Tampereen ammattikorkeakoulu

Tilanhallinta moderneissa verkkosovelluksissa

Jaakko Pullinen

OPINNÄYTETYÖ
Toukokuu 2020

Tieto- ja viestintätekniikan koulutus
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikan koulutus
Ohjelmistotekniikka

PULLINEN, JAAKKO:
Tilanhallinta moderneissa verkkosovelluksissa

Opinnäytetyö 29 sivua
Toukokuu 2020

Tässä opinnäytetyössä kehitetään toimeksiantajan sähkökäyttöisten linja-autojen latauspisteiden hallinnointisovellukselle tehokas tilanhallintajärjestelmä sekä tutkitaan modernien verkkosovellusten tilanhallinnan osa-alueita.

Reaktiivinen ohjelmointi on keskeinen käsite sovellusten tilanhallinnassa. Reaktiivisen ohjelmoinnin periaattein kehitetyt tilanhallintakirjastot suorittavat ja hallinnoivat muutoksia sovelluksessa reaaliaikaisesti ja tehokkaasti. JavaScript-pohjaisten ohjelmistojen reaktiivisen ohjelmoinnin mahdollistava kirjasto on RxJS, josta on jatkokehitetty FCMS-sovelluksessa käytettävä Akita-tilanhallintakirjasto. Akita hallitsee sovelluksissa tilaa käyttämällä tietovarastoja ja niitä muokkaavia kutsupalveluita sekä kyselymetodeja, jotka noutavat tietoa tietovarastosta.

Fleet Charging Management System eli FCMS on sähköisten linja-autojen lataamispisteiden hallinnointiin ja lataamiseen kehitetty sovellus. Sovelluksessa käyttäjälle tarjotaan varikon hallinnointinäkymä päänäkymänä sekä satelliittinäkymä, jossa on myös suodattamistoimintoja. Molemmat näkymät näyttävät varikolla olevat latauspisteet ja niissä olevat latauslaitteet.

FCMS-sovellus hyödyntää tilanhallinnassaan Akitan lisäksi myös selaimen ja palvelimen välisen reaaliaikaisen kommunikoinnin sallivaa WebSockets-teknologiaa. WebSockets on suosittu teknologia muun muassa pikaviestinpalveluissa ja verkkopeleissä. Sovellus vaatii käyttäjältä sisäänkirjautumisen, joka toimii kolmannen osapuolen (Auth0) rajapinnan kautta. Sovellus on kehitetty TypeScriptiä käyttävällä Angular2+-ohjelmistokehyksellä. TypeScript on verkkosovelluksissa suosittu, JavaScriptiksi suoraan kääntyvä ohjelmointikieli, joka tarjoaa vahvan tyypittämisen tuen.

Työssä kehitettiin Akitaa ja WebSocketsia käyttäen työn toimeksiantajan sovellukselle tehokkaasti toimiva tilanhallinta. Sovelluksen kehityksen aikana keskityttiin ohjelmiston kattavaan dokumentointiin, hyvän ohjelmoinnin periaatteita noudattavan koodin tuottamiseen ja sovelluksen luotettavan toiminnan takaamiseen.

Asiasanat: reaktiivinen ohjelmointi, tilanhallinta, FCMS, Akita

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

PULLINEN, JAAKKO:
Modern Web Application's State Management

Bachelor's thesis 29 pages
May 2020

The subject of this thesis was to develop an efficient state management solution to the client's web application which manages electric bus charging stations and to research different areas of state management in modern web applications.

Reactive programming is a central concept in state management software. State management libraries that have been developed with reactive programming principles can manage the application's state changes efficiently and in real-time. RxJS is a library that allows reactive programming in JavaScript-based applications. Akita state management library, which is used in FCMS-application, has been developed on top of RxJS. Akita manages state by using data stores, services that call changes to them, and query methods that fetch information from the stores.

Fleet Charging Management System, or FCMS, is an application developed to manage electrical buses charging stations and their chargers. The app provides two main views to the user, the main view which shows a location's charging stations and its chargers, and a satellite view that shows chargers and filtering functionality.

FCMS uses the Akita library and WebSockets technology in its state management. WebSockets opens a real-time, effective, bidirectional interactive communication session between the user's browser and a server. WebSockets is a popular technology also used in chat applications and online games. FCMS is a closed application that requires the user to log in. The login functionality works through a third-party interface provided by Auth0. The application is developed with a TypeScript framework called Angular. TypeScript is a programming language that builds on JavaScript, offers strong typing functionality, and is widely used in web applications.

The application's state management, developed with Akita and WebSockets, was found to be effective. During the app's development, the focus was on extensive documentation of the software, producing code that is clean and following the principles of good programming and to ensure that the app functions in a reliable way.

Key words: reactive programming, state management, FCMS, Akita

SISÄLLYS

LYHENTEET JA TERMIT

API	Application Programming Interface, ohjelmointirajapinta
CRUD	Create, read, update and delete, muistinkäyttöön liittyvät luonti-, lukemis-, päivitys- ja poistotoiminnot
CSS	Cascading Style Sheets, verkkosivujen tyylittelyyn käytetty kirjoituskieli
FCMS	Fleet Charging Management System, toimeksiantajan sähköisten linja-autojen lataamispisteiden hallinnointisovellus
HTML	Hypertext Markup Language, hypertekstin merkintäkieli, tunnetaan verkkosivujen kirjoituskielenä
HTTP	Hypertext Transfer Protocol, hypertekstin siirtoprotokolla, selainten ja verkkopalvelimien käyttämä tiedonsiirron protokolla
IETF	Internet Engineering Task Force, kansainvälinen organisaatio, joka standardisoi internetin arkkitehtuuria
JavaScript	Pääasiassa verkkoympäristöissä käytettävä dynaaminen ohjelmointikieli
MVP	Minimum Viable Product, pienin toimiva tuote
OCPP	Open Charge Point Protocol, laitekommunikaatioprotokolla sähköisille ajoneuvoille ja niiden latauslaitteille
RxJS	Reactive Extensions Library for JavaScript, reaktiivisen ohjelmoinnin kirjasto JavaScriptille
SCSS	Sassy Cascading Style Sheets, Cascading Style Sheets kirjoituskielestä jatkokehitetty kieli verkkosivujen tyylittelylle
TCP	Transmission Control Protocol, tietoliikenneprotokolla, joka on vastuussa yhteyksien luomisesta laitteiden välillä
TypeScript	JavaScriptin päälle rakennettu ja JavaScriptiksi kääntyvä ohjelmointikieli
W3C	World Wide Web Consortium, kansainvälinen organisaatio, joka standardisoi verkossa käytettäviä teknologioita

1 JOHDANTO

Verkkosovellusten kasvavan määrän ja kysynnän vuoksi myös sovellusten kompleksisuus on kasvanut ajan myötä. Kymmenen vuotta sitten verkkosivut lähinnä näyttivät staattista dataa ja jokainen muutos mitä sivulla tapahtui, täytyi suorittaa palvelinkutsun avulla. Nykyään verkkosovellus hallitsee eri näkymien ja komponenttien tilaa paljolti ilman palvelinkutsuja. Verkkosovellusten monimutkaisuuden kasvun ja niiden kulutuksen kasvun kanssa myös käyttäjien odotukset sille, mitä yksi sovellus tarjoaa, ovat kasvaneet huimasti. Laajoiksi kasvavat ja monimutkaiset verkkosovellukset tarvitsevat älykkäitä ratkaisuja tiedonkulun hallintaan.

Tarjonnan lisäksi myös odotukset sovelluksen suorituskyvylle ovat kasvaneet. Googlen tekemän tutkimuksen mukaan noin 53 % verkkosivuvierailuista hylätään, jos sivu ei lataudu käyttäjän selaimelle 3 sekunnin sisällä, ja yksi kahdesta käyttäjästä odottaa sivun latautuvan alle kahdessa sekunnissa (Think with Google 2016.). Sama tutkimus eriyttää kolme päätekijää mobiilisivustojen pitkille latautumisajoille: suuret tiedostokoot, elementtien latautumisjärjestys ja palvelinkutsut. Tilanhallinta on olennainen osa näistä kahden jälkimmäisen suorituskyvyssä.

