jamk

LCSS - UV-C-desinfiointijärjestelmän diagnostiikkapalvelu

Oskari Seppä

Opinnäytetyö, AMK
Marraskuu 2022
Tekniikan ala
Insinööri (AMK), Tieto- ja viestintätekniikan tutkinto-ohjelma

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Seppä, Oskari

LCSS - UV-C-desinfiointijärjestelmän diagnostiikkapalvelu

Jyväskylä: Jyväskylän ammattikorkeakoulu. **Marraskuu 2022**, 30 sivua.

Tekniikan ala. Tieto- ja viestintätekniikan tutkinto-ohjelma

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

LCSS on diagnostiikkapalvelu UV-C-desinfiointilaitteiden huoltoon ja hallintaan. UV-C on ihmiselle näkymätön valo, jota voi käyttää kontaktittomaan desinfiointiin. Desinfiointilaitteiden UV-C LED-valojen teho heikkenee ajan kuluessa. Asennustilasta johtuen valojen toiminnan seuranta paikan päällä saattaa olla työlästä. Palvelu kehitettiin helpottamaan UV-C-valojen tilan heikkenemisen seurantaa, sekä laitteiden virhetilojen havainnointiin. Tavoitteena palvelun kehittämisessä oli selvittää menetelmä, jolla voi seurata palvelun järjestelmien tilaa. Järjestelmä välittää tiedot reaaliaikaisesti, ylläpitää kerätyn tiedon historiaa ja esittää tiedot käyttäjälle.

Palvelun päätelaite, sekä päätelaitteen UV-C-valon ohjaimet ovat kehitetty ESP-32-mikrokontrollerilla ja toteutuksessa on käytetty valmistajan tarjoamia Arduino-kirjastoja. Palvelun palvelinosuus on TypeScript-kielellä kirjoitettu Node.js HTTP- ja WebSocket-palvelin, joka kommunikoi päätelaitteen ja käyttöliittymän kanssa. Palvelua ajaa Linux-virtuaalipalvelin. Palvelin sisältää myös MariaDB-tietokannan, josta palvelin voi jakaa historiallista tietoa. Palvelun käyttöliittymänä toimii React-pohjainen web-sivu.

Palvelu tuli toimintaan kesäkuussa 2022. Palvelun kehitys toteutui määrittelyjen ja aikataulun mukaisesti. Palvelun kehitys jatkuu kehittämällä palvelua uusilla ominaisuuksilla, kuten UV-C-valon ohjaimien etäohjelmistopäivityksillä.

Avainsanat (asiasanat)

TypeScript, React, Sulautetut järjestelmät, C++, Arduino, MariaDB

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Seppä, Oskari

LCSS – Diagnostics service for UV-C disinfection system

Jyväskylä: JAMK University of Applied Sciences, November 2022, 30 pages.

Technology and engineering. Information and Communication Technology. Bachelor's thesis

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

LCSS is a diagnostics service for servicing and controlling UV-C-disinfection devices. UV-C is a light invisible to the human eye, which can be used in contactless disinfection. The condition of the disinfection devices' lights deteriorates over time and on-site observing of the disinfection lights' condition may be demanding, since the devices may be in a location with difficult access. The service was developed to help keeping an eye out on the condition of the UV-C lights and detecting error states of the devices. The target of the development was to find out a way to follow the condition of the service's systems, which includes relaying the devices' data in real time and historically and displaying the data to a user.

The end-device and the end-device's UV-C-light controllers are developed with ESP-32-microcontroller, using manufacturer-provided Arduino libraries. The server-component of the service is a Node.js HTTP and WebSocket server written with TypeScript language, which communicates with the end-device and the user interface. The server is running on a Linux virtual server. A React-based website functions as the user interface of the service.

The service went to production in June of 2022. Development of the service was successful regarding specifications and schedule. The service's development continues with implementation of new features, such as UV-C-light controller devices' remote software updates.

Keywords/tags (subjects)

TypeScript, React, Embedded systems, C++, Arduino, MariaDB

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Johdanto

LCSS (Led Control System Service) on diagnostiikkapalvelu UV-C-desinfiointijärjestelmälle. Palvelun tavoite on saada järjestelmien huoltajille nopea reaktioaika mahdollisiin järjestelmien virheisiin tai vikatiloihin, sekä antaa mahdollista statistiikkatietoa järjestelmistä asiakkaalle. UV-C-valojen teho heikentyy käytön mukaan. Tehon muutos on nähtävissä palvelussa, jännitte- ja virta-arvojen muutoksena, jolloin valot voidaan vaihtaa hyvissä ajoin ennen käyttöiän loppua.

Työn toimeksiantaja oli SoftRain Blobs Oy. Tavoite opinnäytetyössä oli kehittää palvelu desinfiointilaitteiden tilan reaaliaikaiseen ja historialliseen seurantaan.

2 Työkalut ja teknologiat

2.1 ESP32

ESP32 on Espressif'n valmistama 32-bittinen kaksiytiminen mikrokontrolleri, joka sisältää sisäänrakennetun 2,4GHz radio lähetin-vastaanottimen, joka mahdollistaa WiFi ja Bluetooth yhteyksien käytön. ESP32 mikrokontrolleri ei itsessään vaatii ulkoisia komponentteja, Espressif tarjoaa erilaisia moduuleja, jotka sisältävät tarvittavat oheiskomponentit, kuten haihtumattoman flash-muistin, kellokiteen, sekä metallisen RF-suojan komponenttien päällä (ESP32-WROOM-32 Datasheet 2016, 7).

Projektissa on käytössä AI-Thinker yhtiön NodeMCU-32S kehitysalusta, joka sisältää ESP32-WROOM-32 moduulin. Tämä kehitysalusta sisältää moduulin lisäksi myös LDO-jännitteensäätimen ja CH340-mikropiirin, joka mahdollistaa sarjaliikenteen yhdistämisen tietokoneeseen USB-väylän kautta (NodeMCU-32S Core Development Board N.d.).

2.2 Arduino

Arduino valmistaa avoimeen lähdekoodiin perustuvia elektronisia laitteita, sekä ohjelmistoja. Laitteistot ja ohjelmistot ovat helppokäyttöisiä ja soveltuvat esimerkiksi opetuskäyttöön, mahdollistaen myös ammatilliskäytön. Arduino tarjoaa mahdollisuuden ohjelmoida C++ kielellä, mutta se ei sisällä C++ kielen standardikirjastoja. Sen sijaan Arduino tarjoaa omat, yksinkertaistetut versiot C-kirjastoista (What is Arduino? 2018).

Espressif tarjoaa kääntäjän, kirjastot ja muut työkalut Arduino-tyyliseen ohjelmointiin laitteilleen GitHub-versionhallintapalvelussa (Arduino core for the ESP32, ESP32-S2, ESP32-S3 and ESP32-C3 2016).

2.3 FreeRTOS

Prosessorin ydin voi ajaa vain yhtä ohjelman osaa kerrallaan. RTOS eli Reaaliaikainen käyttöjärjestelmä mahdollistaa moniajon. RTOS-käyttöjärjestelmän ytimenä toimii aikatauluttaja, joka voi käyttäjän määrittämien prosessien avulla antaa prosessoriaikaa eri tehtäville. Normaalit käyttöjärjestelmät, kuten Windows, yrittävät antaa prosesseille prosessoriaikaa sen mukaan, että käyttöjärjestelmä pysyy responsiivisena. Reaaliaikainen käyttöjärjestelmä, kuten FreeRTOS, yrittää tehdä prosessien suorittamisesta arvattavan. Reaaliaikainen käyttöjärjestelmä vaatii, että järjestelmä vastaa tiettyyn tapahtumaan määräajassa.

FreeRTOS on reaaliaikainen käyttöjärjestelmä, joka on suunniteltu tarpeeksi pieneksi käytettäväksi mikrokontrollereilla. Ohjelmiston koon säästämiseksi FreeRTOS sisältää mahdollisuudet vain aikatauluttajan, prosessien välisen kommunikoinnin, ajastuksen ja synkronisoitiin (Why RTOS and What is RTOS? N.d).

2.4 Node.js

Node.js on asynkroninen JavaScript ympäristö, joka mahdollistaa JavaScript kielen ajon palvelinympäristössä. Lähes mikään Node.js toiminto ei estä muun ohjelman kulkua, joka auttaa varsinkin skaalautuvan ympäristön kehityksessä (About Node.js N.d.).

2.5 npm ja npx

”npm” eli Node Package Manager on paketinhallintajärjestelmä, jossa yhteisö voi jakaa ohjelmistoja, joita Node.js voi käyttää (What is npm? 2011). ”npx” on npm paketti, joka asentuu oletuksena -paketinhallintajärjestelmän mukana. npx-työkalulla voi asentaa sekä ajaa ohjelmistoja, npm-paketinhallintajärjestelmän rekistereistä.

2.6 SQLite

SQLite on prosessinsisäinen tietokantakirjasto, joka mahdollistaa nopean ja itsenäisen tietokannan kehittämisen. Erona muihin tietokantaratkaisuihin, SQLite ei vaadi erillistä palvelinta, vaan tietokanta on tallennettuna tiedostoon, josta muut ohjelmat pystyvät itsenäisesti hakemaan tietoja (About SQLite 2021).

2.7 MariaDB

MariaDB on palvelinpohjainen relaatiotietokanta. MariaDB-tietokanta muistuttaa MySQL tietokantaa, jonka alkuperäiset kehittäjät ovat kehittäneet MariaDB'n. MariaDB-tietokanta käyttää MySQL kannan tavoin oletuksena InnoDB-moottoria.

