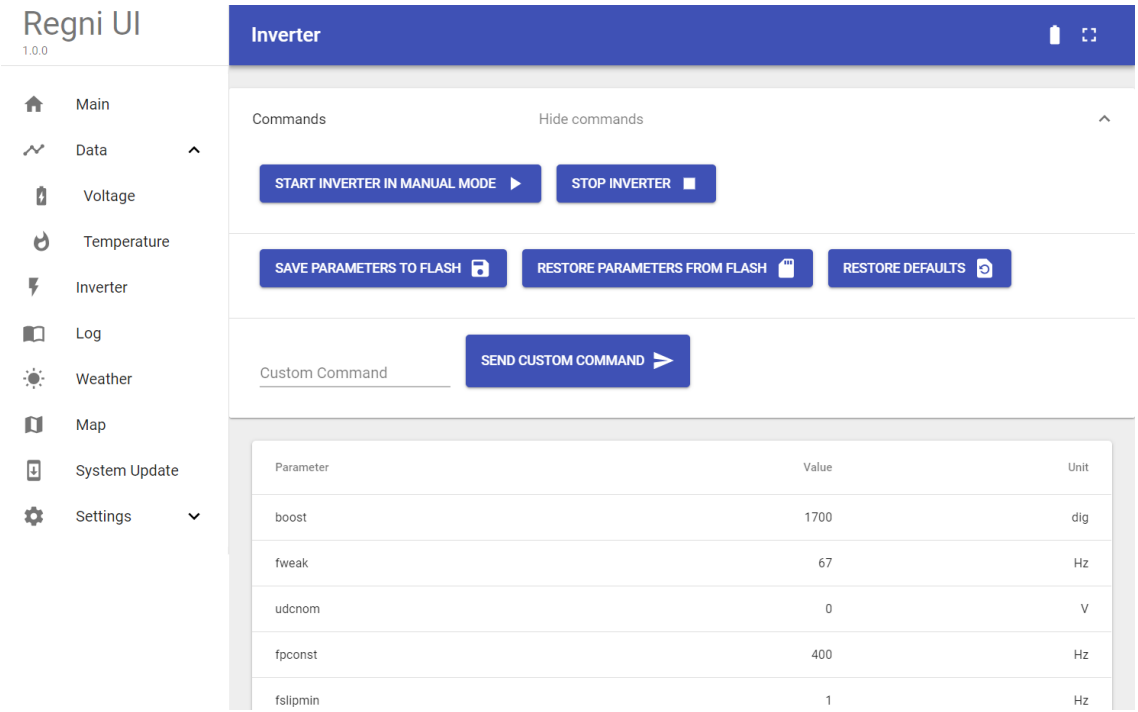
Etäkäyttöliittymässä mittaustulokset saadaan näkyviin antamalla haluttu aloitus- ja lopetuspäivämäärä. Query-nappula tekee haku pyynnön etäpalvelimelle.



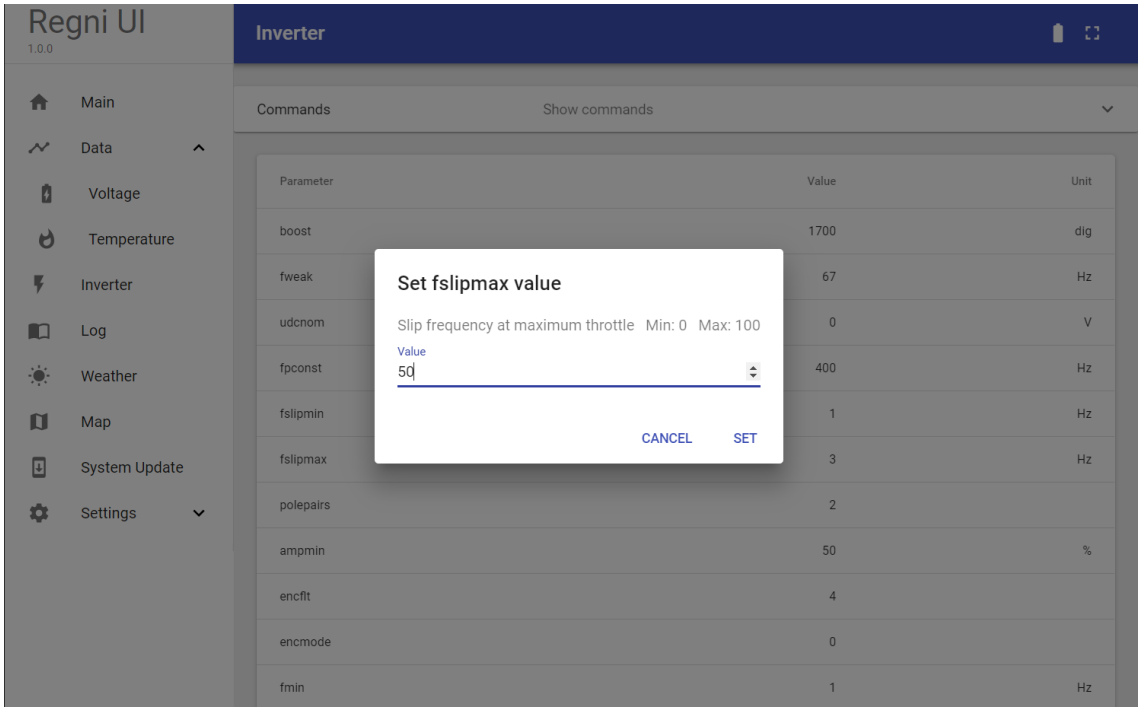
Lämpötila-välilehdellä esitetään lämpökartta akkukennoista ja termoparin- ja akkukennojen mittaustulokset viivadiagrammeilla. Lämpökartan kennojärjestys luetaan alhaalta ylöspäin, vasemmalta oikealle.