Tämän opinnäytetyön tavoitteena on tutkia modernien sovellusten tilanhallintaan kuuluvia osa-alueita ja kehittää työn toimeksiantajan tuotteelle toimiva tilanhallinta. Toimeksiantajana toimiva Plugit Finland Oy on sähköisten ajoneuvojen latausratkaisujen tarjoaja kuluttaja- sekä yritysasiakkaille.

Kehitettävä tuote, jossa opinnäytetyö on osana, on toimeksiantajan Fleet Charging Management System eli FCMS-sovellus, joka toimii käyttöliittymänä sähkökäyttöisten linja-autojen sekä lataamisessa että lataamispisteiden hallinnoinnissa. Opinnäytetyössä tehtävät viittaukset FCMS-sovellukseen tarkoittavat sovelluksen käyttöliittymän MVP-versiota eli pienintä toimivaa tuotetta. Tässä sovelluksen versiossa asiakkaalle tarjotaan yhdessä määritellyt vähimmäisvaatimukset täyttävä sovellus.

2 REAKTIIVINEN OHJELMOINTI

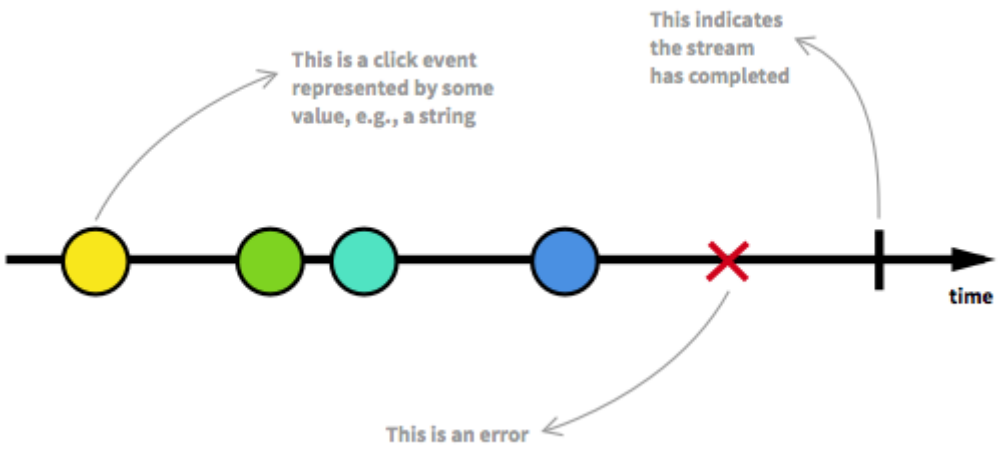
Reaktiivisella ohjelmoinnilla tarkoitetaan ohjelmointia asynkronisilla tietovoilla. Tietovuo (data stream) on jakso olemassa olevia tapahtumia järjestettynä ajassa. Tietovuo voi lähettää kolme erityyppistä signaalia: lähetettävä arvo, virhe ja päättynyt. Päättynyt- ja virhesignaalit päättävät tietovuon (Medeiros 2014.).

Reaktiivinen ohjelmointi on havaittu erityisen tarpeelliseksi moderneissa verkkosovelluksissa niissä tehtävien tapahtumien kasvavan määrän reaaliaikaiseen hallintaan. Sen avulla useasta eri lähteestä eri tavoin saapuvaa tietoa voidaan käsitellä jumiuttamatta tietoa käsittelevää ohjelmistoa.

2.1 Tarkkailija-suunnittelumalli

Reaktiivisessa ohjelmoinnissa määritellään funktio, joka suoriutuu sen vastaanottaessa signaalin tietovuosta. Funktio siis kuuntelee tietovuon muutoksia, tätä kuuntelua kutsutaan tilaukseksi, ja tekee koodissa määriteltyjä toimia muutoksien tapahtuessa. Kyseistä funktiota kutsutaan tarkkailijaksi ja tietovuota, joka on tarkkailun kohde, kutsutaan havaittavaksi.

Tämän kaltaista ohjelmoinnin suunnittelumallia kutsutaan tarkkailija-suunnittelumalliksi (Observer Design Pattern). Tämä on reaktiivisen ohjelmoinnin keskeinen idea.



KUVA 1. Painallustapahtuman tietovuo aikajanalla (Medeiros 2014.)

Esimerkiksi kuvan 1 painallustapahtuman tietovuon tilauksessa tilaus havaitsee painallustapahtuman lähettämiä signaaleita. Signaalit ovat painallustapahtumalle määritettyjä arvoja tai virhesignaaleita tietovuon päättymiseen saakka. Arvoille, joita tilaus vastaanottaa, voidaan suorittaa jatkotoimintoja reaaliaikaisesti arvojen syntyessä.

2.2 RxJS

RxJS on JavaScript-pohjaisten ohjelmistojen reaktiivisen ohjelmoinnin mahdollistava kirjasto. Sen päätarkoituksena on täyttää tarve ideaalille tavalle hallinnoida tapahtumia sovelluksessa (RxJS Introduction n.d.). Kirjasto käyttää tarkkailija-suunnittelumallin elementtejä sekä ajastintyyppejä ja operaattoreita, jotka pystyvät muuntelemaan asynkronisia tapahtumia. RxJS:n perusta on kuitenkin havaittava eli observable.

RxJS on avoimen lähdekoodin ohjelmisto, joka on lisensöity Apache 2.0 -lisenssin alle. Ohjelmisto on Microsoftin kehittämä yhteistyössä sovelluskehittäjäyhteisön kanssa. Kehittäjäryhmä kehittää ohjelmistoa aktiivisesti.

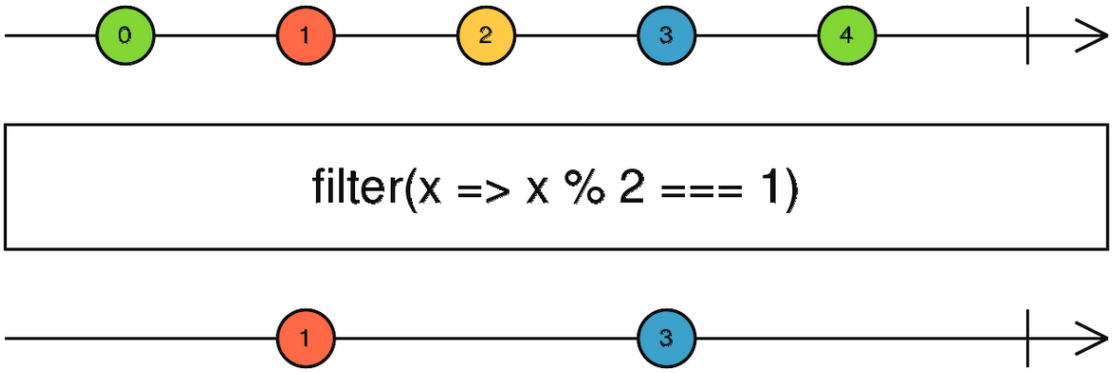
2.2.1 Operaattorit

Operaattorit muodostavat RxJS:n tärkeimmän toiminnallisuuden. Ne sallivat monimutkaisen asynkronisen koodin yksinkertaisen toteutuksen. Operaattorit saavat syötteenä havaittavamuuttujia ja muokkaavat näitä määritetyn toimintansa mukaisesti tai luovat syötteenä niiden perusteella uuden havaittavan (RxJS Operators n.d.). RxJS:llä on hyvin kattava kirjasto operaattoreita. Kirjaston dokumentaatiosivut listaavat yli 100 erilaista muunnosoperaattoria (RxJS API n.d.). Nämä operaattorit ovat funktioita ja niitä on kahta eri tyyppiä, putkitettavia (pipeable operator) ja luomisoperaattoreita (creation operator).