Etuna MariaDB-palvelimessa SQLite-tietokantaan on suorituskyky suurissa tietokannoissa (Pomponio, A. 2021).

2.8 React

React on JavaScript kirjasto, web-käyttöliittymien rakentamiseen. React vaatii Node.js-ympäristön ja käyttöliittymiä suunnitellessa voi käyttää hyväkseen laajasti tarjolla olevia NPM-paketteja. React-kirjastolla toteutettaessa käyttöliittymää, käytetään yleensä XML-tyylistä JSX-syntaksia komponenttien näyttämisessä (reactjs.org N.d).

2.9 WebSocket

WebSocket on protokolla, joka mahdollistaa kaksisuuntaisen kommunikoinnin palvelimen ja käyttäjän välillä. Kuten HTTP-protokolla, WebSocket-protokolla käyttää TCP-yhteyttä kommunikointiin. WebSocket yhteyden kättely muistuttaa myös HTTP-pyynnön header-osuutta.

WebSocket-protokollan etu normaaliin HTTP-pyyntöön verrattuna on se, että yhteys pysyy auki, jolloin ylimääräisiä avausviestejä ei tarvitse jokaisen viestin yhteydessä. Tämä mahdollistaa reaaliaikaisemman viestinnän palvelimen ja käyttäjän välillä (The WebSocket Protocol 2011, 4-5).

2.10 UV-C

UV-C on säteilyä, jonka aallonpituus on 100-280nm välillä (ISO-21348). Tämä säteily ei esiinny luonnossa, sillä ilmakehä suodattaa säteilyn kokonaan. Säteilyä voi käyttää kontaktittomassa desinfioinnissa.

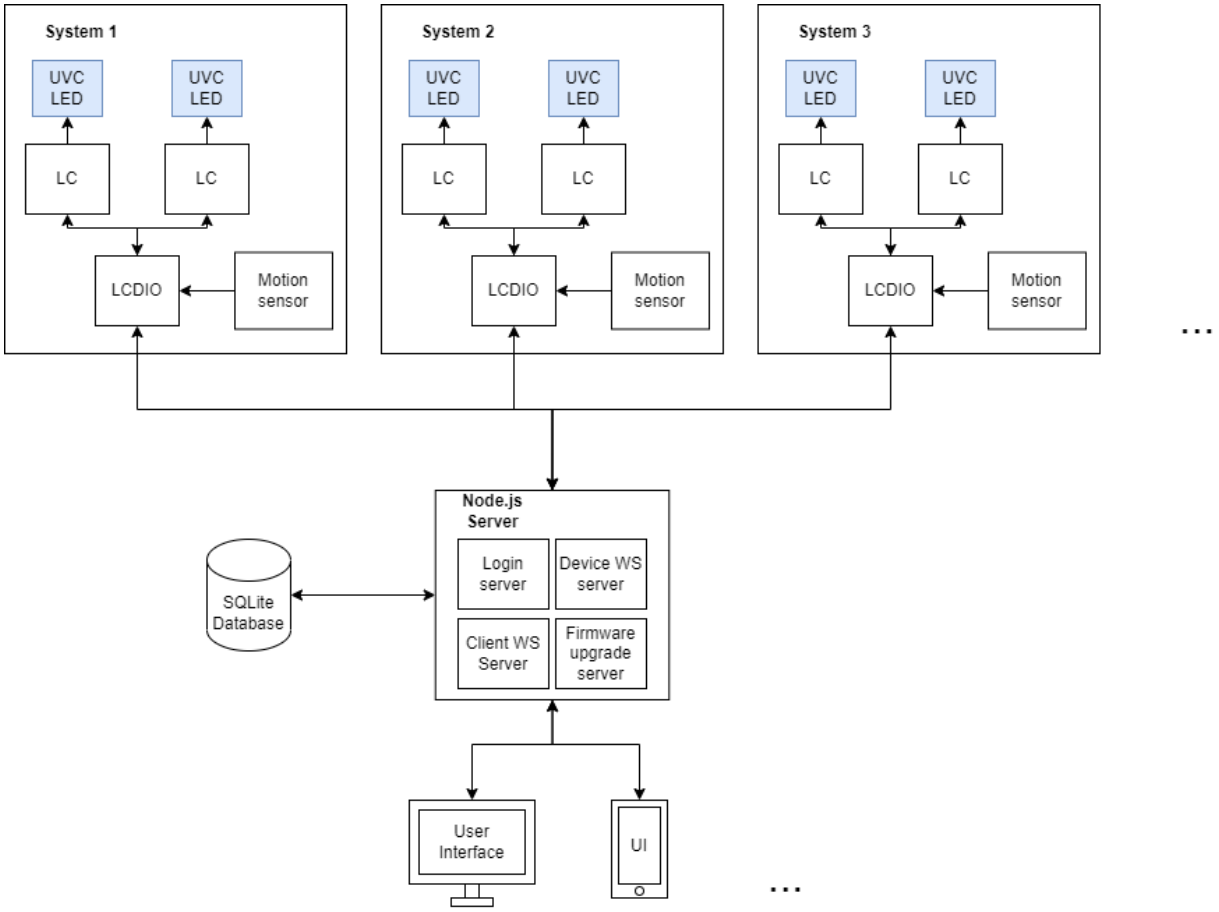
UV-C-desinfiointia voi harkita, jos halutaan nopeaa ja kontaktitonta desinfiointia. UV-C-desinfiointi soveltuu parhaiten ilman ja veden desinfiointiin, mutta sitä voi käyttää myös pintojen desinfiointiin. Desinfiointi toimii polttamalla LED-valoja, jotka tuottavat ihmissilmälle näkymätöntä UV-C-valoa. Tämä valo on kuitenkin haitallinen ihmiselle, eläimille, sekä kasveille (UV-C-säteilyn käyttö desinfioinnissa 2021).

3 Järjestelmän rakenne

3.1 Kokonaisuus

Desinfiointijärjestelmä koostuu kahdesta eri laitteesta: LC-kortista, joka ohjaa UV-C-valoja ja LCDIO-isäntäkortista, joka ohjaa LC-ohjaimia. LCDIO on järjestelmän ainoa laite, joka voi kommunikoida palvelimen kanssa. LC-kortit eivät ole yhteydessä palvelimeen tai verkkoon. LC-kortit kommunikoivat LCDIO-kortin kanssa RS485-väylän välityksellä. LC- ja LCDIO-kortteja ohjaavat ESP-32 mikrokontrollerit.

Desinfiointijärjestelmien tietoja voi tarkastella palvelun web-käyttöliittymästä, joka on yhteydessä palvelimeen WebSocket-yhteydellä.



Kuvio 1. Palvelun ja desinfiointijärjestelmien rakenne

3.2 LCDIO-kortti

LCDIO on järjestelmän päätelaite, joka ohjaa desinfiointijärjestelmän toimintaa, sekä lähettää tietoa palvelimelle laitteista. LCDIO-kortti on projektiin suunniteltu kortti, jossa on mahdollisuus kahdeksaan digitaaliseen IO-kanavaan, sekä neljään analogiseen IO-kanavaan. Nämä kanavat voidaan määrittää ESP-32-mikrokontrollerissa ohjaamaan desinfiointia erilaisista lähteistä.

LCDIO-korttiin tulee sisääntulona signaali kahdelta liiketunnistimelta, joiden mukaan LC-kortteja ohjataan. Kortissa on kiertokytkin, jonka voi asettaa kuuteentoista eri asentoon. Tällä määritetään laitteen ”osoite” järjestelmässä. Osoite 0 on varattu järjestelmän määrittelylle, Osoite 1 on varattu järjestelmää ohjaavalle isäntäkortille, eli LCDIO-kortille. Loput osoitteet, 1…15, ovat rengeille, eli LC-korteille varattuja. Tämän osoitteen mukaan RS485-väylässä voidaan osoittaa viestit tietylle laitteelle, sekä tunnistaa laite palvelussa.

Valittaessa osoitteen 0, LCDIO ei mene normaaliin ajotilaan, vaan laite menee järjestelmän määrittelytilaan. Laitteen ESP-32-mikrokontrolleri avaa WiFi-verkon, johon käyttäjä voi liittyä esimerkiksi mobiililaitteellaan. Mikrokontrolleri avaa myös DNS-palvelimen, jonka avulla käyttäjä voi avata valitsemastaan selaimesta selkokielisen osoitteen IP-osoitteen sijaan. Sivun avattuaan käyttäjä pääsee järjestelmän määrittelysivulle, josta voi asettaa RTC-kellonajan ja järjestelmän lämpötila- ja jänniterajan. Lisäksi määrittelysivulla voi asettaa WiFi-yhteyspisteen tiedot, johon mikrokontrolleri yrittää yhdistää. Tämän yhteyspisteen kautta kontrolleri saa yhteyden LCSS-palveluun.

Määrittelysivulla voi myös asettaa laitteen fyysisen sijainnin koordinaatteina, jonka avulla palvelussa voi näyttää järjestelmän kartalla. Sivulla asetetaan renkikorttien kanavamäärittelyt, mutta kanavamäärittelyt voi myös asettaa myöhemmin LCSS-palvelun käyttöliittymässä. Kanavamäärittelyihin kuuluu kanavan tyyppi, joka voi olla tyhjä, valaisin, sininen valo tai UV-C-valo. Kanavamäärittelyssä asetetaan myös UV-C-valotikkujen määrä, minimi- ja maksimivirrankäyttö.