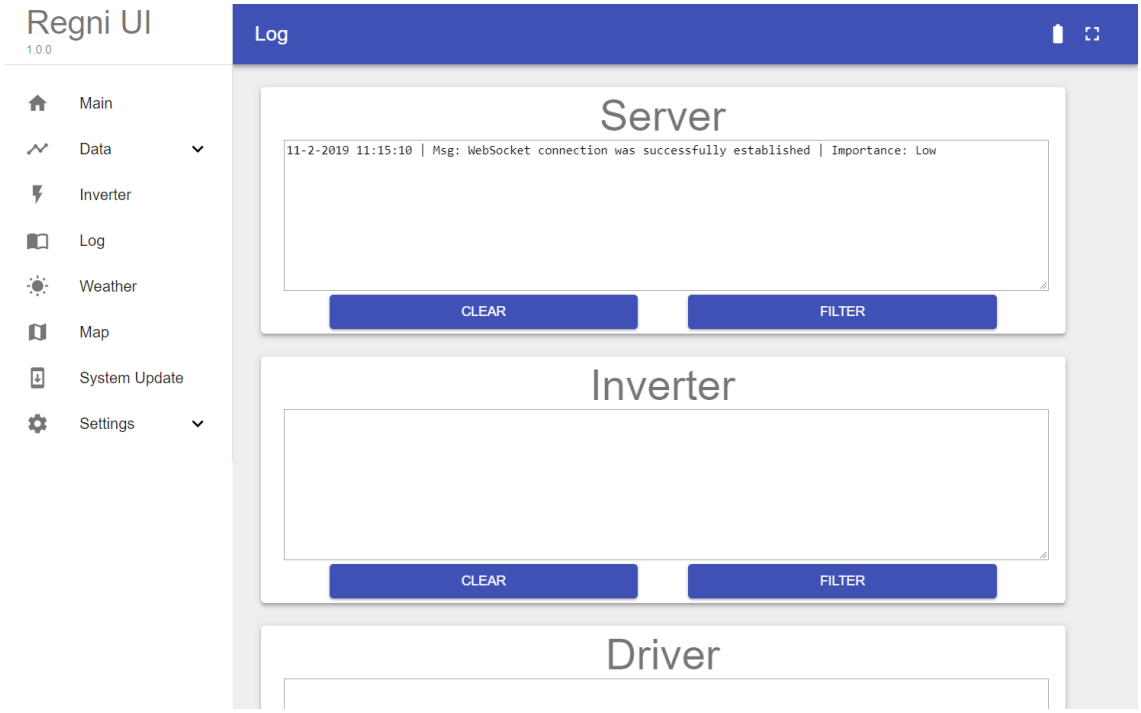
Inverter-välilehdellä voidaan antaa komentoja vaihtosuuntaajalle. Lisäkomentoja saadaan painamalla Commands-palkkia.