2.2.2 Putkitettavat operaattorit

Putkitettavat operaattorit ovat funktioita, jotka ottavat havaittavan muuttujan syötteenä ja palauttavat muunnetun havaittavan ulostulona. Alkuperäinen muuttuja

siis pysyy koskemattomana. Muunnettuun tulostettavaan muuttujaan tehtävä tilaus suorittaa tilauksen myös alkuperäiseen operaattorin syötemuuttujaan.



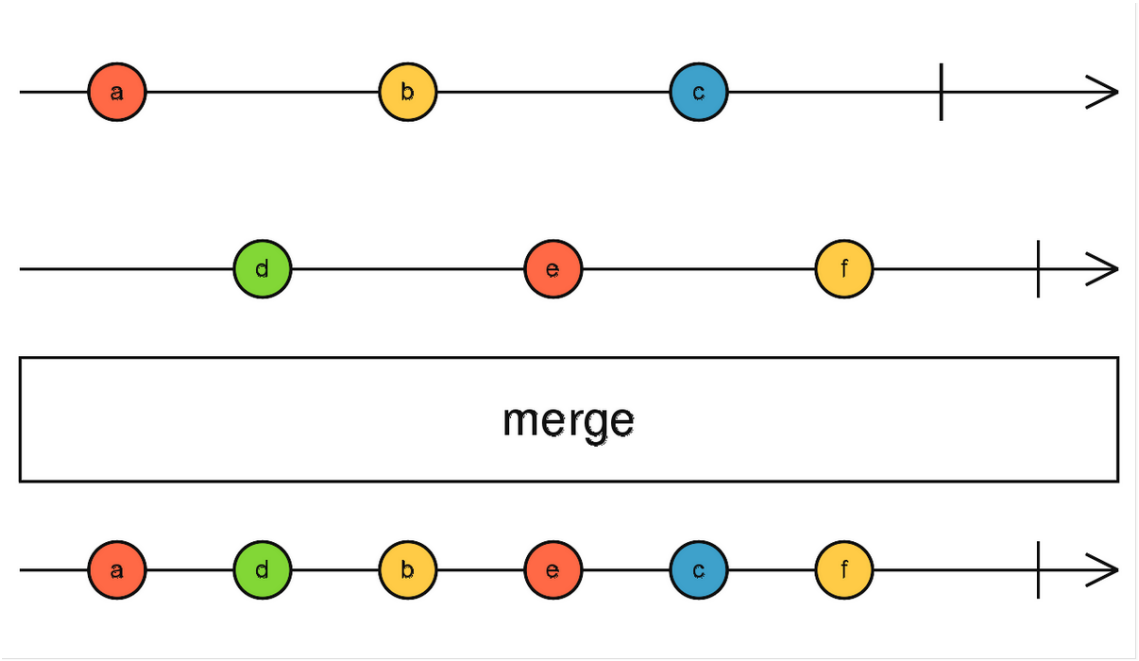
KUVA 2. Filter-operaattorin toiminta aikajanalla (RxJS API N.d.)

Kuvassa 2 kuvataan kahta havaittavamuuttujaa aikajanalla sekä filter-operaattoria. Operaattori saa syötteekseen luonnollisia lukuja lähettävän havaittavamuuttujan ja arvioi jokaisen syötteeltä saaman arvon predikaattinsa mukaisesti. Operaattorin predikaatti on suodattaa syötteestä kahdella jaettavat numeeriset arvot. Predikaatti palauttaa operaattorille arviointinsa mukaan tosi- tai epätosiarvon. Jos arvo on tosi, operaattori päästää arvon eteenpäin tulostehavaittavalle. Operaattori toimii predikaattinsa mukaisesti ja tulostaa suodetut arvot välittömästi aikajanalla. Koska kyseessä on tarkkailija-suunnittelumallin mukainen rakenne, kaikki operaattorin suorittamat muokkaukset tapahtuvat reaaliaikaisesti tilauksessa.

Putkitettavista operaattoreista monilla on samaa toiminnallisuutta JavaScriptin array-luokan operaattoreiden kanssa. Array-luokan tunteminen ja sen muokkaamisen osaaminen on hyvä perusta putkitettavien operaattoreiden käytölle.

2.2.3 Luomisoperaattorit

Luomisoperaattorit ovat funktioita, jotka luovat täysin uuden havaittavan muuttujan. Niillä on yhteinen ennalta määritetty käyttäytymistapa tai ne yhdistävät muita havaittavia muuttujia yhdeksi muuttujaksi.



KUVA 3. Merge-operaattorin toiminta aikajanalla (RxJS API N.d.)

Kuvassa 3 kuvataan kahta syötehavaittavaa aikajanalla sekä merge-operaattoria, joka luo yhdistetyn tulostehavaittavan. Tulostehavaittava samanaikaisesti lähettää kaikki arvot jokaisesta sille annetusta syötehavaittavasta. Merge suorittaa tilauksen jokaiseen sille argumenttina annettuun syötehavaittavaan ja lähettää nämä eteenpäin yhtenä tulostehavaittavana, jonka arvot ovat oikea-aikaisessa järjestyksessä aikajanalla. Nämä arvot pysyvät muuttumattomina. Tulostehavaittava päättyy vasta jokaisen syötehavaittavan päättyessä.

Luomisoperaattorit tekevät tilauksen syötehavaittavaan, mutta jättävät sen muokkaamattomaksi. Tämä sallii syötehavaittavan käytön muilla operaattoreilla tai menetelmillä. Usein tätä havaittavaa käytetäänkin useammalla eri operaattorilla tai menetelmällä. Operaattorin luoman tulostehavaittavan päättymislogiikka, tulostehavaittava päättyy vasta jokaisen syötehavaittavan päättyessä, myös varmistaa kaiken tiedon vastaanottamisen ja muokkaamisen operaattorin toimintalogiikan mukaisesti.

3 TILANHALLINNAN MENETELMÄT

Moderneissa verkkosovelluksissa tilaa hallitaan palvelimen ja selaimen välillä http-kutsujen avulla sekä reaaliaikaisesti WebSocket-yhteyksillä. Selaimen sisällä tilaa hallitaan valitun ohjelmistokehyksen ominaisuuksilla ja ulkoisilla ohjelmakirjastoilla, jotka tarjoavat tilanhallintatoiminnallisuuksia. Selaimen sisäisen tilanhallinnan etuina ovat lyhyemmät vasteajat käyttäjän toiminnon ja tilanmuutoksen tapahtuman sekä sen visuaalisen vasteen välillä. Lyhyet vasteajat ovat vahvasti liitännäisiä sovelluksen positiiviseen käyttäjäkokemukseen.