Kuvio 2. Kaappi ohjainkortteineen

LCDIO-kortissa on myös mahdollisuus järjestelmäpäivitykseen palvelun kautta, jonka ansiosta voidaan korjata mahdollisia ohjelmavirheitä tai lisätä tärkeitä ominaisuuksia.

3.3 LC-kortti

LC-kortti on LCDIO-kortin tavoin räätälöity kortti, jota ohjaa ESP-32-mikrokontrolleri. LC-kortissa on neljä ulostulokanavaa, jotka voivat pulssinleveysmodulaatiolla ohjata kanavaan liitettäviä laitteita. Kanavien käyttöjännite on 24 volttia. Kanaviin voi liittää eri konfiguraatiossa esimerkiksi UV-C-valoja, joiden virtamäärää voidaan rajoittaa kanavamäärittelyissä.

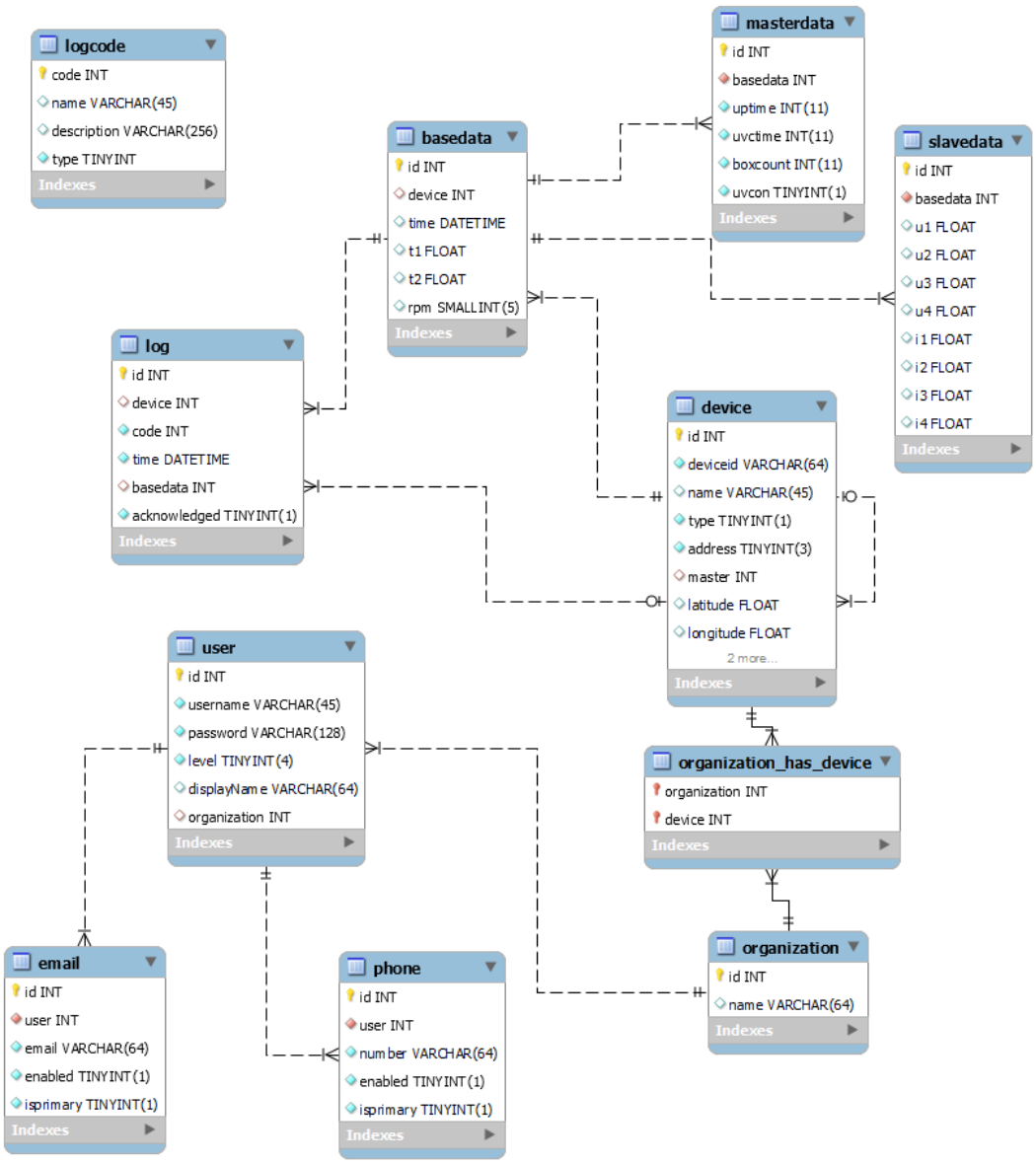
LC-kortit kuuntelevat komentoja LCDIO-kortilta RS485-väylän kautta ja niissä on kiertokytkin LCDIO-kortin tapaan, jolla valitaan osoite 2…15, joka vastaa kortin osoitetta järjestelmässä.

LC-kortti ei ole yhteydessä verkkoon, joten sitä ei voi päivittää LCDIO-kortin tavoin.

3.4 Palvelin

Palvelin vastaa laitteiden ja käyttäjän kommunikoinnista, sekä datan tallennuksesta tietokantaan. Palvelin on TypeScript-kielellä kirjoitettu node.js palvelu, jonka tehtävä on vastaanottaa tietoja laitteelta, jotka voidaan ohjata MariaDB-tietokantaan, sekä reaaliaikaisesti WebSocket-yhteyttä käyttäen käyttöliittymään.

Palvelin sisältää omat eri porteissa olevat WebSocket-palvelimet laitteille sekä käyttöliittymille. Lisäksi palvelimessa on kaksi http-palvelinta, joista toinen hoitaa käyttöliittymän kirjautumista ja toinen LCDIO-kortin laiteohjelmistojen päivityksiä.



Kuvio 3. Tietokannan rakenne

3.5 Käyttöliittymä

Käyttäjät pystyvät tarkastelemaan käyttöliittymästä laitteiden tietoja, asettamaan tiettyjä määrityksiä, sekä tekemään ohjelmistopäivityksen laitteelle. Käyttöliittymä on web-sivu, joka on tehty React-kirjastoja käyttäen. Käyttöliittymä kommunikoi palvelimen kanssa WebSocket-yhteydellä.

Käyttöliittymä sisältää kaksi näkymää ”user”-tason käyttäjälle: ”Home”, eli kotinäkymä, jossa järjestelmien fyysiset sijainnit esitetään kartassa, sekä ”Systems Overview”, eli järjestelmien yleisnäkymä, jossa käyttäjä voi nähdä oman organisaation järjestelmien tila ja yleistä tietoa järjestelmistä.

Käyttöliittymässä on myös kolme erillistä näkymää ainoastaan ”admin”-tason käyttäjälle. ”UVC systems log”, eli laitenäkymä, jossa esitetään järjestelmien yleisnäkymän tavoin järjestelmien tietoja, mutta yksityiskohtaisemmin eri laitteina. Järjestelmänvalvojalle on myös laitteen yksityiskohtainen näkymä, jossa voi tarkastella yksittäisen laitteen tietoja, tai ohjata kyseistä laitetta antamalla esimerkiksi järjestelmäpäivityksen. Käyttöliittymän ”Settings”-näkymässä käyttäjä voi luoda uusia käyttäjiä ja organisaatioita, sekä hallita järjestelmäpäivityksiä.

4 Suunnittelu

4.1 Vaatimukset

Palvelun suunnitteluun kuului vaatimuksien laatiminen. Vaatimukset olivat lopulta melko väljät, joka mahdollisti vapaan kehityksen toimintojen ympärille. Palvelulta vaadittiin, että järjestelmien kuntoa voidaan valvoa lähes reaaliajassa web-käyttöliittymän välityksellä ja huomata mahdolliset vikatilat mahdollisimman nopeasti. Palvelusta täytyy myös nähdä LED-valojen tilan heikkeneminen. Tilan heikkenemisen pystyy havaitsemaan LED-valojen jännitteen ja virrankulutuksen avulla, eli palvelu vaatii myös historiallisen tiedon näyttämisen. Seuraamalla valojen tilaa, epäkunnossa olevien valojen vaihtaminen voidaan ajoittaa paremmin.

AI-Thinkerin NodeMCU-32S kehitysalustalla oleva ESP32 ohjaa desinfiointilaitteita. Etuna kehitysalustan käytössä pelkän moduulin sijaan on mahdollisuus vaihtaa epäkunnossa oleva mikrokontrolleri sekunneissa. Kehitysalustaa valmistetaan myös suuressa mittakaavassa, joten se on halpa, sekä luotettava.

Palvelimen, päätelaitteen, sekä käyttöliittymän välinen kommunikointi vaadittiin olevan lähes reaaliaikaista, joten suunnittelussa päädyttiin WebSocket-yhteyksien käyttöön.

Palvelussa piti myös estää, ettei ulkopuoliset pääse tarkastelemaan tietoja, joten palvelu vaati kirjautumisjärjestelmän, sekä kirjautumisruudun käyttöliittymään. Lisäksi jos tulevaisuudessa useampi eri yhtiö käyttäisi palvelua, palvelussa olisi hyvä olla mahdollisuus lähettää tietoja vain niistä laitteista, jotka ovat kyseisen organisaation käytössä.