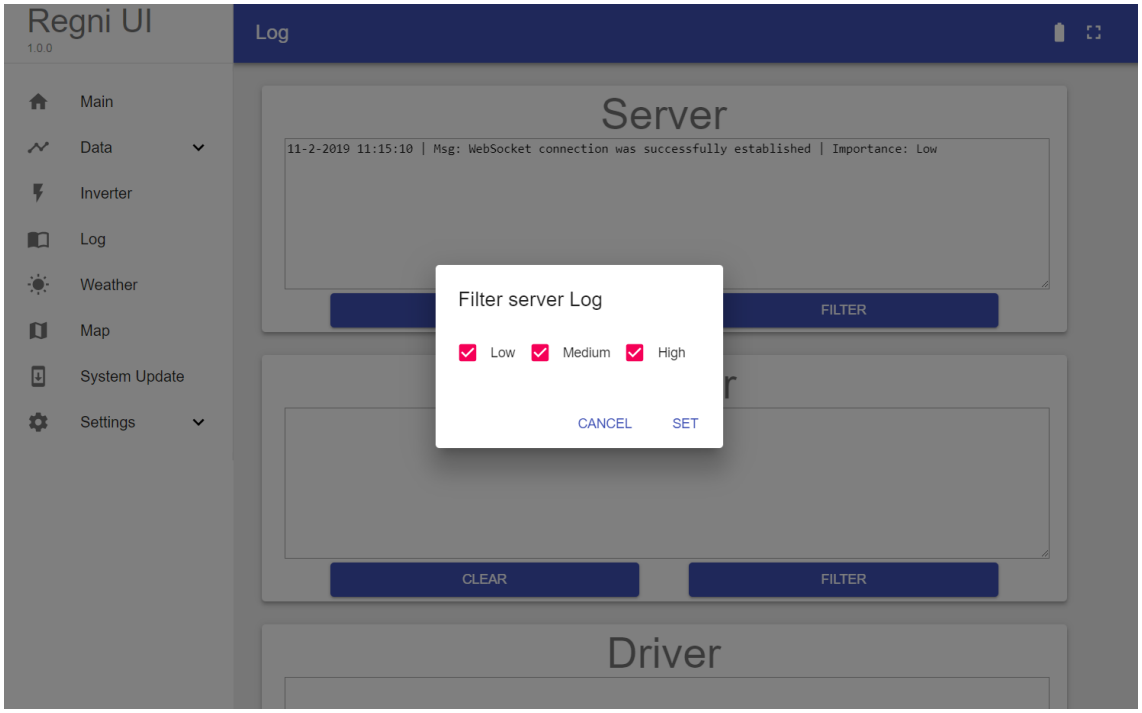
Commands-palkin kautta voidaan käynnistää vaihtosuuntaaja manuaali tilassa tai pysäyttää vaihtosuuntaaja. Vaihtosuuntaajan asetukset voidaan tallentaa, ladata tai palauttaa oletusasetuksille. Oma komento voidaan antaa kirjoittamalla se Custom Command-tekstikenttään ja painamalla lähetä nappulaa.



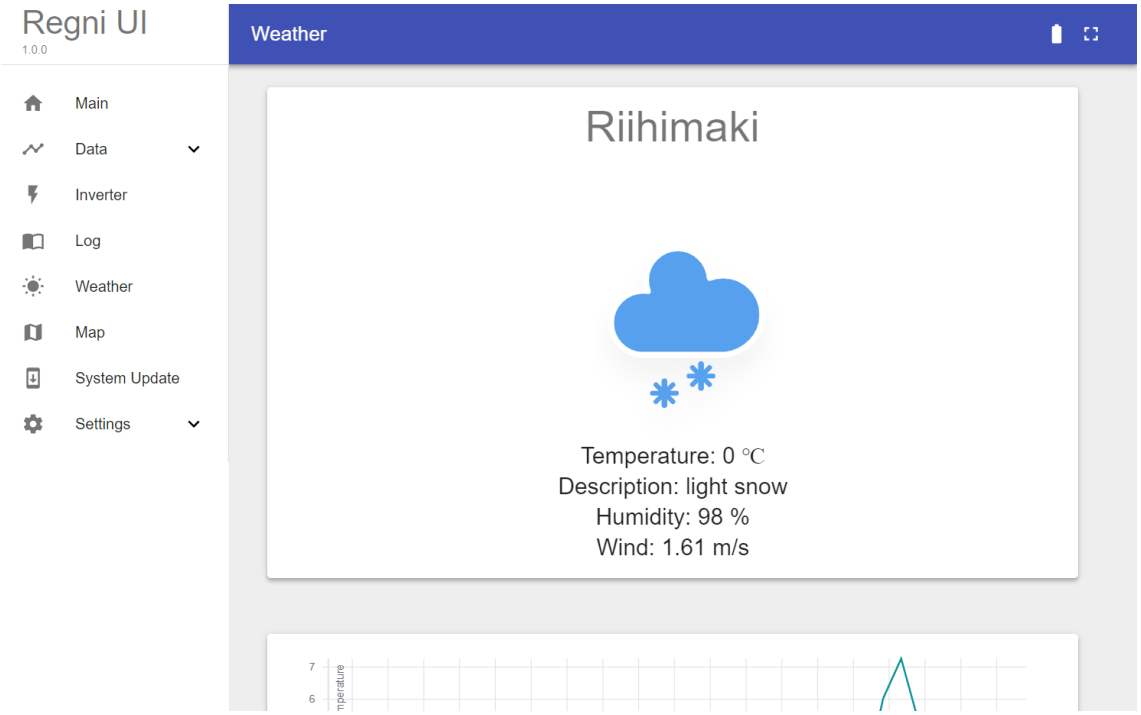
Vaihtosuuntaajan parametrien arvoja voidaan muokata painamalla halutun parametrin riviä taulukosta. Kun riviä painetaan, käyttöliittymä avaa ikkunan, jossa on annettu kuvaus parametrista ja sen minimi ja maksimi arvot. Uusi arvo voidaan asettaa antamalla arvo Value-kenttään ja painamalla SET, jolloin muutos annetaan välittömästi vaihtosuuntaajalle. Jos muutosta ei haluta tehdä, paina CANCEL.



