Opinnäytetyö (AMK)

Tieto- ja viestintätekniikan koulutus

2022

Aleksi Elmroos

SAUNALAUTAN NAVIGOINTIKOMPONENTIN TOTEUTUS HOME ASSISTANTILLA

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2022 | 36 sivua

Aleksi Elmroos

SAUNALAUTAN NAVIGOINTIKOMPONENTIN TOTEUTUS HOME ASSISTANTILLA

Älylaitteiden yleistyessä halutaan laajentaa niiden tarjoamia mahdollisuuksia myös vesillä kulkevaan yksityiseen saunalauttaan. Saunalautalle keskeisen tiedon keruuta, prosessointia ja visualisointia varten hyödynnetään Home Assistant -nimistä automaatioalustaa. Tärkeänä osana lautan toiminnan automaatiota oli mahdollistaa navigointi vesillä Home Assistantin käyttöliittymän avulla, minkä toteutukseen työssä keskityttiin.

Opinnäytetyössä rakennettiin navigointikomponentti Home Assistantin päänäkymään tietokokonaisuuden osaksi korttina eli yhtenä Home Assistantin dataikkunana muiden joukossa. Komponentin piti myös pystyä kommunikoimaan lautan muiden osien kanssa MQTT-viestintäprotokollalla, jotta voidaan kuljettaa koordinaattidataa.

Aluksi toteutettiin komponentin perustana karttanäkymä, jonka päälle kehitettiin vaadittua toiminnallisuutta. Home Assistantin ominaisuuksia hyödyntäen vastaanotettiin viestinvälittäjän kautta lautan sijainti, joka tallennettiin Home Assistantin sisäiseen tietokantaan. Koordinaattihistorian avulla visualisoitiin lautan kulkema reitti karttaan. Sen jälkeen kehitettiin ominaisuus reitin suunnittelua varten karttanäkymää hyödyntäen. Suunnitellun reitin koordinaatit lähetettiin MQTT-pakettina takaisin lautalle, jossa ne myöhemmin käsitellään kulkusuunnan määrittämiseksi.

Lopputuloksena on navigointikomponentti, jonka avulla voidaan helposti seurata lautan kulkemaa reittiä vesillä sekä toteuttaa tulevan reitin hallintaa. Tämän seurauksena saunalautta on askeleen pidemmällä automaatiossa ja älykkyydessä kyeten parempaan toiminnallisuuteen osana esineiden internetiä.

ASIASANAT:

Automaatio, esineiden internet, navigointi, lautat, visualisointi

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2022 | 36 pages

Aleksi Elmroos

IMPLEMENTATION OF A SAUNA RAFT NAVIGATION COMPONENT FOR HOME ASSISTANT

As smart devices become more commonplace, the utility offered by them is employed even in a private sauna raft traveling by water. The raft utilizes an automation platform called Home Assistant for the collection, processing, and visualization of relevant data. As an important part of the automation process, providing a way to navigate the raft from the Home Assistant's user interface comes into focus.

The objective of this thesis was to build a navigation component to function as a part of the Home Assistant dashboard. It must work as a card, which is a window of data among others on the dashboard view. The component must also be able to communicate with other parts of the raft using the MQTT messaging protocol to relay coordinate data.

At first, a map view on which the necessary features would be implemented was created as a base. Using the Home Assistant's built-in functions, the raft's position was received through a message broker, after which it was saved into Home Assistant's internal database. The coordinate history was then used to visualize the route travelled by the raft onto the map. Next, a feature to plan a route utilizing the map view was developed. The planned route's coordinates were then sent over to the raft as an MQTT packet to be processed later for determining the raft's direction of movement.

The result was a navigation component, which can easily track the sauna raft's position and route, as well as manage its future route. In conclusion, the sauna raft has taken a step further along the line of automation and smartness, becoming more capable in its Internet of Things.

KEYWORDS:

Automation, internet of things, navigation, rafts, visualization

SISÄLTÖ

SANASTO

API	Ohjelmointirajapinta
CSS	Nettisivujen muotoiluun käytettävä ohjelmointikieli (Cascading Style Sheets)
Dashboard-näkymä	Home Assistantin näkymä, jonka ylle rakennetaan korttien avulla käyttöliittymä
Docker	Ohjelma, jonka avulla rakennetaan ja ajetaan kontteja
Docker-kontti	Standardoitu suoritettava komponentti, joka koostuu sovelluksen lähdekoodista, kirjastoista ja riippuvaisuuksista [8]
DOM	Dokumenttioliomalli eli web-dokumenttien ohjelmointirajapinta (Document Object Model)
Home Assistant	Avoimen lähdekoodin automaatioalusta
HTML	Nettisivujen luontiin käytetty merkintäkieli (Hyper Text Markup Language)
HTTP	Internetin välityksellä kommunikointiin tarkoitettu protokolla (Hyper Text Transfer Protocol)
ID	Tunniste
JSON	Kevyt datan tallennukseen ja kuljetukseen käytetty formaatti (JavaScript Object Notation)
Kortti	Home Assistantin dashboard-näkymässä toimiva dataa visualisoiva ikkuna
Lovelace	Virallinen nimitys Home Assistantin dashboard-näkymäkokonaisuudelle
MQTT	Kevyt viestintäprotokolla (Message Queue Telemetry Transport)
REST	Ohjelmointirajapintojen toteuttamiseen tarkoitettu arkkitehtuurimalli (Representational State Transfer) [28]

URL	Verkossa sijaitsevan materiaalin osoite (Uniform Resource Locator)
YAML	Datan sarjallistamiseen käytettävä ohjelmointikieli (Yet Another Markup Language)

1 JOHDANTO

Älylaitteiden ja esineiden internetin tuomaa toiminnallisuutta nähdään nykypäivänä entistä enemmän. Kotikäyttöön on jo olemassa monia kaupallisiakin laitteita, joita voidaan hallita verkon välityksellä. Näiden avulla pystyy esimerkiksi seuraamaan sisätilan ilmanlaatua tai hallitsemaan valoja. Mahdollisuuksia on monia, ja ne tuovat uudenlaisen tason ympäristön hallintaan. Tämänlaisia toiminnallisuuksia halutaan myös hyödyntää yksityisessä saunalautassa seuraamaan esimerkiksi saunan lämpötilaa, säätä ja kuljettua reittiä.

Saunalautta hyödyntää Raspberry Pin päällä toimivaa Home Assistant -nimistä automaatioalustaa eri osien hallintaan. Home Assistant on tarkoitettu nimensä mukaisesti kodin automaation yhdistämiseen, mutta sen toiminnallisuus sopii muihinkin ympäristöihin [24]. Sitä voidaan käyttää esimerkiksi visualisoimaan dataa ja toiminnallisuutta ihmisluettavaan muotoon [29]. Home Assistantiin kuuluu visualisoinnin lisäksi muun muassa laite- ja viestihallintaa, kuten MQTT-viestinnän mahdollistaminen [30], mikä on saunalautalle tärkeää. MQTT eli Message Queue Telemetry Transport on yksi tapa viestiä esineiden internetiin yhdistettyjen laitteiden välillä [4]. MQTT-viestintäprotokollan avulla yhdistetään lautan eri komponenttien tarjoamaa dataa ja toiminnallisuutta yhteen paikkaan. Koska viestintä vaatii myös MQTT-viestinvälittäjän, Home Assistant toimii käyttöjärjestelmän sijasta Docker-kontissa, kuten myös viestinvälittäjä Eclipse Mosquitto.

Vaikka Home Assistantissa on olemassa monenlaista valmista toiminnallisuutta, vesillä navigointi on kuitenkin oletuksena sen tarjoaman toiminnallisuuden ulkopuolella. Sitä ei ole myöskään ennen julkisesti käsitelty Home Assistant -alustalla. Karttakortteja on kuitenkin jo olemassa, ja niiden toiminta on pelkän reitin seurauksen kannalta saman tapaista halutun kanssa [1][31]. Kortit ovat Home Assistantin näkymän kokoonpanossa sijaitsevia ikkunoita, jotka visualisoivat määritettyä dataa eri tavoin [29].

Valmiista toteutuksista löytyy esimerkiksi Home Assistantin sisäänrakennettu karttakortti, jonka avulla voi suorittaa laiteseurantaa kartalta halutun pituiselta aikajaksolta [31]. Se on kuitenkin hyvin yksinkertainen eikä mahdollista sen enempää toimintoja. Sen lisäksi löytyy myös mukautettu karttakortti, jonka avulla voi suorittaa laiteseurantaa yksityiskohtaisemmin kuin Home Assistantin omasta kortista. Kyseinen kortti on kuitenkin pääasiassa harjoitusprojekti, joka on toteutettu tekijän itsensä mukaan monimutkaisemmin kuin tarpeellista. [1] Se toimii Google Maps -rajapinnalla, joka ei ole täysin ilmainen

[26], eikä tarjoa vedellä kulkemiseen parempia ominaisuuksia kuin ilmaiset karttapalvelutkaan.

Opinnäytetyössä toteutetaan saunalautan reittiä hallitseva navigointikomponentti Home Assistantin päänäkymään osaksi tietokokonaisuutta mukautettuna korttina. Kehitysvaiheessa keskitytään pääasiassa kortin ja sen toiminnallisuuden luontiin, sillä Home Assistantin ja MQTT-viestinvälittäjän toteutukset ovat jo olemassa. Työssä käydään kuitenkin myös läpi näiden komponenttien toimintaa kortin toteutusprosessin avaamiseksi. Valmiin navigointikomponentin pitää pystyä visualisoimaan kuljettu reitti kartalla sekä mahdollistaa tulevan reitin suunnittelu karttanäkymästä käsin.

2 POHJAELEMENTIT

Opinnäytetyön pohjalla toimivat elementit ovat pääasiassa automaatioalusta Home Assistant ja viestinvälittäjä Eclipse Mosquitto. Saunalauttaan kuuluu muitakin osia, mutta ne eivät kuitenkaan ole navigointikomponentin kannalta tärkeitä. Asiaankuuluvat komponentit toimivat Docker-konttien sisältä, mahdollistaen helpomman kehitystyön. Esimerkiksi Home Assistantin oletusarvoinen asennus loisi kokonaan itsenäisen käyttöjärjestelmän Raspberry Pin päälle.