Käyttäjille vaadittiin käyttöliittymässä käyttäjätaso, joka määrittää mitä tietoja käyttäjä voi tarkastella. Käyttäjillä on kaksi tasoa - ”user”-taso, eli käyttäjätaso, jota palvelun asiakkaat käyttävät, sekä ”admin”-taso, jota palvelun ja laitteiden kehittäjät tai huoltajat pystyvät käyttämään. ”user”-tasolla määriteltiin näkymään vain perustietoa laitteista, kuten virhetila, UV-C-valojen päälläoloaika ja lämpötila. ”admin”-tason käyttäjät pystyvät tarkastelemaan kaikkia järjestelmän tietoja, sekä tekemään erilaisia operaatiota, kuten järjestelmien nimen muokkaaminen tai käyttäjien lisääminen.

Tärkeä vaatimus oli myös tietoturva. Laitteelta palvelimelle tuleva tieto täytyy salata, kuten palvelimelta käyttäjälle. Salauksessa päädyttiin käyttämään SSL/TLS suojauksia http- ja WebSocket-yhteyksissä.

5 Toteutus

5.1 Palvelin

5.1.1 Ympäristö

Palvelimen kehitys alkoi alustamalla NPM projekti. Kirjoittamalla komentolinjalle komennon ’npm init lcss-server’, npm-pakettimanageri alustaa projektin ja lisää tarvittavat tiedostot. Node.js kääntää natiivisti JavaScript-kieltä, mutta palvelimen kieleksi oli valittu TypeScript, joten projekti vaati erityisiä konfiguraatioita.

Käyttäen NPM-pakettimanageria, asennettiin paketti ”ts-node”, joka pystyy ajamaan TypeScript-lähdekoodia. Projektiin lisättiin myös tsconfig.json tiedosto, joka on TypeScript kääntäjän määritystiedosto. Tiedostossa tsconfig.json on määritelty ”outDir”, joka määrittää kääntäjän ulostulopolun, johon käännetyt JavaScript-tiedostot tallennetaan. Tiedostoon määriteltiin myös ”include”-polut, jossa on käännettävät tiedostot. Käännettyjä tiedostoja voi myöhemmin käyttää suoritettavan paketin rakentamisessa.

Kuvio 4. tsconfig.json tiedoston sisältö

npm käyttää package.json-tiedostoa määrittelyihin, kuten erilaisten ohjelmien skriptit tai asennetut paketit. npm alustaa package.json-tiedoston ajaessa ”npm init” komennon. Tähän tiedostoon lisättiin skripti vastaamaan ”npm start” komentoa, joka ajaa projektin päätiedoston.

Tietokannaksi valittiin SQLite3 kehityksen ja testauksen ajaksi. SQLite3-tietokanta oli yksinkertainen asentaa. NPM-pakettimanagerin avulla sai asennettua ohjelman, jolla pystyi määrittelemään ja tarkastelemaan tietokantaa. Koska tietokanta oli jo alustavasti suunniteltu, voitiin edetä ajamalla ”sqlite3 db/lcss.db < db/lcss-db.sql”. SQLite3 alustaa tällä komennolla tietokannan ”lcss.db” tiedostoon. Tietokanta vaihdettiin kuitenkin MariaDB-tietokantaan kehityksen edetessä.

5.1.2 WebSocket-palvelimet

WebSocket valittiin järjestelmän kommunikointikeinoksi monesta syystä. WebSocket yhteys on pysyvä, joten käyttäjän tehdessä palvelimelle pyynnön, käyttäjän ei tarvitse tunnistautua erikseen jokaisessa viestissä palvelimelle. Käyttäjä voi ensimmäisessä viestissään tehdä tunnistautumisen, jonka jälkeen palvelin tietää keltä viesti tulee. WebSocket tukee kaksisuuntaista liikennöintiä. Käyttäjän ei tarvitse tehdä kyselyä palvelimelle, palvelin voi myös itse lähettää viestin käyttäjälle. Tämä mahdollistaa reaaliaikaisen tiedonsiirron. Koska WebSocket yhteys on yllä käyttäjän ja palvelimen välillä, tiedonsiirron määrä pienenee myös merkittävästi.

WebSocket palvelimia on kaksi, toinen palvelin päätelaitteille ja toinen käyttöliittymälle. Käyttäen npm-paketinhallintajärjestelmää, projektiin asennettiin ”ws” niminen paketti, joka sisältää tarvittavat rajapinnat WebSocket-palvelimelle. Tämän jälkeen projektiin luotiin luokka, joka ohjaa palve-

limia. Alustaessa luokan, luokka rakentaa kaksi WebSocket palvelinta ja aloittaa kuuntelemaan yhteyksiä. Samalla luokassa asetetaan funktiot, jotka laukeavat, kun yhteys muodostetaan palvelimeen.

Kun käyttäjä yhdistää käyttöliittymästä palvelimeen, palvelin odottaa käyttäjältä viestin, joka sisältää palvelimen muistiin tallennetun uniikin avaimen, joka vanhenee ajan kuluessa. Jos palvelin ei vastaanota tietyn ajan jälkeen viestiä, palvelin pudottaa käyttäjän yhteyden.

5.1.3 Autentikointi

Käyttäjä voi yhdistää WebSocket-palvelimeen vain, vanhenevan ja uniikin avaimen avulla. Avaimen saa kirjautumalla palvelimeen sisään, joten palveluun päätettiin lisätä erillinen http-palvelin, joka tarkistaa käyttäjän tunnukset ja antaa käyttäjälle avaimen onnistuneella kirjautumisella. Kuten WebSocket-palvelimet, kirjautumispalvelin vaati NPM-paketin toimiakseen, joten ”http” paketti asennettiin palvelinta varten.

Kirjautumispalveluun asetettiin kuuntelemaan http-pyyntöä, joka sisältää käyttäjän käyttäjätunnuksen, sekä salasanan. Näitä verrataan tietokannassa oleviin tietoihin ja viestiin vastataan käyttäjän tiedoilla, jos käyttäjätunnusta vastaava käyttäjä on löytynyt, sekä annetun salasanan ”hash”-kenttä vastaa tietokannassa olevaa. Tämä mahdollistaa käyttöliittymässä palveluun kirjautumisen.

5.1.4 Viestien käsittely

Laitteen autentikoinnin jälkeen ohjelma asetettiin vastaanottamaan tapahtuma-pohjaisesti laitteelta viestejä. Ensimmäisenä vaiheena tietojen käsittelyssä tarkistetaan, sisältääkö viesti ”cmd”-kentän. Tämä kenttä on numero, joka vastaa päätelaitteessa, palvelimessa ja käyttöliittymässä yhteisesti määriteltyä komentoa.

Päätelaitteelta saapuvan dataviestin lukuarvoon täytyi tehdä lisää tarkistuksia. Jos laite ei jostain syystä pysty lukemaan omia- tai renkilaitteiden tietoja, laite asettaa luetut lukuarvot lukuun, jotka ovat laitteelle mahdottomia. Esimerkiksi lämpötilan mittauksen epäonnistuessa, laite palauttaa 16-bittisen kokonaisluvun 0x8000, joka tarkoittaisi, että lämpötila-arvo olisi -32768°C. Tämän kor-

jaamiseksi ohjelmaan lisättiin suodatin, joka suodattaa selvästi virheellisesti luettuja arvoja, sekä kirjaa ylös virheellisesti luetut arvot.

Date	Log
08.09.2022 15:30:40	No errors
08.09.2022 15:30:24	Discarded invalid temperature value

Kuvio 5. Virhearvojen esittäminen logeissa

Käsittelyn jälkeen, päätelaitteen dataviestin tiedot tallennetaan tietokantaan, joka mahdollistaa historiallisien tietojen tutkimisen. Viestin sisältö välitetään myös WebSocket-yhteyden välityksellä käyttöliittymään. Dataviesti välitetään vain käyttöliittymän käyttäjille, jotka ovat laitteelle asetetussa organisaatiossa.

5.1.5 Tietokantaratkaisun vaihtaminen

Palvelun ollessa tuotannossa muutaman kuukauden ajan ja SQLite-kannan koko alkoi lähentelemään yhtä gigatavua, palvelussa ilmeni selvä hitaus. Historiallisen datan kyselyissä saattoi kestää jopa kymmeniä sekunteja. Tässä vaiheessa tehtiin päätös, että tietokantaratkaisu on vaihdettava tietokantaan, joka skaalautuu paremmin.

Uudeksi tietokantaratkaisuksi valittiin MariaDB-palvelin. SQLite-tietokannan kyselyiden syntaksi on hieman erilainen kuin MariaDB'n, joten migraatiota varten kirjoitettiin skripti, joka hoitaa tietokannan rakenteen, sekä datan siirtämisen. Tietokannan rakenteeseen tehtiin myös pieniä muutoksia samalla kannan nopeuttamiseksi. MariaDB-palvelin ei vielä ollut päivityksen jälkeen erityisesti nopeampi ja palvelin vaati "innodb_buffer_pool_size" asetuksen muuttamista, jolla voi säätää InnoDB-moottorin puskurin muistin käyttömäärää. Asetus on oletuksena 128Mt ja sen muuttamalla 2Gt, saatiin tietokannan haut pudotettua noin sekuntiin suurella määrällä dataa.

5.2 Käyttöliittymä

5.2.1 Ympäristö

Käyttöliittymän kehitykseen valittiin React, koska se on melko helppo tapa tehdä moderneja web-käyttöliittymiä. Reactin etu on varsinkin suuremmalla skaalalla, kun ominaisuuksia ja näkymiä alkaa olemaan useampia. Pelkällä HTML, JavaScript ja CSS- työkaluilla tehdyt sivustot saattavat muuttua myöhemmin vaikeammaksi lukea. React-projektista on helppo myös jälkeenpäin tehdä esimerkiksi React-Native-projekti, jonka voi jakaa mobiiliapplikaationa.