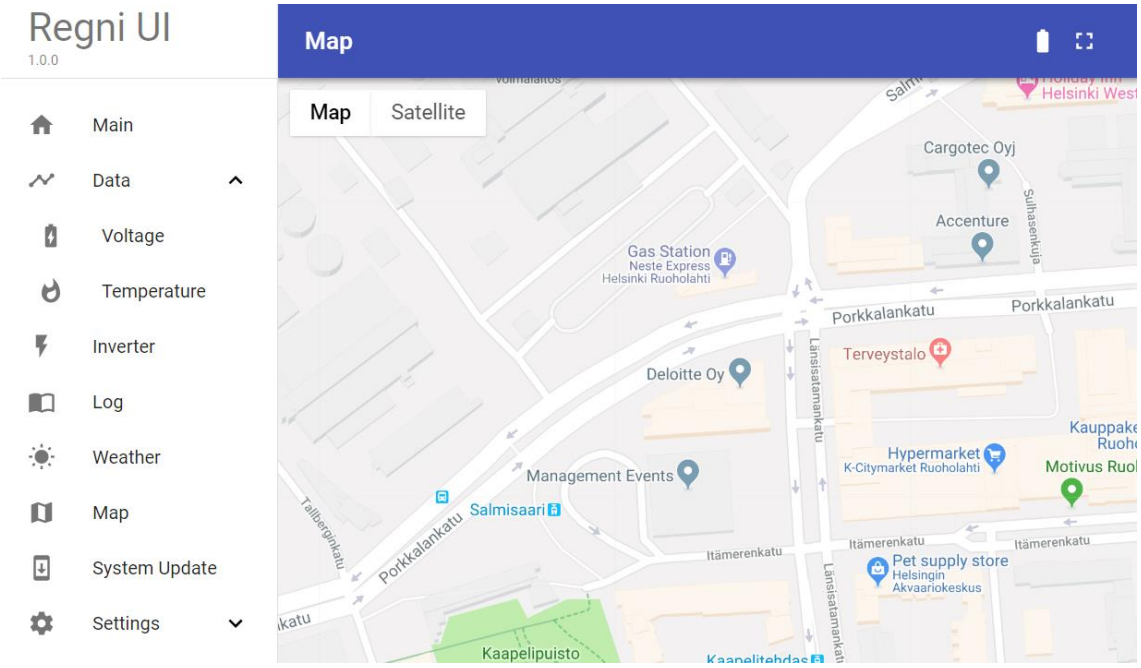
Log-välilehdellä voidaan lukea loki viestejä palvelimelta, vaihtosuuntaajalta, Driver-kortilta ja Controller-kortilta. Lokiviestit voidaan tyhjentää painamalla CLEAR-nappulaa.