Docker on avoimen lähdekoodin niin sanotun konttikuljetuksen alusta. Se mahdollistaa kehittäjien pakata sovelluksia kontteihin, jotka ovat standardoituja suoritettavia komponentteja. Ne yhdistävät sovellusten lähdekoodin käyttöjärjestelmän kirjastoihin ja riippuvuuksiin, jotta kontin sisältämää koodia voidaan ajaa missä tahansa ympäristössä. [8] Dockerin konttien avulla kehitys monelta eri alustalta helpottuu ilman monimutkaisia järjestelyjä kehitysympäristön suhteen. Tämä on myös hyödyllistä, mikäli saunalautassa tapahtuu komponenttimuutoksia.

2.1 Home Assistant

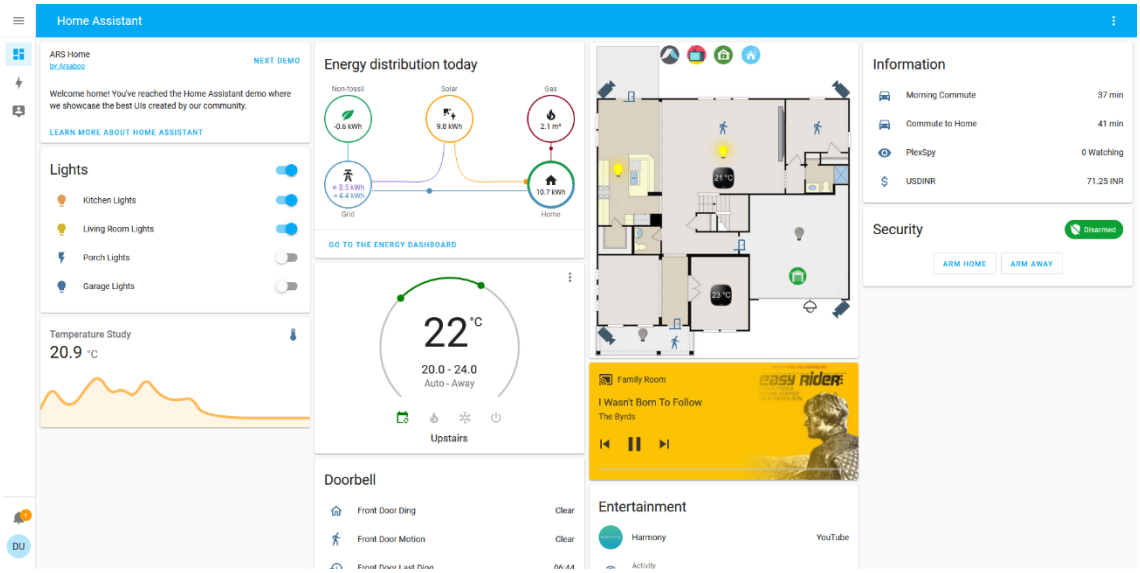
Home Assistant on avoimen lähdekoodin esineiden internetin automaation työkalu, joka keskittyy hallintaan ja yksityisyyteen [24]. Home Assistantin avulla voidaan kerätä eri kohteista dataa, ja kohdistaa se kaikki yhteen paikkaan helppolukuiseen muotoon [29]. Home Assistantin käyttöliittymä myös mahdollistaa monen eri tyylisen prosessin hallinnan käyttäjän määrittelemillä automaatioilla [24]. Näiden työkalujen määrittely onnistuu Home Assistantin konfiguraatiotiedostosta *configuration.yaml*.

Home Assistantin konfiguraatiosta löytyy tärkeimpiä määrityksiä, kuten laiteseurannan laitteet, MQTT:n määritelmä ja tietokannan seuraamat entiteetit. Se on lautan tiedon seurannan kannalta tärkeä tiedosto, joka on jatkuvasti kehityksessä mukana. Tämän takia on tärkeää, että sen versiota hallitaan esimerkiksi GitHubissa. Oletuksena konfiguraation hallinta toimii Home Assistantin käyttöliittymästä, joten tiedosto on määritetty luettavaksi ja käsiteltäväksi YAML-muodossa jakamista varten. YAML eli Yet Another Markup Language on ohjelmointikieli, joka on tarkoitettu datan serialisointiin. YAML-muodon määritys tarkoittaa sitä, että konfiguraatiota muokataan vain suoraan tiedostosta käsin. Jotta

eri data ja toiminnallisuus saadaan tiivistettyä yhteen paikkaan, käytetään hyödyksi Lovelacea eli Home Assistantin käyttöliittymää.

2.1.1 Lovelace-käyttöliittymä

Lovelacen avulla on mahdollista määrittää näkymiä Home Assistantin päälle [29]. Sen avulla on luotu päänäkymä, joka sisältää lautalle asiaankuuluvaa tietoa. Lovelacen konfiguraatiolla saadaan määriteltyä näkymään esimerkiksi sisäänrakennettuja kortteja, joiden kanssa voidaan kuvastaa monenlaista dataa kuvan 1 mukaisesti [29]. Näihin sisältyy muun muassa sensoreiden tiloja, valojen säätökytkimiä tai kuvia. Näiden korttien lisäksi on myös mahdollista luoda mukautetun toiminnallisuuden kortteja JavaScriptin avulla [5], ja tätä toiminnallisuutta hyödyntäen luodaan navigointikomponentin kortti.



Kuva 1. Esimerkki Home Assistantin päänäkymästä [25].

Lovelacen konfiguraatiotiedoston *lovelace-ui.yaml* avulla voidaan määrittää Home Assistantin näkymän sisältö ja toiminnallisuus. Sillä saadaan esimerkiksi määriteltyä korttien käyttämät entiteetit, joka mahdollistaa entiteettien sisältämän datan käsittelyn. Entiteetit voivat olla monenlaisia datan kerääjiä, kuten esimerkiksi device tracker -nimellä toimivia laiteseurannan muuttujia, joihin tallentuu erilaisista laitteista kerättyä statustietoa.

2.1.2 Recorder-integraatio

Koska navigointikomponentti käsittelee lautan kuljetun reitin koordinaattidataa, minkä se vastaanottaa vain kerran koordinaatteja kohden, sen pitää pystyä myös pitämään kyseinen data tallessa. Tätä varten hyödynnetään Home Assistantin sisäänrakennettua recorder-nimistä integraatiota, joka mahdollistaa Home Assistantin komponenttien datan tallennuksen sisäiseen tietokantaan [32].

Recorder-integraatio toimii keräten tapahtumia ja attribuutteja konfiguraatioon määritetyistä komponenteista [32]. Oletuksena tietokanta toimii SQLite-alustalla, mutta muiden tietokantojen käyttö on myös mahdollista [9]. Koska tällä hetkellä ei ole tarvetta monimutkaisempaan tietokannan käsittelyyn, voidaan käyttää oletusarvoa. Tietokantaa käyttämällä saadaan tallennettua koordinaattidataa sekä haettua sitä tarvittaessa. Lautan kulkeman reitin koordinaatit saapuvat MQTT-integraation kautta, joka käyttää viestinvälittäjänä Eclipse Mosquittoa.

2.2 Viestinvälittäjä Eclipse Mosquitto

Eclipse Mosquitto on kevyt avoimen lähdekoodin viestinvälittäjä, joka hyödyntää MQTT-viestintäprotokollaa [33]. Viestinvälittäjät ovat ohjelmistoja, jotka mahdollistavat sovellusten, järjestelmien ja palveluiden kommunikaation ja tiedonvaihdon toistensa kanssa [2]. Eclipse Mosquitton avulla voidaan viestiä muiden komponenttien kanssa käyttäen eri viestikanavia eli aiheita. Ohjelma toimii Docker-kontin sisältä, ja sen osoite on määritelty Home Assistantin konfiguraatioon käytettäväksi MQTT-integraation kanssa. Integraation avulla voidaan manuaalisen viestinnän lisäksi luoda aiheita lukevia automaatioita.

MQTT on viestintäprotokolla eli laitteiden välisen kommunikaation määritelmä, joka on suunniteltu kevyeksi julkaisu/tilaus -menetelmää hyödyntäväksi viestien kuljettajaksi [3]. MQTT-protokolla toimii määrittämällä viestinvälittäjän ja asiakaskoneet. Viestinvälittäjä vastaanottaa kaikki julkaistut viestit asiakaskoneen määrittämältä aiheelta ja ohjaa ne eteenpäin viestin aiheen tilaajille. Asiakaskone pystyy tilaamaan monta eri aihetta viestinvälittäjältä. Kun asiakaskone julkaisee viestin näillä aiheilla, kaikki muut saman aiheen tilanneet asiakaskoneet saavat myös viestin itselleen viestinvälittäjän kautta. [4]

Kevyenä protokollana MQTT on sopiva monenlaiseen mobiiliympäristöön, ja tätä voidaan hyödyntää myös vesillä liikkuessa niin Raspberry Pin suorituskyvyn kuin

mahdollisesti heikon verkkoyhteydenkin kannalta. Tämän mahdollistaa asynkroninen eli ei-reaaliaikainen protokollan toiminta, joka helpottaa viestintää erottamalla asiakaskoneet tilassa ja ajassa [4].

3 NAVIGOINTIKOMPONENTIN RIIPPUVUUDET

Navigointikomponentin luomiseksi tarvitaan ensin sille pohja, joka on Home Assistantin mukautettu kortti eli kortti, joka on luotu tyhjästä käsin. Mukautetut kortit toimivat HTML-pohjaisesti mukautettua elementtiä (engl. custom element) hyödyntäen [5]. HTML eli Hyper Text Markup Language on nettisivujen luontiin käytettävä merkintäkieli. Mukautetut elementit mahdollistavat kehittäjien luoda uudenlaisia HTML-elementtejä. Niiden avulla voidaan laajentaa muiden elementtien toiminnallisuutta, sitoa toiminnallisuutta yhteen ja laajentaa olemassa olevien DOM-elementtien ohjelmointirajapintaa. [10] Tämä mahdollistaa Home Assistantilla korttien yhdenmukaisen käsittelyn.

