Samu Rinne

Resursointityökalu Aurelia-ohjelmointikehystä käyttäen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

23.11.2022

Tiivistelmä

Tekijä:	Samu Rinne
Otsikko:	Resursointityökalu Aurelia-ohjelmointikehystä käyttäen
Sivumäärä:	48 sivua
Aika:	23.11.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Simo Silander CTO Sami Pärnänen

Insinöörityössä oli tarkoitus suunnitella ja ohjelmoida GSGroup Finland Oy:n kehittämään sekä ylläpitämään Paikannin.com-järjestelmään resursointityökalu. Paikannin.com on kaluston paikannukseen ja hallintaan luotu järjestelmä, jonka avulla sen käyttäjät voivat esimerkiksi optimoida laitteidensa käyttöä sekä ylläpitää sähköistä ajopäiväkirjaa. Työkalun tarkoitus on helpottaa järjestelmää käyttävien asiakkaiden laitteiden hallintaa sekä auttaa heitä optimoimaan kalustonsa käyttöä. Työn tavoitteena oli luoda työkalusta helppokäyttöinen sekä integroida se hyvin Paikannin.com-järjestelmään siten, että sen käyttäminen yhdessä muun järjestelmän kanssa on sujuvaa.

Työn käyttöliittymä toteutettiin Aurelia-ohjelmistokehystä käyttäen, joka on kokoelma ominaisuuspainotteisia JavaScript-moduuleita. Se on tarkoitettu käyttöliittymän ohjelmointiin. Ominaisuuden vaatimat palvelinmuutokset sekä REST-rajapinta ohjelmoitiin Javalla. Uudet tietokantataulukot luotiin käyttämällä PostgreSQL:ää.

Työ eteni ilman suuria ongelmia. Suurimmat työn ongelmat liittyivät valmiin JavaScript-kalenterikomponentin käyttämiseen Aurelian kanssa. Tämä johtui siitä, että Aurelia ei ole yleisesti käytetty ohjelmistokehys, joten suurimpaan osaan ongelmista ei löytynyt valmista vastausta eikä valittu kalenterikomponentti tukenut suoraan Aureliaa. Aurelia kuitenkin toimii hyvin JavaScript-komponenttien kanssa, joten ongelmiin löytyi helposti ratkaisu.

Insinöörityön aikana saatiin kehitettyä vaatimuksia vastaava resursointityökalu, joten työtä voidaan pitää sen suhteen onnistuneena. Lopulliset tulokset saadaan kuitenkin vasta, kun järjestelmää käyttävät asiakkaat ehtivät testata ominaisuutta ja heiltä saadaan palaute ominaisuuteen liittyen. Palautteen pohjalta ominaisuuteen tullaan varmasti tekemään muutoksia, jotta se tukee erilaisten yritysten vaatimuksia paremmin. Ominaisuuteen on jo nyt tiedossa joitain jatkokehityskohteita, kuten nyt kerätyn datan visualisointi muualla järjestelmässä.

Avainsanat: Ohjelmointi, Java, JavaScript, Aurelia, Kaluston paikannus

Abstract

Author:	Samu Rinne
Title:	Resourcing Tool Using Aurelia Framework
Number of Pages:	48 pages
Date:	23 November 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Simo Silander, Senior Lecturer Sami Pärnänen, CTO

The goal of the study was to design and implement a resourcing tool for Paikannin.com. Paikannin.com is a system that helps its users to locate and maintain their cars and work machines. It includes features that e.g. allow using an electric drive log and helps with fleet optimization. The goal of the resourcing tool is to help users to optimize their devices usage. The tool has to be easy to use, and it must be integrated to Paikannin.com so that it can be seamlessly used with its other features.

The user interface of the resourcing tool was programmed with Aurelia, a JavaScript framework. It is a collection of feature based modern JavaScript modules, and it is used to program user interfaces. Changes to the program server and REST API (representational state transfer application programming interface) were programmed with Java.

The resourcing tool meets all the set requirements. However, final outcome will only be seen after customers have tested the resourcing tool and given feedback about it. The resourcing tool will most likely be further developed to match different kinds of business area needs for it. Also, there is already some plans to further develop it, e.g. visualize collected data within other features of Paikannin.com

Keywords: Programming, Java, JavaScript, Aurelia, Fleet management

Sisällys

Lyhenteet

1 Johdanto 1

2 Paikannin.com 1

2.1 PPCT Instant 2
2.2 Uusi järjestelmä 5

3 Vaatimukset resursointityökalulle 11

3.1 Varauskalenteri 12
3.2 Integrointi Paikannin.com-järjestelmään 15

4 Teknologiat ja työkalut 17

4.1 Järjestelmän suoritusympäristö 17
4.2 Järjestelmän kehittäminen 21

5 Aurelia 22

5.1 Käyttöönotto 22
5.2 Aurelian perusteet 25
5.3 Vertailu muihin ohjelmointikehyksiin 31

6 Resursointityökalun toteutus 35

6.1 Suunnittelu 35
6.2 Toteutus 37
6.3 Testaus ja jatkokehitys 44

7 Yhteenveto 45

Lähteet 47

Lyhenteet

REST:	*Representational state transfer*. Arkkitehtuurityyli ohjelmointirajapintojen toteuttamiseen.
API:	*Application Programming Interface.* Ohjelmointirajapinta, jonka avulla sovellukset keskustelevat keskenään.
ECMAScript:	JavaScript-standardi, jonka tarkoituksena on varmistaa internetsivujen toiminta eri selaimilla.
Npm:	*Node Package Manager.* Maailman suurin sovellusrekisteri, jonka avulla voidaan esimerkiksi hallinnoida projektien riippuvuuksia.
HTML:	*Hypertext Markup Language.* Avoimesti standardoitu merkkikieli, joka on suunniteltu internetsivujen esittämiseen.
CSS:	*Cascading Style Sheets.* Internetsivujen tyylin määrittämiseen käytetty koodikieli.
SQL:	*Structured Query Language.* Standardoitu kyselykieli, jota käytetään relaatiotietokantojen kyselyissä.
JUnit:	Javassa käytössä oleva ohjelmistokehys, jota käytetään yksikkötestien tekemiseen.

1 Johdanto

Tämä insinöörityö on tehty GSGroup Finland Oy:lle, joka on osa norjalaista GSGroup-konsernia. GSGroup Finland kehittää sekä ylläpitää Paikannin.com-järjestelmää, jonka on tarkoitus sekä helpottaa että tehostaa ajoneuvojen sekä kaluston paikantamista ja hallintaa. Sen kautta on esimerkiksi mahdollista tarkastella kalustonsa reaaliaikaista sijaintia tai tarkastella ajoneuvoilla ajettuja reittejä yksityiskohtaisemmin.

Insinöörityössä toteutettiin Paikannin.com-järjestelmään resursointityökalu helpottamaan järjestelmää käyttävien asiakkaiden töiden suunnittelua. Ennen työssä tehtävää ominaisuutta järjestelmässä ei pystynyt varaamaan laitteita etukäteen eikä tarkastelemaan laitteiden varaustilannetta. Uuden ominaisuuden on tarkoitus mahdollistaa nämä seikat ja näin tehostaa järjestelmää käyttävien asiakkaiden työntekoa.

Tässä työssä käsitellään kehitystyön eri vaiheita suunnittelusta testaamiseen sekä julkaisuun asti. Työssä käydään myös läpi Paikannin.com-järjestelmän pääpiirteitä.

Lisäksi työssä käsitellään Aurelia-ohjelmointikehystä. Se on kokoelma ominaisuuspainotteisia JavaScript-moduuleita ja sitä käytetään käyttöliittymän ohjelmointiin. Työssä käsitellään Aurelian käyttöä sekä vertaillaan sitä muihin suosittuihin käyttöliittymän ohjelmointiin käytettäviin ohjelmointikehyksiin.

2 Paikannin.com

Paikannin.com on suomalainen ajoneuvojen sekä kaluston GPS-paikantamiseen tarkoitettu järjestelmä. Sen on kehittänyt suomalainen GSGroup Finland. Yritys tunnettiin ennen nimellä Pirkanmaan PC-Tuotteet Oy (PPCT), mutta norjalainen GSGroup osti yrityksen vuonna 2016. Järjestelmää alettiin kehittämään alun perin nimellä PPCT Instant.

Järjestelmän toiminta perustuu ajoneuvoihin sekä työkoneisiin asennettaviin GPS-paikantimiin ja niistä kerättävään dataan. Paikantimia on monenlaisia, ja ne soveltuvat erilaisiin tarkoituksiin. Paikannin voidaan kytkeä kiinni esimerkiksi suoraan auton akkuun, OBD-porttiin, tupakansytyttimeen tai muuten suoraan auton virtoihin. Laitemallista ja kytkennästä riippuen laite saattaa lähettää paikkatietodatan lisäksi muuta dataa. Laite saattaa esimerkiksi antaa tiedon siitä, milloin aura-auton aura tai hiekoitustoiminto on käytössä.

Paikannin.com-järjestelmän käyttäjiä ovat esimerkiksi monet Suomen kaupungit ja kunnat. Heillä saattaa olla esimerkiksi kadunhuoltoon liittyviä velvoitteita, jotka he pystyvät osoittamaan suoritetuksi järjestelmän kautta. Tämän lisäksi käyttäjiä on paljon esimerkiksi kiinteistönhuollon, ajoneuvo- ja työkonevuokraamoiden sekä kuljetuspalveluiden aloilta. Erilaisilla aloilla työnkuva saattaa vaihdella paljon, joten on tärkeää, että järjestelmä soveltuu jokaisen työnkuvan erilaisiin tarpeisiin.

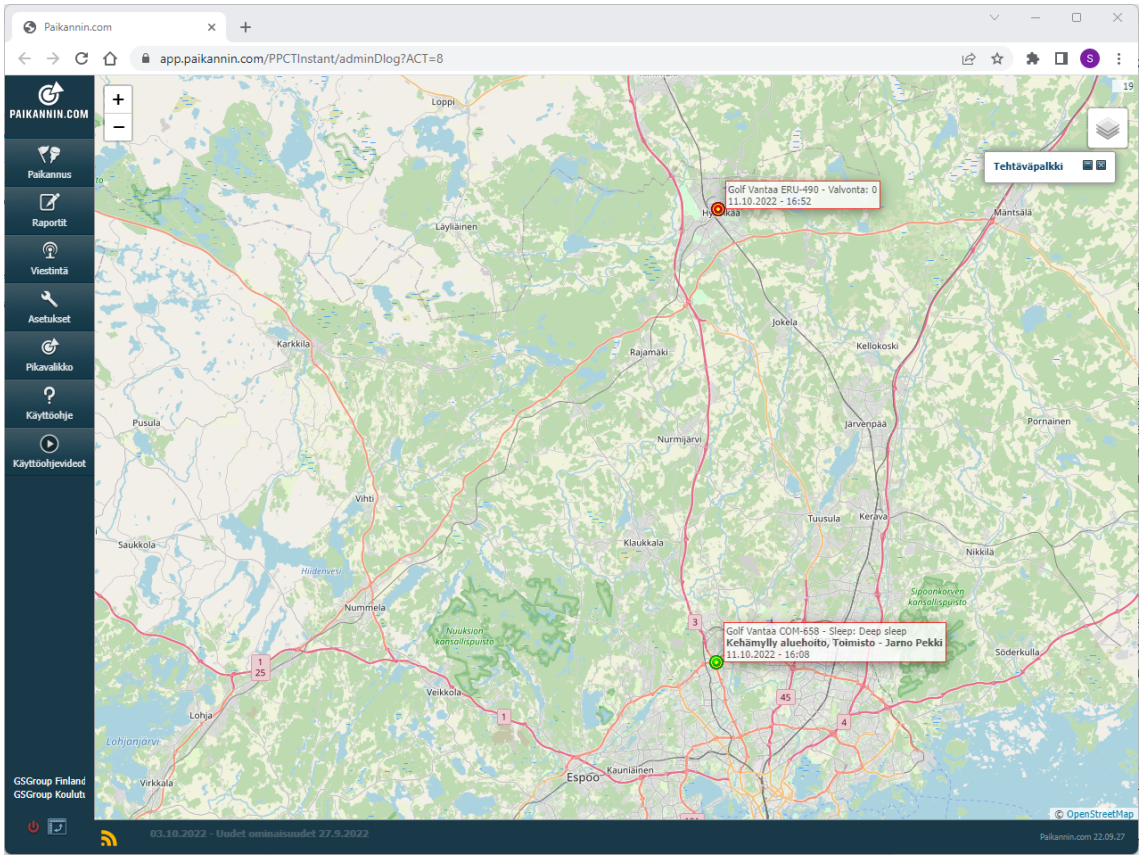
2.1 PPCT Instant

PPCT Instant -järjestelmää alettiin kehittämään vuonna 2005. Siihen aikaan Suomessa ei ollut vastaavaa kaluston GPS-paikannukseen perustuvaa järjestelmää. Järjestelmän ensimmäinen versio julkaistiin vuonna 2006 ja sitä kehitettiin aktiivisesti vuoteen 2016 asti. [1, s. 89.]

Vaikka järjestelmästä on nykyään tehty uusi versio, PPCT Instant on edelleen joillain GSGroup Finlandin asiakkailla käytössä. Tämän vuoksi järjestelmää ylläpidetään edelleen, joskin siihen tehdyt päivitykset ovat harvinaisia. Vaikka PPCT Instant eroaa huomattavasti sekä ulkoasullisesti että toiminnollisesti uudesta Paikannin.com-järjestelmästä, niiden pohjalla käytettävä data ei eroa toisistaan, mikä on mahdollistanut vanhan järjestelmän tukemisen.

Järjestelmä sisältää suurimman osan Paikannin.com-järjestelmän tärkeimmistä ominaisuuksista. Näitä ovat esimerkiksi ajoneuvojen reaaliaikainen tarkastelu kartalla, laitteiden ryhmittely, nimetyt paikat, paikkaryhmät sekä lukuisat erilaiset raportit.

Kuvassa 1 on esitetty vanhan käyttöliittymän etusivu. Etusivulla näkyy kartta, jonka kautta voidaan tarkastella laitteiden liikkeitä reaaliajassa. Siitä näkee myös, että järjestelmän ulkoasu on nykypäivän standardien mukaan jo vanhentunut. Tämän näkee esimerkiksi siitä, että laitteet ovat merkittyinä pelkillä ympyröillä kartassa.



Kuva 1. Vanhan käyttöliittymän etusivu.

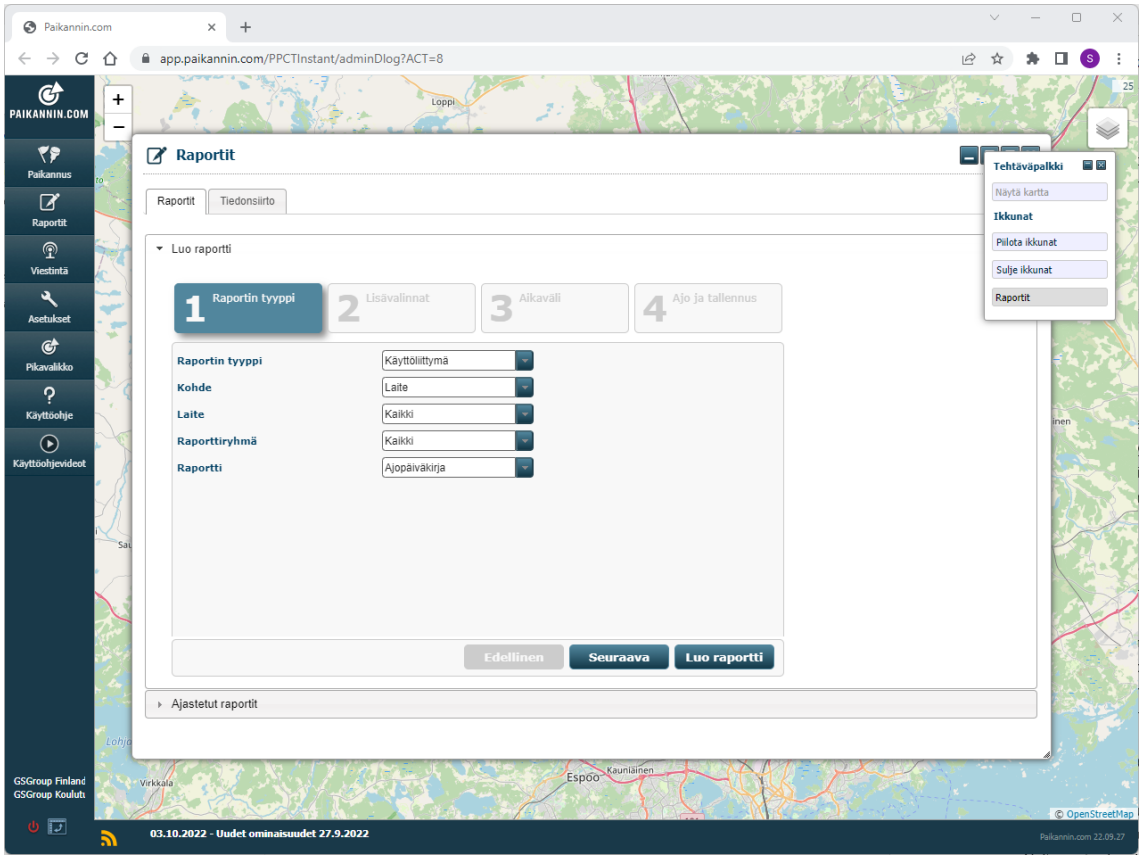
Laitteita pystyy esimerkiksi ryhmittelemään omiin ryhmiin, minkä avulla järjestelmässä pystyy määrittelemään laitteiden paikannusoikeuksia käyttäjäkohtaisesti. Tämä on tietoturvan kannalta tärkeä ominaisuus, sillä yritykset eivät halua, että kaikki heidän työntekijänsä pystyvät paikantamaan esimerkiksi kaikkia autoja. Paikannusoikeuksia laitteeseen voidaan antaa myös laitekohtaisesti, joten laiteryhmiä ei ole pakko käyttää.