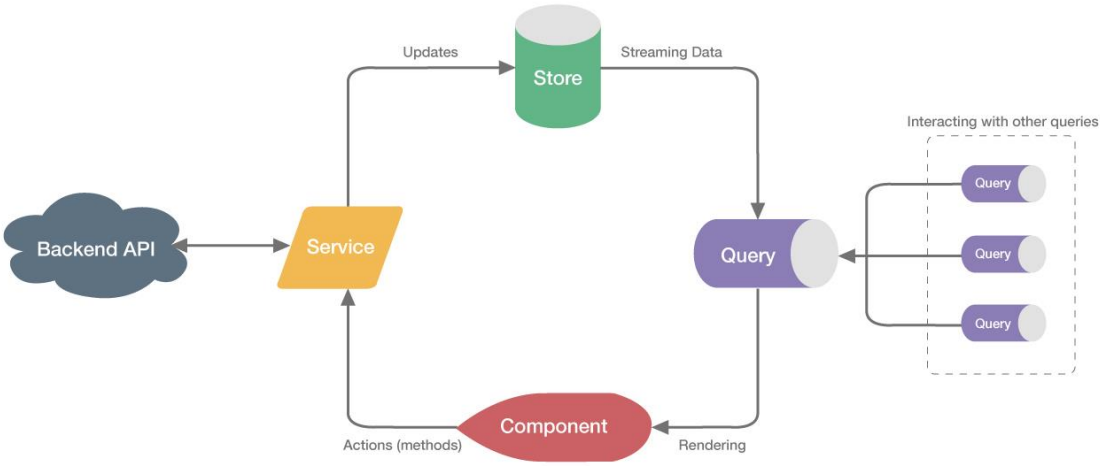
3.1 Akita-tilanhallintakirjasto

Akita on JavaScript-pohjaisten ohjelmistojen tilanhallintakirjasto, joka on kehitetty RxJS-kirjastoa käyttäen. Sen keskeisenä toimintaideana on datavarastojen käyttö, joita voidaan manipuloida ja joista voidaan hakea tietoa reaktiivisen ohjelmoinnin tavoin tietovuon tilaamalla.

Akitan kehittäjien kuvauksen mukaan Akita on suunniteltu kannustamaan yksinkertaisen koodin tuottoa ja se on luotu olio-ohjelmoinnin suunnitteluperiaatteilla. Sillä on tarkasti määritelty rakenne, joka tarjoaa selkeyttä kehyksen käyttöön (Basal 2018.). Akitan kehittäjäryhmä on aktiivisesti jatkokehittämässä kirjastoa ja tarjoaa paljon tukea ohjelmistoa käyttäville kehittäjille verkossa. Akitasta on myös hyvin löydettävissä oppimateriaalia verkosta.

3.1.1 Rakenne

Kuvassa 4 kuvataan Akitan olennaisimmat käsitteet. Nämä ovat kysely (query) ja tietovarasto (store). Tyypilliseen rakenteeseen kuuluvat myös komponentti (component), kutsupalvelu (service) ja taustarajapinta (Backend API).



KUVA 4. Akitan arkkitehtuuri (Basal 2018.).

Rakenteen käsitteistä kysely ja tietovarasto ovat Akitan tarjoamia toiminnallisuuksia, joita voidaan tuoda sovellukseen ja määritellä sen koodissa. Kutsupalvelut ja komponentit ovat sovelluksen omia ohjelmiston osia, jotka hyödyntävät ja toimivat yhdessä Akitan kanssa. Taustarajapinnat voivat olla sovellukseen tuotavia kolmannen osapuolen rajapintoja tai palvelimien kommunikointiin itse sovellusta varten kehitettyjä rajapintoja.

Kyselyistä tulee huomioida, että niitä voidaan saattaa yhteen ja ne voivat vuorovaikuttaa toisiinsa. Tietovarastoja taas tulee käsitellä aina tarkasti määritellyn tietomallin mukaan.

3.1.2 Tyypillinen toiminta

Tyypillisessä Akitan käyttötapahtumassa tilaansa muuttava sovelluksen komponentti tekee rakentuessaan kutsun, joka muodostaa aktiivisen tietovuo-tilauksen koodissa määritettyyn tietovarastoon. Tämä tilaus päivittää komponenttia tietovaraston muutoksien tapahtuessa. Tietovarastoa voidaan muuttaa kutsupalvelun kautta lähettämällä päivityskäskyjä. Tietovarasto vastaanottaa päivityskäskyn ja päivittää tietomallinsa tämän mukaisesti. Samalla komponentin tietovuo-tilaus suorittaa päivityksen komponentin tilassa.

Ohjelmiston koodissa määritetään tarkasti tietovaraston tietorakenne, tietovaraston eri kokonaisuuksien tietomalli ja lähetettävät kutsut.

3.1.3 Store

Store eli tietovarasto on objekti, joka sisältää määritellyn tietovaraston tilan ja toimii keskitettynä tiedonlähteenä tilanhallinnalle. Tietovarasto tyypitetään sen tietomallilla. Tietomalli rajaa tarkasti tietovaraston rakenteen ja sallii kyselyt ja päivitykset vain mallin mukaisesti.

Storesta on erikoistuneempi luokitus, entity store, jonka tyyppisiä suurin osa Akitassa käytetyistä tietovarastoista on. Entity store -tyyppistä tietovarastoa voidaan muokata kutsupalvelun kautta useammalla metodilla. Näihin kuuluvat tietokantojen käsittelystä tutut CRUD-toiminnallisuudet eli tietoalkioiden luonti, lukeminen, päivittäminen ja poistaminen. Metodeihin kuuluvat myös upsert eli tietoalkion lisääminen tai päivittäminen, upsertMany eli usean tietoalkion lisääminen tai päivittäminen, replace eli tietoalkion korvaaminen tunnistetta lukuun ottamatta, move eli tietoalkion siirtäminen indeksistä toiseen, setLoading eli tietovaraston latautumistilan päivittäminen, setError eli tietovaraston virhetilan päivitys sekä destroy eli tietovaraston tuhoaminen.

Tietovarasto tukee tietueen aktiivisen tilan asettamista. Osa tietovarastosta voidaan merkitä aktiiviseksi, jolloin sitä on helpompi käsitellä lisämetodeilla ja sovelluksessa voidaan näyttää relevanttia tietoa helpommin käyttämällä tietovaraston aktiivista tilaa.

3.1.4 Query

Query eli kysely on Akitan luokka, joka suorittaa tietovaraston kyselyt. Kysely saa parametreinaan tietovaraston, joihin kyselyitä lähetetään, sekä valinnaisesti muita koodissa määritettyjä kyselyluokkia. Akitan arkkitehtuurin (kuva 4) mukaisesti kyselyluokkia voidaan yhdistää ja ne voivat olla vuorovaikutuksessa keskenään. Kyselyluokalla on metodeinaan select, getValue, selectLoading sekä selectError jotka suorittavat eri tyyppisiä kyselyitä.

Select-metodi valitsee sille annetun määritelmän mukaisesti tietyn osuuden tai yksittäisen tietoalkion tietovarastosta. Metodi palauttaa aina havaittavan (observable), jolle voidaan suorittaa tilaus. Metodi laukeaa tietovaraston tilan muuttuessa. Muut select-metodit selectLoading ja selectError toimivat samankaltaisesti mutta rajatummin. SelectLoading valitsee tietovaraston loading- eli latautumistilan ja selectError valitsee tietovaraston error- eli virhetilan. GetValue-metodi palauttaa tietovaraston raaka-arvot.

Yleisen Akitan kyselyluokan lisäksi Akitalla on erikoistuneempi kyselyluokka, entity query. Entity queryt ovat aina liitännäisiä tietovaraston erikoistuneempaan entity store -luokkaan. Entity queryt ovat lisätoiminnallisuuksia kyselyiden tekemiseen. Niihin kuuluvat muun muassa selectAll, kaikkien tietoalkioiden valinta, selectMany, usean tietoalkion valinta, selectFirst, ensimmäisen tietoalkion valinta, selectLast, viimeisen tietoalkion valinta ja selectCount, tietovaraston alkioiden lukumäärän valinta.

3.1.5 Service

Service eli kutsupalvelu on Akitan kehittäjien suosittelema rakenne tietovarastoon tehtävien muutoksien tekemiseen. Kutsupalvelua käyttämällä voidaan välttää rakenteellisia virheitä ohjelmistossa, joita saattaa syntyä, mikäli tietovarastoa muokattaisiin suoraan siihen liittyvistä ohjelmiston komponenteista. Kutsupalvelussa määritetään ja suoritetaan tietovaraston muokkaamismetodit (kts luku 3.1.3).