DOM eli dokumenttioliomalli on ohjelmointirajapinta käytettäväksi HTLM- ja XML-dokumenteissa. Se määrittää dokumenttien rakenteen, sekä tavan päästä niihin käsiksi ja käsitellä niitä. [11] Korttien luonti tapahtuu JavaScriptilla, jolla käsitellään kortin DOMia esimerkiksi ohjelmistokehyksen avulla, kuten monissa nettisivuissakin.

Kortin sisällön eli kartan rakentamiseksi tarvitaan koodikirjasto, joka osaa renderöidä kartan ja antaa pääsyn sen sisältöön. Sen avulla tehdään myös mahdolliseksi koordinaattien visualisointi kartalla pisteinä tai viivoina pisteiden välillä. Koodikirjastot ovat yleensä myös ylläpidettyjä ja dokumentoituja, ja valmista vapaaseen käyttöön tarkoitettua kirjastoa käyttämällä säästytään turhalta työltä.

3.1 Ohjelmistokehys ja -kirjasto

Ohjelmistokehys on rakennelma, jonka päälle rakennetaan ohjelmistoa. Se toimii perustana, ettei kehitystyötä tarvitse aloittaa tyhjästä. [12] Sen avulla lisätään yksinkertaistettua toiminnallisuutta kehitysympäristöön.

Ohjelmistokehyksen valintaan vaihtoehtoja on monia, kuten suositut Polymer tai Angular, muttei kuitenkaan React, joka aiheuttaa ongelmia mukautettujen elementtien kanssa [5]. Kehyksen ei kuitenkaan aina tarvitse olla monimutkainen, ja saunalautan mobiiliympäristön tarpeisiin yksinkertainen toteutus on paras vaihtoehto. On olemassa ohjelmistokehyksen lailla toimivia kirjastoja, jotka tarjoavat samantapaista toiminnallisuutta yksinkertaisemmassa muodossa, ja näitä hyödyntäen saadaan kevennettyä kokonaisuutta. Valitaan kirjasto, joka on kevyt ja toimii myös Home Assistantin ympäristössä.

LitElement on yksinkertainen pohjaluokka nopeiden ja kevyiden web-komponenttien luontiin, ja se toimii kaikissa nettisivuissa myös ohjelmistokehysten kanssa [13]. Web-komponentit ovat kokoelma eri teknologioita, joiden tarkoitus on mahdollistaa uudelleenkäytettävien mukautettujen elementtien luonti, ja kapseloida niiden toiminnallisuus pois muusta koodista [14]. LitElement-kirjasto on käytössä niin Home Assistantin kuin sen sisällönkin kehittäjillä [15], joten tiedetään jo valmiiksi sen olevan yhteensopiva mukautettujen korttien kehitykseen.

LitElement toimii renderöimällä niin sanotun varjo-DOMin. Varjo-DOM viittaa selaimen kykyyn sisällyttää DOM-elementtien alipuu dokumentin renderöintiin, muttei päädokumentin DOM-puuhun [16]. Käytännössä tämä tarkoittaa kortin renderöintiä erillisenä näkymänä päänäkymän sisällä. Varjo-DOMin päivitys on tärkeä osa LitElementin elinkaarta.

LitElement-pohjaiset komponentit päivittyvät asynkronisesti vastauksena havaittuihin ominaisuusmuutoksiin [17]. Tätä hyödyntämällä voidaan reagoida kortin osien DOM-tapahtumiin dynaamisesti elinkaaren eri askeleissa. Korkealla tasolla elinkaari koostuu jonkun uuden ominaisuuden asettamisesta, siihen liittyvästä päivityspyynnöstä ja päivityksen käsittelystä renderöiden kortin sisällön näytettäväksi [17]. Elinkaaren askeleita hyödyntämällä voidaan esimerkiksi alustaa kortin elementtejä luotettavasti heti, kun LitElement saa renderöinnin valmiiksi.

3.2 Karttakirjasto

Kartan interaktiivista havainnollistamista varten tarvitaan siihen erikoistuva JavaScriptillä toimiva koodikirjasto. Vaihtoehtoihin sisältyy muun muassa OpenLayers, jonka avulla voidaan luoda dynaaminen kartta millä tahansa nettisivulla [27], sekä kevyempään karttojen piirtämiseen erikoistuvaa kirjastoa, kuten Leaflet. Leaflet on käytössä myös Home Assistantin sisäänrakennetussa karttakortissa [31], joten tiedetään, että sen toteutuksesta ei pitäisi aiheutua ongelmia. Kartan toteuttavaksi kirjastoksi valitaan siten Leaflet.

Leaflet on mobiiliystävällinen avoimen lähdekoodin kirjasto interaktiivisia karttatoteutuksia varten [18]. Sen avulla saadaan korttiin karttanäkymä, johon voidaan piirtää erilaisia merkkejä koordinaattien visualisointia varten. Piirtäminen toimii koordinaattipohjaisesti, eli sen avulla voidaan toteuttaa vastaanotettujen koordinaattien kautta kuljetun reitin

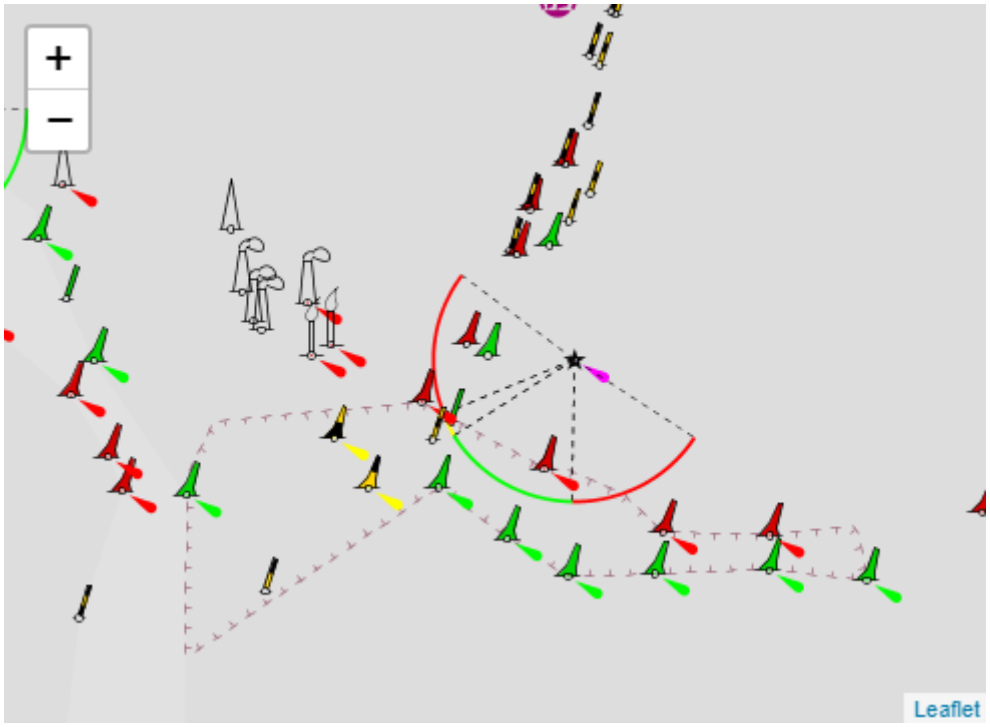
piirto karttaan. Sillä voidaan myös välittää koordinaatteja kartasta koodiin, joka auttaa tulevan reitin suunnittelua.

Leaflet vaatii toimiakseen kartan määritelmän eli karttapohjan. Karttapohja on kartta itsessään. Se on jonkin karttapalvelun tarjoama kartta, joka renderöidään Leafletin avulla. Karttapohjassa ei ole mitään erikoisuuksia, ja se toimii haettuna joltain palvelimelta Leafletin kautta.

Karttapohjia tarjoavia palveluja on olemassa erilaisia, kuten hyvin tunnettu Google Maps, mutta kaikki eivät sovellu saunalautan käyttöön. Jotkut karttapohjat eivät tarjoa vesillä navigoinnille hyötyä, joten on tarpeen valita siihen tarkoitukseen sopiva palvelu. Tähän tarkoitukseen soveltuu hyvin karttapalvelu nimeltä OpenStreetMap. OpenStreetMap on tuhansissa nettisivuissa, sovelluksissa ja laitteissa käytössä oleva yhteisön ajama projekti, joka on myös avoimen datan karttapalvelu [19]. Avoin data tarkoittaa tässä tapauksessa organisaation tuottamaa julkista tietoa, joka on avattu rakenteisessa muodossa vapaasti ja maksutta kaikkien hyödynnettäväksi [20]. OpenStreetMap itsessään ei auta vesillä navigointia, kuten kuvasta 2 näkee, mutta se tarjoaa lisäosia, jotka muuttavat tilanteen.

Kuva 2. OpenStreetMap-karttapohja.

Kartan lisäosat eli ylimääräiset kerrokset karttapohjan päällä mahdollistavat karttapohjan kanssa yhteensopivien kuvakkeiden käytön kartassa paremman visualisoinnin ja tiedon tarjonnan avuksi. Vesillä kulkemisen kannalta hyödyllinen karttakerros on tässä tapauksessa kuvassa 3 esitetty avoimen lähdekoodin ilmainen OpenSeaMap, jonka avulla näemme karttapohjan lisäksi merikarttaan liittyvät symbolit.



Kuva 3. Kartan lisäkerros OpenSeaMap.

4 NAVIGOINTIKOMPONENTIN RAKENTAMINEN

Jotta voidaan aloittaa mukautetun kortin kehitys, pitää ensin määrittää sen koodin sisältävän tiedoston polku Home Assistantin konfiguraatioon. Luodaan ensin Home Assistantin pääkansion alle uusi kansio nimellä *www*. Tätä polkua hyödyntäen toteutetaan kaikki mukautetut komponentit, jotka ajetaan paikallisina tiedostoina.