Ympäristön alustaminen alkoi tekemällä uusi React-projekti. Helpoiten React-projektin sai tehtyä käyttämällä NPX-työkalua. Kirjoittamalla komentolinjalle ”npx create-react-app lcss-web”, NPX asentaa ja ajaa automaattisesti ”create-react-app”-työkalun, joka tekee hakemiston ”lcss-web” ja alustaa projektin siihen hakemistoon. Työkalu tekee oletushakemistorakenteen ja lisää muutamia tarvittavia tiedostoja projektin juureen. Tämän jälkeen hakemiston sisällä voi ajaa ”npm start” komennon, joka käynnistää sisäisen palvelimen, josta selaimella voi tarkastella käyttöliittymää.

5.2.2 WebSocket-yhteys

WebSocket-yhteys palvelimeen avataan heti sivun avautuessa. Sivu lähettää selaimen ”localStorage”-muistiin tallennetun avaimen palvelimelle, joka tarkastaa, onko avain vielä voimassa. Jos avain on vanhentunut, palvelin vastaa virheviestillä ja käyttöliittymä ohjautuu kirjautumisnäkymään. Yhteyden avautuessa käyttöliittymä aloittaa palvelimelta lähetettyjen viestien kuuntelemisen, vastaanottaakseen reaaliaikaista dataa laitteilta.

5.2.3 Navigointi

Käyttöliittymään vaadittiin useampi näkymä, joka vaatii käyttöliittymältä navigointiratkaisun. Navigoinnissa päätettiin käyttää react-router-kirjastoa. Kirjaston avulla kehittäjä voi lisätä sivuille linkkejä, jotka vaihtavat palvelun näkymää päivittämättä sivua. Sivut, joissa näytetään laitteiden yksityiskohtaisia tietoja, löytyy myös osoitteesta, jonka osoitteen kyselyssä on laitteen ID. Tämän avulla yksittäisen laitteen yksityiskohtasivun osoitteen voi jakaa muille käyttäjille.

5.3 Päätelaite

5.3.1 Ympäristö

Päätelaitteen ympäristö vaati Arduinon ohjelmistoja. Asentamalla Arduino IDE-kehitysympäristön, sai käyttöön kaikki tarvittavat Arduinon kirjastot, sekä käyttöliittymät. Oletuksena Arduino asentaa ainoastaan Arduinon omien kehitysalustojen kirjastot, mutta päätelaitteen mikrokontrollerit ovat Espressif'n ESP32-mallia, joten ympäristö vaati vielä Arduino-ESP32-kirjaston.

Arduino IDE-kehitysympäristössä on käyttöliittymä kolmannen osapuolen kirjastojen ja kehitysalustojen asentamiselle, kuten ESP-32-mikrokontrollerille. Arduino IDE ei ole tekstieditorina kovin monipuolinen, joten ohjelmointi toteutettiin Visual Studio Code-ohjelmalla. VS Code-tekstieditoriin pystyi asentamaan Arduino-lisäosan, joka mahdollisti koodin kääntämisen ja lähettämisen mikrokontrollerille suoraan VS Coden käyttöliittymästä.

5.3.2 Yhteystestaus

Vaikka päätelaitteen desinfiointiosuuden ohjelma oli jo enimmäkseen kirjoitettu, niin kehitys alkoi tekemällä yhteystestaus ESP32-mikrokontrollerilta palvelimeen tyhjällä projektilla. Desinfiointijärjestelmän laitteisto ei ollut vielä saatavilla, joten tällä projektilla pystyi testaamaan yhteyden muodostamista, sekä alustavia viestirakenteita.

Ensimmäisenä yhteydentestausprojektiin lisättiin ohjelma, joka yhdistää WiFi-yhteyspisteeseen. Tämä avaa laitteelle mahdollisuuden yhdistää sisäisessä tai ulkoisessa verkossa olevaan palvelimeen. Testaus vaati myös WebSocket-yhteyden muodostamisen palvelimeen. Tämän ratkaisemiseksi asennettiin Arduino IDE-käyttöliittymästä ”ArduinoWebSockets”-niminen kirjasto. Tämä kirjasto antoi helpon rajapinnan WebSocket-yhteyden muodostamiseen, sekä viestien lähetykseen ja vastaanottoon. Palvelussa vaadittiin viestien formaatiksi JSON-muoto, jonka lukemiseen ja kirjoittamiseen päätelaitteessa soveltui ”ArduinoJSON”-kirjasto. Tämän kirjaston pystyi myös asentamaan Arduino IDE-käyttöliittymästä.

Ohjelman kirjoittaminen alkoi määrittelemällä WiFi-yhteyspisteen SSID ja salasana, sekä WebSocket-palvelimen IP-osoite ja portti. Tämän jälkeen ohjelman alussa käytetään ESP32:n ”WiFi”-kirjastoa, joka hoitaa yhteyspisteeseen yhdistämisen helposti. Kun yhteys on muodostettu,

käytetään ”ArduinoWebSockets”-kirjastoa WebSocket-yhteyden avaamiseksi. Jos WebSocket-yhteys aukeaa palvelimeen, ohjelma asetettiin menevään silmukkaan, joka kymmenen sekunnin välein luo satunnaista tietoa, joka muistuttaa palvelussa lähetettävää tietoa. Tiedon luonnin jälkeen tieto pakataan JSON-formaattiin ”ArduinoJSON”-kirjaston avulla ja lähetetään WebSocket-yhteyden välityksellä palvelimelle. WebSocket-palvelimella voidaan tarkastella saapuvia viestejä ja niiden eheyttä. Yhteystestauksen tulokset todettiin toimivaksi, joten tätä toiminnallisuutta voi alkaa integroimaan desinfiointijärjestelmän ohjelmistoon.

5.3.3 Yhteysprosessi

Desinfiointijärjestelmän LCDIO- ja LC-laitteet hoitavat useita toimintoja samanaikaisesti. Ohjelmassa on tärkeää, ettei yksi toiminto estä muiden toimintojen ajoa pidempiä aikoja. Laitteissa on käytössä FreeRTOS-käyttöjärjestelmä, jonka avulla voi ajaa prosesseja omissa säikeissään asynkronisesti. Palvelun yhteyden ylläpito siis täytyi hoitaa myös omassa prosessissaan.

Palvelulle tehtiin projektiin uudet tiedostot LCSS.cpp ja LCSS.h, johon toiminnallisuuksia ja määrityksiä alettiin rakentaa. LCSS.h tiedostoon määriteltiin väliaikaisesti kovakoodattuna WiFi-yhteyspisteen ja palvelimen tiedot. LCSS.cpp tiedostossa muodostettiin funktio prosessille, nimeltä ”LCSStask”.

LCDIO-kortin koodin alustuksessa määritellään prosessit. Tähän alustukseen lisättiin myös palvelun yhteydenhallintaprosessi.

```
#ifdef LCSS_IN_USE
    delay(1);
    xTaskCreatePinnedToCore(
        LCSStask,
        "LCSStask",
        8192,
        NULL,
        1,
        NULL,
        ARDUINO_RUNNING_CORE
    );

#endif // LCSS_IN_USE
```

Kuvio 6. Yhteydenhallintaprosessin määrittely

Prosessin määrittelyssä asetetaan pinon koko, johon valittiin alustavasti 8192 tavua. Suuren pinon syy on enimmäkseen ArduinoJson-kirjaston muistinkäyttö, kun luetaan tai muodostetaan pitkiä viestejä. Lisäksi määrittelyssä asetetaan prioriteetti, joka voi tällä prosessilla olla melko matala, sillä muiden prosessien ajo on tärkeämpää ja tulisi olla etusijalla.

Yhteysprosessi ei voi alkaa liian aikaisin, koska se muuten mahdollisesti sekoittaisi muiden prosessien alustusta, tai lähettäisi palvelimelle mittaamatonta tietoa. Prosessin täytyy myös pysähtyä, jos laite on ”setup”-tilassa, koska tässä tilassa mittauksia ei tehdä. Tämän vuoksi prosessi alkaa odottamalla, kunnes pääprosessi ei ole enää alustustilassa. Alustustilan loppuessa, yhteysprosessi hakee kiertokytkimen asennon, joka määrittää onko laite ”setup”-tilassa. Jos kiertokytkimen asento on ”1”, prosessi jatkuu, muussa tapauksessa prosessi lopetetaan.

Laite vaatii tavan tunnistautua palveluun, että palvelin tietää, miltä laitteelta viestit tulevat. Tämä ratkaistiin käyttämällä ESP32-mikrokontrolleriin esiohjelmoitua uniikkia MAC-osoitetta. Ohjelmaan lisättiin funktio, joka hakee MAC-osoitteen ja tallentaa sen muistiin, jonka jälkeen sitä voi käyttää laitteen tunnisteena.