3.2 WebSockets

WebSocketsia käytetään kaksisuuntaiseen kommunikointiin palvelimen ja selaimen välillä. Se on suosittu teknologia moderneissa verkkosovelluksissa. WebSocketsin pääkäyttökohteina ovat pikaviestintäpalvelut, reaaliaikaisesti päivittyvät analytiikkatyökalut, kaksisuuntainen mediatiedostojen suoratoisto sekä verkkodokumenttien samanaikainen muokkaus usean käyttäjän kesken (Socket.io n.d.).

3.2.1 Protokolla

WebSocket-protokolla mahdollistaa kaksisuuntaisen reaaliaikaisen kommunikaation selaimen ja palvelimen välillä avaamalla yksittäisen TCP-yhteyden. Protokolla luotiin kasvaneeseen kaksisuuntaisen kommunikaation tarpeeseen esimerkiksi verkkopeleissä ja pikaviestinpalveluissa. Ennen protokollan luontia monet palvelut luottivat suureen määrään http-kutsuja luomaan päivityskyselyitä palvelimelle, joka johtaa palvelimen luomaan useita TCP-yhteyksiä jokaiselle selaimelle tai käyttäjälle. Tämä aiheuttaa raskaan ja tehottoman yhteyden palvelimen ja selaimen välillä (Fette, Melnikov 2011.).

Protokolla on IETF:n vuonna 2011 standardisoima ja WebSocket API on W3C:n standardisoima. Molemmat ovat luotettavia ja vaikuttavia standardisoinnin yhteisöjä verkkoteknologioissa. Vaikka WebSocket-protokolla on irrallinen http-protokollasta, on se suunniteltu olemaan yhteensopiva http:n kanssa. Lähes kaikki verkkoselaimet tukevat protokollaa, mukaan lukien kaikki käytetyimmät selaimet kuten Google Chrome, Firefox ja Safari.

3.2.2 Toiminta

Verkkosovelluksessa WebSocketsia käytetään JavaScriptin WebSocket API:n kautta. Sovelluksessa avataan WebSocket-yhteydelle oma palvelin, ja selaimella määritellään WebSocket-objekti, jolle annetaan parametreinä osoite ja valinnaisesti käyttöprotokolla. Kun objekti on luotu, se voi lähettää ja vastaanottaa viestitapahtumia tai yhteyden lopetustapahtuman. Viestitapahtuma voi sisältää erilaisia tietotyyppejä, kuten multimedia- tai tekstidataa (Socket.io Overview n.d.). Jos yhteydessä tapahtuu virhe, WebSocket-objekti vastaanottaa virhetapahtuman. Virhetapahtuma kutsuu objektin virheenhallintafunktiota, joka lähettää objektille takaisin sulkeutumistapahtumaviestin, jossa ilmaistaan syy yhteyden sulkemiselle.

WebSockets on hyvin joustava toiminto yhteyden rakentamiseen, sillä on yksinkertaiset komennot yhteyden avaamiseen, sulkemiseen ja tiedonlähettämiseen.

Tietoa voi lähettää hyvinkin erilaisissa muodoissa, mikä tarjoaa sille paljon erilaisia käyttömahdollisuuksia. Protokollan standardisointi ja verkkoselainten sille tarjoama tuki sille tekee siitä suositun teknologian sovelluksissa.

4 FLEET CHARGING MANAGEMENT SYSTEM -SOVELLUS

FCMS-sovellus toimii käyttöliittymänä sähkökäyttöisten linja-autojen sekä latauksessa että latauspisteiden hallinnoinnissa. Työssä osallistuttiin sovelluksen MVP-version tilanhallinnan kehitykseen.

Tässä versiossa käyttäjälle tarjotaan näkymä varikon latauslaitteisiin ja tietoa niiden käyttötilasta, karttanäkymä latauslaitteiden sijainnista varikolla, autentikointi sekä navigointi sovelluksen satelliitit- ja päänäkymän välillä ja eri varikoiden välillä.

4.1 Teknologiat

Sovelluksen käyttöliittymän ohjelmoinnissa käytettiin Angular2+ TypeScript -ohjelmistokehystä. Käyttöliittymän tilanhallinnassa käytettiin Akita-tilanhallintakirjastoa. Sovellus liittyy autentikointia varten Auth0:n rajapintaan sekä karttanäkymää varten Google Mapsin rajapintaan. Auth0 on yksi suosituimmista verkkosovellusten käyttämistä kolmannen osapuolen rajapinnoista autentikointiin. Palvelu suorittaa yli 2,5 miljardia kirjautumistapahtumaa kuukausittain (Auth0 Overview n.d.). WebSocket-yhteyksien luomisessa käytetään socket.io-kirjastoa.

4.1.1 TypeScript

TypeScript on ohjelmointikieli, joka kääntyy suoraan JavaScriptiksi. TypeScript lähinnä lisää ominaisuuksia JavaScriptille ja on sen kanssa täysin yhteensopiva. TypeScriptin lisäominaisuuksista tärkeimpänä pidetään vahvan tyypittämisen tukea.

Vahvasti tyypitetyissä muuttujissa määritellään muuttajan datatyyppi (esimerkiksi string). Kun tyyppi on määritelty, muuttujaa ei voi käsitellä millään muun kaltaisella tietotyypillä. Vahvasti tyypitetyt muuttujat tarjoavat ohjelmistolle helpommin ennakoitavaa käyttäytymistä (Strongly-typed programming language 2003.). Tämä ennakoitava käyttäytyminen tarjoaa ohjelmistolle parempaa ohjelmistovirheiden ennaltaehkäisyä.

Tyypittäminen TypeScriptissä on valinnaista, mutta sen käyttö voi olla hyvin hyödyllistä ohjelmiston kehittäjälle. Useat modernien verkkosovellusten sovelluskehykset käyttävät TypeScriptiä, ja sen suosio on kasvamassa (TypeScript 2016.).

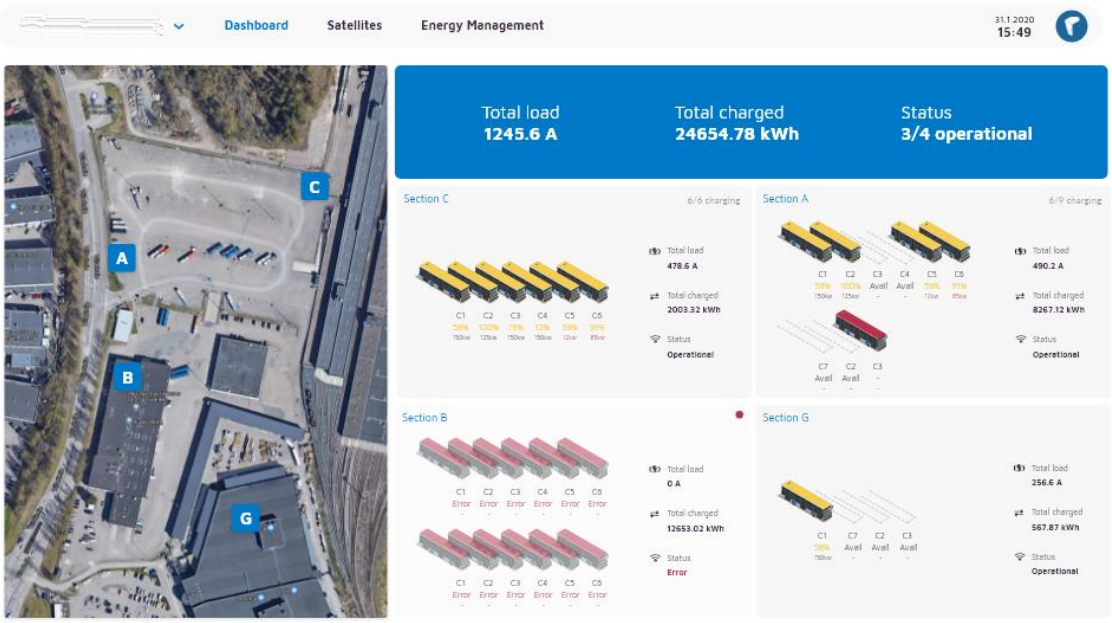
4.1.2 Angular2+

Angular2+ on sovelluskehityksessä käytettävä TypeScript-ohjelmointikielen ohjelmistokehys. Suurin osa Angular-komponentin tai näkymän toiminnallisuudesta luodaan TypeScript-tiedostojen avulla, mutta tyypillinen Angular-komponentti on yhdistelmä, joka muodostuu TypeScript-, HTML- ja CSS- tai SCSS-tiedostoista. HTML-tiedosto toimii sovelluksen näkymän luomisessa ja CSS- tai SCSS-tiedostolla tyylitellään visuaalisesti HTML-tiedoston luomaa näkymää.