Kortti itsessään toimii JavaScript-tiedostona, joka luodaan edellä mainittuun hakemistoon. Kun tiedosto on olemassa, voidaan sen polku määrittää Home Assistantin konfiguraatioon *Lovelace*-osion alle *resources*-avainta hyödyntämällä. *Resources*-avaimen arvoihin lisätään *url*-avaimella local-hakemistosta alkava kortin polku. Localista alkavalla polulla haetaan paikalliset komponentit, sillä se yhdistyy *www*-kansioon. Tyypiksi määritetään *module*. Korttitiedoston polkuun lisätään myös parametrina tiedoston versio kehittämistä varten, ja sitä korotetaan aina muutosten yhteydessä. Ilman versionumeroa tiedoston sisältö ei välttämättä päivity aina Home Assistantin mukana, sillä vanha versio voi pysyä esimerkiksi selaimen välimuistissa.

Kun Home Assistant tunnistaa kortin olemassaolon, voidaan sitä käyttää samalla tavalla muiden korttien kanssa, eli lisäämällä se Lovelacen käyttöliittymän konfiguraatioon. Kortin avaimeksi *type* annetaan kortin mukautetun elementin luonnin yhteydessä määritetty nimi. Kun mukautettu luokka määritetään koodissa *define*-metodilla, siitä tulee mukautettu elementti, jota Home Assistant voi käyttää. Lisätään mukautettu kortti päänäkymään etuliitteellä *custom:*, jotta Home Assistant tunnistaa sen erilliseksi sisäänrakennetuista korteista.

Mukautetun kortin sisältämä mukautettu luokka luodaan LitElementin tarjoaman pohjaluokan jatkeeksi, jotta saadaan sen tarjoama toiminnallisuus käyttöön. Kun mukautettu luokka on määritelty, voidaan sen sisältöä alkaa renderöimään kortiksi Home Assistantin päänäkymään. LitElementin *render*-metodilla määritetään merkkijonona HTML-koodi, joka toimii kortin pohjana. Lisäksi *static get styles* -metodilla voidaan tyylitellä kortin HTML-sisältöä palauttamalla CSS-pohjainen merkkijono. CSS eli Cascading Style Sheets on nettisivujen muotoiluun käytetty ohjelmointikieli. Kortin sisällön kehitykseen vaadittavan pohjan valmistuessa voidaan alkaa rakentamaan navigaatiokomponentin alkeita eli karttanäkymää.

4.1 Leafletin alustus

Jotta voidaan renderöidä Leafletin tarjoama karttakomponentti, pitää sille ensin luoda paikka kortin HTML-elementissä. Lisätään elementtiin *div*-tagi, joka määrittää karttanäkymän paikan kortissa. Tagille pitää myös lisätä uniikki ID eli tunniste, jotta Leaflet osaa renderöidä kartan oikeaan sijaintiin kortin sisällä. Sille määritetään myös tyylissä jokin koko, jotta kartalla on tilaa renderöityä. Tälle riittää pelkkä *height*-määritys, ja koolla ei itsessään ole rajauksia. Kortin suuresta koosta aiheutuva mahdollinen ongelma on kuitenkin kortin renderöinnin epäonnistuminen esimerkiksi selainikkunan koon riittämättömyydestä tai muutoksesta. Tätä varten pitää lisätä kortin koon määritys Home Assistantin sisällä käyttämällä *getCardSize*-metodia mukautetussa luokassa. Sen avulla voidaan palauttaa kokoluokka, jonka perusteella Home Assistant luo tilaa kortille. Tämä tila on määrityksen jälkeen kortin käytettävissä esimerkiksi dynaamisen korttien paikoituksen yhteydessä. Tämä estää myös kortin hajoamisen, mikäli siltä loppuu selainikkunasta tila kesken.

Koska HTML-elementin pitää olla olemassa kartan määrityksen yhteydessä, pitää varmistaa, että se renderöityy ensin. LitElementin elinkaaressa renderöinti tapahtuu jo ennen *firstUpdated*-askelta, jota kutsutaan vain ensimmäisen varjo-DOMin päivityskutsun yhteydessä [17]. Edellä mainitun askeleen nimen jakavaa metodia hyödyntäen voidaan parhaiten alustaa Leaflet, sillä se ajetaan vain kerran ja lähes heti HTML-elementtien lataamisen jälkeen.

Leafletin käyttöönottoa varten pitää ensin määrittää karttaobjekti, jonka päälle voidaan ladata kartan sisältö eli näyttää kartta kortin sisällä. Karttaobjektin pitää myös olla muodossa, josta siihen pääsee käsiksi ympäri koodia. Tätä varten lisätään ensin *firstUpdated*-metodin sisälle kutsu Leafletin Map-luokkaan. *L.map*-kutsun avulla luotu objekti tallennetaan *this*-avainsanalla luokan kontekstin muuttujaksi, jotta sitä voidaan käyttää helpommin luokan sisällä. Parametriksi vaaditaan aikaisemmin karttaa varten luodun *div*-tagin tunniste, jotta kartta on mahdollista sijoittaa sen sisälle. Normaalisti määritys toimisi yksinkertaisesti syöttämällä tunniste merkkijonomuodossa, mutta tässä tapauksessa Leaflet yrittää hakea sitä globaalilla tasolla pää-DOMista, mikä ei toimi kapseloidun varjo-DOMin ympäristössä. *L.map* mahdollistaa kuitenkin syötön myös HTLMElement-muodossa, mikä on toimivampi ratkaisu.

Div-tagin saaminen HTLMElement-muodossa onnistuu *querySelector*-metodia hyödyntämällä, sillä sen avulla voidaan hakea HTLMElement-muotoisia elementtejä DOMin sisältä. Tämä web-ohjelmointirajapinnan tarjoama metodi on normaalisti käytössä dokumentin tasolla pää-DOMissa, mutta sitä voidaan käyttää myös kapseloidun varjo-DOMin sisällä shadowRoot-käyttöliittymän avulla. ShadowRoot on yksinkertaisesti varjo-DOMin sisäisen DOM-puun juuri, joka renderöidään erikseen dokumentin pää-DOMista [21]. Lisäksi pitää käyttää *this*-avainsanaa, jotta saadaan haettua elementti oikean luokan kontekstista. Yhdessä näistä saadaan lause *this.shadowRoot.querySelector('#mapid')*, joka palauttaa *mapid*-tunnisteella määritetyn *div*-tagin HTMLElement-muodossa.

Nyt on olemassa pohja Leafletin kartalle, mutta siinä ei kuitenkaan vielä näy mitään. Tämä johtuu siitä, ettei kartalle ole määritetty laattakerrosta (engl. tile layer). Laattakerros on joukko web-saatavia laattoja palvelimella, mitä haetaan suorilla web-osoitepyynnöillä selaimesta [22]. Laattakerrokset määritetään koodiesimerkin 1 mukaisesti Leafletin alustuksen yhteydessä *firstUpdated*-metodin sisällä. Leaflet ottaa vastaan palvelimen osoitteen *TileLayer*-objektiin, ja täyttää karttalaatan sijainnin ja koon määrittävät muuttujat dynaamisesti karttanäkymän keskipisteen ja zoomauksen perusteella. Sen jälkeen Leaflet lataa palvelimelta oikeat karttalaatat karttanäkymän täytteeksi kuvan 4 mukaisesti. Kuvassa 5 näkyy esimerkki ladattavasta karttalaatasta.

Karttanäkymän koordinaatit alustetaan saunalautan yleiseen sijaintiin, kun se ei ole vesillä, jotta voidaan hahmottaa karttaa paremmin ennen koordinaattien vastaanottoa. Karttaobjektin metodia *setView* käyttämällä saadaan parametrinä asetettua koordinaatit, jotka siirtävät kartan näkymän keskittyen määritettyyn pisteeseen.

```
// OpenStreetMap
var map_base = 'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png';
map.addLayer(L.tileLayer(map_base, {
  attribution: '© OpenStreetMap contributors'
}));
// OpenSeaMap
var map_overlay =
  'http://tiles.openseamap.org/seamark/{z}/{x}/{y}.png';
map.addLayer(L.tileLayer(map_overlay, {}));
```

Koodiesimerkki 1. Laattakerrosten määrittäminen.

Kuva 4. Karttanäkymä laattakerrosten määrityksen jälkeen.

Kuva 5. Esimerkki ladattavasta karttalaatasta [23].

4.2 Koordinaattien käsittely

Leafletin alustuksen jälkeen aletaan määrittämään kartan toiminnallisuutta. Jotta voidaan näyttää reitti kartalla, pitää ensin vastaanottaa lautan koordinaatit. Koordinaatit saapuvat MQTT-paketin välityksellä JSON-muodossa sisältäen lautan sen hetkisen sijainnin pituus- ja leveyspiirin. JSON eli JavaScript Object Notation on datan tallennukseen ja kuljetukseen käytetty formaatti. Paketin vastaanottoa varten hyödynnetään Home Assistantin automaatiota, ja rakennetaan MQTT-aiheeseen reagoiva toiminnallisuus.

4.2.1 Koordinaattien vastaanotto

Kun MQTT:n välityksellä saapuu viesti haluttuun aiheeseen, halutaan tallentaa viestin sisältö käytettäväksi mukautetun kortin koodiin. Luodaan automaatio, joka reagoi Home Assistantiin määritetyn MQTT-integraation avulla koordinaattidatan sisältävän viestikanavan aiheeseen saapuvaan viestiin. Kun automaatio laukeaa, sen pitää suorittaa toimenpide, joka tallentaa koordinaatit koodissa käytettävään sijaintiin. Tämä sijainti on device tracker, jota voidaan käsitellä mukautetun kortin koodin sisältä. Koska device tracker luodaan automaation yhteydessä, sitä ei tarvitse määrittää Home Assistantin konfiguraatiossa erikseen. Konfiguraatioon pitää kuitenkin lisätä *device_tracker*-avain, jotta Home Assistant osaa lukea uusia laiteseurannan muuttujia. Device tracker itsessään lisätään entiteettinä kortin määritelmään Lovelacen konfiguraatiossa *entity*-avaimella.