Nimetyt paikat puolestaan ovat kartalle luotavia alueita, jotka voivat olla esimerkiksi yrityksen toimipisteitä tai työkohteita. Paikkaryhmillä taas voidaan

ryhmitellä nimetyt paikat halutulla tavalla. Näiden avulla järjestelmässä voidaan esimerkiksi ajaa raportti, joka analysoi ryhmien tai paikkojen välisiä reittejä sekä niissä vietettyä aikaa. Muita järjestelmästä löytyviä tärkeitä raportteja ovat esimerkiksi sähköinen ajopäiväkirja, päästöraportti sekä ajoneuvojen käyttöastetta laskeva raportti.

Järjestelmässä on myös joitain ominaisuuksia, joita ei ainakaan vielä ole tuotu uuteen Paikannin.com-järjestelmään. Yksi merkittävimmistä ominaisuuksista on viestien lähettäminen laitteille auton navigaattorin kautta. Sen avulla työnjohto pystyy esimerkiksi lähettämään uuden työkeikan tiedot haluamalleen autolle. Viestiin on myös mahdollista sisällyttää työkohteen koordinaatit, jolloin auton kuljettaja voi helposti aloittaa navigoinnin uuteen työkohteeseen. Ominaisuutta ei ole tuotu uuteen järjestelmään siksi, että sen käyttö on ollut vähäistä vaadittuun työmäärään nähden. Toinen merkittävä uudesta järjestelmästä puuttuva ominaisuus on reititystoiminto. Vanhasta järjestelmästä löytyvä reititystoiminto mahdollistaa nopeimman reitin hakemisen pisteiden välille.

Monet PPCT Instant -järjestelmään tehdyt ohjelmakomponentit ovat edelleen käytössä Paikannin.com-järjestelmässä. Näitä ovat esimerkiksi raportointiin käytettävät moduulit, joita lisätään ja ylläpidetään tarvittaessa. Vanhan käyttöliittymän raporttityökalu on nähtävissä kuvassa 2. Myös tässä kuvassa on havaittavissa järjestelmän vanhanaikaisuus. Kuvasta käy myös hyvin ilmi vanhan järjestelmän rakenne. Siinä kartta on aina auki, ja sivuvalikosta valittavat toiminnot avautuvat uutena ikkunana kartan päälle. Uudet ikkunat sisältävät useimmiten muutaman välilehden ja jokaisella välilehdellä on aiheeseen liittyviä toimintoja.

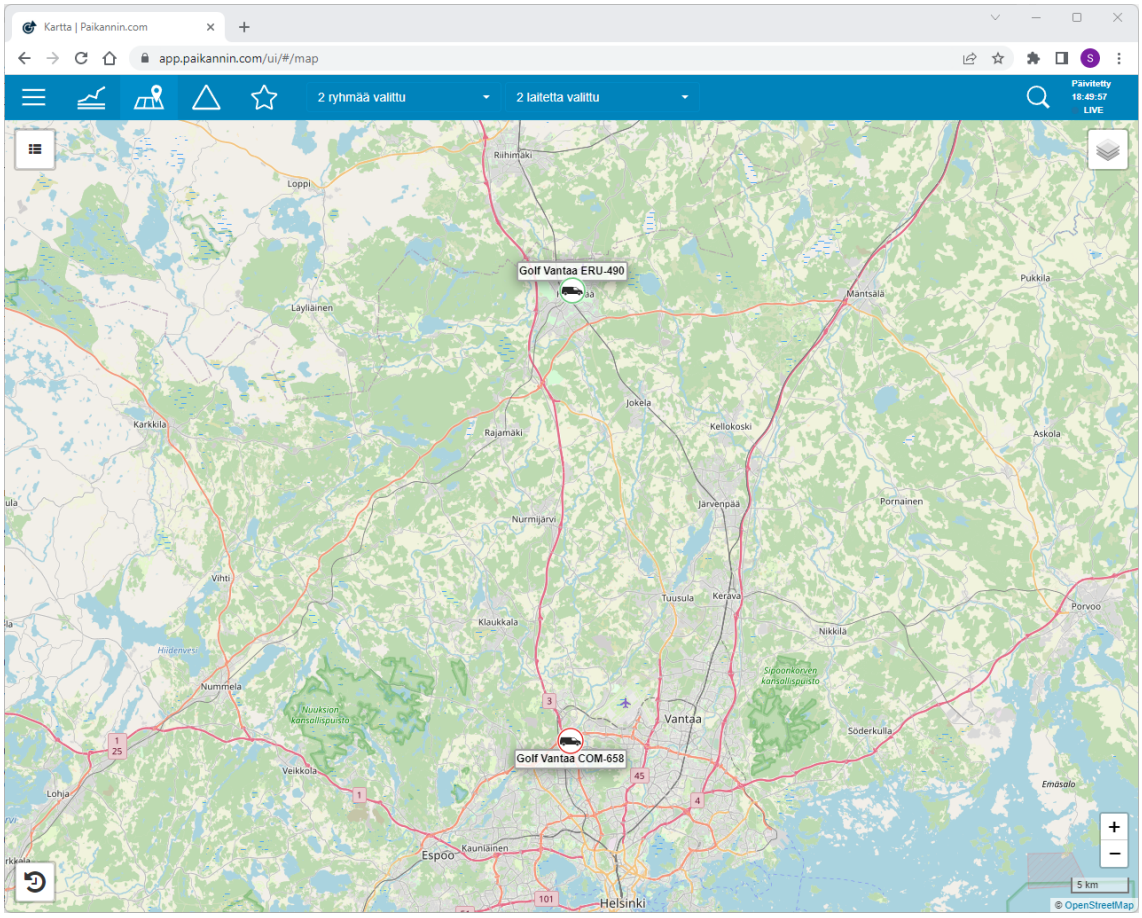


Kuva 2. Vanhan käyttöliittymän raportointityökalu.

Tämän lisäksi kaikki järjestelmän asiakkuuksiin liittyvät työt tehdään edelleen vanhalla järjestelmällä. Näitä ovat esimerkiksi uuden asiakkuuden luonti järjestelmään sekä laitteiden lisääminen järjestelmään.

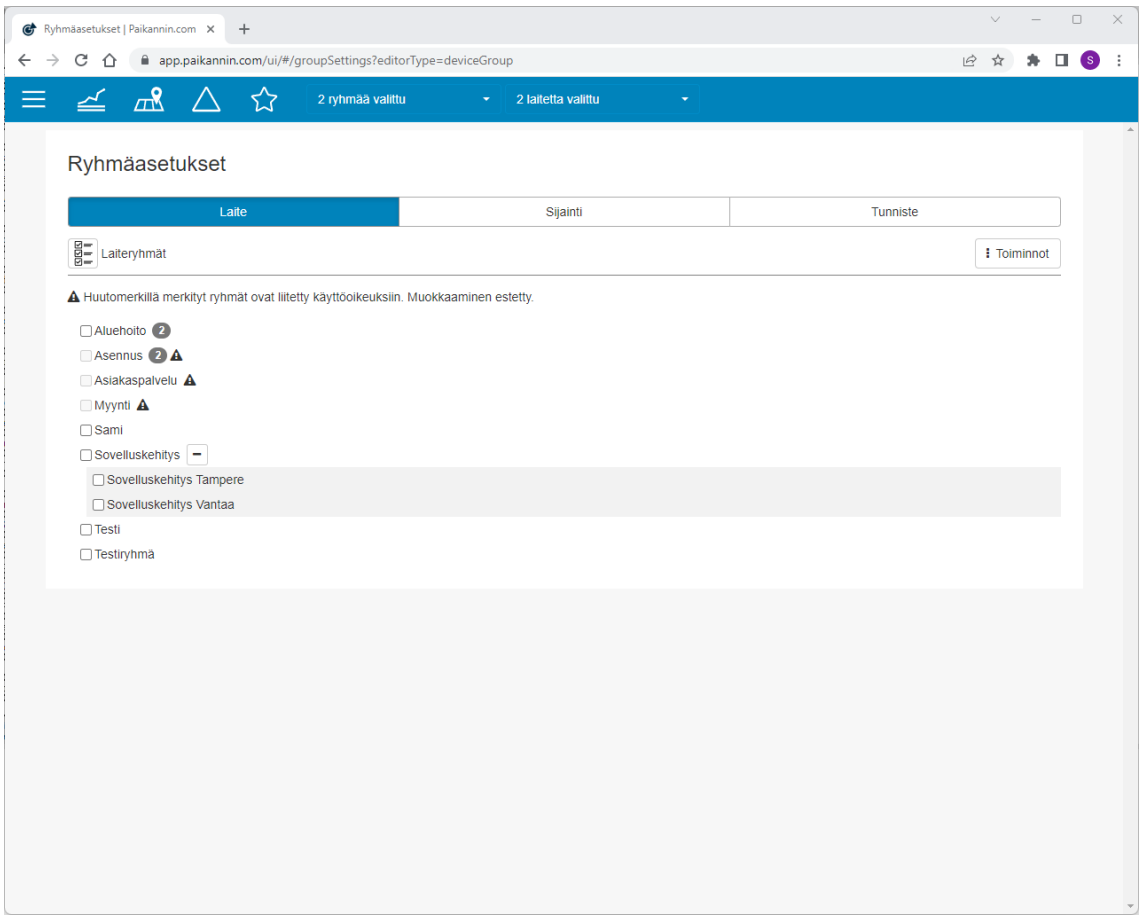
2.2 Uusi järjestelmä

Nykyistä Paikannin.com-järjestelmää alettiin kehittämään vuonna 2015 ja siitä julkaistiin ensimmäinen versio vuoden 2016 lopussa. Suurin yksittäinen syy uudelle järjestelmälle oli vanhan järjestelmän ulkoasun vanhanaikaisuus. Uusi käyttöliittymä myös mahdollisti modernimpien teknologien käyttöönoton. Uusi käyttöliittymä päätettiin ohjelmoida Aurelia-ohjelmointikehystä käyttäen. Se on huomattavasti modernimman näköinen kuin vanha käyttöliittymä. Vertailukohtana voidaan käyttää esimerkiksi uuden järjestelmän karttanäkymää, joka on nähtävissä kuvassa 3. Laitteiden paikalla siinä on vaihdettavia kuvakkeita, ja tyyli on muutenkin siistimmän näköinen.



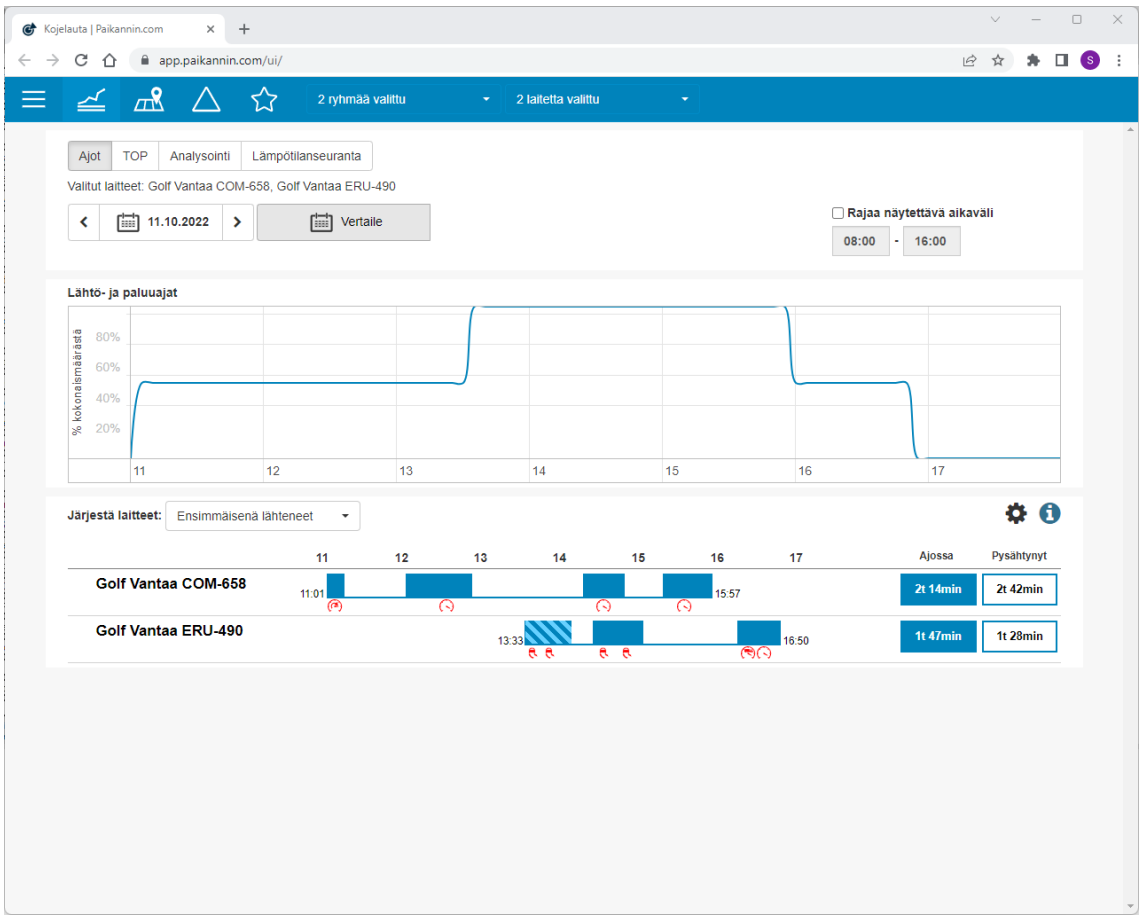
Kuva 3. Uuden järjestelmän karttanäkymä.

Järjestelmään on tuotu kaikki PPCT Instant -järjestelmän tärkeimmät ominaisuudet. Osa ominaisuuksista on pysynyt toiminnaltaan lähes samanlaisena, mutta joihinkin on tehty vuosien saatossa merkittäviä parannuksia. Esimerkiksi vanhasta järjestelmästä tuttu laitteiden ryhmittely päätettiin toteuttaa hyödyntämällä puurakennetta (kuva 4). Siinä laiteryhmillä voi olla alaryhmiä, joiden avulla käyttöoikeuksien hallinnointi helpottuu. Esimerkiksi yrityksen laitteet pystytään jakamaan osastoittain, jolloin osastojen johtajille voidaan antaa paikannusoikeudet oman osaston laitteisiin. Osaston alle voidaan puolestaan luoda uusia ryhmiä, vaikka työkonetyyppien perusteella, jolloin työkoneista vastaaville henkilöille voidaan antaa ryhmiin tarvittavat oikeudet. Näin yrityksen ylin johto voi halutessaan tarkastella kaikkia yrityksen laitteita, osastojen johtajat voivat nähdä vain omaan osastoonsa kuuluvat laitteet ja muilla työntekijöillä on vielä rajatummat oikeudet.



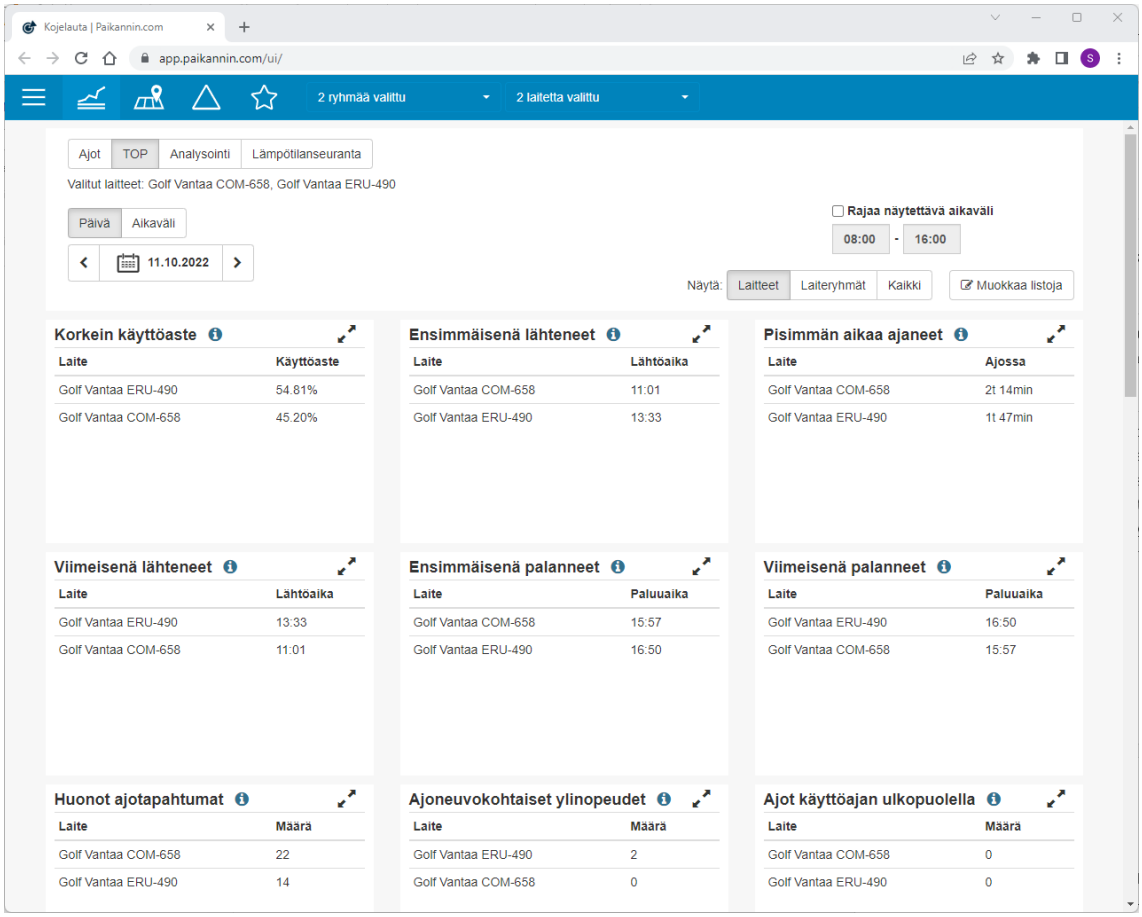
Kuva 4. Laiteryhmien puunäkymä.