Angular tarjoaa lisätoiminnallisuutta HTML-tiedostoihin Angularin omalla HTML-sapluunan syntaksilla, joka muokkaa HTML-tiedostoa sovelluksen logiikan mukaisesti. Tämä syntaksi tuo paljon eri toimintoja HTML-elementteihin, joista päätoimintoina voidaan pitää if-ehtolauseita, for-silmukoita ja datan välittämistä suoraan elementtiin (Angular n.d.).

4.2 Sovellus

Sovellus on suljettu ohjelmisto, joka vaatii käyttäjältä autentikoinnin. Autentikointi suoritetaan kolmannen osapuolen (Auth0) rajapinnasta. Autentikoinnin jälkeen sovellus saa rajapinnalta käyttäjätiedot ja ohjaa käyttäjän hallinnointinäkymään (kuva 5), joka toimii sovelluksen päänäkymänä.



KUVA 5. Luonnos FCMS-sovelluksen käyttöliittymän hallinnointinäkymästä.

Sovelluksessa voidaan navigoida päänäkymän ja satelliitit-näkymän välillä ja käyttäjä voi valita tietojensa perusteella määritellystä listasta varikon, jota tarkastella.

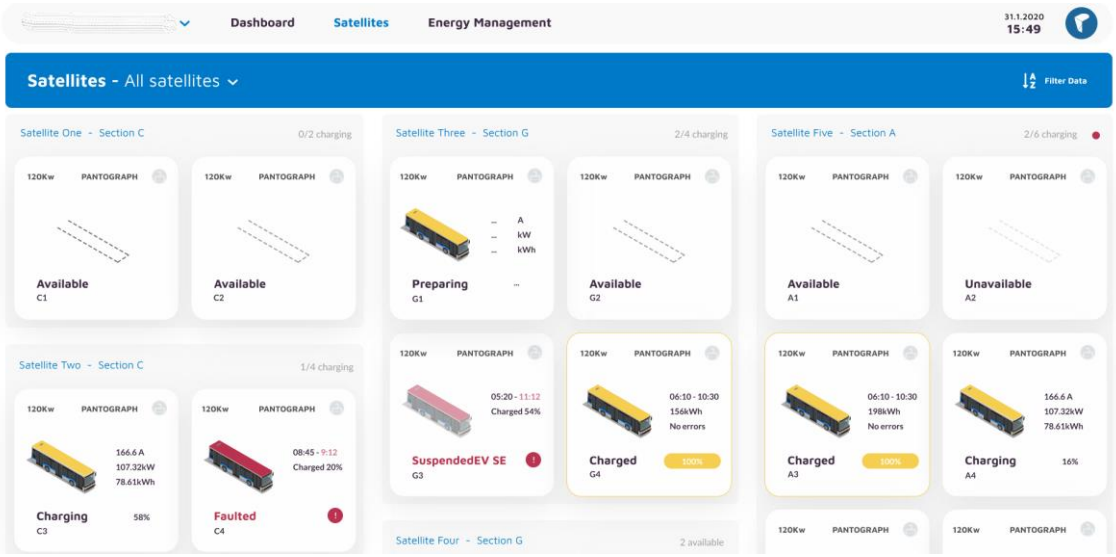
4.2.1 Päänäkymä

Päänäkymässä näytetään yläpalkki, joka toimii paikkana käyttäjän navigointi- ja kirjautumistoiminnoille, käyttäjän valitun varikon satelliittikuva, yhteenveto lataustapahtumista varikolla sekä varikon jaetut osastot (section). Osastojen sijainnit varikolla osoitetaan ikoneilla satelliittikuvassa.

Osastojen latauslaitteet, latauslaitteiden tila, osaston tila sekä tärkeimpiä lataustietoja osastosta näytetään päänäkymän osastokorteissa. Latauslaitteiden tilaa osoitetaan eri värisinä bussi-ikoneina korttien sisällä.

4.2.2 Satelliitit-näkymä

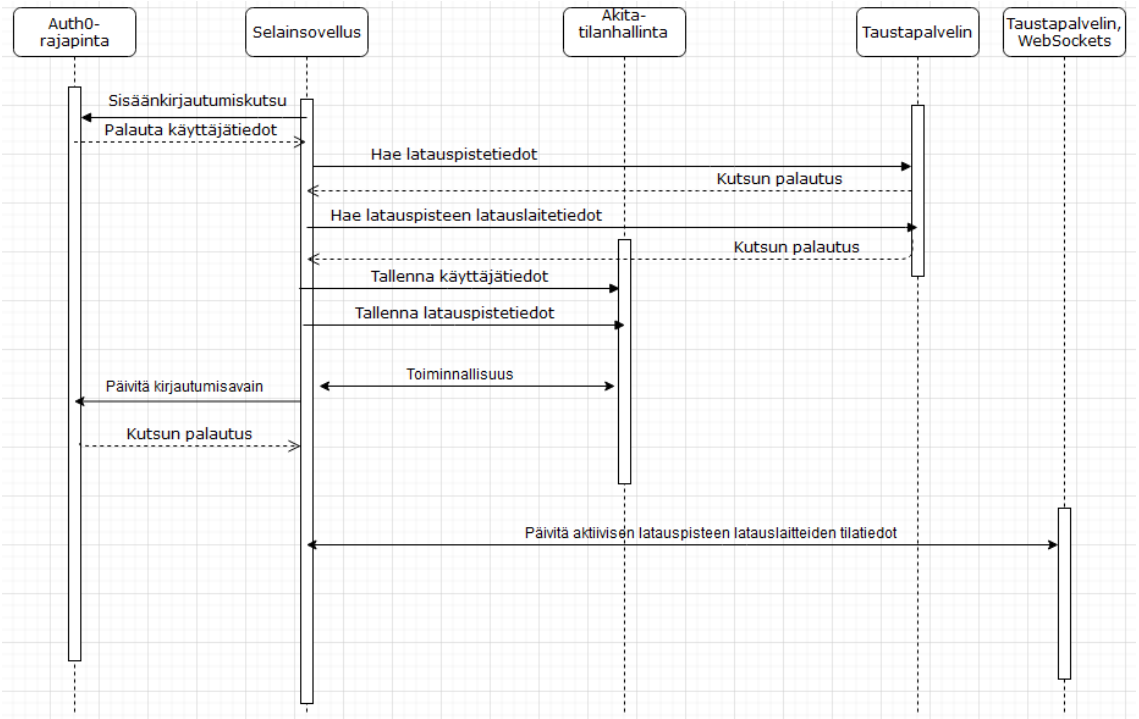
FCMS-sovellus tarjoaa myös toisen näkymän, satelliittinäkymän (kuva 6). Tämä sovelluksen osa-alue jakaa varikon latauslaitteet helpommin hallittavaan näkymään ja tarjoaa mahdollisuuksia suodattaa nähtäviä latauslaitteita.