Automaatioon reagoivaksi tapahtumaksi määritetään palvelu *device_tracker.see*, joka päivittää halutun device trackerin sisällön automaation lauettua. Palvelun attribuutiksi määritetään *data_template*, jonka sisälle lisätään korttiin määritetyn device trackerin nimi *dev_id*-avaimella, sekä vastaanotettu koordinaattidata *gps*-avaimella. Koska MQTT-viestin sisältö ei kuitenkaan koostu pelkästään koordinaattidatasta ja sisältää myös muun muassa viestin otsikkodataa, pitää haluttu sisältö hakea viestikuorman sisältä. Tätä varten hyödynnetään automaation *trigger*-muuttujia.

Home Assistantin automaatio tarjoaa MQTT-viestiin reagoinnin mahdollisuuden lisäksi myös pääsyn viestin sisältöön erilaisilla muuttujilla. Koska viestin sisältö saapuu JSON-paketissa, käytetään muuttujaa *trigger.payload_json* koodiesimerkin 2 mukaisesti. JSON-kentästä *latitude* saadaan koordinaattien leveysaste, ja kentästä *longitude*

saadaan koordinaattien pituusaste. Automaation ansiosta MQTT-viestin saapuessa saadaan nyt tallennettua koordinaatit device trackeriin helppopääsyisempään muotoon.

```
gps:
  - '{{trigger.payload_json.latitude}}'
  - '{{trigger.payload_json.longitude}}'
```

Koodiesimerkki 2. Koordinaattidatan vastaanotto MQTT-paketista.

Mukautetun kortin koodissa käytetään metodia *set hass(hass)*, joka suoriutuu aina, kun Home Assistant eli tässä tapauksessa *hass*-objekti päivittyy. Päivitys tapahtuu muun muassa silloin, kun uusi koordinaattidata ajetaan aiemmin luodussa automaatiossa device trackeriin, joten tämän metodin avulla on mahdollista suorittaa koodia aina uuden MQTT-viestin saapuessa. Home Assistant voi kuitenkin päivittyä muidenkin komponenttien toimesta, joten lisätään myös metodin sisälle vain uuden koordinaattidatan päivitykseen reagoiva tarkistus. Koodiesimerkin 3 mukaisesti saadaan haettua lautan koordinaatit device trackerista sen sisällön päivittyessä. Ensin haetaan kortin Lovelace-konfiguraatioon määritelty entiteetti, josta tallennetaan sen tila. Tilan avulla päästään käsiksi attribuutteihin, jotka sisältävät päivityksen yhteydessä kerätyt koordinaatit.

```
var state = hass.states[this.config.entity];
// Leveysaste
var raft_lat = state.attributes.latitude;
// Pituusaste
var raft_lon = state.attributes.longitude;
```

Koodiesimerkki 3. Lautan koordinaattien haku kortin koodista.

4.2.2 Koordinaattihistorian tallennus

Kun MQTT-viestillä saapuneet koordinaatit ovat tallessa, niitä voidaan tallentaa esimerkiksi taulukkomuotoiseen muuttujaan. Taulukkomuuttujan avulla voidaan pitää kuljettu

reitti session muistissa Leafletin luettavaksi reitin piirtämiseen. Historian luku onnistuu taulukon kautta, mutta vain nykyisen session sisällä. Muuttujien arvot säilyvät session läpi, mutta jos sessio katkeaa niin muuttujien arvot nollaantuvat.

Mikäli Home Assistantin näkymän päivittää esimerkiksi lataamalla verkkosivun uudestaan, istunnon muuttujat katoavat, joten koordinaattien säilytystä varten pitää saada ne talletettua pysyvämmin. Tämä toteutetaan määrittämällä Home Assistantin konfiguraatioon *recorder*-avaimella koordinaattien seuraukseen luotu device tracker -entiteetti recorder-integraation seurantaan. Määrityksen jälkeen Home Assistant alkaa keräämään device trackerin historiallista dataa, jonka sisältä löytyy myös kerätyt koordinaatit.

Jotta päästään käsiksi tietokantaan, pitää joko tehdä sinne suoria kyselyjä, tai hyödyntää Home Assistantin RESTful ohjelmointirajapinnan toteutusta. REST eli Representational State Transfer on ohjelmointirajapintojen toteuttamiseen tarkoitettu arkkitehtuurimalli [28], ja RESTful tarkoittaa REST-mallia toteuttavaa ohjelmointirajapintaa. Ohjelmointirajapintaa käyttämällä voidaan hakea device trackerin historia suhteellisen helposti, joten hyödynnetään sitä.

Ohjelmointirajapinnan dokumentaatiota tutkimalla selviää, että tiedon haku tietokannasta onnistuu HTTP-pyynnön välityksellä GET-metodilla Home Assistantin palvelimen polusta */api/history/period/<timestamp>* [6]. HTTP eli Hyper Text Transfer Protocol on verkon välityksellä tapahtuvaan kommunikaatioon käytettävä protokolla. Pyyntöön lisätään myös entiteetin suodatus parametrilla *filter_entity_id*. Lautan yksittäisten matkojen seurantaan riittää oletusarvoinen päivän historian haku, joten *<timestamp>* voidaan jättää tyhjäksi. Tästä seuraa ohjelmointirajapinnan polku */api/history/period?filter_entity_id=device_tracker.raft_pos*. Kyselyn suoritus koodista onnistuu sisäänrakennetulla Fetch-ohjelmointirajapinnalla. *Fetch*-metodin avulla on mahdollista luoda HTTP-pyyntöjä tarvittavilla parametreilla, jotka ovat pyynnön osoite ja pyynnön omat parametrit, kuten viestin otsikkodata JSON-formaatissa. Pyynnön jälkeen *fetch* palauttaa lupauksen vastaukseen [7], eli se palauttaa palvelimen vastauksen heti, kun se on mahdollista. Tämän vastauksen sisältö on recorder-integraatiosta haetun entiteetin historia.

Jotta voidaan lukea historia aina, kun sitä tarvitaan, toteutetaan historian säilytys kortissa *firstUpdated*-metodia hyödyntämällä. Lisäämällä historian haku *firstUpdated*-metodin sisään se suoritetaan aina kortin ensimmäisen latauksen yhteydessä koodiesimerkin 4 mukaisesti. *RequestURL* sisältää polun HTTP-pyynnön osoitteeseen, ja *requestOptions* sisältää viestin otsikkodatan, johon kuuluu Home Assistantin profiilisivulta luotava

pitkäikäinen käyttöoikeustunnus, sekä viestisisällön tyyppi *application/json*. Nämä lähetetään merkkijonomuotoisessa JSON-formaatissa. Käyttöoikeustunnus on vaadittu, jotta Home Assistant voi todentaa pääsyn ohjelmointirajapintaan hyväksytystä lähteestä. Historian käyttöä varten kerätään *fetch*-metodin vastaus *json*-metodin avulla JSON-muotoon jäsennettynä, jonka jälkeen käsitellään device trackerin sisältämä koordinaattidata siistittyyn muotoon ilman mahdollisia duplikaatteja *getHistory*-funktion sisällä. Sen jälkeen lisätään koordinaatit session taulukkomuuttujaan samassa funktiossa. Tämä helpottaa koordinaattihistorian käyttöä ja uusien koordinaattien lisäystä ilman ylimääräisiä ohjelmointirajapintakutsuja. Koordinaattidatan käsittelyfunktion kutsussa käytetään *bind*-metodia, jolla lukitaan funktion konteksti luokkaan. Tämän avulla päästään *this*-avainsanaa käyttämällä käsiksi luokan sisältöön, kuten taulukkomuuttujaan, joka on käytössä eri metodeissa luokan sisällä.

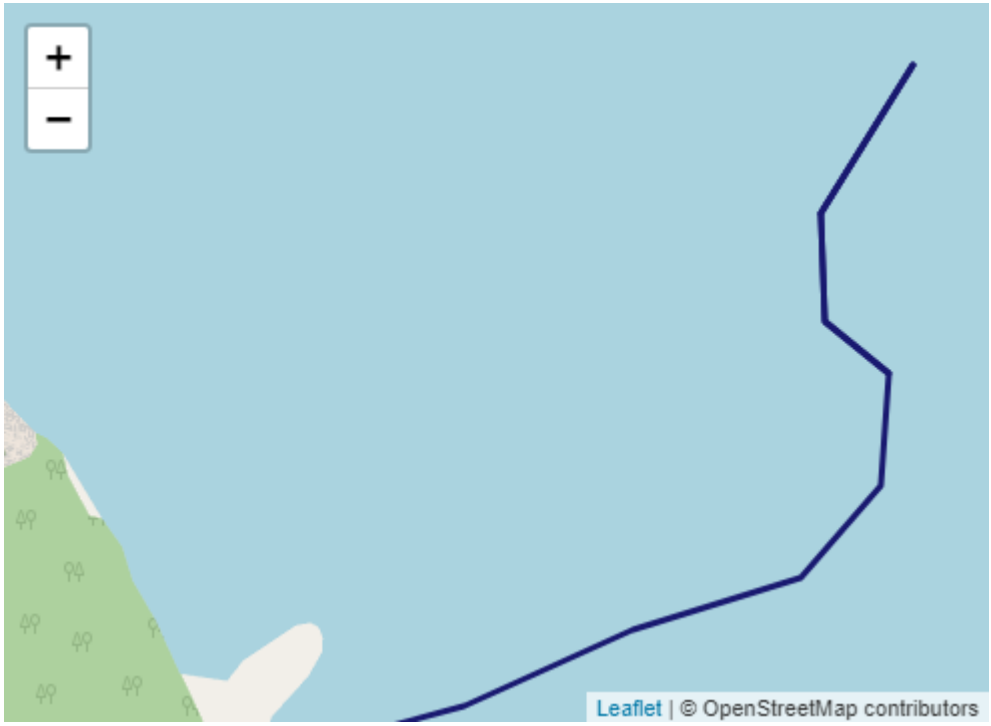
```
fetch(requestURL, requestOptions)
  .then(res => res.json())
  .then(data => {
    // Bind-ominaisuuden käyttö lukitsee oikean kontekstin.
    data[0].forEach(this.getHistory.bind(this));
  })
  .catch(error => console.log(error));
```

Koodiesimerkki 4. HTTP-pyyntö Home Assistantin ohjelmointirajapintaan *fetch*-metodilla, ja historian lähetys käsittelyfunktioon.