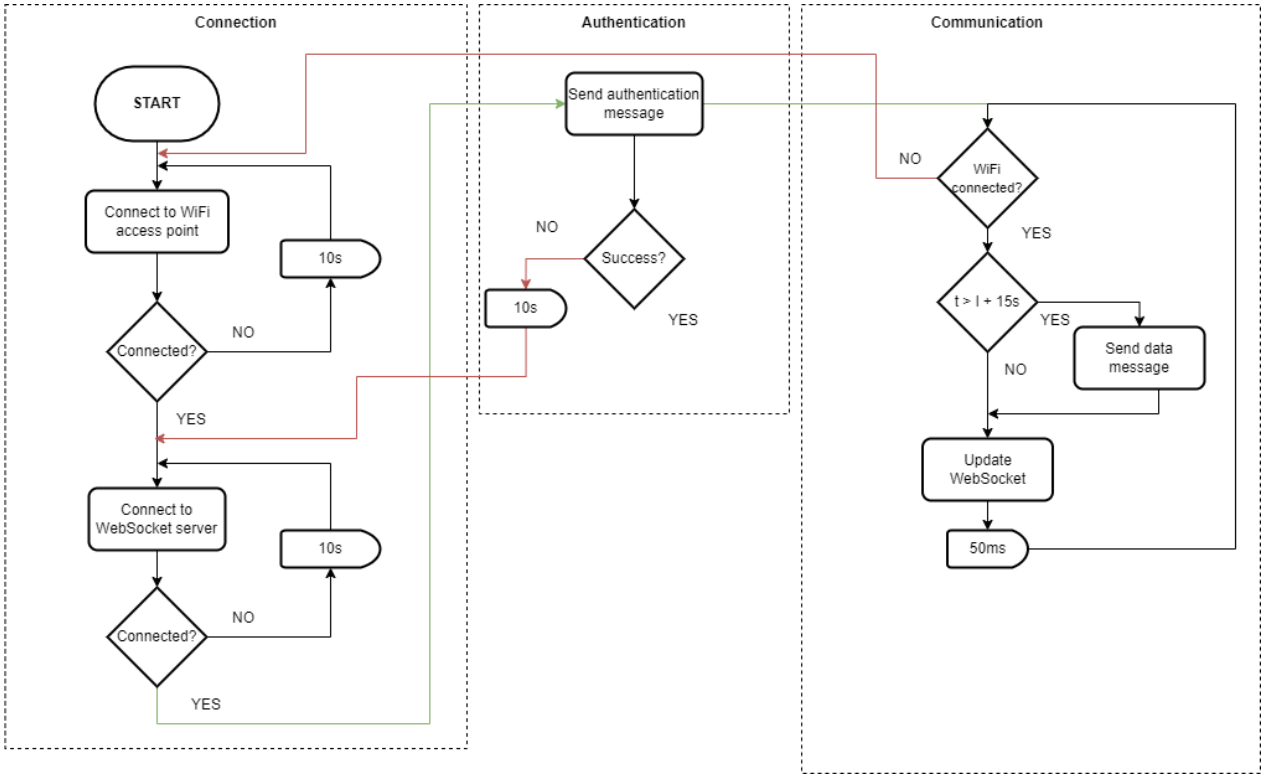
5.3.4 Yhdistäminen palvelimeen

Palvelimeen yhdistäminen vaatii internet-yhteyden, joka muodostetaan projektissa WiFi-yhteydellä. Kuten yhteyden testauksessa, projektiin on määritelty vakiona WiFi-yhteyspisteen SSID, sekä salasana.

Ohjelmaan lisättiin silmukka, joka toistuvasti yrittää yhdistää WiFi-yhteyspisteeseen. Yhteyden muodostuksen epäonnistuessa, ohjelma odottaa kymmenen sekuntia ja yrittää uudelleen. Yhteyden muodostuessa prosessi poistuu silmukasta, sekä aloittaa palvelimeen yhdistämisen.

Palvelimen IP-osoite ja portti ovat myös kovakoodattu ohjelmaan. ”ArduinoWebSockets”-kirjaston yhteyden kuunteleminen on tapahtumapohjainen. Täten kirjasto ei estä prosessia ajamasta kuunnellessa tapahtumia, vaan kirjaston WebSocketsClient-luokan ”loop”-funktion ajaessa, kirjasto tarkastaa vastaanotettua tietoa ja sen perusteella laukaisee tapahtuman. Tapahtumille määritetään ohjelmassa funktio.

WebSocket-yhteys luodaan prosessissa WiFi-yhteyden luonnin jälkeen ja prosessi siirtyy ikuiseen lähetyssilmukkaan.



Kuvio 7. Yhteysprosessin kaavio

5.3.5 Tietojen lähetys

Tietojen lähetys alkaa autentikointiviestillä, joka sisältää laitteen tunnisteen. Laitteen tunnisteeksi valittiin ESP32-kontrollerin MAC-osoite. Viesti sisältää myös uniikin tunnistautumisavaimen, laitteen fyysiset koordinaatit, laitteiston versio, laiteohjelmiston versio ja ID, sekä PWM-kanavien jännite- ja virtarajat. Viesti kootaan ArduinoJSON-kirjastoa käyttäen JSON-muotoon, joka lähetetään palvelimelle. Palvelin vastaa autentikointiviestiin viestillä, jossa kerrotaan, onko todennus hyväksytty vai hylätty. Palvelimen mahdollisen hylkäyksen jälkeen ohjelma yrittää yhdistää uudelleen palvelimeen.

Hyväksytyn autentikoinnin jälkeen ohjelma siirtyy lähetystilaan, joka lähettää laitteiden tietoja vakion aikavälin välein. Yhteyden ylläpito vaatii tiheää päivitystä ja tietojen lähetys tapahtuu samassa prosessissa, joten prosessia estävä viive ei käy. Tämä ratkaistiin vertaamalla viime viestin lähetysaikaa nykyisellä ajalla prosessin silmukassa ja lähettämällä viestin vasta aikaeron ollessa määritettyä vakiota suurempi.

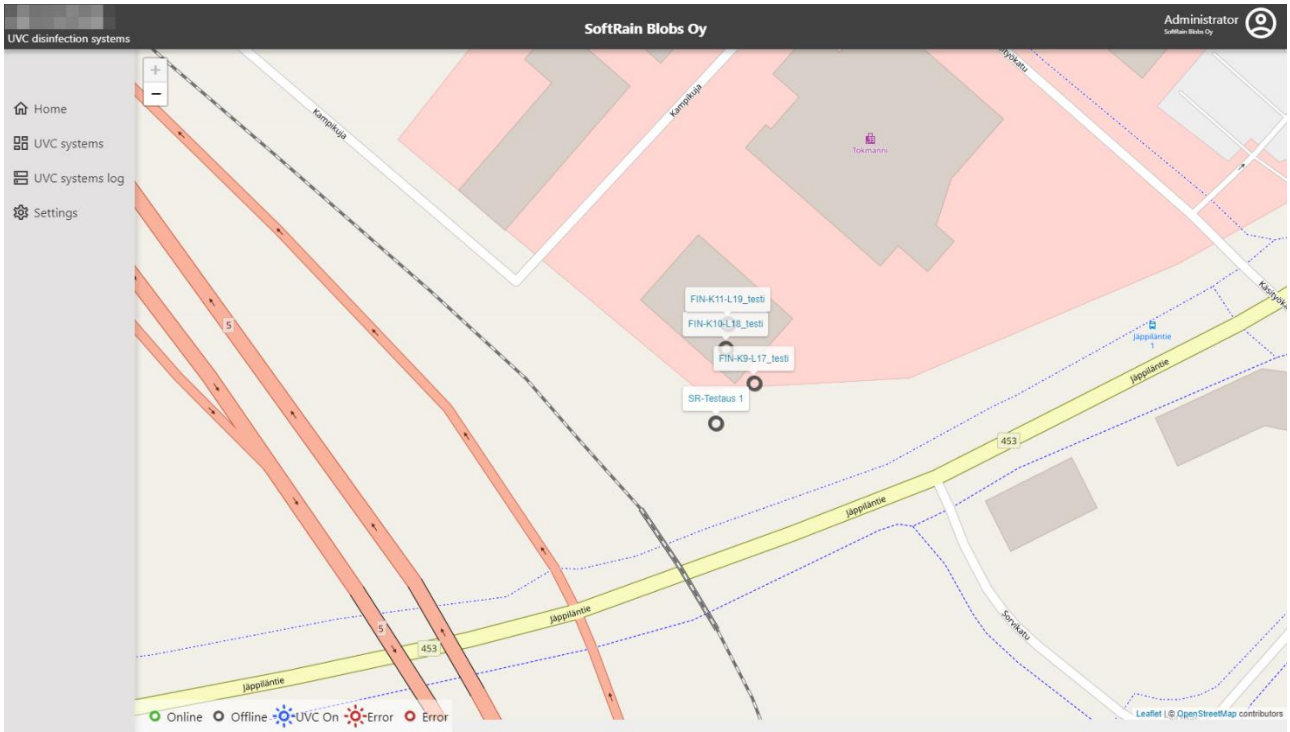
5.3.6 Päätelaitteen ohjelmistopäivitys

Järjestelmää kehittäessä huomattiin, että järjestelmä saattaa vaatia uusia ominaisuuksia jatkossa, joita ei ehditty ensimmäiseen tilaukseen toteuttamaan. Järjestelmien päivittäminen paikallisesti on työlästä laitteiden fyysisen sijainnin vuoksi, joten suunnitelmiin lisättiin mahdollisuus etäpäivitykselle.

ESP32:n kirjastoissa toteutetaan OTA (Over The Air) päivityksen mahdollisuus, jota pystyi hyödyntämään laitteen päivittämisessä palvelimen kautta. Päivitysjärjestelmän kehityksessä kesti vain muutama päivä ja sillä turvattiin laitteen toiminta, vaikka ohjelmistossa olisi virheitä.