KUVA 6. Luonnos FCMS-sovelluksen käyttöliittymän satelliittinäkymästä

4.3 Sovelluksen tilanhallinta

Sovelluksen tilanhallinta eriytyy kommunikaatioon palvelimen ja autentikointirajapinnan kanssa sekä tilanhallintakirjaston toiminnallisuuksiin ja ohjelmistokehyksen käyttöliittymän komponenttien visuaalisiin muutoksiin.



KUVA 7. Sekvenssikuvaaja sovelluksen tilanhallinnasta

Kuvassa 7 on esitettynä, mistä selainsovellus hakee tietonsa ja miten tiedot on liitetty Akita-tilanhallintakirjastoon. Akita on sekvenssikuvaajassa eriteltynä selainsovelluksesta selkeyden vuoksi, vaikka se onkin osa selainsovelluksen kokonaisuutta. Sovellus tarvitsee toiminnassaan tietoja käyttäjältään autentikoinnin suorittamiseksi sekä palvelimelta latauspisteen tietoja latauslaitteiden ja lataustapahtumien tietojen esittämiseen. Latauspisteen latauslaitteiden tilatietoja päivitetään reaaliaikaisesti WebSocket-yhteydellä.

Selainsovellus ottaa yhteyden taustapalvelimeensa sille kehitetyn rajapinnan kautta, jota ei kuvassa 7 erikseen eritellä, vaan kommunikoinnin osoitetaan tapahtuvan suoraan palvelimelle. Tämä rajapinta toimii vain taustapalvelimen yhteyksien käsittelijänä.

4.3.1 Autentikaatio

Sovelluksessa määritetään käyttäjälle tilaobjekti (kuva 8), jonka tiedot haetaan autentikointirajapinnalta ja jonka aktiivista kirjautumistilaa käsitellään onnistuneen sisäänkirjautumiskutsun tuloksen perusteella.

```
export interface AuthState {
  loggedUser: any;
  isLogged: boolean;
  permissions: Permissions;
}
```

KUVA 8. Sovelluksen autentikaatio-tietovaraston tietomalli.

Sovelluksen avautuessa käyttäjälle sovellus tarkistaa, onko käyttäjällä aktiivista kirjautumistilaa (isLogged). Mikäli tätä ei löydy, sovellus avaa sisäänkirjautumisnäkymän.

```
isLoggedIn$ = this.authQuery.isLogged$;
ngOnInit(): void {
  // If user is already logged in, redirect to the charge points page.
  this.isLoggedIn$
    .pipe(untilDestroyed(this))
    .subscribe((loggedIn) => {
      if (loggedIn) {
        this.router.navigate(['/charge-points']);
      }
    });
}
```

KUVA 9. Sovelluksen sisäänkirjautumiskomponentin sisäänkirjautumistilan tarkistus.

Tämä tarkistus (kuva 9) tapahtuu Akitan query-metodia käyttämällä. Sovelluksen kirjautumiskomponentin rakentuessa komponentti tarkastaa aktiivisen sisäänkirjautumisen (isLoggedIn) boolean-muuttujan tilan tarkkailija-suunnittelumallin mukaisesti tilaamalla havaittavamuuttujan Akitan query-metodilla autentikaatio-tietovarastosta (kuva 8).

Jos tila on tosi, käyttäjä ohjataan sovelluksen päänäkymään. Jos tila on epätosi, sovelluksen sisäänkirjautumisnäkymä avautuu. Näkymässä käyttäjä voi painaa sisäänkirjautumisnappia, joka avaa linkin Auth0 -autentikaatiorajapinnan sisäänkirjautumiskomponenttiin. Varmennetuilla kredentiaaleilla Auth0 palauttaa sovellukselle käyttäjätiedot ja sovellus ohjaa käyttäjän päänäkymään. Käyttäjätietojen mukana käyttäjälle annetaan kirjautumisavain, joka tarkistetaan ja päivitetään määritellyillä aikaväleillä. Kirjautumisessa Akitan tietovarastoon päivitetään käyttäjäobjekti ja aktiivinen kirjatumistila sekä luvat sille, mitä käyttäjä voi tehdä sovelluksessa.

4.3.2 Akita

Jokaisella Akitan tietovarastoon tallennettavalla objektilla on tarkasti määritelty tietomalli. Jos tietovarastoon koitetaan asettaa tietoa, joka ei ole mallin mukaista, palauttaa Akita virhetapahtuman sovellukselle. Sovelluksen tietovarastot ovat myös vahvasti tyypitetyt.

```
export interface ChargePoint {
  _id: string;
  chargePointGroupId: string;
  name: string;
  description?: string;
  address: Address;
  location: Location;
  assets: string[];
  tags: string[];
  hasError: boolean;
  sections?: Section[];
}
```

KUVA 10. Latauspisteen tietomalli

Latauspisteen tietomalli (kuva 10) määrittelee latauspisteen tietovaraston rakenteen. Tietovarastoa muokataan ja hyödynnetään kuvan 4 Akitan arkkitehtuurimallin mukaisesti. Latauspisteen kutsupalveluun on määritetty toiminnot tietovaraston tyhjäämiseen sekä aktiivisen latauspisteen määrittämiseen Akitan tietovaraston setActive-metodilla.

```
@Injectable({ providedIn: 'root' })
export class ChargePointsQuery extends QueryEntity<ChargePointsState> {
  chargePoints$ = this.selectAll();
  activeChargePoint$ = this.selectActive() as Observable<ChargePoint>;
  loading$ = this.selectLoading();

  constructor(protected store: ChargePointsStore) {
    super(store);
  }

}
```

KUVA 11. Latauspisteen tietovaraston kyselyt

Latauspisteen tietovarastolle määritellyt kyselyt ovat Akitan query-metodeista selectAll, selectActive sekä selectLoading. Näillä kyselyillä (kuva 11) saadaan käyttäjälle näytettyä listaus saatavilla olevista latauspisteistä, joista käyttäjä voi valita yhden käytettäväksi eli aktiiviseksi latauspisteeksi. Sovelluksella voi olla myös käyttäjätiedoissa muistissa käyttäjän oletusaktiivinen latauspiste. Loading$-muuttujalla määritellään käyttöliittymän latautumiskäyttäytymistä.

4.3.3 Palvelin

Sovelluksen taustapalvelimelta haetaan http-kutsuilla latauspisteiden, latauslaiteosastojen ja yksittäisten latauslaitteiden tiedot. Kutsun palautuessa haetut tiedot asetetaan Akitan tietovarastoihin, mikäli vastaava tietovarasto sovelluksessa on (kuva 12). Nämä kutsut tapahtuvat useimmiten sovelluksen eri näkymien rakentuessa.

```
/**
 * Get charge points from api and sets the response to the store.
 */
getChargePoints() {
  return this.http.get<ChargePoint[]>(`${environment.API.URL}/charge-points`).pipe(tap((entities) => {
    this.chargePointsStore.set(entities.map(entity => createChargePoint(entity)));
  }));
}
```

KUVA 12. Latauspisteiden tietojen haku

Sovelluksessa myös asetetaan aktiivinen latauspistekohtainen WebSocket-kanava. Kanavan avulla latauspisteen latauslaitteiden tilatietoja voidaan päivittää sovelluksessa reaaliaikaisesti.

Latauslaitteiden tilatietoja käsitellään ja päivitetään sähköajoneuvojen ja niiden latauslaitteiden välisen OCPP-protokollan mukaisesti, jolla määritellään kansainvälisesti yhtenäinen rakenne sähköajoneuvojen, latauslaitteiden ja niitä hallitsevien ohjelmistojen kommunikointiin (Chargelab 2020.).