4.3 Koordinaattihistorian piirtäminen karttaan

Leaflet sisältää monenlaisia työkaluja kartan päälle piirtämiseen, kuten yksinkertaisia muotoja ja merkkejä, mutta reitin visualisointiin sopivin on suora viiva luokasta Polyline, joka luodaan *L.polyline*-kutsulla. Se mahdollistaa viivan piirtämisen kartan pisteiden välillä koordinaattitaulukon perusteella. Taulukko on jo olemassa, joten välitetään se eteenpäin reitin piirtoa varten *firstUpdated*-metodin sisällä. Mikäli sessio nollautuu, tietokannasta haetut koordinaatit piirretään kortin ladattua karttaan. Koordinaatit välitetään

L.polyline-kutsuun kaksiulotteisena koordinaattitaulukkona, jossa jokainen arvo on leveys- ja pituusasteen yhdistelmä. Jotta piirretty reitti pysyy ajan tasalla, lähetetään uudet koordinaatit Polyline-luokalle reaktiona uusiin koordinaatteihin *set hass(hass)* -metodin avulla niiden vastaanoton yhteydessä. Koordinaattien välittämisen seurauksena kuljetut koordinaatit näkyvät nyt kartalla pisteiden välisenä monikulmaisena viivana, kuten kuvassa 6.



Kuva 6. Reitin piirtäminen koordinaattien välityksellä.

5 REITTISUUNNITTELUN TOTEUTUS

Lautan tulevan reitin suunnittelua varten pitää ensin parantaa vuorovaikutuksen tasoa kartassa. Oletuksena kartan toiminnallisuus rajoittuu kartan liikuttamiseen ja zoomauksen tason muuttamiseen. Leaflet tarjoaa kuitenkin toiminnallisuutta myös kartan päällä tapahtuvien tapahtumien hallintaan. Esimerkiksi karttaa klikatessa rekisteröidään *click*-tapahtuma, johon voidaan lisätä tapahtumankuuntelija *on*-metodin avulla. Sillä saadaan määritettyä klikkausta käsittelevä funktio. Leafletin alustuksen yhteyteen lisätään määritys funktiolle, joka suoriutuu aina, kun karttaa klikataan. Käytetään myös *bind*-toiminnallisuutta, jotta voidaan lukita suorituskonteksti kortin luokkaan.

Koodiesimerkki 5:n klikkaustapahtumaan reagoiva funktio *onMapClick* laukeaa aina kartan klikkauksen yhteydessä vastaanottaen argumentin *e*, joka sisältää muun muassa klikkauksen sijaintiin liittyvää dataa. Tärkein argumentista vastaanotetun datan osa on *e.latlng*, jonka sisältö on kartasta klikatun pisteen koordinaatit. Kun koordinaatit on vastaanotettu, ne voidaan tallentaa esimerkiksi taulukkomuuttujaan lähetettäväksi suunnitelmana eteenpäin lautalle, jossa ne käsitellään suunnan määrittelyä varten. Jotta saadaan luotua moniosainen suunnitelma, pitää ensin rakentaa käyttöliittymä reitin suunnittelua varten.

```
this.map.on('click', this.onMapClick.bind(this));
```

Koodiesimerkki 5. Kartan klikkaustapahtuman *e* välitys funktiolle *onMapClick(e)*.

5.1 Reitin suunnittelun käyttöliittymä

Käyttöliittymässä pitää pystyä valitsemaan monta pistettä kokonaista suunnitelmaa varten, joten luodaan ominaisuus, jonka avulla se on mahdollista. Pisteet tallennetaan taulukkomuuttujaan suunnitelmaksi, minkä avulla voidaan lähettää koordinaatit oikeassa järjestyksessä eteenpäin. Jotta pisteiden järjestys on selvää myös karttanäkymässä, käytetään merkkejä, joissa järjestys on selkeästi näkyvillä.

Leafletin karttaan voidaan merkata sisällöllisiä pisteitä käyttäen Popup-luokkaa, joka luo ponnahdusikkunamaisia kuplia haluttuihin kohtiin. Kuplan sisältö voi olla esimerkiksi merkkijono. Tätä luokkaa hyödyntämällä voidaan kirjoittaa valittuihin pisteisiin järjestysnumero, joka kertoo pisteen järjestyksen suunnitelmassa. *L.popup*-kutsulla saadaan luotua uusi kupla *onMapClick*-funktion sisällä kartan klikkauksen yhteydessä. Samalla saadaan lisättyä kuplan sisällöksi järjestysnumero, joka lasketaan suunnitelmataulukon pituuden perusteella. Suunnitelmataulukkoon lisätään luotu *popup*-olio, jonka kautta sen sisältöä ja järjestystä voidaan käsitellä, sekä hakea kuplan sijainnin koordinaatit. Lisätään myös funktioon vahingollisia tuplaklikkauksia estävä duplikaattikoordinaattien esto.

Kun kartasta valitaan piste, on mahdollista, että esimerkiksi kosketusnäytön herkän syötön takia sama piste luodaan useamman kerran. Tämä vaikeuttaa järjestyksen käsittelyä koodissa sekä lukemista kartalla. Saman pisteen valitseminen tarkoituksella esimerkiksi paluumatkalla on epätodennäköistä koordinaattien suuren tarkkuuden takia, joten on hyödyllistä lisätä duplikaattien esto. Tarkistetaan siis ettei suunnitelmataulukosta löydy jo valmiiksi uuden kuplan koordinaatit luonnin yhteydessä.

Kuplan sijainti kartalla määritetään *onMapClick*-funktion argumenteista saatujen koordinaattien perusteella *e.latlng*-arvosta. Jotta kupla ei voi vahingossa kadota kartalta, annetaan vielä sen lisämääritykseksi arvot *autoClose:false* ja *closeOnClick:false*. Lisämääritykset kertovat kuplalle, että se ei saa sulkeutua, kun toinen kupla aukeaa tai kuplaa klikattaessa. Suunnittelun yhteydessä pitää kuitenkin pystyä poistamaan käsin valittuja pisteitä kartasta, jota varten hyödynnetään vähemmän vahinkoaltista kuplasta oletuksena löytyvää rastia.

Rastia painamalla kupla katoaa kartasta, mutta mikäli poistaa pisteen, joka ei ole suunnitelman viimeinen, kuplien esittämään järjestykseen jää yhden järjestysnumeron tyhjä väli. Kupla jää myös suunnitelmataulukon sisään vaikkei se ole kartassa näkyvillä. Jotta haluttu piste saadaan poistettua oikein, pitää lisätä kuplan sulkeutumisen yhteydessä tapahtuvaa *remove*-tapahtumaa seuraava kuuntelija. Tapahtuman käsittelyä varten määritetään uusi funktio samoin kuin koodiesimerkissä 5 määritettiin *click*-tapahtumalle käsittelijäfunktio.

Remove-tapahtumaan vastaavan funktion sisälle määritetään kupla poistettavaksi suunnitelmataulukosta, kunhan se omaa samat koordinaatit kuin tapahtuman kohde. Tämä toteutus hyötyy myös siitä, ettei duplikaattikoordinaatteja ole olemassa suunnitelmataulukossa. Kuplan poistoa varten käytetään taulukon *splice*-metodia. Sen avulla saadaan

poistettua kupla jättämättä tyhjää väliä taulukkoon. Tämän jälkeen päivitetään kuplien sisällöt suunnitelmataulukon järjestyksen mukaisiksi, jotta voidaan välttää järjestysnumeroihin syntynyt tyhjä väli.

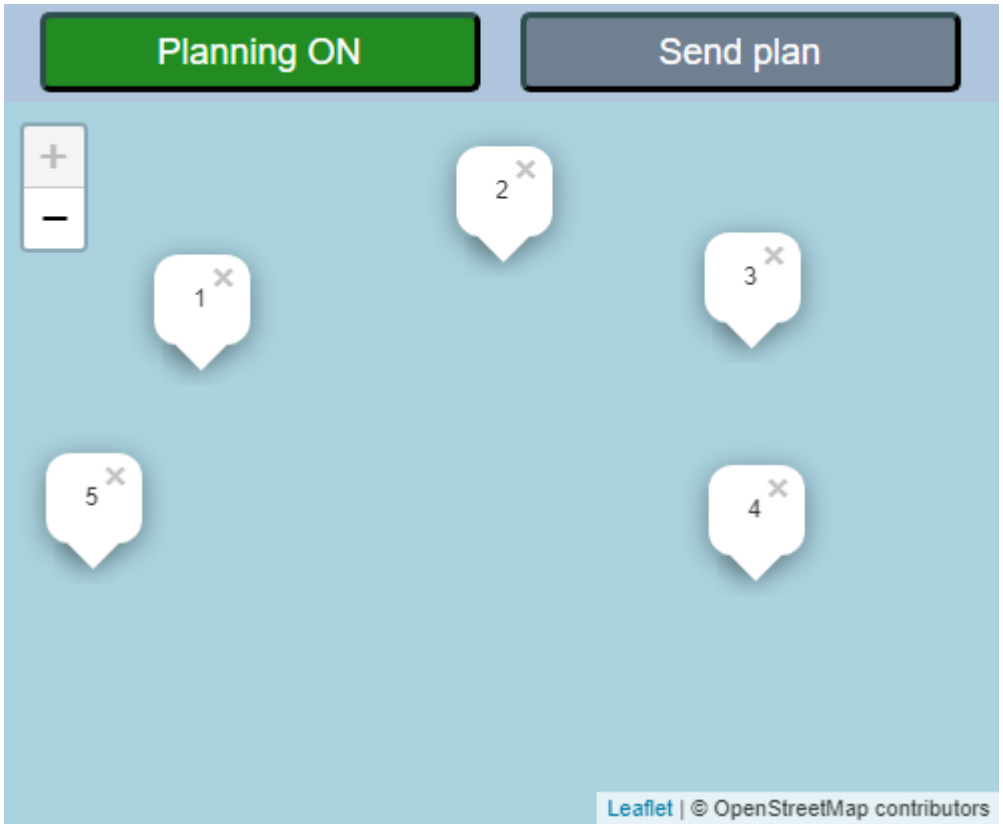
Suunnittelun ulkopuolisten kuplien luonnin välttämiseksi luodaan korttiin nappi, jonka avulla voidaan vaihtaa suunnittelutila päälle tai pois päältä. Kun suunnittelutila on pois päältä, kartan klikkaus ei suorita kuplan luomisen koodia. Lisätään renderöinnin yhteydessä luotavaan HTML-elementtiin nappi, jonka painallukseen yhdistetään uusi toiminnallisuus lisäämällä sille määritelmä *@click="${this.planModeToggle}"*. Kun nappia painetaan, ajetaan funktio, joka vaihtaa suunnittelun tilaa. Määritetään napin painallusta varten totuusarvomuuttuja, jonka arvoksi tulee *true* mikäli suunnittelutila on päällä ja muuten *false*. Suunnittelutila on oletuksena pois päältä, joten alustetaan muuttuja arvolla *false*. Napin painalluksen yhteydessä vaihdetaan totuusarvomuuttujan arvo, jotta voidaan seurata nykyistä tilaa. Lisätään *onMapClick*-funktioon suunnittelutilan tarkistus, jottei kartta reagoi klikkauksiin sen ollessa pois päältä.