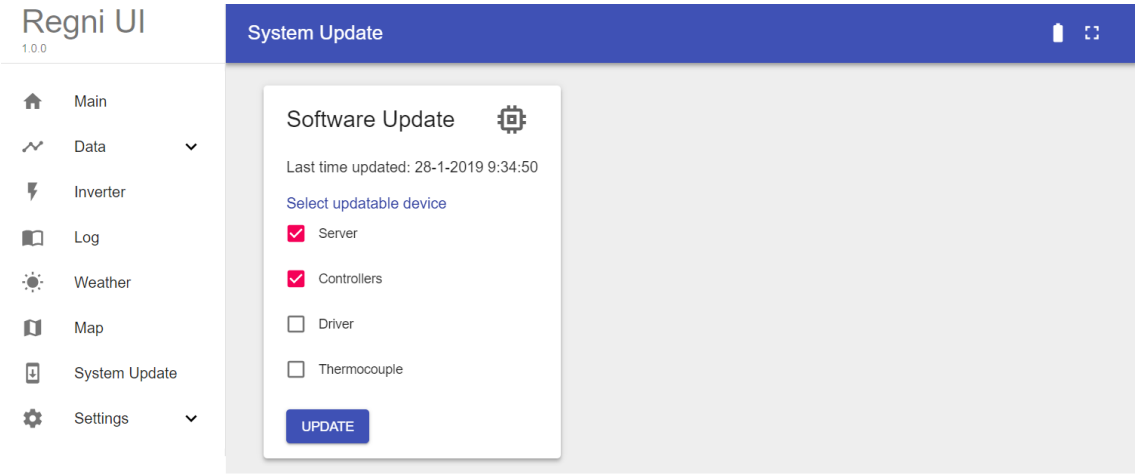
Loki viestejä voidaan suodattaa painamalla FILTER-nappulaa ja ottamalla valitsin pois sen tason viesteistä, joita ei haluta tarkastella.



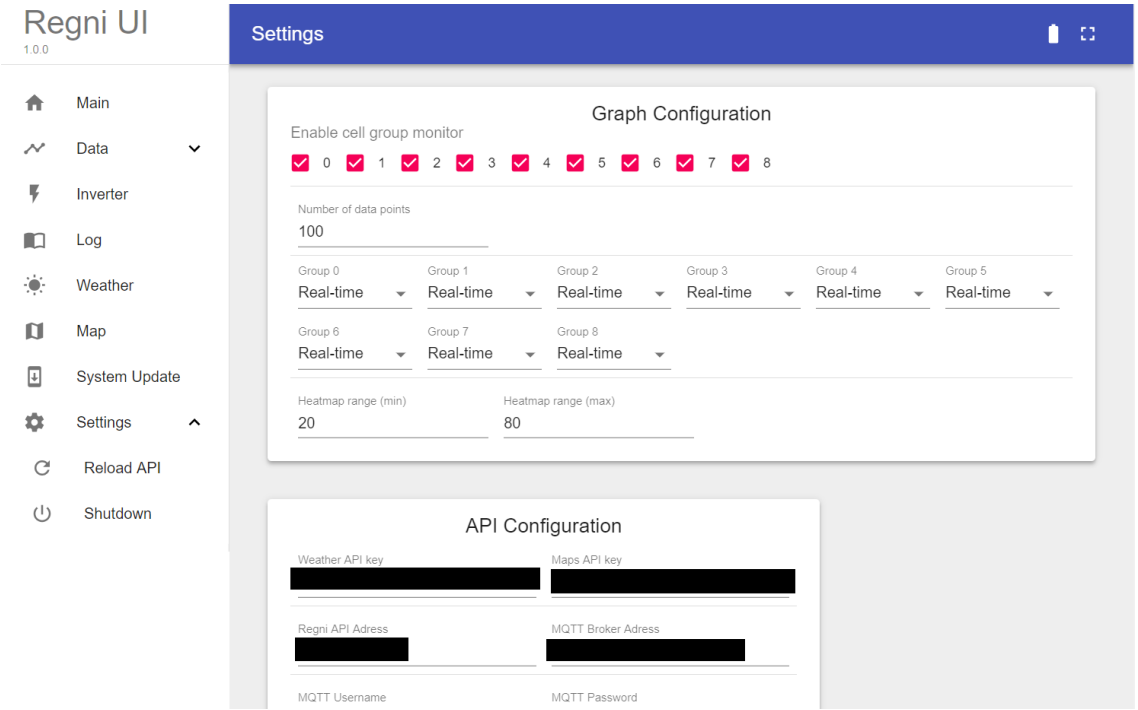
Weather-välilehdellä voi lukea paikallisen säätiedon ja viiden päivän ennusteen. Näyttölaitteessa pitää olla GPS-sijaintipalvelu päällä ja käyttöliittymälle pitää antaa oikeus käyttää GPS-sijaintitietoja, jotta säätiedot olisivat tarkat.