Vanhasta järjestelmästä tuotujen ominaisuuksien lisäksi uuteen järjestelmään on myös toteutettu monia uusia ominaisuuksia. Yksi näistä on kojelautanäkymä (kuva 5), jossa pystyy yhdellä silmäyksellä seuraamaan laitteiden ajoja päivän aikana. Se toimii myös käyttöliittymän etusivuna.



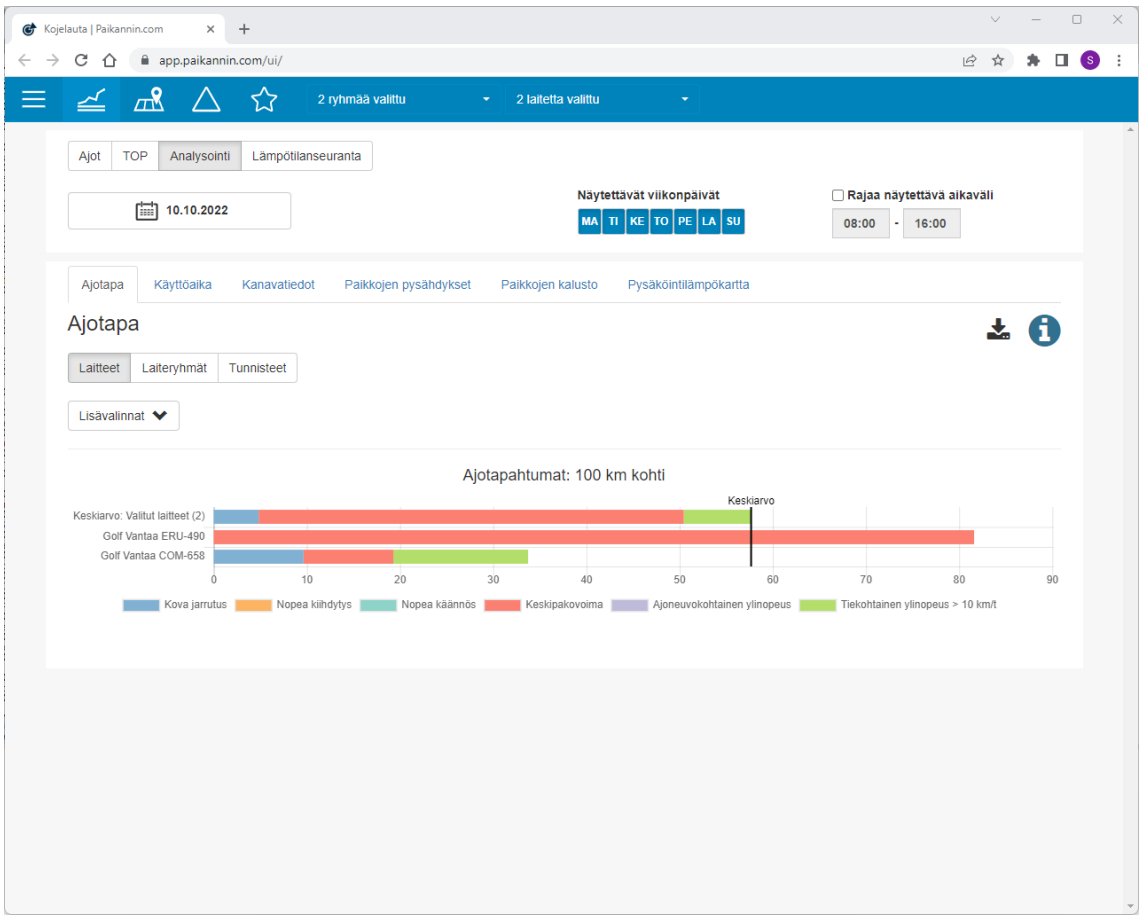
Kuva 5. Paikannin.com-kojelauta.

Uudessa järjestelmässä pystyy myös esimerkiksi seuraamaan laitteiden käyttöaikaa sekä käyttöastetta suoraan käyttöliittymän kautta. Käyttöastetta pystyy seuraamaan järjestelmän kautta useilla eri tavoilla. Valittujen laitteiden tai laiteryhmien keskimääräistä käyttöastetta ajan suhteen pystyy tarkastelemaan suoraan kojelaudalta löytyvästä kuvaajasta, joka on nähtävissä kuvassa 5. Siinä on esitetty käytössä olevien laitteiden prosentuaalinen osuus päivän mittaan. Toinen vaihtoehto tarkastella laitteiden käyttöastetta on tarkastella sitä varten suunniteltua Top-listaa (kuva 6). Top-listoja löytyy järjestelmästä useita erilaisia, ja niissä näytettävän datan aikaväli on täysin käyttäjän itse valittavissa. Näiden ominaisuuksien avulla Paikannin.com-järjestelmää käyttävien yritysten on mahdollista tehostaa toimintaansa karsimalla ylimääräisiä laitteita pois.



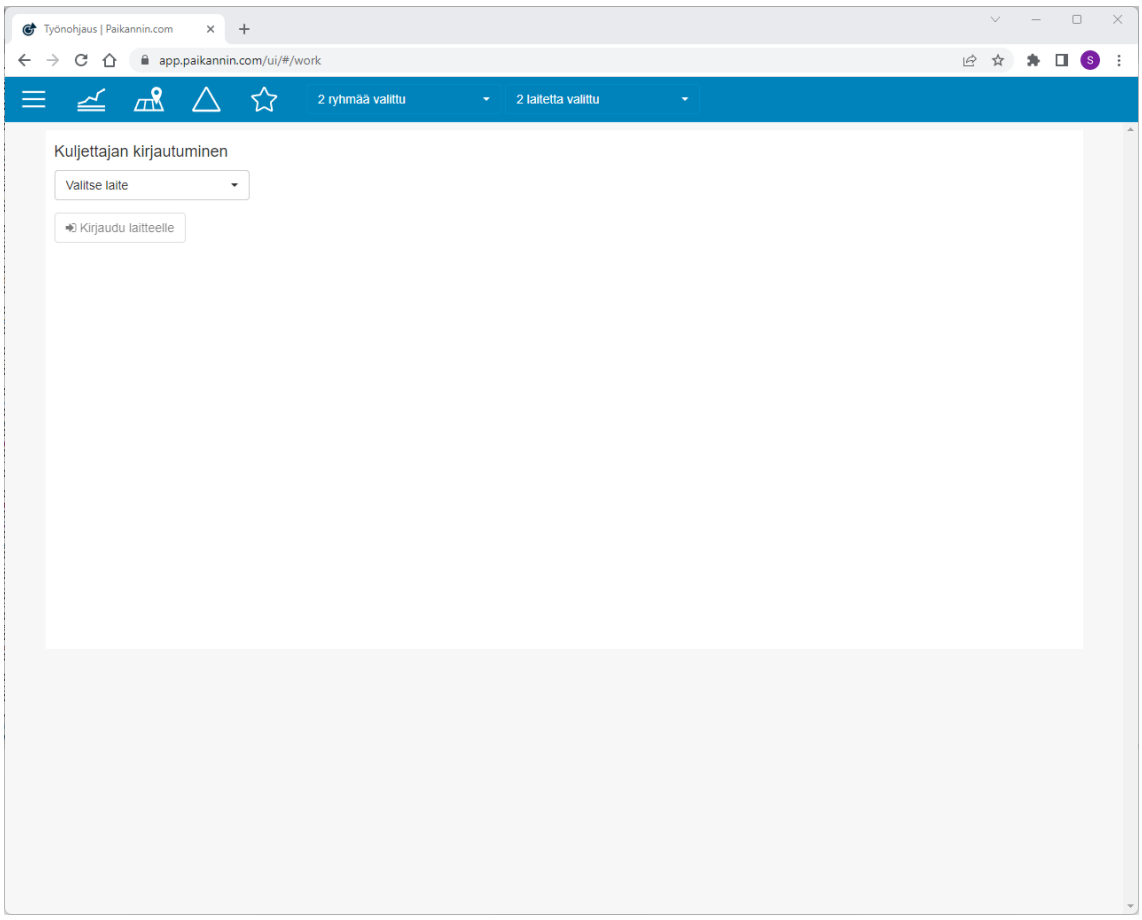
Kuva 6. Paikannin.comin top-listat.

Uuden järjestelmän avulla on myös mahdollista seurata laitteilla ajettujen matkojen ajotapaa. Ajotapa-analyysin avulla pystytään esimerkiksi puuttumaan ylinopeuksiin, liian voimakkaisiin kiihtyvyyksiin sekä äkkijarrutuksiin. Ajotapa-analyysi on nähtävissä kuvassa 7. Tämän ominaisuuden avulla järjestelmän käyttäjät voivat olla varmoja siitä, että heidän kuljettajansa noudattavat yhteisiä sääntöjä ajotavan suhteen. Ajotapa-analyysin saa halutessaan esittämään datan absoluuttisina määrinä tai suhteutettuna ajettuihin kilometreihin.



Kuva 7. Ajotapa-analyysi.

Järjestelmä mahdollistaa myös kuljettajan kirjautumisen laitteelle (kuva 8). Sen avulla järjestelmän käyttäjät voivat ominaisuuden nimen mukaisesti kirjautua laitteille, joihin hänellä on oikeus kirjautua. Ominaisuus mahdollistaa myös erilaisen ajotavan valinnan, jonka avulla käyttäjä voi määrittää ajon olevan työajoa tai omaa ajoa. Ominaisuus on joillekin asiakkaistamme tärkeä, sillä jos autoa käytetään työmatkojen lisäksi omiin ajoihin, ne pitää pystyä erottelemaan toisistaan.



Kuva 8. Kuljettajan kirjautuminen.

Kuljettajan kirjautuminen mahdollistaa myös erilaisten työsignaalien käyttämisen järjestelmän kautta. Esimerkkejä signaaleista voisi olla auraus ja hiekoitus. Laitteelta löytyvät signaalit voidaan laittaa järjestelmän kautta päälle, jolloin järjestelmä rekisteröi, että kyseisenä aikana on esimerkiksi aurattu.

3 Vaatimukset resursointityökalulle

Resursointityökalun on tarkoitus helpottaa järjestelmää käyttäviä asiakkaita tarkastelemaan heiden laitteidensa käyttöä sekä antaa heidän työntekijöidensä tarkastella työtehtäviään. Laitteiden varaustilanteen tarkastelulla asiakkaat pystyvät optimoimaan omien laitteidensa käyttöä.

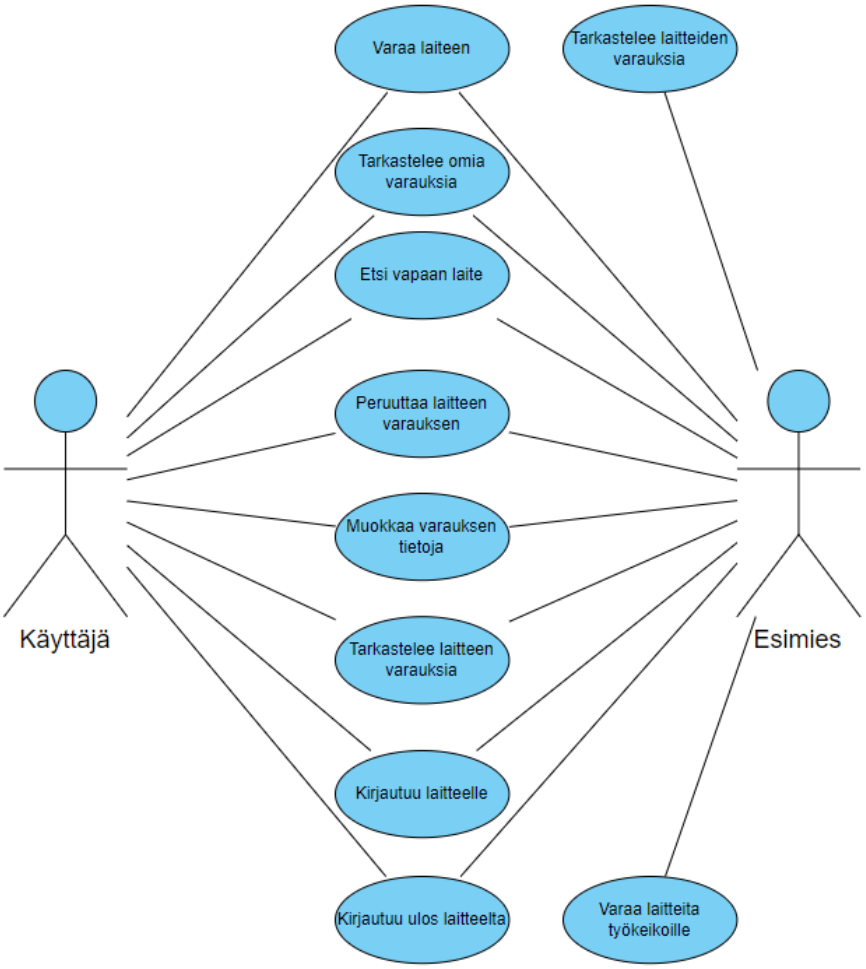
Jo ennen uuden resursointityökalun suunnitteluvaihetta pohdimme sekä ohjelmistokehittäjien että johtoryhmän kanssa, mitä vaatimuksia uudella

ominaisuudella on. Nämä pohdinnat työkalun vaatimuksista olivat tärkeitä, sillä näin saimme monenlaisia näkökulmia resursointityökalun käyttöön liittyen. Kaikilla on kuitenkin erilaiset työnkuvat ja kokemukset. Tämän vuoksi jokainen osaa tuoda esiin erilaisia vaatimuksia, joita he tarvitsisivat omien päivittäisten työtehtävien organisoinnissa.

Usein isoimpien uusien ominaisuuksien kanssa projektissa on mukana myös jokin asiakkaamme. Asiakkaalta saamme uusista ominaisuuksista nopeasti palautetta, jolloin pystymme vastaamaan heidän tarpeisiinsa nopeammin. Tässä tilanteessa emme kuitenkaan päätyneet ottamaan asiakasta mukaan projektiin, sillä ominaisuuden tulee toimia hyvin kaikenlaisilla asiakkailla. Jos ottaisimme projektiin mukaan asiakkaan, jonka työtehtävät liittyvät esimerkiksi logistiikkaan, heidän vaatimuksensa ominaisuuteen eroaisivat huomattavasti konevuokraamoiden vaatimuksista. Tämän vuoksi pyrimme tuomaan asiakkaillemme hyvin toimivan resursointityökalun, jota voimme tarvittaessa jatkokehittää vastaamaan eri alojen tarpeita.