Käyttäjälle selvennykseksi määritetään dynaamisesti napin sisältö painalluksen yhteydessä totuusarvomuuttujaa hyödyntäen. Jos suunnittelutila on päällä, napissa lukee ”ON” ja tausta on vihreä. Suunnittelutilan ollessa pois päältä napissa lukee ”OFF” ja taustaväri muuttuu harmaaksi. Napin sisällön muuttamista varten haetaan napin elementti ensin varjo-DOMista *getElementById*-metodilla sen tunnisteen avulla. Sen jälkeen päästään käsiksi sen sisältöön ja tyyliin kentillä kuten *textContent* ja *style.background*.

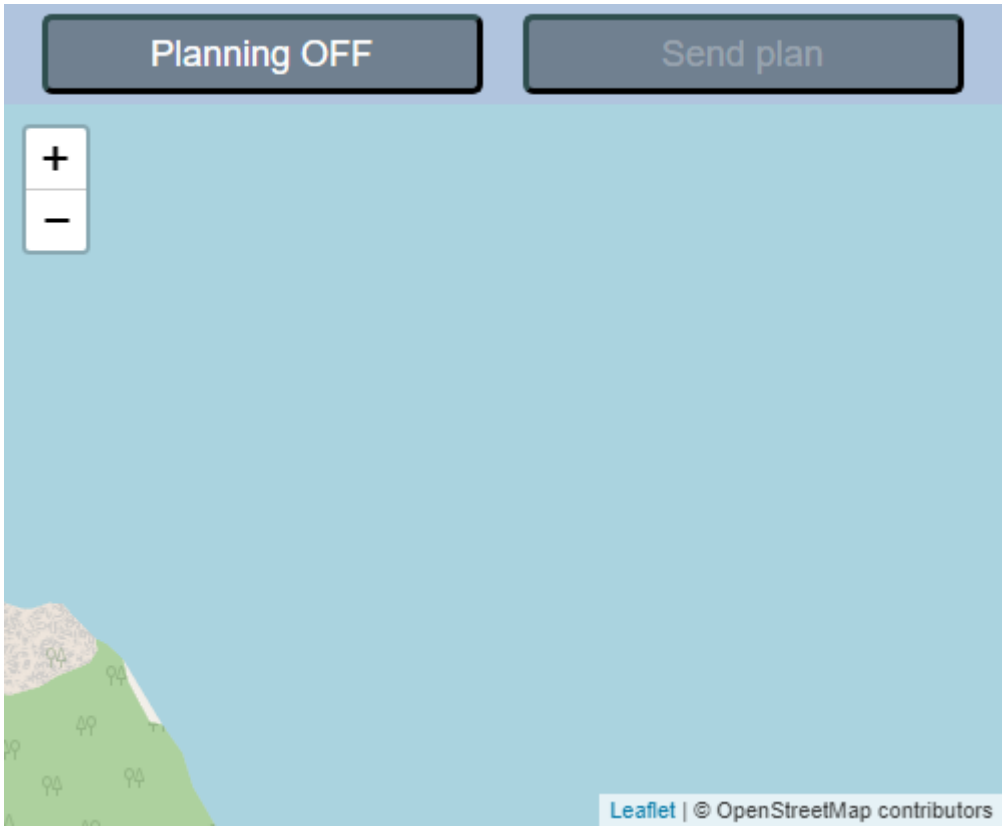
Kun suunnittelutila suljetaan tämän napin kautta, lopetetaan koko suunnitteluprosessi eli poistetaan kaikki suunnitellut pisteet kartasta. Lisätään suunnittelutilan sulkemisen yhteyteen toiminta tyhjentää olemassa oleva suunnitelmataulukko, ja poistaa kaikki kuplat karttanäkymästä *remove*-metodin avulla. Tätä varten pitää ensin poistaa kuplien *remove*-tapahtumaa seuraava toiminnallisuus, sillä muuten kuplan sulkiessa aikaisemmin määritetty poistofunktio ajaa *splice*-metodin suunnitelmataulukkoon. Tämän seurauksena kuplien poistosilmukka ei pääse enää käsiksi jokaiseen taulukon indeksiin. Lisätään siis ennen kuplien sulkemista kutsu niiden *off*-metodiin, joka poistaa niistä tapahtumien kuuntelijat.

Kun suunnitelma on valmis, se pitää lähettää eteenpäin lautalle. Suunnitelmataulukon koordinaattien lähetystä varten luodaan ensin nappi, joka mahdollistaa lähetyksen. Lisätään HTML-elementtiin uusi nappi, ja sille *@click*-määritelmä omaan funktioonsa. Mikäli suunnittelutila ei ole päällä, tällä napilla ei ole tarkoitusta, joten lisätään sen määritykseen

oletuksena *disabled*-arvo. Tämä poistaa sen käytöstä, kunnes määritetään toisin. Kun suunnittelutila avataan, määritetään tämä nappi taas käytettäväksi, kuten kuvassa 7 näkyy. Haetaan napin elementti *getElementById*-metodilla ja määritetään *disabled*-kentän arvoksi *false.* Suunnittelutilan sulkemisen yhteydessä nappi poistetaan uudelleen käytöstä kuvan 8 mukaisesti antamalla *disabled*-kentälle arvo *true*.



Kuva 7. Suunnittelutila päällä.



Kuva 8. Suunnittelutila pois päältä.

5.2 Koordinaattien lähetys

Kun suunnitelma halutaan lähettää eteenpäin painamalla lähetysnappia, ajetaan funktio, joka suorittaa valittujen pisteiden koordinaatit sisältävän MQTT-viestin lähetyksen. Luodaan ensin uusi taulukko pelkkiä koordinaatteja varten. Aikaisempaa suunnitelmataulukkoa hyödyntäen saadaan kuplaolioiden *_latlng*-arvoista pisteiden leveys- ja pituusasteet. Näistä jokainen pari lisätään omana taulukkomuotoisena elementtinä lähetettävään taulukkoon, josta saadaan kaksiulotteinen taulukko. Kaksiulotteisella taulukolla saadaan pidettyä koordinaatit ja niiden suhde toisiinsa tallessa lähetyksen yli. Koordinaattitaulukkoa käyttäen voidaan lähettää viesti MQTT:n välityksellä lautalle. Lähetystä varten käytetään uudestaan aikaisempaa *fetch*-metodia, jolla tällä kertaa luodaan POST-pyyntö. POST-pyyntö on GET-pyynnön vastakohta, ja sillä lähetetään dataa sen pyynnön sijasta.

Fetch-metodin parametreihin lisätään tällä kertaa polku Home Assistantin MQTT-integraation julkaisumetodiin. Sen avulla voidaan lähettää MQTT:n välityksellä viesti, joka

sisältää halutunlaista dataa. MQTT:n julkaisumetodi löytyy Home Assistantin palvelimelta polusta */api/services/mqtt/publish* [6]. Luodaan pyyntö samalla tavalla kuin aikaisemmassa GET-pyynnössäkin lisäten siihen otsikkodata, joka sisältää käyttöoikeustunnuksen ja viestisisällön tyypin *application/json*. Lisätään otsikkodataan myös *Accept*-avaimella sama arvo *application/json*, jotta pyynnön muoto välittyy oikein. Otsikkodatan lisäksi määritetään *Method*-avaimella pyynnön muodoksi *POST*, jotta *fetch*-metodi suoriutuu POST-pyyntönä. MQTT-viestin sisältö määritetään *body*-avaimeen JSON-oliona, joka sisältää julkaistavan MQTT-aiheen *topic* ja viestikuorman *payload*. Jotta viestikuorma eli koordinaattitaulukko kelpaa lähetettäväksi, se pitää vielä muuttaa merkkijonomuotoon ajamalla se metodin *JSON.stringify* läpi.

Kun viesti on lähetetty, suljetaan suunnittelutila, mutta jätetään merkit kartalle. Merkkien avulla saadaan seurattua lautan kulkemaa reittiä suunnitelmaan verrattuna toisin kuin jos merkit poistettaisiin. Mikäli halutaan muokata reittiä, voidaan avata suunnittelutila uudelleen. Tämä mahdollistaa valittujen pisteiden muokkauksen halutunlaisiksi, ja uuden suunnitelman lähetyksen lautalle.

6 POHDINTA

Lopputuloksena saatu navigointikomponentti toimii halutunlaisesti ja osaa kommunikoida molempiin suuntiin lautan kanssa. Alussa asetettu tavoite kortin toiminnasta saunalautan navigaation avustamiseen toteutui niin kuljetun reitin visualisoinnin kuin reitin suunnittelunkin kannalta. Kortin sisältö jäi suhteellisen yksinkertaiseksi, mutta myös helppokäyttöiseksi. Koodin ajosta ei myöskään ole käytössä aiheutunut ongelmia, muttei se ole jatkossa täysin mahdotontakaan.