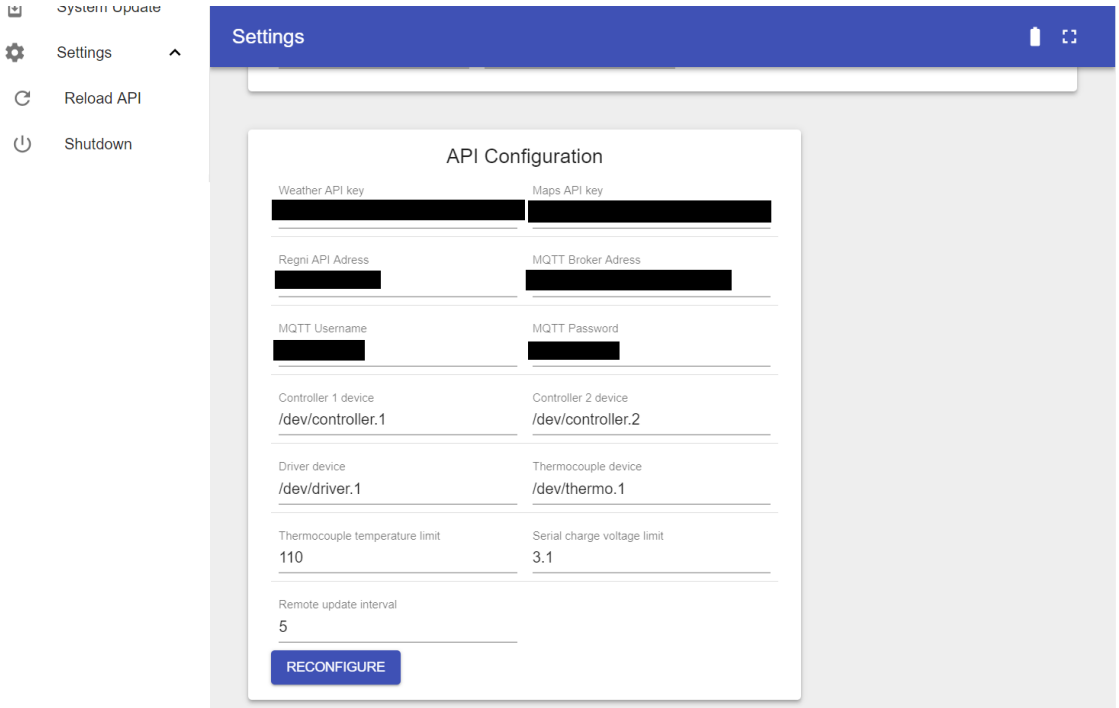
Map-välilehdellä voi selata Google-karttaa. Kartta näkymäksi voidaan valita Map- tai Satellite-näkymä. Karttaa voidaan liikuttaa liikuttamalla sormea tai kursoria näytöllä ylös, alas, oikealle tai vasemmalle. Karttaa voidaan tarkentaa tai loitontaa liikuttamalla kahta sormea kohti toisiaan tai poispäin toisistaan.



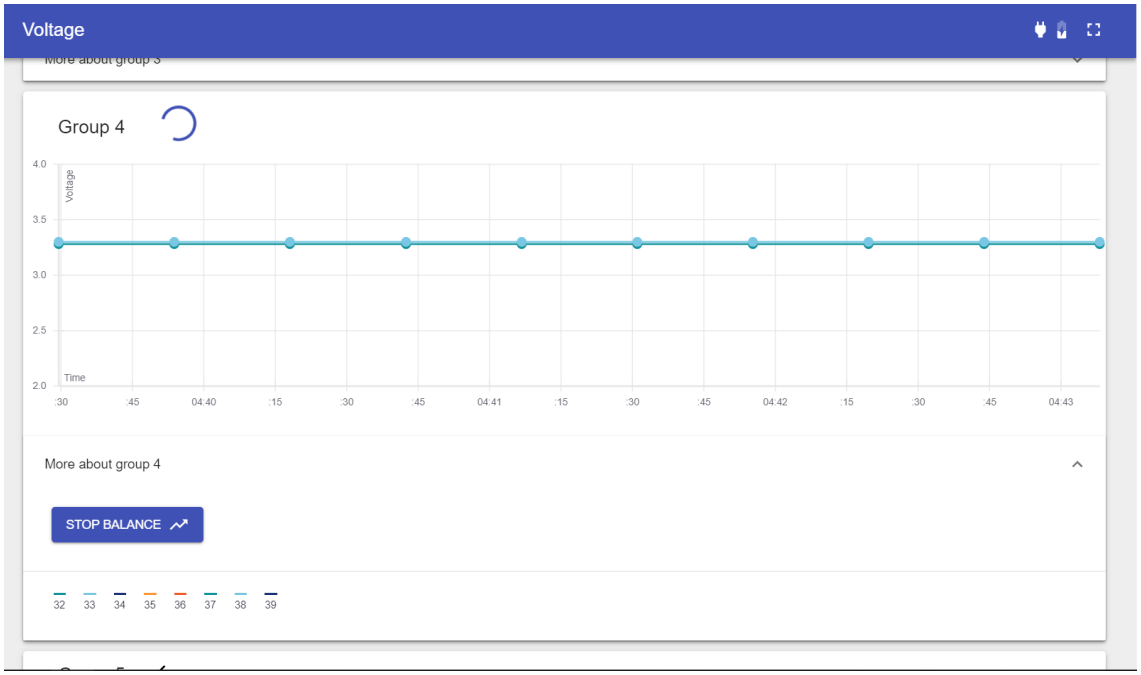
System Update-välilehdellä voidaan päivittää rajapinta, Controller-, Driver ja Thermocouple-kortit. Päivitettäviä laitteita voidaan valita yksi tai useampi. Kun päivitys on valmis, käyttöliittymä avataan uudestaan automaattisesti.