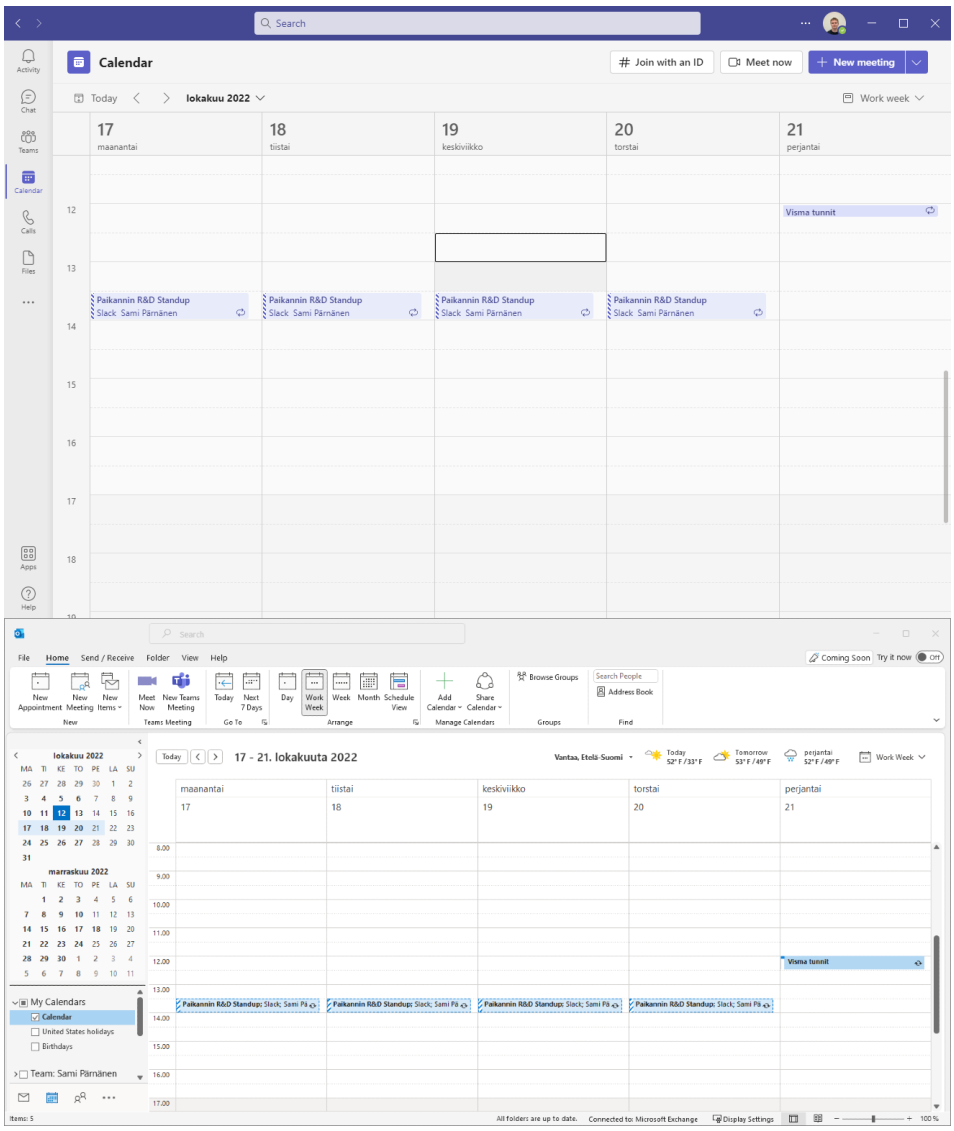
3.1 Varauskalenteri

Resusointityökalun tarkoituksena on mahdollistaa käyttäjille laitteiden varaustilanteen tarkastelun sekä uusien varauksien luominen. Työkalun tärkeimmät käyttötarkoitukset ovat näkyvissä kuvan 9 käyttötapauskaaviossa. Kuten käyttötapauskaaviosta näkee, suurin osa käyttötapauksista liittyy suoraan tai välillisesti varauskalenteriin. Esimerkiksi varauskalenterin kautta kuljettaja voi etsiä itselleen vapaan laitteen sekä muokata itselleen merkittyjen varausten tietoja. Työnjohtajat puolestaan voivat tarkastella laitteiden varauksia laajemmin sekä varata laitteita heidän alaistensa käyttöön. Tämän vuoksi varauskalenteri on käyttäjän näkökulmasta ominaisuuden tärkein osuus. Onkin siis luonnollista, että varauskalenterin ulkoasuun sekä toimintaan liittyy lukuisa vaatimuksia.



Kuva 9. Resursointityökalun käyttötapauskaavio.

Resursointityökalun pohjana toimivan kalenterin tulee ensinnäkin olla helppokäyttöinen. Ihmiset ovat tottuneet käyttämään erilaisia elektronisia kalentereita sekä töissä että vapaa-ajalla. Monella on töissä käytössä esimerkiksi kalenteri suoraan Outlookin tai Teamsin kautta. Tämä asettaa tiettyjä vaatimuksia myös resursointityökalun kalenterille. Esimerkiksi molempien kalenterien ulkoasu muistuttaa paljon toisiaan. Kuten kuvasta 10 näkee, merkityt tapahtumat näkyvät samanlaisina palkkeina sekä Teamsin että Outlookin kalentereissa. Muutenkin elementtien sijoittelu on suunnilleen samanlaista.



Kuva 10. Teamsin kalenterinäkymä yläpuolella ja Outlookin kalenterinäkymä alapuolella.

Myös kalentereiden toiminnot ovat pääpiirteittäin samanlaisia. Molemmissa voidaan siirtyä päivien, viikkojen ja kuukausien välillä helposti käyttöliittymästä löytyvien nuolien avulla. Molemmissa voidaan painaa kalenterista tiettyä päivämäärää tai tuntia ja näin luoda kalenteriin uusi tapahtuma. Toki tapahtumien lisäämisessä on myös pieniä eroavaisuuksia: Teamsin kautta varaus on tehtävissä yhdellä painalluksessa, kun taas Outlookin kalenterissa vaaditaan kaksoispainallusta tai hiiren oikeaa painiketta. Lisäksi molemmissa kalentereissa on käytössä kaikille tutut päivä-, viikko- ja kuukausinäkymät.

Koska suosituimmat elektroniset kalenterit muistuttavat toisiaan sekä ulkoasullisesti että toiminnallisesti, on tärkeää, että myös resursointityökalun kalenteri muistuttaa näitä. Kalenterinäkymä on kuitenkin hyvin vakiintunut, joten suurin osa ihmisistä osaa käyttää sitä ainakin kohtalaisen hyvin. Jos kalenterin toiminnallisuus tai ulkoasu eroaisi radikaalisti jo totutuista malleista, käyttäjät joutuisivat tarpeettomasti opettelemaan uuden kalenterin käyttöä.

Varauskalenterissa tulee myös pystyä siirtämään varauksia vetämällä niitä toisiin ajankohtiin. Myös tämä on nykyajan elektronisissa kalentereissa hyvin tavallista. Tämän lisäksi kalenteriin tulee myös pystyä lisäämään tapahtumapohjia, joiden avulla voidaan kalenteriin luoda helposti uusia usein toistuvia tapahtumia. Jollain käyttäjällä saattaa esimerkiksi olla tunnin mittaisia siivousurakoita, joita hänen täytyy pystyä luomaan kalenteriin helposti.

Paikannin.com-järjestelmän käyttäjistä osa käyttää järjestelmää lähinnä mobiililaitteilla, joten myös heidän tarpeensa tulee ottaa huomioon. Mobiililaitteiden näytöt ovat usein pieniä, joten varauskalenterin käyttöliittymän tulee skaalautua hyvin myös pienille näytöille. Tämän lisäksi mobiililaitteita käytetään lähes poikkeuksetta kosketusnäytön avulla. Tämän vuoksi käyttöliittymäkomponenttien tulee olla tarpeeksi isoja, jotta niitä on helppo käyttää myös mobiilisti.

Vaikka varauskalenterin tulee toimia hyvin mobiililaitteilla, sen käytön ei tarvitse olla identtistä työpöytäkäytön kanssa. Kalenterin käyttö mobiililaitteilla tulee todennäköisesti olemaan lähinnä yksittäisten käyttäjien omien varausten tarkastelua sekä muokkaamista, jolloin kaikki työnjohtajan käyttämät ominaisuudet eivät välttämättä ole mobiilikäyttöliittymässä pakollisia. Mobiililaitteella ruudun koko on rajallinen, ja vähemmän käytettyjen ominaisuuksien pois jättäminen sekä selkeyttää että helpottaa mobiilikäyttöä.

3.2 Integrointi Paikannin.com-järjestelmään

Kalenterin ulkoasuun ja toimintaan liittyvien vaatimusten lisäksi resursointityökaluun liittyy myös paljon vaatimuksia järjestelmän suhteen. Paikannin.com on monimutkainen järjestelmä ja uuden ominaisuuden tulee toimia järjestelmän

kanssa saumattomasti. Esimerkiksi siirtyminen resursointityökalun sekä järjestelmän muiden ominaisuuksien välillä tulee olla sujuvaa.

Resursointityökalu tulee toteuttaa siten, että yksittäiset käyttäjät pystyvät seuraamaan omaa kalenteriaan ja tarkastelemaan heille varattuja laitteita. Käyttäjän tulee myös pystyä varaamaan laite omaan käyttöön, mikäli käyttäjällä on oikeus käyttää laitetta. Käyttäjän tulee myös pystyä tarkastelemaan niiden laitteiden varauksia, joihin hänellä on käyttöoikeus, jotta hän voi tietää, milloin laite on vapaana. Tietosuojan takia käyttäjä saa kuitenkin nähdä muiden tekemistä varauksista vain välttämättömät tiedot, kuten varauksen aloitus- ja lopetusajat.

Järjestelmää käyttävän yrityksen määrittelemille työnjohtajille ominaisuuden vaatimukset ovat hieman erilaiset. He voivat toki käyttää järjestelmää kaikilla samoilla tavoilla kuin normaalit käyttäjät, mutta tämän lisäksi heillä tulee olla oikeus tarkastella käyttöoikeuden puitteissa kaikkien varausten tarkempia tietoja sekä varaamaan laitteita toisille käyttäjille.

Resursointityökalun on myös tarkoitus toimia hyvin käyttöliittymän kautta tehtävien kuljettajan kirjautumisten kanssa. Järjestelmä mahdollistaa jo ennestään kuljettajan kirjautumisen laitteelle, jota asiakkaat käyttävät esimerkiksi yhteiskäyttöisten autojen kuljettajan tunnistautumiseen. Sen avulla on mahdollista osoittaa, kuka on milloinkin ajanut kyseistä laitetta. Tämä ominaisuus on tarkoitus yhdistää resursointityökaluun siten, että käyttäjä voi halutessaan jättää laitteen varaamatta ja kirjautua suoraan vapaalle laitteelle. Kuljettajan kirjautuessa laitteelle kalenteriin tulee automaattisesti uusi varaus, jossa näkyvät esimerkiksi kirjautuneen kuljettajan sekä laitteen tiedot. Uusi varaus kestää joko ennalta määrätyn ajan tai seuraavan varauksen alkuun asti. Keston määrittelyyn vaikuttaa laitteen varaustilanne. Toisaalta jos käyttäjä onkin suoriutunut tehtävästään ennakoitua nopeammin ja hän kirjautuu ulos laitteelta, tulee nykyinen varaus katkaista, jotta työnjohto näkee laitteen olevan taas käytössä. Tämän lisäksi kuljettajan kirjautumisen tulee ilmoittaa, jos kirjauduttavalla laitteella on kohta alkamassa jonkun toisen käyttäjän varaus. Tämän avulla laitteelle kirjautuja ei tietämättään pysty ottamaan käyttöön toiselle varattua laitetta.

Kuljettajan kirjautumisen yhdistäminen resursointityökaluun tulee myös olla asiakaskohtaista. Järjestelmää käyttävien asiakkaiden tarpeet voivat olla hyvin erilaiset ja jotkut eivät välttämättä halua kirjauksista muodostuneita varauksia käyttöön.

Järjestelmä mahdollistaa laitteelle kirjautumisen myös muilla tavoilla kuin käyttöliittymän kautta. Esimerkiksi autossa sijaitsevan fyysisen laitteen kautta voidaan kirjautua laitteelle erilaisten tunnisteiden avulla, mutta sen yhdistäminen resursointityökaluun jätetään ainakin toistaiseksi toteuttamatta.

Resursointityökalun tulee myös tukea Paikannin.com-järjestelmän raportointia. Työkalun keräämän datan avulla tulee pystyä tekemään erilaisia raportteja esimerkiksi laitteiden varausasteen sekä varausten ja todellisen käyttöajan suhteesta. Työkalun ensimmäisessä vaiheessa raportoinnin suhteen riittää yksinkertainen näkymä tai raportti, jonka avulla pystyy helposti hakemaan ja tarkastelemaan tiettyyn aikaan vapaana olevia laitteita. Raportteja pystytään helposti lisäämään myös myöhemmissä vaiheissa. Esimerkki myöhemmin luotavasta raportista voisi olla esimerkiksi kalenterin merkintöihin perustuva käyttöasteraportti.

4 Teknologiat ja työkalut

Paikannin.com-järjestelmään ylläpidetään sekä kehitetään käyttäen monia erilaisia työkaluja ja teknologioita. Näitä kaikkia on joko suoraan tai välillisesti käytetty myös resursointityökalun kehityksessä.

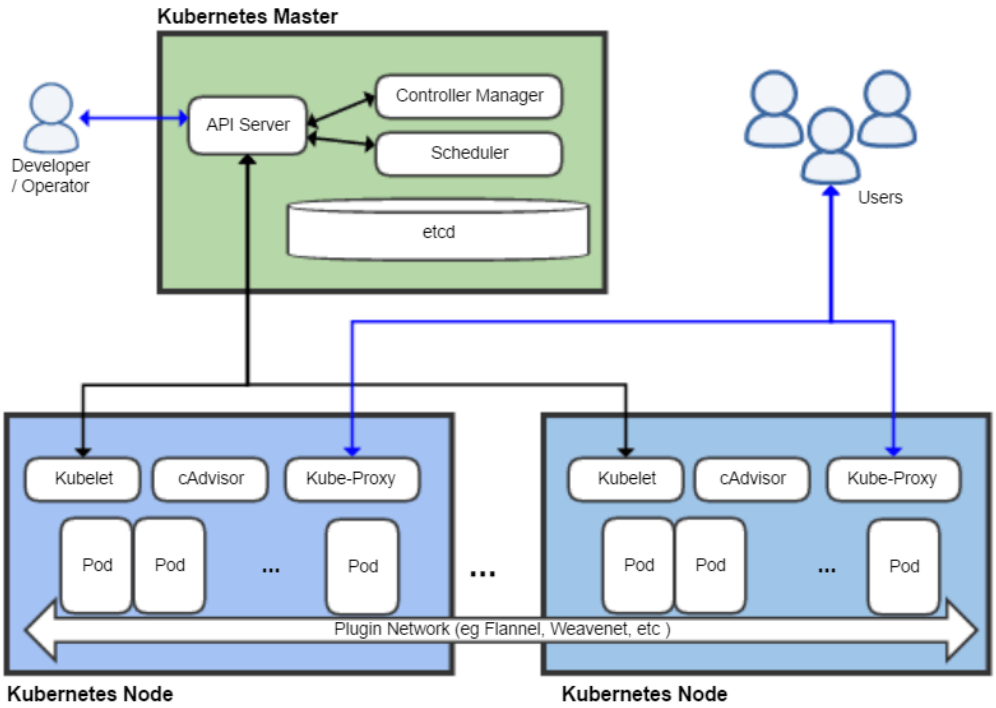
4.1 Järjestelmän suoritusympäristö

Järjestelmä tallentaa keräämänsä tiedon PostgreSQL-tietokantaan, joka on avoimena lähdekoodin relationaalinen tietokantajärjestelmä, jota on kehitetty jo yli kolmenkymmenen vuoden ajan [2]. Järjestelmä sekä tallentaa dataa tietokantaan että hakee sitä tietokannasta käyttäen Javalla ohjelmoitua palvelinta. Palvelimella pyörii myös järjestelmän sisäisessä käytössä oleva REST-rajapinta, jonka kautta järjestelmän käyttöliittymä sekä hakee että päivittää tietoja.

Järjestelmän käyttöliittymä on ohjelmoitu jo aiemmin mainitun Aurelia-ohjelmistokehyksen avulla. Ohjelmistoympäristöjä on käytössä kaksi: Javaa ohjelmoidaan käyttäen NetBeans-ohjelmistoympäristöä ja JavaScriptiä ohjelmoidaan käyttäen Visual Studio Code -ohjelmistoympäristöä.

Järjestelmä sijaitsee kokonaisuudessaan Amazonin AWS-palvelimilla. Jokainen järjestelmän erillinen ohjelmistokomponentti pyörii omassa Docker-kontissa. Kontti on puolestaan säiliö, jonne ohjelmistokomponentti sekä kaikki sen riippuvuudet sekä muut sen tarvitsemat ohjelmistokomponentit säilötään [3]. Sen tarkoituksena on luoda vakaa ja hallittava ympäristö ohjelmistolle, jossa sen suorittaminen onnistuu käyttäjän järjestelmästä rippumatta.

Kontti sijaitsee Kubernetes-podissa. Se on Kuberneteksen pienin komponentti, joka vastaa yksittäistä prosessia Kubernetes-klusterin sisällä. Yksinkertainen klusterin rakenne on nähtävissä kuvassa 11. Kubernetes on myös avoimen lähdekoodin ohjelmisto, jolla hallinnoidaan kontteja. Sen avulla monet konttien hallintaa liittyvät tehtävät voidaan automatisoida. Se pystyy esimerkiksi uudelleenkäynnistämään vioittuneita tai sammuneita palveluita automaattisesti sekä jakamaan järjestelmään varattuja resursseja tasaisesti.



Kuva 11. Kubernetes-klusterin rakenne.

Kubenrnetes-podit pyörivät Kubernetes-noodien sisällä. Noodit ovat joko fyysisiä tai virtuaalisia tietokoneita, riippuen klusterista, jolla tarkoitetaan noodien ryhmää. Noodien sisällä on useimmiten yksi tai useampi podi. Podien lisäksi noodit sisältävät kubelet-komponentin, jonka tehtävänä on varmistaa, että noodilla olevien podien kontit toimivat normaalisti [4]. Tämän lisäksi jokaisessa noodissa on kube-proxy, joka toimii rajapintana, kun noodi haluaa keskustella noodin ulkopuolisen maailman kanssa. Esimerkiksi käyttäjän hakiessa käyttöliittymän kautta tietoa sovelluksen tietokannasta, käyttöliittymä-podi keskustelee tarvittaessa kube-proxyn avulla toisen podin kanssa, jossa palvelin sijaitsee.