6 Tulokset

6.1 Kotinäkymä

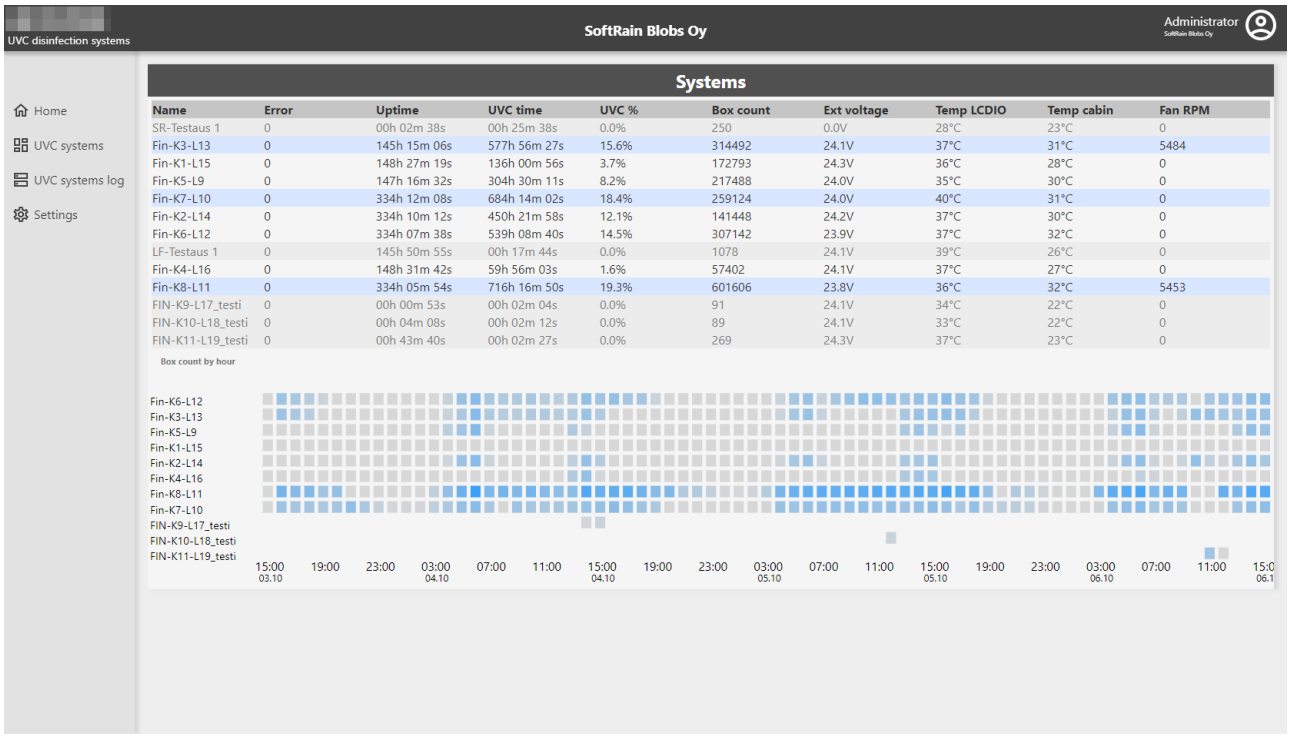


Kuvio 8. Koti- ja karttanäkymä

Kotinäkymä tulee esiin karttana, jossa näytetään kaikkien laitteiden fyysinen sijainti kartalla. Laitteiden karttaikoneissa myös tulee esiin laitteen tila, joka kertoo, onko laite päällä, pois päältä, vir-

hetilassa tai onko laitteen UV-valot päällä. Kotinäkymä luo helpon yleiskatsauksen järjestelmien tilasta. Kaikilla käyttäjätyypeillä on pääsy kotinäkymään.

6.2 Järjestelmien yleisnäkymä



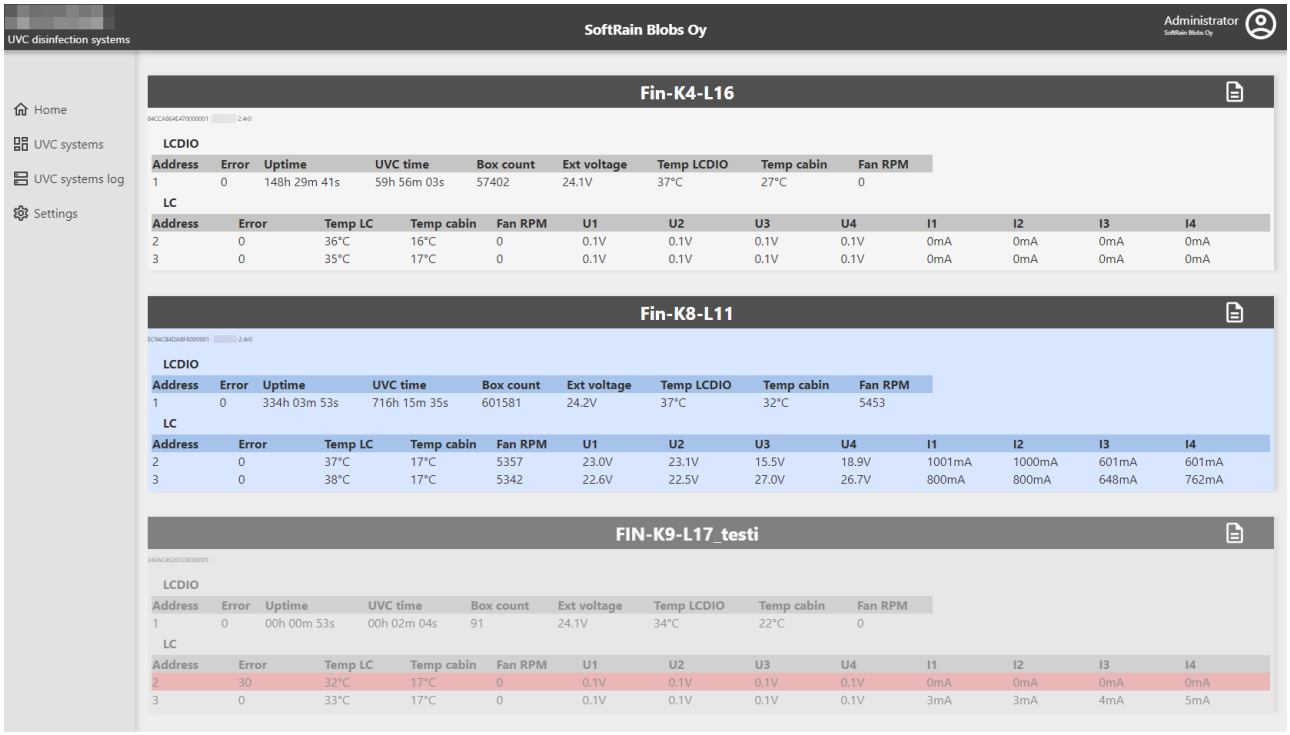
Kuvio 9. Järjestelmien yleisnäkymä

Järjestelmien yleisnäkymässä käyttäjä näkee kaikkien järjestelmien yleiset tiedot. Näkymä on enimmäkseen asiakkaalle, joka voi tarkkailla laitteiden tietoja, kuten UV-C valon päälläoloaikaa, sekä joitain asiakaskohtaisia arvoja kuten ”Box count”-kenttä tässä tapauksessa. Yleisnäkymässä järjestelmien rivit ovat värikoodatut, esittäen yhteydessä olevat järjestelmät valkealla taustalla, virhetilassa olevat punaisella taustalla, sekä sinisellä järjestelmät, joiden UV-C-valot ovat päällä. Epäaktiiviset laitteet ovat harmaalla taustalla.

Yleisnäkymässä on myös lämpökartta-tyylinen esitys ”Box count”-kentästä jokaisella laitteella kolmen vuorokauden ajalta.

Järjestelmien yleisnäkymään pääsee kaikilla käyttäjätyypeillä.