Settings-välilehdellä Graph Configuration-valikossa voidaan piilottaa/esittää viivadiagrammeja ryhmäkohtaisesti, valita esitettävien mittauspisteiden määrän, asettaa viivadiagrammin päivitys nopeuden ja asettaa lämpökartan minimi ja maksimi lämpötilat. Kun Settings-välilehti on auki, vasemmalla olevasta valikosta voidaan myös uudelleen käynnistää rajapinta ja sammuttaa Raspberry Pi.

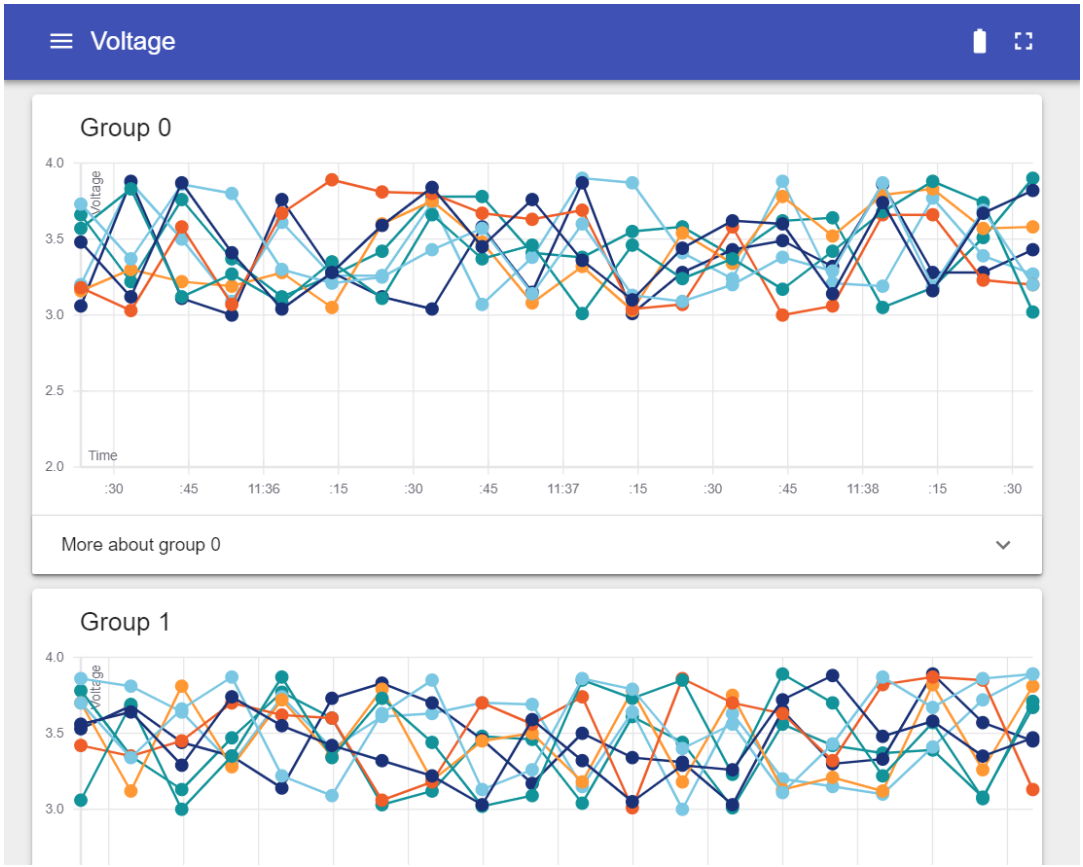


Settings-välilehdellä API Configuration-valikossa voidaan muuttaa rajapinnan asetuksia. Tärkeimmät asetukset ovat Thermocouple-kortin lämpötila (Thermocouple temperature limit), jonka ylittyessä kytketään hätätila päälle. Sarjalatausraja (Serial charge voltage limit), jonka ylittyessä laite siirtyy jännitteentasaustilaan. Aikaväli (Remote update interval), jolloin mittaustulokset lähetetään etärajapintaan.

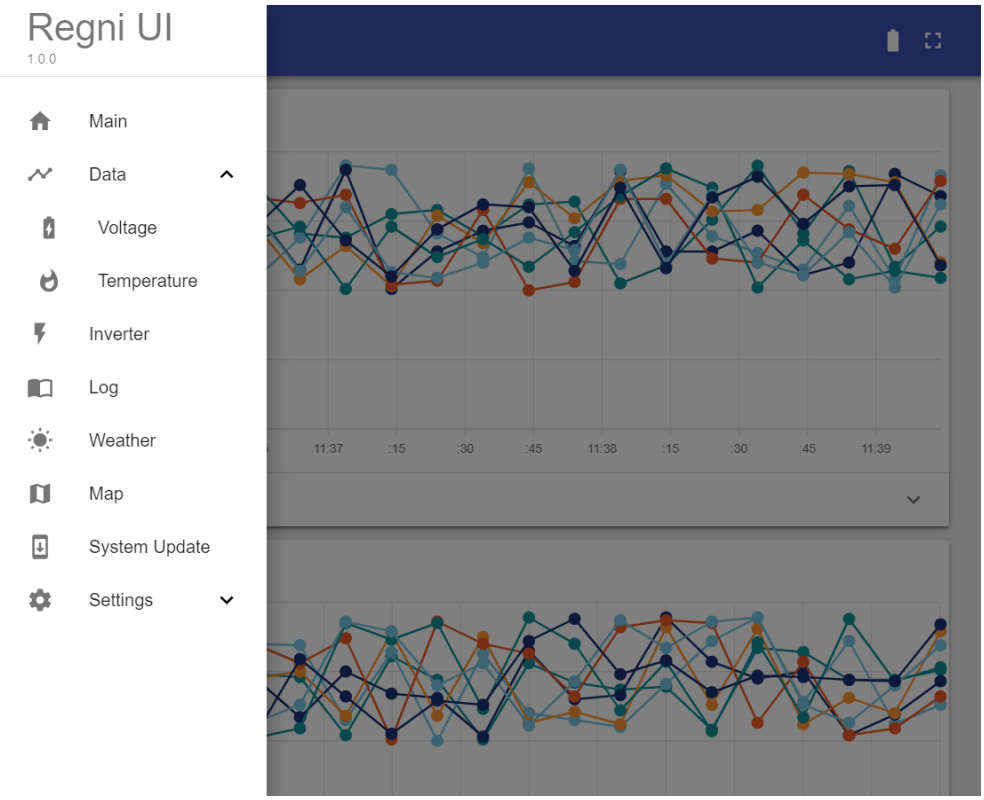


Kun laturi on kytketty ajoneuvoon, oikeaan yläkulmaan ilmestyy pistokkeenkuva. Pistokkeen vieressä oleva pariston kuva on animoitu, niin kauan kuin on ajoneuvo lataustilassa. Kun lataustila on päättynyt, pariston kuva näyttää täydeltä. Pistokkeenkuva häviää, kun laturi irrotetaan ajoneuvosta.

Jännitteentasaus voidaan asettaa manuaalisesti akkukennoryhmälle Data-Voltage valikosta. Valitse kenno ryhmä ja paina More about group X-palkkia. Kun jännitteentasaus on aktiivisena, ryhmän nimen vieressä on animoitu ympyräkuvake. Kun jännitteentasaus ei aktiivinen, mutta laturi on kytkettynä ajoneuvoon, ryhmän nimen vieressä on ✓-merkki.



Jos näyttölaiteen leveys on tarpeeksi kapea, vasemmanpuolen valikko on piilotettu. Valikon saa auki, painamalla kolmea viivaa vasemmassa yläkulmassa.



Navigaatiovalikon saa piilotettua painamalla tummennettua aluetta, valikon ulkopuolelta.