Jokainen Kubernetes-klusteri sisältää myös yhden master-noodin, jonka tehtävänä on hallita muita noodeja. Noodi sisältää API-palvelimen (API Server), ohjainhallitsijan (Controller Manager), aikatauluttajan (Scheduler) sekä etcd:n. API-palvelimen kautta master-noodi kommunikoi muiden noodien kubelet-komponenttien kanssa, sekä ohjelmiston ylläpitäjät pystyvät hallitsemaan Kubernetes-klusteria. Ohjainhallitsijan tehtävänä on hallita klusterin ohjaimia ja

esimerkiksi käynnistää uudelleen vikaantuneita komponentteja. Etcd puolestaan sisältää tietokannan, jonka sisälle on tallennettu klusterin konfiguraatiotiedot. Ajastajan tehtävänä on taas määritellä, minkä noodin sisälle podit sijoitetaan. [4; 5.]

Versionhallintana järjestelmänä käytössä on git, joka sekin on avointa lähdekoodia. Se mahdollistaa usean ohjelmistokehittäjän samanaikaisen työskentelemisen projektien kanssa. Se toimii käytännössä siten, että jokaisella projektilla on oma yhteinen säilytyspaikka, josta jokainen ohjelmistokehittäjä kopioi itselleen gitin avulla paikallisen kopion. Kaikki projektiin tehdyt muutokset tehdään paikalliseen kopioon, josta ne gitin avulla yhdistetään takaisin alkuperäiseen säilytyspaikkaan. Gitin toiminta perustuu siihen, ettei se kopioi käyttäjän paikallisesta kopiosta kaikkea alkuperäiseen säilytyspaikkaan, vaan se vertailee kopion ja alkuperäisen säilytyspaikan eroja ja päivittää vain eroavat kohdat. Tämä tekee gitin käytöstä paljon tehokkaampaa kuin koko säiliön päivittämistä jokaisen muutoksen yhteydessä.

Java-projektien kokoamisen hoitaa Maven. Maven on projektinhallintatyökalu, joka mahdollistaa esimerkiksi projektin kokoamisen tai yksikkötestien suorittamisen. Kokoaminen suoritetaan Azure DevOps-ympäristössä, joka on ohjelmistokehitystä helpottavien menetelmien palvelu, joka sisältää esimerkiksi useita projektin hallintaan liittyviä ominaisuuksia, kuten mahdollisuuden koota jatkuvan kehityksen putkia. Azure DevOpsiin on rakennettu jatkuvan kehityksen putket, jotka varmistavat, että projektin koonnin jälkeen kaikki siitä riippuvaiset projektit kootaan myös uudestaan. Näin myös päivitetystä projektista riippuvaiset projektit saavat aina käyttöönsä päivitetyn projektin uusimman version. Kun kaikki projektin riippuvuudet ovat valmiina, projekti päivittyy automaattisesti käytössä olevaan kehitysympäristöön. Projektien kokoaminen voidaan aloittaa joko manuaalisesti tai se voidaan automatisoida. Helpoin tapa hoitaa projektien kokoaminen on automatisoida se siten, että joka kerta, kun joku yhdistää paikalliseen kopioon tehdyt muutokset alkuperäiseen säilytyspaikkaan, Maven kokoaa projektin sekä projektit, jotka hyödyntävät juuri koottua projektia.

4.2 Järjestelmän kehittäminen

Järjestelmän kehitystyö toteutetaan omassa kehitysympäristössä, joka sijaitsee myös AWS-palvelimilla. Kehitysympäristöstä löytyy tuotantoa vastaava tietokanta sekä palvelin, joita kehitysympäristön käyttöliittymä käyttää. Kehitysympäristö on täysin erillinen ympäristö tuotantoympäristöön verrattuna, jolloin sovelluksen kehittäminen sitä käyttäen on turvallista. Kehitysympäristöstä päivitetyt ohjelmistokomponentit siirretään ennalta tarkkaan määritellyn prosessin avulla testiympäristöön, jossa tulevan tuotantojulkaisun päivityksiä on mahdollista kokeilla vakaassa ympäristössä. Kuten kehitysympäristö, myös testiympäristö on täysin erillinen ympäristö tuotantoympäristöön nähden. Testiympäristöstä toimivaksi todetut ohjelmistokomponentit päivitetään tuotantoympäristöön, johon on myös käytössä ennalta tarkasti määritelty prosessi.

Järjestelmää kehitetään Scrum-mallia myötäillen, kuitenkaan sitä liian tarkasti noudattaen. Pienen tuotekehitysosaston ansiosta kehitystyö pystytään toteuttamaan hyvin ketterästi sekä joustavasti. Scrumin tapaisesti kehitystyö on jaettu noin kuukauden mittaisiin sprint-jaksoihin. Jokaisen jakson alussa johtoryhmä katsoo työjonosta tärkeimmät tehtävät ja siirtää niitä tulevaan jaksoon. Tehtävät voivat olla esimerkiksi uusia ominaisuuksia, bugikorjauksia tai järjestelmän ylläpitoon liittyviä tehtäviä. Jaksoon siirretyt tehtävät jaetaan pienempiin työtehtäviin ja työtehtäville annetaan tehtävään kuluvan ajan arvio. Viimeistään tässä vaiheessa tehtäville annetaan myös jonkinlaiset hyväksymiskriteerit, joiden täytyttyä tehtävä on valmis. Tehtäviä aletaan suorittamaan yksi työtehtävä kerrallaan, kunnes ne on kaikki suoritettu. Kun kaikki tehtävän alle määritellyt työt on suoritettu, on tehtävä valmis, ja se annetaan toiselle ohjelmistokehittäjälle tarkastettavaksi. Jos tarkastaja löytää jotain korjattavaa, alkuperäinen tehtävän omistaja korjaa ne, jonka jälkeen tehtävä annetaan samalle tarkastajalle uudelleen tarkastettavaksi. Kun tarkastaja katsoo tehtävän tulleen hyväksytysti suoritetuksi, tehtävä merkitään valmiiksi, jolloin se on valmiina odottamassa uutta tuotantojulkaisua.

Kehitykseen kuuluu myös oleellisena osana päivittäiset Stand up -palaverit, joiden aikana käydään lyhyesti läpi, mitä kukin on saanut aikaiseksi sekä, mitä

kenelläkin on seuraavana vuorossa. Näin jokaisella kehitykseen osallistuvalla työntekijällä on aina hyvä kuva siitä, miten kyseinen jakso etenee. Tämä on tärkeää, jotta jokainen projektin kehitykseen osallistuva työntekijä voi siirtyä tekemään toisen kehittäjän tekemää tehtävää esimerkiksi sairastapauksen takia.

5 Aurelia

Aurelia on kokoelma moderneja JavaScript-moduuleita, jotka yhdessä muodostavat tehokkaan alustan internet-sivujen, työpöytäohjelmistojen sekä mobiilisovellusten kehittämiseen. Aurelian lähdekoodi on täysin avointa lähdekoodia sekä sen dokumentaatiot ovat täysin vapaasti käytettävissä.

Aurelia on kirjoitettu ECMAScriptin mukaisesti, joka on JavaScript-standardi, jonka tarkoituksena on mahdollistaa ohjelmistojen toimiminen kaikilla yleisesti käytössä olevilla internetselaimilla. Se on myös suunniteltu DOM-standardien mukaisesti. Aurelia toimii sekä JavaScriptin että TypeScriptin kanssa.

5.1 Käyttöönotto

Aurelian käyttöönotto on hyvin yksinkertaista ja siihen löytyvät esimerkiksi Aurelian omilta sivuilta yksinkertaiset ohjeet. Tässä esimerkissä näytetään, miten Aurelian käyttäminen aloitetaan. Käyttöönottoa varten tarvitsee NodeJs-version 10 tai siitä uudemman version.

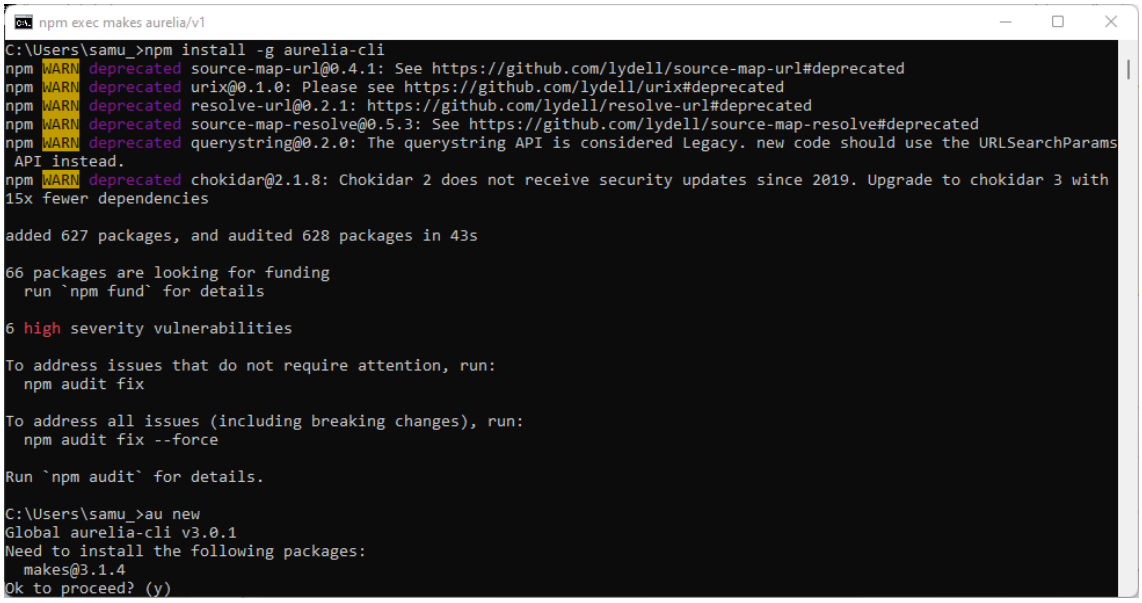
Aluksi asennetaan Aurelia CLI, joka on Aurelian virallinen komentorivityökalu. Sen avulla pystyy esimerkiksi luomaan uusia Aurelia-projekteja sekä kasaamaan sovelluksen julkaisua varten. Aurelia CLI:n asentaminen tapahtuu komentorivikomennolla hyödyntäen noden paketinhallintatyökalua npm:ää (Node Package Manager):

```
npm install -g aurelia-cli
```

Komento asentaa Aurelia CLI:n globaalisti tietokoneelle, jonka jälkeen sen tarjoamat työkalut ovat käytettävissä komentorivin kautta.

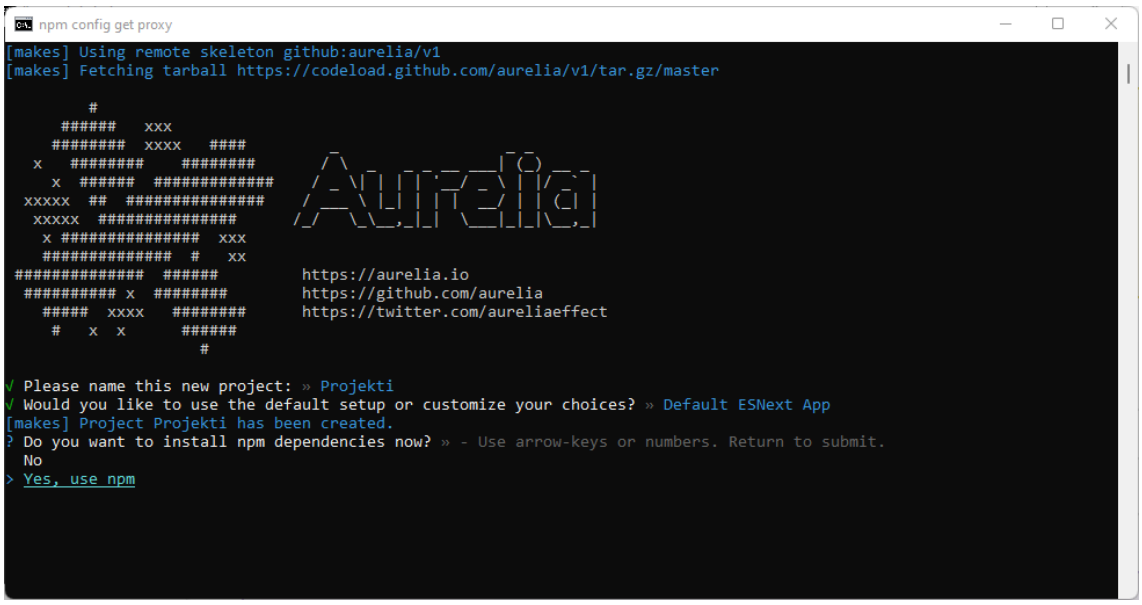
Uuden projektin luominen onnistuu helpoiten käyttämällä Aurelian CLI:n au new -komentoa. Projekti luodaan siihen kansioon, missä komento syötetään.

Komennon jälkeen saattaa joutua asentamaan joitain Aurelian käyttämiä riippuvuuksia, jos niitä ei ennestään asennettu. Se onnistuu vastaamalla komentokehotteen esittämiin kysymykseen. Tässä tapauksessa asennettavana oli makes-kirjasto versionumerolla 3.1.4 (kuva 12).



Kuva 12. Aurelian uuden projektin tarvitsemien riippuvuuksien asennus.

Mahdollisten riippuvuuksien asentamisen jälkeen komentokehote kysyy luotavan projektin nimeä. Tässä esimerkissä nimeksi annetaan ”Projekti”. Tämän jälkeen Aurelia kysyy, minkälaisen projektin haluaa luoda. Valittavana on JavaScript-sovellus, TypeScript-sovellus, JavaScript-lisäosa, TypeScript-lisäosa tai kustomoitu projekti. Tässä esimerkissä valitaan ensimmäinen vaihtoehto eli JavaScript-sovellus. Vielä lopuksi komentokehote kysyy, asennetaanko sovelluksen tarvitsemat npm-riippuvuudet heti, johon tässä tilanteessa vastataan myöntävästi (kuva 13).



Kuva 13. npm-riippuvuuksien asennus.

Kun riippuvuudet on asennettu, voidaan siirtyä komentokehotteella juuri luotuun projektiin käyttämällä alla olevaa komentoa:

```
cd Projekti
```

Tämän jälkeen voidaan käyttää alla olevaa Aurelian omaa komentoa, joka kokoaa projektin sekä avaa oletusselaimella osoitteen localhost:8080, jossa juuri luotu projekti pyörii:

```
au run --open
```

Projektin luominen on onnistunut, jos sivulla lukee Hello World! Vaihtoehtoisesti voidaan käyttää esimerkiksi alla olevaa npm:n omaa komentoa ja avata sen jälkeen osoite localhost:8080 halutulla selaimella:

```
npm start
```

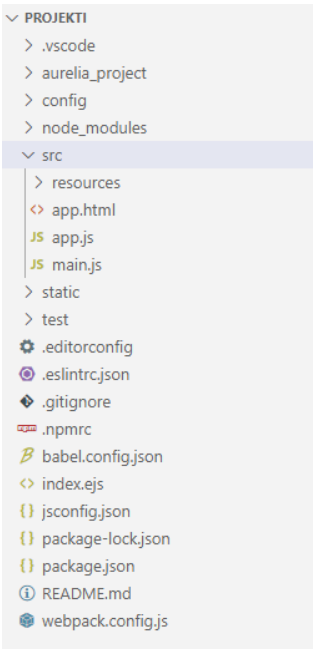
Aurelian oma komento on yksinkertaisesti alias yllä olevalle komennolle, jossa parametri open avaa konfiguraatiossa määritellyn osoitteen internetselaimella. Jos portti 8080 halutaan jostain syystä vaihtaa, se onnistuu aurelia.json-tiedoston kautta. Tiedosto löytyy projektin kansiosta nimeltä aurelia_project. Tiedoston alin objekti on platform-objekti, jonka port-arvon vaihtamalla voidaan käyttää

haluttua porttia. Saman objektin open-arvoa vaihtamalla myös npm start -komento saadaan avaamaan projekti internetselaimella automaattisesti.

5.2 Aurelian perusteet

Aurelian käyttäminen on helppoa, sillä sitä käyttäessä voi ohjelmoida suoraan JavaScriptiä sekä HTML:ää käyttäen. Tämän vuoksi sen perusteet on helppo oppia. Aureliaa käyttäessä JavaScript- sekä HTML-tiedostot erotellaan toisistaan omiin tiedostoihinsa, jolloin koodin logiikka sekä ulkoasulliset komponentit pysyvät erillään toisistaan. Lisäksi käytössä on CSS, joka sekin sijaitsee omassa tiedostossa, jolla voidaan vaihtaa HTML-komponenttien tyyliä.

Luvussa 5.1 luotiin uusi Aurelia-projekti. Projektin luonnin yhteydessä Aurelia loi projektille kuvan 14 mukaisen rungon. Projektista löytyy esimerkiksi kaikki aikaisemmin asennetut npm-riippuvuudet, sekä package.json-tiedosto, jonka kautta voidaan esimerkiksi lisätä sovellukseen uusia riippuvuuksia. Tärkeimmät tiedostot tässä kohtaa ovat kuitenkin src-kansiosta löytyvät app.html sekä main.js. Tämän lisäksi src-kansiossa on myös main.js-tiedosto, joka sisältää hyvin yksinkertaisen Aurelian automaattisesti luodun konfiguraation.