6.3 Laitenäkymä



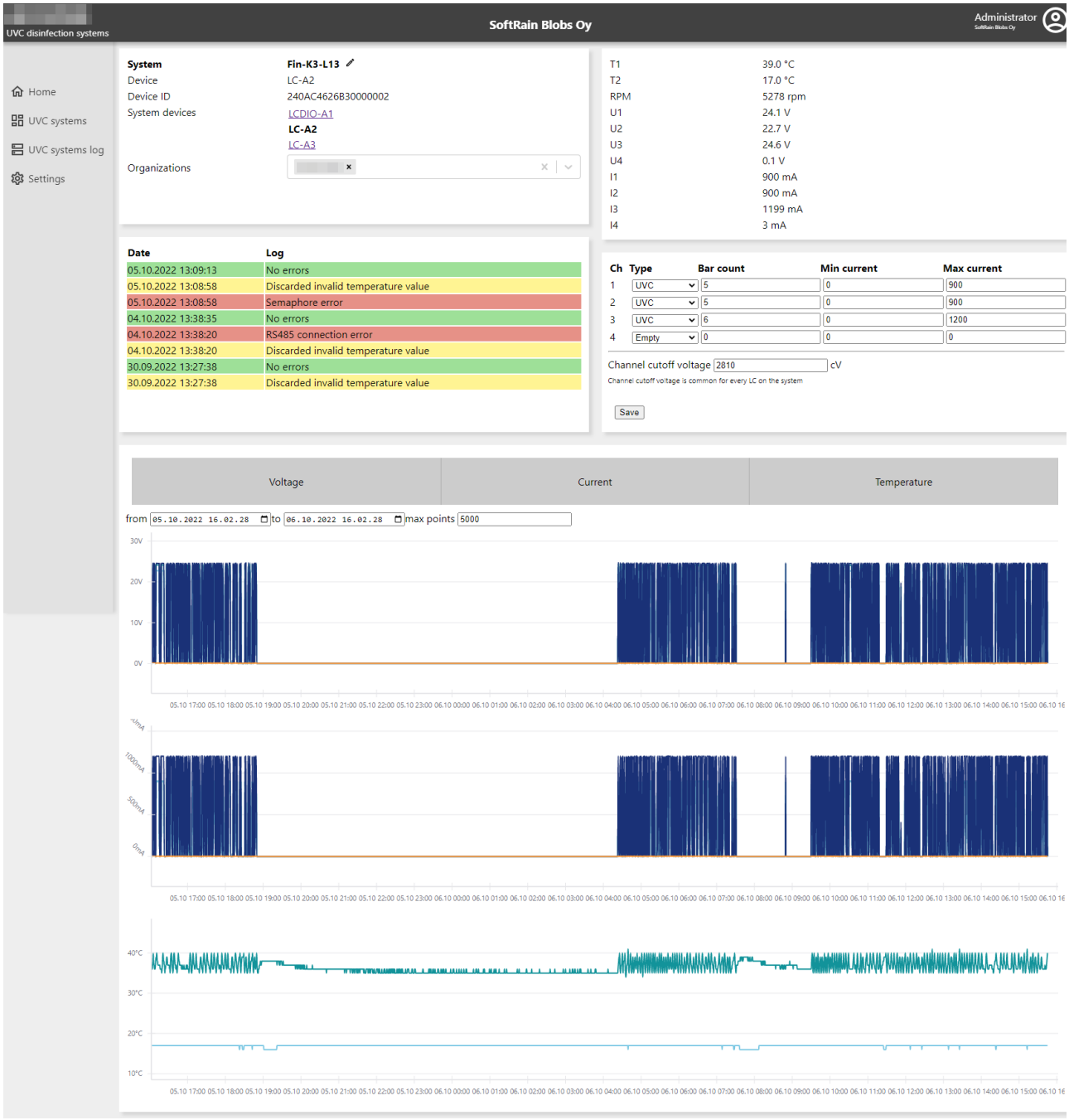
Kuvio 10. Laitenäkymä

Laitenäkymässä näytetään tarkempia tietoja järjestelmistä ja sen sisältämistä laitteista. Näkymä on tarkoitettu järjestelmien huoltajille, jotka pystyvät tutkimaan laitteista yksityiskohtaisempia tietoja. Laitenäkymässä näkyy LCDIO-laitteen tiedot, kuten yleisnäkymässä, mutta laitenäkymässä näkee myös LC-laitteiden tiedot.

Näkymästä voi siirtyä laitteen yksityiskohtaiseen näkymään klikkaamalla jotain laitteen riviä, sekä lisäämään huoltotoimenpiteisiin liittyviä merkintöjä oikeassa ylänurkassa olevasta ikonista.

Tämä näkymä on tarkoitettu ainoastaan ”admin”-tason käyttäjille.

6.4 Laitteiden yksityiskohtainen näkymä



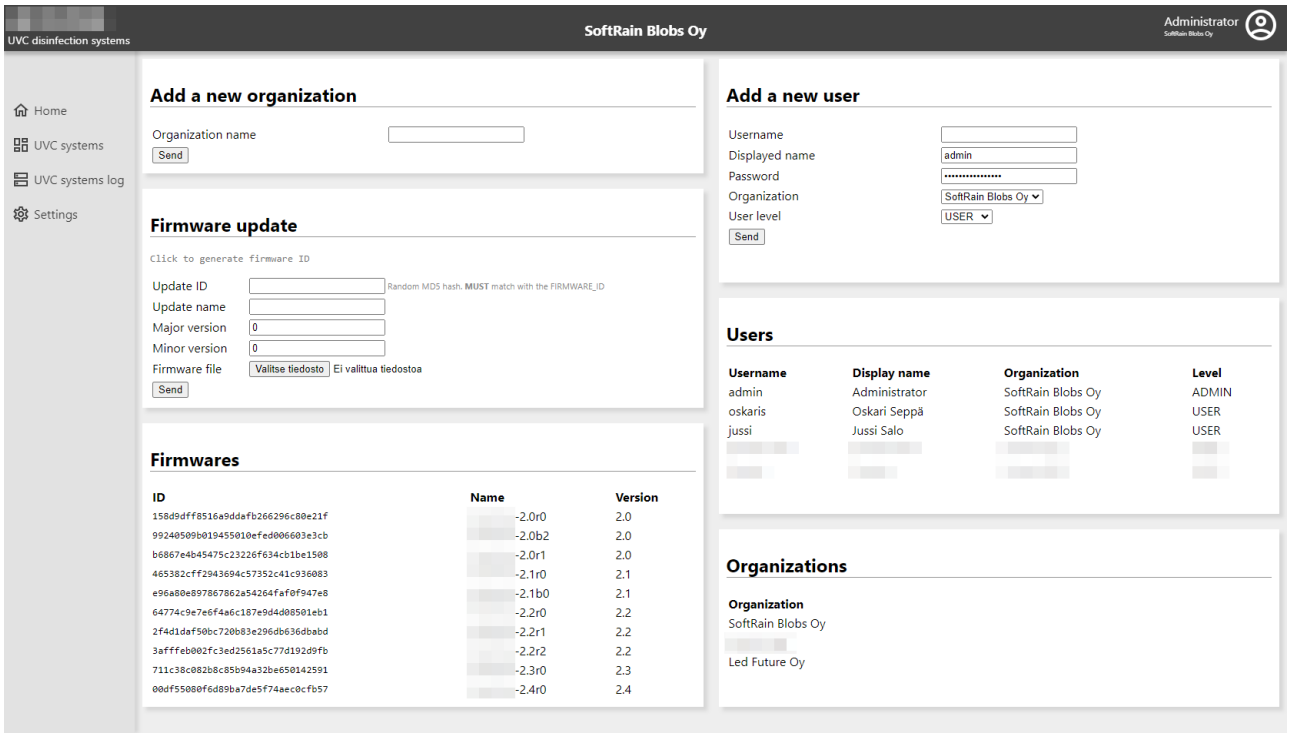
Kuvio 11. LC-laitteen yksityiskohtainen näkymä

Laitteiden yksityiskohtaisessa näkymässä voi tarkastella yksittäisen laitteen kaikkia tietoja. Näkymästä näkee kirjaukset laitteella tapahtuneista virheistä tai varoituksista, sekä trendin jännitteestä, virrasta ja laitteen lämpötilasta käyttäjän valitsemalla aikavälillä.

Näkymässä voi tehdä myös laitekohtaisia muutoksia, kuten järjestelmän nimen tai organisaation asettaminen, sekä LC-korttien kanavamäärittelyt.

Tämä näkymä on tarkoitettu ainoastaan ”admin”-käyttäjille.

6.5 Palvelimen asetusnäkymä



Kuvio 12. Palvelun asetusnäkymä

Palvelimen asetusnäkymässä voi hallita käyttäjiä ja järjestelmäpäivityksiä.

7 Pohdinta

Palvelua kehitettiin noin kolme kuukautta ennen palvelun käyttöönottoa. Päätelaitteen ohjelmistopäivityksen ansiosta palvelu voitiin ottaa käyttöön lyhyen kehityksen jälkeen, vaikka kaikki ominaisuudet eivät olleet välttämättä toivotulla tasolla.

Palvelun kehitys sujui vaatimusten ja aikataulun mukaisesti. Palvelua kehitettiin palvelun jatkokehitystä ja uusia ominaisuuksia ajatellen. Palvelu otettiin käyttöön kesäkuussa 2022, jolloin kahdek-

san järjestelmää vietiin asiakkaalle ja toiminta aloitettiin. Palvelun hyödyllisyys voitiin todistaa muutaman kuukauden käyttöönoton jälkeen. Palvelusta pystyi havaitsemaan epäkunnossa olevan UV-C-valon ja valoa ohjaava kanava voitiin asettaa etänä pois päältä.

Lähteet

ESP32-WROOM-32 Datasheet. 2016. Viitattu 20.10.2022. https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

NodeMCU-32S Core Development Board. N.d. Viitattu 20.10.2022. https://docs.ai-thinker.com/en/esp32/boards/nodemcu_32s

What is Arduino? 2018. Viitattu 21.10.2022. https://www.arduino.cc/en/Guide/Introduction

Arduino core for the ESP32, ESP32-S2, ESP32-S3 and ESP32-C3. 2016. Viitattu 21.10.2022. https://github.com/espressif/arduino-esp32#readme

About Node.js. N.d. Viitattu 21.10.2022. https://nodejs.org/en/about/

What is npm? 2011. Viitattu 21.10.2022. https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/

About SQLite. 2021. Viitattu 21.10.2022. https://www.sqlite.org/about.html

UV-C-säteilyn käyttö desinfioinnissa. 2021. Viitattu 17.5.2022. https://www.stuk.fi/aiheet/uv-sateily-aurinko-ja-solarium/uv-c-sateilyn-kaytto-desinfioinnissa

Fette, I., Google, Inc., Melnikov, A., Isode, Ltd. 2011. The WebSocket Protocol. Viitattu 12.7.2022. https://datatracker.ietf.org/doc/html/rfc6455

Why RTOS and What is RTOS? N.d. Viitattu 17.7.2022. https://www.freertos.org/about-RTOS.html

Pomponio, A. 2021. Viitattu 24.8.2022. https://www.openlogic.com/blog/mysql-vs-sqlite