Kortti ja sitä ajava koodi reagoi halutunlaisesti MQTT-viestikanavan kautta vastaanotettuun JSON-pakettiin, jonka sisällöstä haetaan lautan koordinaatit. Viestin vastaanottoosion suoritus on suhteellisen turvassa ongelmilta. Vääränmuotoisen paketin vastaanotto on epätodennäköistä, sillä paketti muodostetaan automaattisesti aina samaan muotoon toisessa lautan komponentissa. Paketin sisällön tarkistukset esimerkiksi GPS-laitteen lukemien oikeudesta kuuluvat myös siihen komponenttiin. Suunnitelman lähetyksen paketti toisaalta kuuluu navigointikomponentin vastuisiin.

Suunnitelmataulukon muodostuksen yhteydessä ei myöskään yleisesti pitäisi syntyä ongelmia, sillä koordinaatit tulevat suoraan Leafletin kanssa luoduista olioista. Mikäli Leafletin kanssa olisi ongelma, se näkyisi todennäköisesti jo aikaisemmassa vaiheessa, kuten käynnistyksen yhteydessä kartan sisällön lataamisen epäonnistumisena. Jos näissä toiminnoissa esiintyisi ongelmia, ne johtuisivat todennäköisesti MQTT-viestinnän epäonnistumisesta.

Viestinnän epäonnistuminen on epätodennäköistä sen asynkronisuuden ansiosta, mutta mikäli siinä esiintyisi ongelmia, se voisi aiheuttaa esimerkiksi reitin piirrossa niin sanottuja oikoreittejä eli joidenkin pisteiden ohitusta viestien vastaanoton epäonnistumisen yhteydessä. Tässä tapauksessa haitan suuruus olisi suhteessa menetettyjen viestien määrään, muttei todennäköisesti olisi normaalissa käytössä huomattavaa. Navigointikomponentti pystyy käsittelemään suuria määriä pisteitä kartalla, ja GPS-laitteelta niitä voidaan vastaanottaa myös hyvin tiheään tahtiin. Kunhan viestinnässä ei ole ajoittaisia katkoksia suurempia häiriöitä, navigointikomponentti pystyy vielä suorittamaan tehtävänsä. Sama on totta myös suunnitelman lähetyksen yhteydessä.

Suunnitelmataulukko lähetetään yhtenä pakettina, joten sen epäonnistuminen ei aiheuttaisi suurta vahinkoa, koska sen voi lähettää uudestaan. Suurin ongelma syntyisi siitä,

jos viestintä pysähtyisi kokonaan. Siinä tapauksessa ongelma ei liittyisi kuitenkaan enää pelkästään navigointikomponenttiin, vaan jonkinlaiseen häiriöön verkkoyhteydessä tai MQTT-integraatiossa, mikä pitää korjata ulkoisesti.

7 LOPUKSI

Opinnäytetyössä rakennettiin saunalautan navigointikomponentti toimimaan osana tietokokonaisuutta Home Assistantin avulla. Tämän komponentin eli Home Assistantin päänäkymän kortin piti pystyä visualisoimaan lautan kulkemaa reittiä kartalla, sekä mahdollistaa tulevan reitin suunnittelu samalla kartalla. Karttanäkymä toteutettiin Leaflet-kirjaston avulla hyödyntäen OpenStreetMapin ja OpenSeaMapin yhdistelmää. Näistä syntynyt karttapohja tarjosi informatiivisen tavan seurata ja suunnitella lautan reittiä kartalla. Sen lisäksi luotiin suunnitelman hallintaa varten kortin käyttöliittymään yksinkertaiset napit suunnittelutilan vaihtoon, sekä suunnitelman lähettämiseen lautalle. Viestintä rakennettiin MQTT-viestintäprotokollaa hyödyntäen, ja lopputuloksena saatiin suhteellisen kevyt ja mobiiliystävällinen toteutus toimimaan Raspberry Pin Home Assistant -asetelmassa.

Navigointikomponentin valmistuminen oli merkityksellistä, sillä samoilla toiminnallisuuksilla varustettua Home Assistantin korttia ei ole aiemmin julkaistu. Uuden komponentin rakentamisen yhteydessä selvisi myös paremmin täysin mukautettujen komponenttien mahdollisuus saunalautan tulevaisuutta varten. Prosessin ymmärrys on suuri etu saunalautan jatkokehityksen kannalta, mutta myös navigointikomponenttia itsessään on mahdollista parantaa.

Home Assistantin yllä toimiva kortti saunalautan navigointia varten vastaa tarpeita, muttei ole kuitenkaan täydellinen. Osana kokonaisuutta siihen voi saunalautan kehittyessä tulla jatkuvasti lisää vaatimuksia, joita pitää toteuttaa ja testata. On myös mahdollista, että tulee vastaan korjaustarpeita, jotka selviävät vasta oikeassa käytössä. Tulevaisuudessa halutaan, että navigointikomponentti toimii myös offline-tilassa, joten esimerkiksi koodikirjastot pitää toteuttaa paikallisesti eikä erilliseltä palvelimelta. Tämän lisäksi ulkoasumuutokset ovat todennäköisesti tarpeen ennemmin tai myöhemmin, sillä kortin sisältö on toteutettu hyvin yksinkertaisesti oletuksena saatavilla olevilla osilla ja kuvakkeilla.

LÄHTEET

[1] Covrig 2018. Tracker map for Home Assistant. GitHub. Viitattu 14.11.2021 https://github.com/covrig/homeassistant-trackermap

[2] IBM Cloud Education 2020. Message Brokers. Viitattu 27.11.2021 https://www.ibm.com/cloud/learn/message-brokers

[3] MQTT 2020. MQTT: The Standard for IoT Messaging. Viitattu 13.11.2021 https://mqtt.org/

[4] Yuan, Michael 2017. Getting to know MQTT. IBM Developer https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/

[5] Lovelace: Custom Cards. Home Assistant Developer Docs. Viitattu 7.11.2021 https://developers.home-assistant.io/docs/frontend/custom-ui/lovelace-custom-card/

[6] REST API. Home Assistant Developer Docs. Viitattu: 7.11.2021 https://developers.home-assistant.io/docs/api/rest/

[7] Fetch API. MDN Web Docs. Viitattu 7.11.2021 https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

[8] IBM Cloud Education 2021. Docker. Viitattu 29.11.2021 https://www.ibm.com/cloud/learn/docker

[9] Database. Home Assistant Developer Docs. Viitattu 29.11.2021 https://www.home-assistant.io/docs/backend/database/

[10] Bidelman, Eric 2013. Custom Elements. HTML5 Rocks. Viitattu 29.11.2021 https://www.html5rocks.com/en/tutorials/webcomponents/customelements/

[11] Robie, Jonathan, Texcel Research. What is the Document Object Model? W3C. Viitattu 29.11.2021 https://www.w3.org/TR/REC-DOM-Level-1/introduction.html

[12] Codecademy Team 2021. What Is a Framework? Codecademy. Vitattu 30.11.2021 https://www.codecademy.com/resources/blog/what-is-a-framework/

[13] What is LitElement? Lit. Viitattu 30.11.2021 https://lit-element.polymer-project.org/guide

[14] MDN contributors. Web Components. MDN Web Docs. Viitattu 30.11.2021 https://developer.mozilla.org/en-US/docs/Web/Web_Components

[15] Thomasloven 2020. GitHub. Viitattu 30.11.2021 https://gist.github.com/thomasloven/1de8c62d691e754f95b023105fe4b74b

[16] Glazkov, Dimitri 2011. What the Heck is Shadow DOM? Viitattu 30.11.2021 https://glazkov.com/2011/01/14/what-the-heck-is-shadow-dom/

[17] Lifecycle. Lit. Viitattu 30.11.2021 https://lit-element.polymer-project.org/guide/lifecycle

[18] An open-source JavaScript Library for mobile-friendly interactive maps. Leaflet. Viitattu 30.11.2021 https://leafletjs.com/

[19] About-sivu. OpenStreetMap. Viitattu 30.11.2021 https://www.openstreetmap.org/about

[20] Admin 2017. Mitä on avoin data? Helsinki Region Infoshare. Viitattu 30.11.2021 https://hri.fi/fi/ohjeet/mita-on-avoin-data/

[21] MDN Contributors. ShadowRoot. MDN Web Docs. Viitattu 6.12.2021 https://developer.mozilla.org/en-US/docs/Web/API/ShadowRoot

[22] Tile layers. ArcGIS Enterprise. Viitattu 6.12.2021 https://enterprise.arcgis.com/en/portal/10.4/use/tile-layers.htm

[23] OpenStreetMap karttalaatta. https://a.tile.openstreetmap.org/13/4599/2358.png

[24] Home Assistantin kotisivu. Home Assistant. Viitattu 7.11.2021 https://www.home-assistant.io/

[25] Home Assistantin demonäkymä. Home Assistant. Viitattu 12.12.2021 https://demo.home-assistant.io/#/lovelace/0

[26] Billing Account Credits. Google Maps Platform. Viitattu 12.12.2021 https://developers.google.com/maps/billing-credits

[27] OpenLayersin kotisivu. OpenLayers. Viitattu 13.12.2021 https://openlayers.org/

[28] Rajapinnat ja REST. Web-palvelinohjelmointi Java 2019. Viitattu 13.12.2021 https://web-palvelinohjelmointi-19.mooc.fi/osa-6/4-rajapinnat-ja-rest

[29] Lovelace. Home Assistant. Viitattu 18.12.2021 https://www.home-assistant.io/lovelace/

[30] MQTT. Home Assistant. Viitattu 18.12.2021 https://www.home-assistant.io/integrations/mqtt/

[31] Map Card. Home Assistant. Viitattu 18.12.2021 https://www.home-assistant.io/lovelace/map/

[32] Recorder. Home Assistant. Viitattu 18.12.2021 https://www.home-assistant.io/integrations/recorder/

[33] Eclipse Mosquitton kotisivu. Eclipse Mosquitto. Viitattu 18.12.2021 https://mosquitto.org/