Kuva 14. Luodun projektin rakenne.

Kuten esimerkkikoodeista 1 ja 2 nähdään, automaattisesti luodut app.js sekä app.html -tiedostot ovat hyvin yksinkertaisia. Niistä näkee kuitenkin yhden Aurelian perusteen tuoda koodissa määriteltyjä arvoja käyttöliittymään näkyviin.

```
export class App {
  message = 'Hello World!';
}
```

Esimerkkikoodi 1. Automaattisesti luotu app.js-tiedosto

```
<template>
  <h1>${message}</h1>
</template>
```

Esimerkkikoodi 2. Automaattisesti luotu app.html-tiedosto

Esimerkkikoodeista 1 ja 2 nähdään, että app.js-tiedostossa on määritelty muuttuja message. Samaa muuttujaa käytetään app.html-tiedostossa, jolloin muuttujaan määriteltyä ”Hello World”-teksti ilmestyy käyttöliittymään. Koska tiedostot ovat nimettyjä samalla tavalla, Aurelia osaa automaattisesti hakea message-muuttujan JavaScript-tiedostosta, eikä HTML-tiedoston käyttämää JavaScript-tiedostoa tarvitse erikseen määritellä millään tavalla.

HTML-elementtien luonti on myös helppoa. Se onnistuu luomalla erilliset JavaScript- sekä HTML-tiedostot elementille, kuten custom-select.js- sekä custom-select.html-tiedostot. Seuraavaksi määritellään HTML-tiedostossa pudotusvalikko käyttämällä select-tagia ja annetaan sille vaihtoehtoja option-tagilla. Option-tagin sisällä voimme käyttää Aurelian repeat.for-attribuuttia. Sen avulla voimme dynaamisesti määritellä pudotusvalikon vaihtoehdot käyttämällä JavaScript-tiedostosta löytyvää muuttujaa. Yksinkertaisen pudotusvalikon koodi on nähtävissä esimerkkikoodissa 3 ja 4.

```
export class CustomSelect {

  constructor() {
    this.options = [
      {name: 'Ensimmäinen vaihtoehto'},
      {name: 'Toinen vaihtoehto'},
      {name: 'Kolmas vaihtoehto'},
      {name: 'Neljäs vaihtoehto'},
      {name: 'Viides vaihtoehto'}
    ];
  }
}
```

Esimerkkikoodi 3. Yksinkertaisen pudotusvalikon JavaScript-tiedosto

```
<template>
  <select>
    <option repeat.for="option of options">${option.name}</option>
  </select>
</template>
```

Esimerkkikoodi 4. Yksinkertaisen pudotusvalikon HTML-tiedosto

Lopputulos on nähtävissä kuvassa 15

Kuva 15. Yksinkertainen pudotusvalikko erillisenä komponenttina.

Ylemmässä esimerkissä vaihtoehdot on annettu suoraan komponentille, jolloin komponentilla olisi aina samat vaihtoehdot käyttötarkoituksesta riippumatta. Vaihtoehdot voidaan kuitenkin antaa komponentille dynaamisesti sieltä, missä komponenttia käytetään. Tämä onnistuu sitomalla komponenttia käyttävässä koodissa valinnat sen itse käyttämään muuttujaan. Muuttujan sitomiseen on käytössä viisi eri vaihtoehtoa:

- One-way siirtää dataa koodista näkymään.
- Two-way siirtää dataa sekä koodista näkymään että näkymästä koodiin.
- One-time siirtää dataa kerran koodista näkymään, eikä sen jälkeen reagoi mahdollisiin muutoksiin.
- From-view siirtää dataa näkymästä koodiin.
- Bind valitsee automaattisesti sidontatyypin. Suurimmassa osassa tilanteissa bind valitsee one-way-sidonnan, mutta esimerkiksi lomakkeiden kanssa valitaan two-way-sidonta.

Yleisesti ottaen automaattinen sitominen toimii juuri siten kuin halutaan, jolloin muiden käyttö rajoittuu lähinnä erikoistapauksiin, kuten one-way-sidonnan käyttämistä muuttujaan, joka ei muutu, jolloin näkymän on turha käyttää resursseja muuttujan seuraamiseen. Koodiesimerkin 5 mukaisesti custom-select.js-tiedostossa olevan options-muutujan alustus voidaan poistaa ja siitä voidaan tehdä sidottava, jolloin sen arvoa voidaan muokata toisen ohjelmakomponentin toimesta.

```
import {bindable} from 'aurelia-framework';

export class CustomSelect {
  @bindable options;
}
```

Esimerkkikoodi 5. Muokattu pudotusvalikon JavaScript-koodi, jossa valinnat tulevat komponentin ulkopuolelta.

Jotta valinnat saadaan asetettua komponenttiin, sitä kutsuvan ohjelmakomponenttiin tulee tehdä pieniä muutoksia. Yllä olevista sidontamenetelmistä voidaan valita tilanteeseen sopiva menetelmä ja sen avulla custom-select-komponentin options-muuttujalle saadaan annettua arvo. Tämä on nähtävissä koodiesimerkissä 6.

```
<custom-select options.bind="optionsFromOtherFile"></custom-select>
```

Esimerkkikoodi 6. Custom-select-komponentin valintojen sitominen app.js-tiedostosta löytyvään optionsFromOtherFile-muuttujaan.

Tämän lisäksi esimerkkikoodissa 6 mainittu optionsFromOtherFile-muuttuja tulee lisätä app.js-tiedostoon. Esimerkkikoodissa 7 muuttuja on alustettu komponentin luonnin yhteydessä, mutta sen arvoja voisi myös muokata muualta komponentin sisältä.

```
export class App {
  message = 'Hello World!';

  constructor() {
    this.optionsFromOtherFile = [
      {name: "Vaihtoehdot"},
      {name: "Toisesta"},
      {name: "Tiedostosta"}
    ]
  }
}
```

Esimerkkikoodi 7. Päivitetty app.js-koodi, jossa muuttujan optionsFromOtherFile arvo on sidottu custom-select-komponentin options-muuttujaan.

Kuvassa 16 on nähtävissä, että app-komponentin muuttujan optionsFromOtherFile arvot ovat siirtyneet pudotusvalikon vaihtoehdoiksi.

Kuva 16. Custom-select-komponentin valinnat toisesta komponentista tuotuna.

Aurelia mahdollistaa ehdollisen HTML-elementtien näyttämisen kahdella eri tavalla: show.bind sekä if.bind. Kuten koodiesimerkissä 8 näkyy, näiden käyttäminen toimii samalla tavalla kuin yllä mainittu muuttujan sitominen HTML-elementtiin. Nämä eroavat toisistaan siten, että ensimmäisen kohdalla elementti luodaan, mutta sitä ei näytetä. Jälkimmäisen kohdalla taas elementtiä ei luoda ollenkaan, jos siihen sidottu ehto ei täyty. Yleisesti ottaen show.bind on tarkoitettu tilanteisiin, jossa esimerkiksi suodatetaan käyttöliittymän puolella tietokannasta

haettua dataa. Tällöin elementtiin sidottu ehto saattaa muuttua useita kertoja lyhyenkin ajan sisällä. If.bind puolestaan sopii hyvin esimerkiksi tilanteeseen, jossa elementin näyttämiseen tarvitaan jokin käyttöoikeus, joka ei yleensä muutu. Toisaalta edellä mainitussa datan suodattamisessa voi myös käyttää if.bind-sidontaa, jos esimerkiksi muuten luotavien HTML-elementtien määrä kasvaa suureksi, jolloin käyttöliittymän pyörittäminen vaatii turhan paljon resursseja.

```
<template>
  <require from="custom-select"></require>
  <!-- booleanVariable is set as true, someValue as 10-->
  <h1 if.bind="!booleanVariable">${message}</h1>
  <h1 style="color: red" if.bind="booleanVariable">${message}</h1>
  <h1 style="color: gray" if.bind="someValue == 10">${message}</h1>
  <h1 style="color: blue" if.bind="someValue == 9">${message}</h1>
  <custom-select options.bind="optionsFromOtherFile"></custom-select>
</template>
```

Esimerkkikoodi 8. Elementtien ehdollinen näyttäminen. BooleanVariable on koodissa määritelty todeksi ja someValue arvoon 10.

Kuten yllä olevasta kuvasta voidaan päätellä, tässä tilanteessa käyttöliittymässä näkyy sekä punainen että harmaa teksti. Mustaa ja sinistä tekstiä ei ole näkyvissä (kuva 17).

Kuva 17. Ehdollisesti näytetyt tekstit.

Useimmat Aurelian komponentit käyvät käytön aikana läpi viisi eri metodia:

- Constructor-metodia kutsutaan, kun komponentti luodaan.
- Activate-metodia kutsutaan, kun komponentti otetaan käyttöön.

- Attached-metodi kutsutaan, kun kaikki komponentin html-elementit ovat ladattuna.
- Deactivate-metodia kutsutaan, komponentti aiotaan poistaa näkymästä vaikka toiselle sivulle siirtymisen seurauksena.
- Detached-metodia kutsutaan, kun komponentti on poistettu kokonaan näkymästä.

Näillä metodeilla voidaan esimerkiksi alustaa komponentteja tai varmistua, että kaikki komponentin HTML-elementit on ladattuina ennen, kuin niitä kutsutaan koodissa. Lisäksi joissain tilanteissa komponenteista saatetaan joutua tuhoamaan osia tai niille joutuu tekemään joitain muita toimenpiteitä. Näitä voidaan toteuttaa tilanteen mukaan detached- ja deactivate-metodeissa.

Tämän lisäksi Aurelian mukana tulee esimerkiksi datan validointiin käytettävä moduuli, jolla pystyy esimerkiksi validoimaan käyttäjän antaman syötteen lomakkeissa. Lisäksi paketin mukana tuleviin toiminnallisuuksiin sisältyy reititin, jolla pystyy siirtymään sovelluksen sisällä sivulta toiselle.

5.3 Vertailu muihin ohjelmointikehyksiin

Aureliaa voi vertailla muihin samantyyppisiin ohjelmointikehyksiin, joita ovat esimerkiksi React ja Vue. Molemmat niistä ovat Aurelian tapaan suunnattuja käyttöliittymän ohjelmointiin. Molemmat ovat myös avointa lähdekoodia.

React on näistä kolmesta ohjelmointikehyksestä suosituin. Sen GitHub-sivuston [6] mukaan yli 11 miljoonaa GitGup projektia käyttää Reactia. Vuen vastaava luku on noin 3 miljoonaa [7] ja Aurelian vain noin 4 tuhatta [8]. Koska Aureliaa käytetään huomattavasti vähemmän kuin Vuea ja Reactia, sen ympärillä oleva yhteisö on myös huomattavasti pienempi. Tämän vuoksi Reactin ja Vuen käyttäjät löytävät usein helpommin ongelmiinsa vastauksia, sillä on todennäköistä, että useat kyseisten ohjelmointikehysten käyttäjät ovat törmänneet samaan ongelmaan aikaisemmin, jolloin ongelmiin löytyy valmiit ratkaisut. Toisaalta Aurelian pienen yhteisön vuoksi apua saattaa saada jopa Aurelian kehittäjiltä.

Reactiin verrattuna Aurelia tarjoaa helpommin lähestyttävän kokonaisuuden. Kun Aurelia-projekti luodaan Aurelia-CLI-komentorivityökalua käyttäen, uusi projekti sisältää jo valmiiksi useita tärkeitä komponentteja, kuten reitittimen sekä validoinnin. React-projektia luodessa puolestaan pystyy tai joutuu valitsemaan, mitä komponentteja haluaa esimerkiksi näihin tarkoituksiin käyttää. Jos projektin kehittäjiä on paljon, joudutaan väkisinkin tekemään kompromisseja, sillä erilaisia komponentteja on paljon. Tämä johtaa siihen, että kaikki joutuvat opettelemaan jotain uutta, sillä kukaan ei todennäköisesti ole käyttänyt kaikkia mukaan valittuja komponentteja. Tämän lisäksi React-yhteisössä on hyvin löysät standardit esimerkiksi projektin rakenteiden suhteen. Aurelia-yhteisössä puolestaan noudatetaan vahvasti standardeja, joten projektista toiseen siirtyminen on huomattavasti helpompaa kuin Reactin suhteen. [9]

Tämän lisäksi Aurelian syntaksi on usein helpommin ymmärrettävää. Tämän näkee esimerkiksi seuraavista koodiesimerkeistä. Esimerkkikoodissa 9 on nähtävillä, kuinka yksinkertaisen lomakkeen lähetys toimii Reactilla ja Aurelialla. Aurelian koodi vie hieman vähemmän tilaa ja on muutenkin helposti ymmärrettävää, jos on aiemmin käyttänyt HTML:ää. Reactin koodissa puolestaan on sekaisin sekä HTML:ää että JavaScriptiä, mikä tekee siitä sekavamman näköisen.

```
// React
render() {
    return (
        <form onSubmit={this.handleSubmit}>
        <label>
            Name:
            <input type="text" value={this.state.value} on-
Change={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
        </form>
    );
}

// Aurelia
<form submit.trigger="handleSubmit()">
    <label>
        Name:
        <input type="text" value.bind="state.value" />
    </label>

    <input type="submit" value="Submit" />
</form>
```

Esimerkkikoodi 9. Reactin ja Aurelian yksinkertaisen lomakkeen lähetys. [9]

Esimerkkikoodissa 10 puolestaan vertaillaan yksinkertaisen listan näyttämistä käyttöliittymässä. Tässä esimerkissä eron huomaa vielä selkeämmin. Aurelian koodissa on käytetty jo aikaisemmin mainittua repeat.for-toimintoa, joka on yksinkertainen ja kompakti. Reactin keino puolestaan vaatii erillisten lista-elementtien palauttamista, joka vie selvästi enemmän tilaa ja jonka koodi näyttää muutenkin huomattavasti sekavammalta.

```
//React
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li>{number}</li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);

//Aurelia
<let numbers.bind="[1, 2, 3, 4, 5]"></let>

<ul>
    <li repeat.for="n of numbers">${n}</li>
</ul>
```

Esimerkkikoodi 10. Reactin ja Aurelian yksinkertaisen listan näyttäminen käyttöliittymässä. [9]

Aurelian syntaksi on myös helpommin lähestyttävää kuin Vuen syntaksi. Esimerkiksi yksinkertaisen nappulan, joka kasvattaa muuttujan lukuarvo yhdellä ja näyttää lukuarvon. Koodi Vuella kirjoitettuna, on nähtävissä esimerkkikoodissa 11.

```
//JavaScript
import { createApp } from 'vue'

createApp({
  data() {
    return {
      count: 0
    }
  }
}).mount('#app')

<!-- HTML -->
<div id="app">
  <button @click="count++">
    Count is: {{ count }}
  </button>
</div>
```

Esimerkkikoodi 11. Yksinkertainen painike ohjelmoituna Vuella [10]

HTML-koodi on hyvin saman tyylinen kuin Aurelian koodissa, joka on nähtävissä esimerkkikoodissa 12. JavaScript-koodit puolestaan eroavat jo huomattavasti toisistaan. Vuessa muuttujien arvot, tässä tilanteessa count-muuttuja, on määritelty data-metodin sisällä, joka on Vuen tapa määritellä HTML:n käyttämiä muuttujia. Koska muuttujat määritellään ylimääräisen metodin palautusarvona, koodin luettavuus kärsii. Aurelialla muuttujan arvon on tallennettu perinteisen JavaScriptin tyyliin, joten lähes jokainen ohjelmoimaan vähänkään opetellut ymmärtää koodia välittömästi.

```
//JavaScript
export class App {

  constructor() {
    this.count = 0;
  }
}

<!-- HTML -->
<template>
  <button click.delegate="count = count+1">
    Count is: ${count}
  </button>
</template>
```

Esimerkkikoodi 12. Yksinkertainen painike ohjelmoituna Aurelialla.

Vuen eduksi voidaan laskea mahdollisuus toteuttaa koko komponentti yhdellä tiedostolla, jolloin HTML, JavaScript sekä CSS ovat kaikki samassa tiedostossa. Tämä ei onnistu yhtä kattavasti Aurelialla. Aureliaa käyttäen voidaan luoda

komponentteja pelkästään HTML:ää käyttäen, mutta tällä tavalla luodun komponentin toiminnallisuus ei ole yhtä hyvä Vueen verrattuna.

6 Resursointityökalun toteutus

Resursointityökalun toteutus aloitettiin keväällä 2022. Tätä ennen työkalulle oli jo annettu karkeat vaatimukset, jotka on kuvattu luvussa 3. Työ alkoi uuden ominaisuuden suunnittelulla, erityisesti tietokantataulukoihin sekä palvelimeen liittyen.

6.1 Suunnittelu

Resursointityökalun suunnittelu aloitettiin miettimällä, minkälaisia uusia taulukoita tietokantaan tulee lisätä. Jotkut taulukot, kuten varauksen ja varauspohjien tallentamiseen tarvittavat taulut, olivat ilmeisiä valintoja ja tulisivat sisältämään varauksien ja varauspohjien tietojen tallentamiseen tarvittavat kentät. Näiden lisäksi tarvittiin kuitenkin muita taulukoita, kuten taulukko asiakaskohtaisten varauskenttien ja arvojen tallentamiseen sekä taulukko varaustietojen muutosten seuraamista varten. Taulukoiden suunnittelun yhteydessä pohdittiin myös, minkälaista dataa tuleva resursointityökalu tulee tarvitsemaan, josta pystyttiin päättelemään, minkälaisen logiikan avulla tietoa haetaan tietokannasta. Erilaisia metodeja, joilla tietoa haetaan tietokannasta, tarvittaisiin useita. Näitä ovat esimerkiksi varausten hakeminen tiettyjä parametreja käyttäen halutulle aikavälille sekä vapaiden laitteiden hakeminen.