4.3.4 Käyttöliittymä

Sovelluksen ohjelmistokehys Angular2+ tarjoaa paljon toiminnallisuutta tilanhallintaan sekä erilaisille vasteille käyttäjän toiminnoista. Sovelluksessa eri HTML-sapluunojen (templates) näyttämiseen käytetään Angularin asynkronisia toimintoja sekä ehtolauseita HTML-elementtien sisällä.

```
<div class="sectionCard__header">
    <ng-container *ngIf="!(sectionsLoading$ | async); else LoadingSectionsTpl">
        <span>{{ section.name }}</span>
    </ng-container>
</div>
<ng-template #LoadingSectionsTpl>
    <span>{{ 'sectionCard.loading' | translate }}</span>
</ng-template>
```

KUVA 13. Sovelluksen HTML-elementtien toiminnallisuutta

Kuvassa 13 näkyvään Angular-elementtiin on merkitty if-ehtolause, jolle annetaan sectionsLoading$-havaittava boolean-muuttuja, jonka tilan mukaan näytetään HTML-näkymässä eri sapluunat.

Tässä tapauksessa Angularin HTML-elementtien asynkronisen toiminnallisuuden ansiosta elementti voi itse muodostaa tietovuon tilauksen tarkkailija-suunnittelumallin mukaisesti havaittavasta sectionsLoading$-muuttujasta. Kyseinen muuttuja on komponentin koodissa määritetty olevan Akitan tietovaraston query-metodi. Elementti siis havaitsee suoraan Akitan tietovarastossa tapahtuvat muutokset tähän muuttujaan liittyen.

5 POHDINTA

Tilanhallinnan tarve verkkosovelluksissa on ollut luonnollinen osa sovelluskehityksen kaarta. Tarvetta on onnistuneesti täytetty avoimen lähdekoodin periaatteella kehitettyjen ohjelmakirjastojen ja niitä kehittävän ohjelmoijayhteisön avulla. Avoin lähdekoodi ohjelmistojen kehittämismenetelmänä on ollut erittäin tärkeä kaikkien ohjelmistojen kehitykselle ja verkkosovellusten piirissä tämä periaate on hyvin yleinen. Tässä opinnäytetyössä mainituista teknologioista RxJS, Akita, Angular ja TypeScript ovat kaikki jonkinlaisen avoimen lähdekoodin lisenssin alla. Jokainen näistä on verkkosovelluksissa käytettyjen teknologioiden kärkipäästä.

Verkkosovellusten tilanhallinnan ratkaisuissa suositaan usein tietyillä variaatioilla reaktiivista ohjelmointia käyttävää tietovarasto-komponentti-kysely-päivityskutsu -rakennetta, jota myös Akita käyttää (kuva 4). React-ohjelmistokehyksen puolella tätä käyttävät Redux- ja MobX-tilanhallintakirjastot ja itseasiassa kyseistä rakennetta usein kutsutaan Redux-malliksi. Tietovaraston ”ainoa totuuden lähde” -periaate tilanhallinnassa yksinkertaisuutta ohjelman kehityksessä ja debuggauksessa.

Redux-mallin lisäksi on olemassa myös toisenlaisia verkkosovellusten tilanhallintaratkaisuja. React-ohjelmistokehyksen kehittäjät julkaisivat Reactin 16.8 versiossa ”Hooks” -periaatteella toimivan tilanhallintamenetelmän. On mielenkiintoista seurata, millä tavalla erilaisilla periaatteilla toimivat tilanhallintamenetelmät tulevat kehittymään tulevaisuudessa.

Tässä työssä käytetty Akita-tilanhallintakirjasto havaittiin hyvin toimivaksi ratkaisuksi FCMS-sovellukselle. Sovelluksen kehittäjätiimillä oli jo aikaisempaa kokemusta kirjaston käytöstä aiemmista projekteistaan, ja siksi sen käyttöä suositeltiin tämän opinnäytetyön tekijälle. Työn aikana ohjelmistokirjaston käyttö ei osoittautunut erityisen haastavaksi, sillä sen syntaksi ja käyttötavat olivat hyvin samassa linjassa tekijän aiemman ohjelmointikokemuksen kanssa. Akitan rakenne on myös hyvin intuitiivinen ja helppo oppia vähänkin kokeneelle ohjelmistokehittäjälle. Akitan kehittäjäryhmä tarjoaa myös aktiivisesti apua verkon kautta kirjastoa käyttäville kehittäjille. Sen piirteet on hyvin dokumentoitu ja verkosta löytyy paljon artikkeleita ja muuta oppimismateriaalia kirjaston käyttöön.

Myös RxJS, jonka päälle Akita on rakennettu, on laajalti käytetty verkkosovelluksissa sekä hyvin dokumentoitu. Myös RxJS:stä löytyy paljon oppimismateriaalia verkosta. Nämä ovat hyvin tärkeitä ominaisuuksia näitä toimintoja käyttäville ohjelmistokehittäjille. Artikkelit ja muut oppimismateriaalit tarjoavat pintakosketusta uuden oppimiseen tai kokeneemmille syvempääkin oppimista. Tarkka dokumentaatio antaa täyden kuvan teknologian ominaisuuksista, jotta sitä voidaan mahdollisimman tehokkaasti hyödyntää.

Työssä onnistuttiin luomaan toimeksiantajan sovellukselle tehokas tilanhallinta, jota on helppo jatkokehittää sovelluksen muiden ominaisuuksien kanssa. Koska kyseessä on sovelluksen MVP-versio, tehtyjen ratkaisujen yhteensopivuutta jatkokehitykseen on harkittu tarkasti. Ohjelmiston kehitystoiminnassa myös keskityttiin ohjelmiston kattavaan dokumentointiin, toimintatapoja kehitystiimin sisällä yhtenäistettiin ja sovelluksen kehitystä seurattiin sekä työnjakoa tehtiin päivittäisissä tapaamisissa.

LÄHTEET

Angular. N.d. Introduction to components and templates. Luettu 3.4.2020. https://angular.io/guide/architecture-components

Auth0. N.d. Overview. Luettu 26.4.2020. https://auth0.com/overview/

Basal, N. 2018. Akita State Management Tailored-Made for JS Applications. Luettu 2.3.2020. https://netbasal.gitbook.io/akita/

Basal, N. 2018. Introducing Akita: A New State Management Pattern for Angular Applications. Luettu 2.3.2020. https://netbasal.com/introducing-akita-a-new-state-management-pattern-for-angular-applications-f2f0fab5a8

Chargelab. N.d. Luettu 22.4.2020. https://www.chargelab.co/what-is-ocpp/

Fette, I., Melnikov, A. 2011. The WebSocket Protocol. Luettu 16.4.2020. https://tools.ietf.org/html/rfc6455#section-1.1

Medeiros, A. 2014. The introduction to Reactive Programming you've been missing. Luettu 30.3.2020. https://gist.github.com/staltz/868e7e9bc2a7b8c1f754

MDN. N.d. The WebSocket API. Luettu 16.4.2020. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

Reactive Extensions for JavaScript. N.d. Luettu 1.4.2020. https://github.com/Reactive-Extensions/RxJS

RxJS. N.d. Introduction. Luettu 1.4.2020. https://rxjs.dev/guide/overview

RxJS. N.d. Operators. Luettu 1.4.2020. https://rxjs.dev/guide/operators

RxJS. N.d. API. Luettu 1.4.2020. https://rxjs.dev/api

Socket.io. N.d. Luettu 16.4.2020. https://socket.io/

Socket.io. N.d. Overview. Luettu 16.4.2020. https://socket.io/docs/

Strongly-typed programming language. N.d. *The Free On-line Dictionary of Computing*. 2003. Luettu 19.4.2020. https://encyclopedia2.thefreedictionary.com/Strongly-typed+programming+language

Think with Google. 2016. The need for mobile speed: How mobile latency impacts publisher revenue. Luettu 1.4.2020. https://www.thinkwithgoogle.com/intl/en-154/insights-inspiration/research-data/need-mobile-speed-how-mobile-latency-impacts-publisher-revenue/

TypeScript. 2016. TypeScript Language Specification: Introduction. Luettu 15.4.2020. https://github.com/Microsoft/TypeScript/blob/master/doc/spec.md#1-introduction