Suunnittelun toisen vaiheena oli käyttöliittymän ulkoasun suunnittelu. Kuten aiemmin luvussa 3 mainittiin, kalenterinäkymä on hyvin vakiintunut, joten oli ilmeistä, että myös resursointityökalun varauskalenterin tulisi näyttää ja toimia suunnilleen samalla tavalla kuin muiden suosittujen kalentereiden.

Tässä vaiheessa pohdimme myös, voisiko varauskalenteriin käyttää jo valmiiksi löytyviä kalenterikomponentteja. Päädyimmekin vertailemaan muutamaa eri vaihtoehtoa. Näitä ovat esimerkiksi FullCalendar [11], Mobiscroll-Scheduler [12] sekä Syncfusionin Flutter Calendar [13]. Näissä jokaisessa on selkeät hyvät ja

huonot puolensa. FullCalendar on ilmainen avoimen lähdekoodin kalenterikomponentti, joka sisältää suoraan suurimman osan tarvitsemistamme ominaisuuksista. Se vaikuttikin hyvältä valinnalta, mutta se skaalautui mobiilinäytölle erittäin huonosti. Tämän vuoksi se osoittautui sopimattomaksi tarkoituksiimme. Flutter Calendar puolestaan täyttää kaikki vaatimuksemme. Se myös tukee suoraan Aureliaa, jolloin sen tuominen osaksi projektia on helppoa. Se osoittautui kuitenkin turhan kalliiksi vaihtoehdoksi, joten sekään ei ollut hyvä valinta.

Myös Mobiscroll-Scheduler täytti kaikki vaatimuksemme. Se sisältää suurimmaksi osaksi samat ominaisuudet kuin FullCalendar sekä joitain resursointityökalun kannalta tärkeitä ominaisuuksia. Sen lisäksi se skaalautuu huomattavasti paremmin mobiililaitteelle. Tämän lisäksi sen hinta oli kohtuullinen, joten se sopisi meidän tarkoituksiimme. Ainoana huonona puolena siinä oli se, ettei se suoraan tukenut Aureliaa, ja sen sovittaminen Aureliaan tulisi vaatimaan työtä. Päätimme kuitenkin, että valmiin komponentin sovittaminen Aureliaan tulisi olemaan helpompaa kuin uuden komponentin luonti tyhjästä varsinkin, kun sen skaalautuvuus mobiililaitteella oli todella hyvä. Tämän vuoksi päätimme, että kokeilemme Mobiscroll-Schedulerin ilmaisversiota, ja jos se osoittautuu hyväksi, tulemme käyttämään sen maksullista versiota osana resursointityökalua.

Varauskalenterin lisäksi piti päättää, miten kuljettajan kirjautuminen laitteelle yhdistettäisiin resursointityökaluun sekä minkälaisia raportteja kerätystä datasta tulisi muodostaa. Tässä vaiheessa teimme päätöksen, että raportointi jätetään toistaiseksi kokonaan pois, kunnes saamme asiakkailta lisätietoa siitä, minkälaisia raportteja he oikeasti tarvitsevat. Kuljettajan kirjautumisen yhdistäminen päätettiin toteuttaa siten, että se on mahdollisimman yksinkertainen ja helppo sitä käyttäville asiakkaille. Tähän päädyttiin siksi, että kuljettajan kirjautuminen on jo asiakkaillemme tuttu toiminto, emmekä halunneet monimutkaistaa sitä turhaan. Tämän lisäksi pohdimme myös yksinkertaista näkymää, josta käyttäjä voisi nähdä vain sama päivän sisällä olevia varauksia, eikä varauksia pystyisi muokkaamaan. Tätä näkymään voisi käyttää esimerkiksi pakettien kuljettajat, jotka näkisivät päivän keikat helposti yhdellä silmäyksellä. Näkymään ei kuitenkaan ainakaan vielä tässä vaiheessa päätetty toteuttaa.

6.2 Toteutus

Työn toteutus aloitettiin luomalla aiemmin suunnitellut tietokantataulukot sekä niiden käyttämiseen tarvittavat luokat ja metodit. Hyvän suunnitelman ansiosta tämä vaihe oli nopeasti valmis. Tämän vaiheen ainoa ongelma oli yhdistää SQL:n datetime-datatyyppi Javan Instant-olion kanssa. Yhdistäminen onnistui kuitenkin ilman suurempia ongelmia hyödyntämällä muuntajaa, joka muuntaa ohjelmakoodissa olevan muuttujan arvon yhteensopivaksi tietokannan datatyypin kanssa aina, kun tietoa tallennetaan. Sama muuntaja myös huolehti, että tietoa haettaessa datetime-datatyyppi saadaan muunnettua Javan Instant-olioksi.

Tämän jälkeen projektia oli luonnollista jatkaa rajapintaan sekä muuhun palvelimen toteutukseen. Suunnitteluvaiheessa oli jo tarkkaan mietitty, minkälaisia päätepisteitä rajapintaan tarvitaan, jotta sen käyttäminen olisi tehokasta ja intuitiivista, joten niiden toteuttaminen oli helppoa. Palvelimeen tehtiin myös paljon muutoksia, jotta luodut päätepisteet saatiin yhdistettyä dataa hakevien ja muokkaavien metodien kanssa. Palvelimen muutokset sisälsivät myös paljon validointia. Datan validointi palvelimella on erittäin tärkeää, sillä se parantaa sovelluksen turvallisuutta [14]. Tämä johtuu siitä, että käyttöliittymän puolella tehty validointi on huomattavasti helpompi kiertää kuin palvelimella tehty validointi. Turvallisuuden lisäksi validoinnilla pyrittiin estämään vääränlaisen datan pääsy tietokantaan, joka saattaisi joissain tilanteessa haitata ominaisuuden käyttöä tulevaisuudessa.

Validoinnin lisäksi palvelimen on tärkeää varmistaa, että dataa käsittelevällä käyttäjällä on oikeus käsitellä dataa. Ominaisuuden käyttäminen vaatii erillisiä käyttöoikeuksia, joten jokaisen julkisen metodin yhteydessä on tarkistettava, että käyttäjällä on tarvittavat oikeudet. Tämän lisäksi on erittäin tärkeää varmistua myös siitä, että käyttäjällä on oikeus tarkastella dataan kohdistuvan laitteiden tietoja. Huonoimmassa tapauksessa käyttäjä voisi muuten esimerkiksi hakea dataa kokonaan toisen asiakkaan laitteista, joka olisi vakava tietoturva-aukko. Tämän lisäksi palvelimella piti huomioida myös luvussa 3 mainitut erityyppiset käyttäjät. Työnjohdon tulee pystyä hakemaan ja muokkaamaan dataa

monipuolisemmin kuin normaalin käyttäjän. Tämä oli helppo toteuttaa kahdella eri käyttöoikeudella, joista toinen antaa hieman enemmän oikeuksia kuin toinen.

Kun palvelimen ja rajapinnan koodimuutokset olivat valmiit ja vaikuttivat toimivan, oli aika siirtyä käyttöliittymän toteutukseen. Käyttöliittymän ohjelmointi aloitettiin tutkimalla, miten luvussa 6.1 mainittu Mobiscroll-Scheduler toimii Aurelian kanssa. Tutkiminen aloitettiin kopioimalla Mobscrollin sivuilta käyttötarkoituksiimme sopiva yksinkertainen esimerkki ja muokkaamalla koodia siten, että se on yhteensopiva Aurelian kanssa [15].

Aluksi komponentti ei näyttänyt toimivan hyvin Aurelian kanssa ollenkaan. Kalenterikomponentti tuli sivulle näkyviin, mutta siitä puuttui tyylit. Kalenterin toiminnot vaikuttivat toimivan moitteettomasti. Sivulle tuli napit, joiden avulla pystyi siirtymään kalenterissa haluttuun päivään, mutta kalenteri itse oli vain lista numeroita. Minkäänlaista virheviestiä ei kuitenkaan ilmestynyt, joten jouduimme arvailemaan mistä tämä johtuu. Aluksi epäilimme tämän johtuvan siitä, että Aurelian oma css-tiedoston tuonti on epäonnistunut, sillä tiedostot tuodaan hieman eri tavalla Aureliassa kuin perinteisessä HTML-koodissa. HTML:ää käyttäessä css-tiedoston voi ottaa käyttöön esimerkiksi laittamalle head-tagin sisään linkin käytettävään css-tiedostoon:

```
<head>
  <link rel="stylesheet" href="cssfile.css">
</head>
```

Aureliaa käytettäessä head-tagia ei yleisesti ottaen käytetä, joten css-tiedosto tuodaan HTML-koodin käyttämälle require-tagia:

```
<require from="cssfile.css"></require>
```

Ongelma ei kuitenkaan tuntunut ratkeavan, riippumatta siitä, miten css-tiedosto yritettiin tuoda HTML-koodin käyttöön. Ongelman syyksi osoittautui se, että HTML-komponentit eivät latautuneet samalla tavalla kuin pelkkää HTML:ää käyttäen. Mobiscrollin tyylit toimii siten, että komponenteille annetaan erilaisia luokkia, joihin Mobiscrollin css-tiedosto vaikuttaa. HTML-komponenttien tulee olla valmiina, kun Mobiscroll alkaa antamaan niille luokkia, jotta tyylit toimivat.

Ongelma pystyttiin ratkaisemaan Mobiscrollin enhance()-metodilla, joka antaa komponenteille oikeat luokat metodia kutsuttaessa. Metodia käytetään suoraan HTML-komponenttiin, jolloin komponentille sekä kaikille sen lapsikomponenteille, annetaan oikeat luokat. Laittamalla yllä mainitun metodin luvusta 5 tuttuun attached-metodiin, saimme ongelman ratkaistua.

Aurelia toimii yleisesti ottaen hyvin JavaScript-komponenttien kanssa, eikä tämäkään komponentti ollut alun vaikeuksien jälkeen poikkeus. Hyvän dokumentaation sekä valmiin koodiesimerkin ansiosta yksinkertaisen varauskalenterinäkymän tuominen käyttöliittymään osoittautui helpoksi tehtäväksi. Yksinkertaiseen varauskalenterinäkymään oli myös tuotavissa ominaisuuksia muista Mobiscrollin esimerkkikoodeista. Yksi tämänlainen ominaisuus on kalenterin ulkopuolisten tapahtumien tuonti kalenteriin, mitä hyödynnämme varauspohjien kanssa. Uusien ominaisuuksien yhdistäminen kalenterin runkoon ei kuitenkaan ollut aina yksinkertaista. Pystyimme kuitenkin varsin nopeasti päättämään, että käytämme komponenttia resursointityökalussamme.

Kun varauskalenterista oli saatu luotua toimivan oloinen kokonaisuus, oli aika varmistaa, että aikaisemmin luodut palvelin- sekä tietokantamuutokset toimivat varauskalenterin kanssa hyvin. Yhdistäminen onnistui suurimmaksi osaksi hyvin, mutta yhdistämisessä ilmeni myös muutamia uusia ongelmia. Näistä ehkä suurin oli JavaScriptin Date-olion ja Javan Instant-olion yhdistäminen rajapinnan läpi. Ongelmaan oli olemassa helppoja ratkaisuja, kuten ajan muuttaminen Unix-ajaksi, jolla tarkoitetaan kulunutta aikaa päivämäärästä 1.1.1970. Se on standardi tapa mitata aikaa käytännössä kaikissa käyttöjärjestelmissä sekä käyttöjärjestelmien välillä [16]. Tämä vaikeuttaa kuitenkin mahdollista virheiden etsintää, sillä dataa tarkistaessa jouduttaisiin muuntamaan Unix-aikoja helpommin luettavaan muotoon. Tämän vuoksi päätimme muuntaa ajan käyttöliittymässä merkkijonoksi ja muuntaa merkkijono palvelimessa Instant-olioksi, jonka ansiosta mahdollinen virheiden etsintä on huomattavasti helpompaa, kun aikaleimat ovat jo valmiiksi helposti luettavassa muodossa.

Varauskalenterista oli tarkoitus tehdä itsenäisesti toimiva komponentti, joka tuotaisiin resursointinäkymään luvun 5 esimerkkien mukaisesti. Tämä onnistuisi

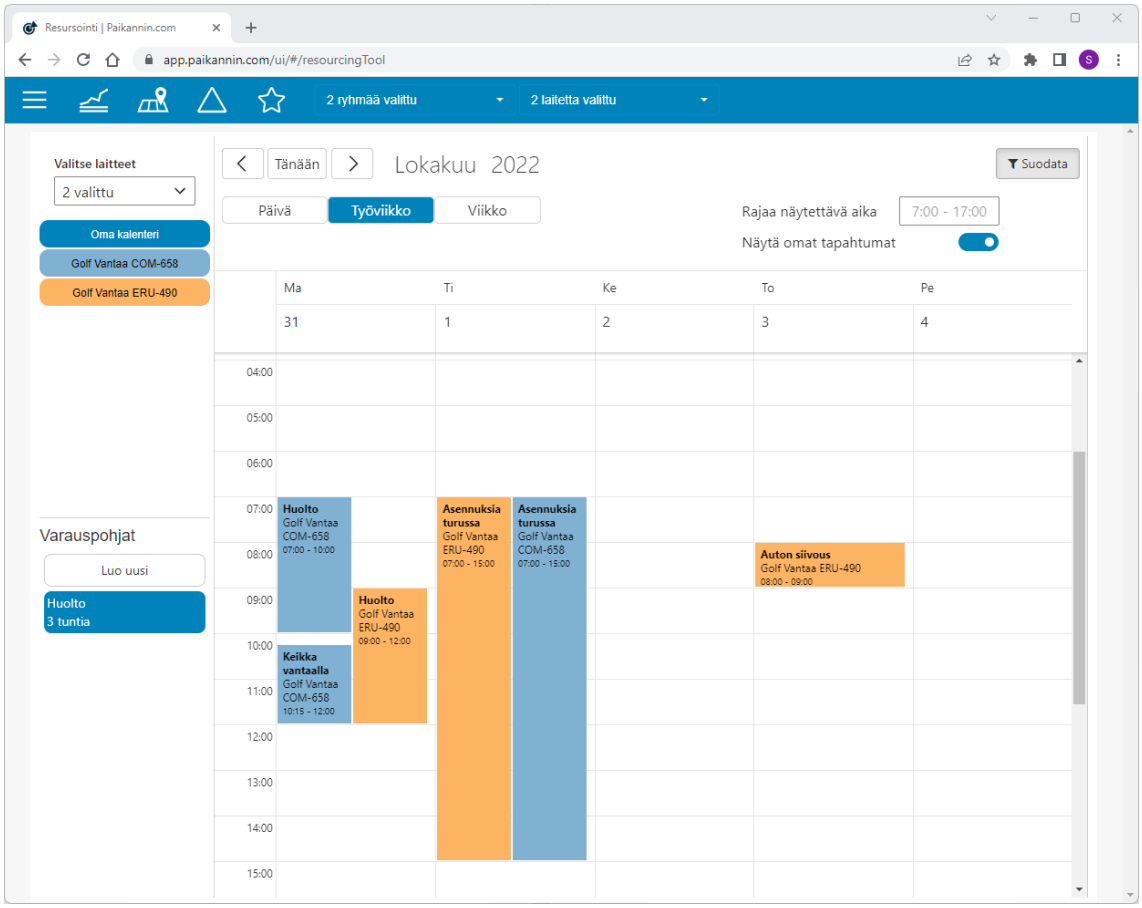
hyvin toteuttamalla omana komponenttinaan kalenterin runko ja sen lisäksi luomalla siinä käytettävistä elementeistä omia komponentteja. Näin samaa kalenterikomponenttia voisi halutessaan käyttää helposti myös muiden ominaisuuksien kanssa. Tämä osoittautui kuitenkin turhan monimutkaiseksi, sillä aiemmin muodostettua yksinkertaista kalenterikomponenttia joutui muokkaamaan niin paljon, että sen jakaminen komponentteihin ei olisi auttanut käytännössä yhtään. Vaikka jokaisesta kalenterin elementistä olisi luonut oman komponenttinsa, ne sisältäisivät niin paljon parametreja, että olisi yhtä helppoa luoda uusi kalenteri tyhjästä. Tämän lisäksi osa elementeistä eivät toimineet halutulla tavalla, jos niistä teki oman komponentin. Yksi tämänlainen elementti on esimerkiksi varausta luodessa tai muokatessa käytetty avautuva popup-ikkuna.

Vaikka emme tässä vaiheessa saaneet tehtyä kalenterista omaa uudelleen käytettyä komponenttia, pystyimme kuitenkin luomaan osasta uusista elementeistä omat komponenttinsa. Yksi esimerkki on Mobiscrollin itse käyttämä pudotusvalikko, sillä siitä sai suhteellisen yksinkertaisesti luotua oman toimivan komponenttinsa, jota voi nykyään tarvittaessa käyttää myös muiden ominaisuuksien kanssa. Komponentti mahdollistaa helposti uuden pudotusvalikon luonnin, ja se on parametrisoitu siten, että sillä voidaan esimerkiksi tilanteen mukaan valita vain yksi valinta tai toisessa tilanteessa useita valintoja.

Yleisesti ottaen varauskalenterin luonti sujui ilman suurempia murheita. Suurimmat varauskalenteriin liittyvät ongelmat johtuivat siitä, ettei se tukenut suoraan Aureliaa. Tämän vuoksi suoraan JavaScript-dokumentaatiosta katsotut ominaisuudet eivät välttämättä toimineetkaan siten kuin dokumentissa oli annettu ymmärtää. Esimerkiksi jotkin varauskalenterin käyttämät komponentit piti erikseen tuhota aina, kun varauskalenterista siirryttiin käyttämään muita järjestelmän toimintoja tai ne eivät enää seuraavalla kerralla toimineet.

Varuskalenteri on nähtävissä kuvassa 18. Kuvasta näkee ominaisuuden tärkeimmät komponentit. Vasemmalta sivupalkista voi valita laitteet, joiden varauksia haluaa tarkastella. Sen alta löytyy valitut laitteet värikoodattuna. Värikoodauksen ansiosta käyttäjä voi yhdellä silmäyksellä erottaa varauksista, mille laitteelle varaus on tehty. Tämän lisäksi sivupalkista voidaan luoda varauspohjia.

Näitä voivat olla esimerkiksi usein toistuvat tapahtumat. Varauspohjalle voidaan tallentaa tapahtumalle aina yhteisiä arvoja. Näitä voivat olla esimerkiksi tapahtuman otsikko, tapahtuman kesto, paikka tai varattava laite. Tämän ominaisuuden avulla käyttäjä ei usein toistuvien tapahtumien kohdalla joudu syöttämään aina kaikkia arvoja uudestaan vaan voi luoda varauspohjan ja siirtää sen aina haluamaansa kohtaan kalenteriin sekä täydentää puuttuvat tiedot. Kalenterin yläpuolelta löytyy kalenterin ohjaamiseen käytettävät painikkeet. Tämän lisäksi kalenterin yläpuolelta löytyy oletuksena piilotettuna oleva suodatusvalikko, jonka kautta voi esimerkiksi rajata kalenterissa näkyvää aikaa sekä vaihtaa näkymän tyyppiä kolmen valinnan kesken.

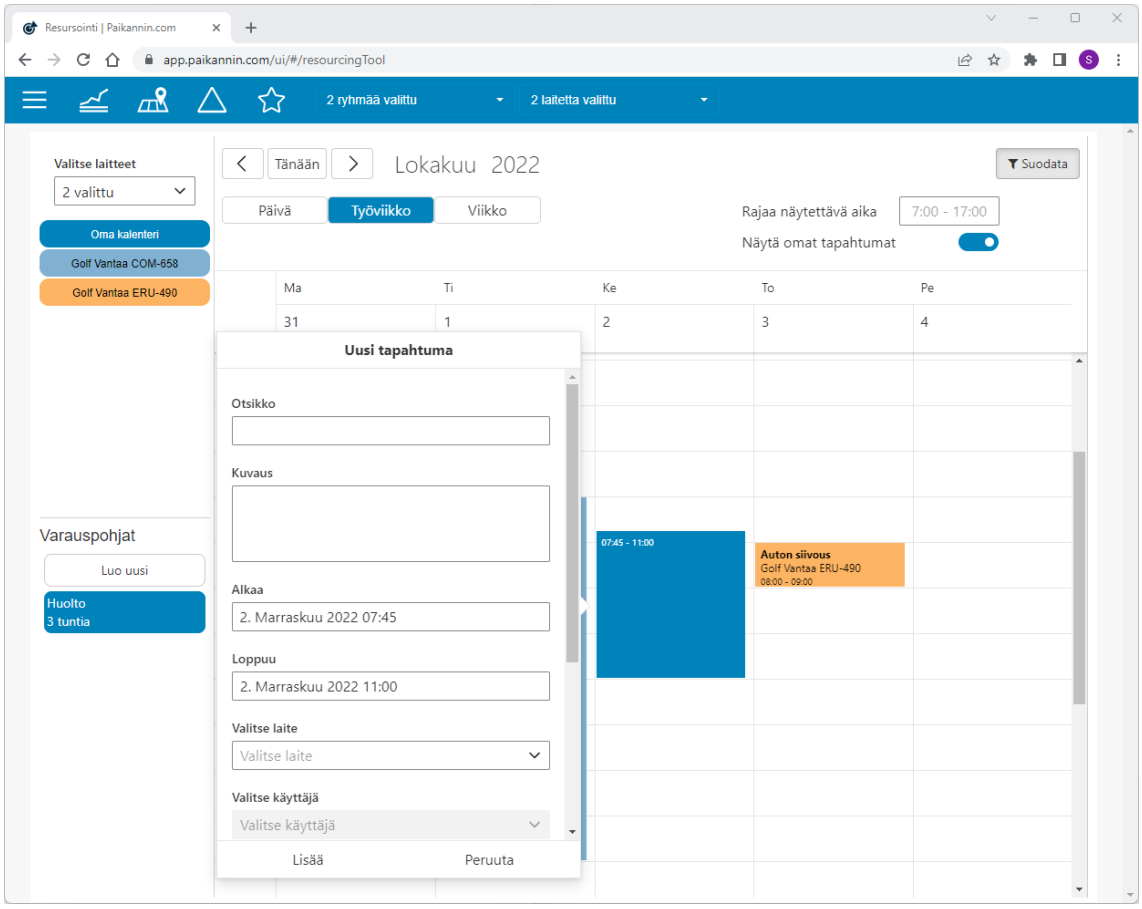


Kuva 18. Kahden auton varaustilanne.

Vaikka päädyimme käyttämään valmista kalenterikomponenttia, se sisälsi vain kalenterin keskeisimmät toiminnot, joita ovat esimerkiksi kalenterin tyylit, siirtyminen päivästä toiseen, varausten näyttämisen oikeissa paikoissa sekä

varausten siirtämisen hiirellä vetämällä. Pelkästään valmiiksi komponentin mukana tulevilla toiminnoilla käyttökokemus ei kuitenkaan olisi riittävän hyvä. Tämän vuoksi toiminnallisuutta jouduttiin parantamaan toteuttamalla esimerkiksi kaikki suodatusvalikon toiminnot itse. Ne kaikki muokkaavat kalenterin parametrejä ominaisuuden vaatimalla tavalla. Tämän lisäksi datan hakemin tietokannasta, sekä sen tallentamiseen ei ollut valmista ratkaisua. Niiden toteuttaminen oli kuitenkin helppoa yksinkertaisten rajapintakutsujen avulla.

Myös varausten luontiin ja muokkauksiin käytetty popup-ikkuna on luotu itse käyttäen Mobiscrollin komponentteja. Se on nähtävissä kuvassa 19. Se mahdollistaa varauksen tietojen lisäämisen varaukselle ja sisältää yleisesti kalentereissa olevia kenttiä, kuten otsikon, vapaan tekstikentän varauksen lisätiedoille sekä varauksen aloitus- ja lopetusajat. Tämän lisäksi siinä on kenttiä resursointityökalun vaatimiin tietoihin kuten varaukseen kohdistettavalle laitteelle, käyttäjälle, paikalle sekä varauksen tilalle. Lisäksi sinne on mahdollista tuoda asiakaskohtaisia kenttiä. Lähes samanlainen popup-ikkuna avautuu myös, kun käyttäjä halua luoda tai muokata varauspohjia. Suurin ero näiden kahden välillä on se, ettei varauspohjalla ole aloitus- ja lopetusaikoja. Tämän sijasta sillä on kesto päivinä ja tunteina.



Kuva 19. Varauskalenterin popup-ikkuna.

Varauskalenterin valmistuttua aloimme ohjelmoimaan kuljettajan kirjauksen yhdistämistä varauskalenteriin. Kuten luvussa 3 kerrottiin, tämän ominaisuuden on tarkoitus helpottaa käyttäjää varaamaan yksittäinen laite itselleen kirjautumalla laitteelle. Tämä ei saa kuitenkaan liikaa hankaloittaa laitteelle kirjautumista. Tämän ominaisuuden ohjelmointi oli helpohko operaatio.

Ominaisuus toimii siten, että laitteelle kirjautuessa tarkistetaan laittaan lähiaikoina olevat varaukset. Mikäli laitteelle löytyy varauksia rajatun ajan sisällä, sovellus päättää business-logiikan avulla esimerkiksi, luodaanko laitteelle uusi varaus. Toisaalta jos laite on varattuna toiselle käyttäjälle, ilmoitetaan siitä kirjautujalle, jotta hän voi tarvittaessa perua kirjautumisen ja valita toisen vapaana olevan laitteen.

Myös laitteelta uloskirjautuminen toimii samalla periaatteella: haetaan laitteen viimeaikaisia varauksia ja niiden perusteella päätetään, pitääkö esimerkiksi kuljettajan varaus katkaista kyseiseen ajanhetkeen. Kuljettajan kirjautumisen yhdistäminen resursointiin sijoitettiin kokonaan asiakaskohtaisen asetuksen taakse, jolloin asiakkaat voivat itse päättää haluavatko he, että kuljettajan kirjaukset vaikuttavat resursointiin.

6.3 Testaus ja jatkokehitys

Ominaisuutta testattiin tässä vaiheessa vain manuaalisesti. Palvelimen metodeja testattiin esimerkiksi käyttämällä niitä eri parametreilla ja tarkastelemalla tietokantaan tulevia muutoksia sekä mahdollista haettua dataa. Tämän lisäksi palvelimen metodeja testattiin paljon käyttöliittymän kautta.

Myös käyttöliittymää testattiin ainoastaan manuaalisesti. Käyttöliittymää testattiin kattavasti sekä internetselaimella että mobiililaitteella. Tämän lisäksi ominaisuuden valmistuttua se annettiin käyttämämme kehitysprosessin mukaisesti toiselle ohjelmistokehittäjälle testattavaksi, joka katselmoi kirjoitetun koodin, sekä varmisti ominaisuuden toimivan halutulla tavalla.

Alun perin oli myös tarkoitus, että palvelimen metodeille luotaisiin JUnit-yksikkötestejä, mutta ne päätettiin tässä vaiheessa jättää toteuttamatta. Yksikkötestien tarkoitus on varmistaa, että mahdolliset muutokset koodiin eivät riko mitään aiemmin kirjoitettua koodia. Tämän vuoksi ne saatetaan toteuttaa myöhemmin, jos ominaisuutta aletaan muokkaamaan jollain tavalla.

Tämän lisäksi ominaisuus on myös joillain asiakkaillamme pilotoinnissa. Kuten luvussa 2 mainittiin, Paikannin.com-järjestelmää käytetään hyvin monipuolisesti erilaisilla aloilla. Resursointityökalun on tarkoitus auttaa heitä kaikkia esimerkiksi optimoimaan laitteiden käyttöä. Tämän vuoksi on tärkeää, että saamme mahdollisimman paljon palautetta eri aloilta. Loppujen lopuksi asiakkaamme tietävät paremmin itse, minkälaisia ominaisuuksia he tarvitsevat liiketoiminatansa tueksi.

Resursointityökalulle on jo nyt tiedossa joitain jatkokehitysideoita. Suurin osa niistä on kuitenkin liikesalaisuuksia, joten niitä ei voi tässä työssä luetella kovin tarkasti. Yksi luvussa 6.1 mainittu jatkokehityskohde on raportointi. Se tullaan todennäköisimmin toteuttamaan, kunhan saamme asiakkailta palautetta asiaan liittyen. Raportoinnin toteuttaminen on työlästä, joten tässä vaiheessa emme kokeneet järkeväksi alkaa luomaan raporttia, jota asiakkaat eivät välttämättä käytä tai tarvitse. Tämän lisäksi mahdolliset jatkokehityskohteet liittyvät esimerkiksi nyt kerätyn datan visualisointiin eri paikoissa. Monet jo järjestelmässä olevat ominaisuudet voisivat hyödyntää resursointityökalun keräämää dataa. Toisaalta jotkut jo järjestelmässä olevat ominaisuudet, kuten laitteelle määritetyt käyttöajat, voisi yhdistää resursointityökaluun, jolloin laitteen käyttöaikojen ulkopuolelle ei esimerkiksi voisi tehdä varauksia ollenkaan. Tämän lisäksi tulemme yhdistämään resursointityökalun joihinkin tuleviin ominaisuuksiimme.

7 Yhteenveto

Insinöörityössä toteutettiin resursointityökalu GSGroup Finland Oy:n kehittämään ja ylläpitämään Paikannin.com-järjestelmään. Se on kaluston paikantamiseen sekä hallintaan tarkoitettua järjestelmä, jonka avulla pystyy esimerkiksi optimoimaan laitteiden käyttöä. Uuden ominaisuuden tärkeimmät vaatimukset olivat asiakkaan laitteiden käytön suunnittelun tehostaminen, laitteiden reaaliaikaisen varaustilanteen tarkastelu sekä se, että ominaisuus sopii erityyppisten asiakkuuksien käyttöön.

Työssä toteutettu työkalu ohjelmoitiin Aurelia-ohjelmistokehystä käyttäen, jolla järjestelmän käyttöliittymä on muutenkin ohjelmoitu. Varauskalenterista oli tarkoitus luoda helposti uudestaan käytettävä komponentti. Se ei kuitenkaan onnistunut, sillä sen parametrisointi olisi ollut turhan monimutkaista verrattuna uuden komponentin luontiin.

Resursointityökalu täyttää sille annetut vaatimukset hyvin, vaikka joitain ominaisuuksia, kuten raportointi, päätettiinkin jättää tässä vaiheessa toteuttamatta. Työkalu antaa GSGroup Finland Oy:n asiakkaille mahdollisuuden sekä suunnitella laitteiden käyttöä etukäteen että tarkastella laitteiden sen hetkistä

varaustilannetta. Työkalu sopii myös hyvin erilaisten asiakkaiden käyttöön. Nämä olivat uuden työkalun tärkeimmät tavoitteet, joten työ vastaa sille annettuja tärkeitä vaatimuksia. Tämän lisäksi järjestelmää käyttävät asiakkaat pystyvät halutessaan yhdistää myös kuljettajan kirjautumisen uuteen resursointityökaluun. Lopulliset tulokset saamme kuitenkin vasta, kun tarpeeksi moni järjestelmän käyttäjä on kokeillut uutta ominaisuutta sekä antanut siitä palautetta.

Uuteen työkaluun on jo nyt jonkin verran jatkokehitysideoita. Näitä ovat esimerkiksi varauskalenterin kautta kerättävän datan visualisointi muissa järjestelmän ominaisuuksissa sekä raportoinnin toteuttaminen, kunhan asiakkaamme kertovat, minkälaisia raportteja he haluavat.

Lähteet

1 Kulmala, Marko & Sjövik, Maria. 2014. PPCT – aina ajan hermolla. Insano.

2 PostgreSQL-kotisivut. Verkkoaineisto. <https://www.postgresql.org>. Luettu 9.10.2022.

3 Docker-kotisivut. Verrkoaineisto. <https://www.docker.com/resources/what-container>. Luettu 11.10.2022.

4 Wallenius, Niklas. 2022. Mikä on Kubernetes ja mitä hyötyä siitä on? Verkkoaineisto. <https://niklaswallenius.fi/kubernetes/>. Luettu 6.10.2022.

5 Mohamed, Rania. 2019. Kubernetes Cluster vs Master Node. Verkkoaineisto. <https://www.suse.com/c/kubernetes-cluster-vs-master-node>. Luettu 11.10.2022.

6 Reactin GitHub-sivusto. Verkkoaineisto. <https://github.com/facebook/react>. Luettu 18.10.2022.

7 Vuen GitHub-sivusto. Verkkoaineisto. <https://github.com/vuejs/vue>. Luettu 18.10.2022.

8 Aurelian GitHub-sivusto. Verkkoaineisto. <https://github.com/aurelia/framework>. Luettu 18.10.2022.

9 I Like Kill Nerds. Why you should choose aurelia over react (mostly). Verkkoaineisto. < https://ilikekillnerds.com/2021/07/why-you-should-choose-aurelia-over-react/>. Luettu 18.10.2022.

10 Introduction. Vue-kotisivut. Verkkoaineisto. <https://vuejs.org/guide/introduction.html>. Luettu 20.10.2022.

11 FullCalendar-kotisivut. Verkkoaineisto. <https://fullcalendar.io>. Luettu 31.10.2022.

12 Event Calendar & Scheduler for Mobile and Desktop. Mobiscroll-kotisivut. Verkkoaineisto. <https://mobiscroll.com/event-calendar-scheduler>. Luettu 31.10.2022.

13 Flutter Calendar Widget - Flexible Event Scheduler. Syncfusion-kotisivut. Verkkoaineisto. <https://www.syncfusion.com/flutter-widgets/flutter-calendar>. Luettu 31.10.2202.

14 Input Validation: Client-Side & Server-Side Cybersecurity Deterrent. Verkkoaineisto. <https://www.securecoding.com/blog/input-validation/>. Luettu 22.10.2022.

15 Mobiscroll. Scheduler demot. Verkkoaineisto. <https://demo.mobiscroll.com/javascript/scheduler/desktop-week-view#>. Luettu 22.10.2022.

16 Unix-aika. Linux kotisivut. Verkkoaineisto. <https://www.linux.fi/wiki/Unix-aika>. Luettu 22.10.2022.