Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Veera Pohjola

Ratatyökoneiden paikkatietorajapinta

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

14.10.2018

Tekijä Otsikko Sivumäärä Aika	Veera Pohjola Ratatyökoneiden paikkatietorajapinta 32 sivua 14.10.2018
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikan tutkinto-ohjelma
Ammatillinen pääaine	IoT and Cloud Computing
Ohjaajat	IT Service Manager Jari Pentti yliopettaja Markku Nuutinen

Insinöörityön tavoitteena oli kehittää rautatiealan yritykselle ratatyökoneiden paikkatietojärjestelmä, jonka avulla työkoneiden tekemiä työsuoritteita voi tarkastella kartalla. Järjestelmän tulisi havaita myös suoritteetta liikkuvat työkoneet. Toiminnallisuuksien toteuttamista varten järjestelmän tulisi tallentaa työkoneiden paikkatietohistoriaa ja yhdistää sitä työsuoritteiden historiatietoihin. Insinöörityö toteutettiin käyttäen jatkuvan integraation toimintamallia.

Insinöörityö tehtiin käyttäen Node.js-ympäristöä ja PostgreSQL-tietokantaa. Tietokantaan tallennettiin tasaisin väliajoin työkoneiden paikkatietoa. Näin saatiin kerättyä työkoneiden paikkatietohistoriaa. Karttatoteutusta varteen kehitettiin rajapinta, joka vastaa lähetettyihin kyselyihin verkkosivulla. Verkkosivu sisältää kartan, jossa on esitetty työkoneiden liikkeitä ja tietoja kyselyparametrien mukaan. Rajapintaan sisältyi myös ominaisuus, joka palauttaa haetut paikkatiedot JSON-objekteina jatkokäyttöä varten.

Insinöörityön lopputuloksena syntyi järjestelmä, jossa työkoneiden tekemiä työsuoritteita pystyy tarkastelemaan kartalla. Tehtyjä suoritteita voi tarkastella reaaliajassa tai hakea erikseen tietyltä aikaväliltä. Järjestelmä seuraa myös koneiden reaaliaikaista tilaa ja lähettää ilmoituksia ilman suoritetta liikkuvista työkoneista. Järjestelmä helpottaa työsuoritteiden seurantaa ja tarkistusta sekä toimii apuna toiminnan suunnittelussa.

Avainsanat	paikkatietorajapinta, Node.js, PostgreSQL, karttarajapinta

Author Title Number of Pages Date	Veera Pohjola Location API for Railway Maintenance Vehicles 32 pages 14 October 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	IoT and Cloud Computing
Instructors	Jari Pentti, IT Service Manager Markku Nuutinen, Principal Lecturer

The goal of this thesis was to develop a tracking system for railway maintenance vehicles. The system was made for a Finnish railway infrastructure company. The system has three main purposes. First, the system needs to store vehicle location data. Secondly, the system needs to track the state of the maintenance vehicles and find out which vehicles are moving without a work status. Last, the system must combine vehicle location and work data to display it on a map.

Continuous integration practices, part of the DevOps-model used in the company, were used when developing the system. The functionality of the system was made using PostgreSQL database and two Node.js processes. One process is dedicated to fetching and saving latest location data to the database. The other process acts as an API between the user and the database.

The result was a fully operational system. The system uses external API to fetch latest location data for the vehicles. The latest location data is stored to a database. The vehicle location data can be combined successfully to corresponding work data. When a user sends a request to the API with parameters that specify the wanted information, a web page consisting of a map or a text-based JSON response is sent back. The system can also keep track of the vehicles that move without a work status. When such incident occurs, an alarm is generated.

The system can be used to inspect and verify maintenance work that has been done with the vehicles. The work status of the vehicles can be inspected in real time or past time. It is possible to use the system to make an operations center for the company and to assist in business planning.

Keywords	API, location data, map-based API, Node.js, PostgreSQL

Sisällys

Lyhenteet

1 Johdanto 1

2 Järjestelmän kuvaus 2

2.1 Yleiskuvaus 2
2.2 Toimintaperiaate 3
2.3 Yhteydet muihin järjestelmiin 3

3 Käytetyt menetelmät 5

3.1 Jatkuva integraatio 5
3.2 Yksikkötestaus 5

4 Käytetyt teknologiat 6

4.1 PostgreSQL 6
4.2 JavaScript 6
4.3 Node.js 7
4.4 Git-versionhallinta 7

5 Järjestelmän toteutus 8

5.1 Järjestelmän toteutuksen yleiskuvaus 8
5.2 Karttarajapinnan käyttäjäkokemus 9
5.3 Haasteet tiedon yhdistämisessä eri rajapinnoista 10
5.4 Tietokannan indeksointi 16

6 Rajapinnan kuvaus 16

6.1 Rajapinnan rakenne 16
6.2 Tekstirajapinta 17
6.3 Karttarajapinta 19
6.3.1 Karttarajapinnan tarjoamat palvelut 21
6.3.2 OpenLayers-kirjasto 28

7 Suoritteetta liikkuvien koneiden havaitseminen 29

8 Yhteenveto 30

Lähteet 31

Lyhenteet

ES	EcmaScript. Skriptauskielen määritelmä.
HTML	Hypertext Markup Language. Verkkosivujen ohjelmointiin käytetty kieli.
HTTPS	Hypertext Transfer Protocol Secure. Internet-protokolla jota käytetään suojattuun tiedonsiirtoon.
JSON	Javascript Object Notation. Yleinen tiedonsiirtoformaatti.

1 Johdanto

Insinöörityön tavoitteena oli kehittää rautatiealalla toimivalle yritykselle ratatyökoneiden paikkatietojärjestelmä. Järjestelmän tehtävänä on tallentaa työkoneiden paikkatietoa, yhdistää siihen työsuoritteita ja tarjota tietoa eteenpäin visuaalisessa muodossa ja perinteisenä rajapintana. Järjestelmän tulee myös havaita ja ilmoittaa, jos kone liikkuu ilman työsuoritetta. Kehitettävä järjestelmä on osa tilaajalla meneillään olevaa työsuoritteiden digitalisointiprojektia.

Tilaajalla on käytössä paikkatietojärjestelmä, josta löytyy työkoneiden uusin paikkatieto. Järjestelmä ei kuitenkaan tallenna paikkatietohistoriaa. Työkoneiden työsuoritteiden kirjaus siirrettiin uuteen sähköiseen järjestelmään hiljattain. Siirron myötä nousi tarve uudelle järjestelmälle, joka yhdistäisi työsuoritetietoja työkoneiden paikkatietoon. Päätettiin kehittää järjestelmä, joka sekä tallentaa työkoneiden paikkatietohistoriaa että jakaa tietoa eteenpäin rajapintana. Rajapinnan tärkein ominaisuus on sijainti- ja työsuoritetiedon yhdistäminen. Rajapinta tarjoaa myös karttanäkymän, jonka avulla yhdistetty sijainti- ja työsuoritetieto esitetään käyttäjäystävällisessä muodossa.

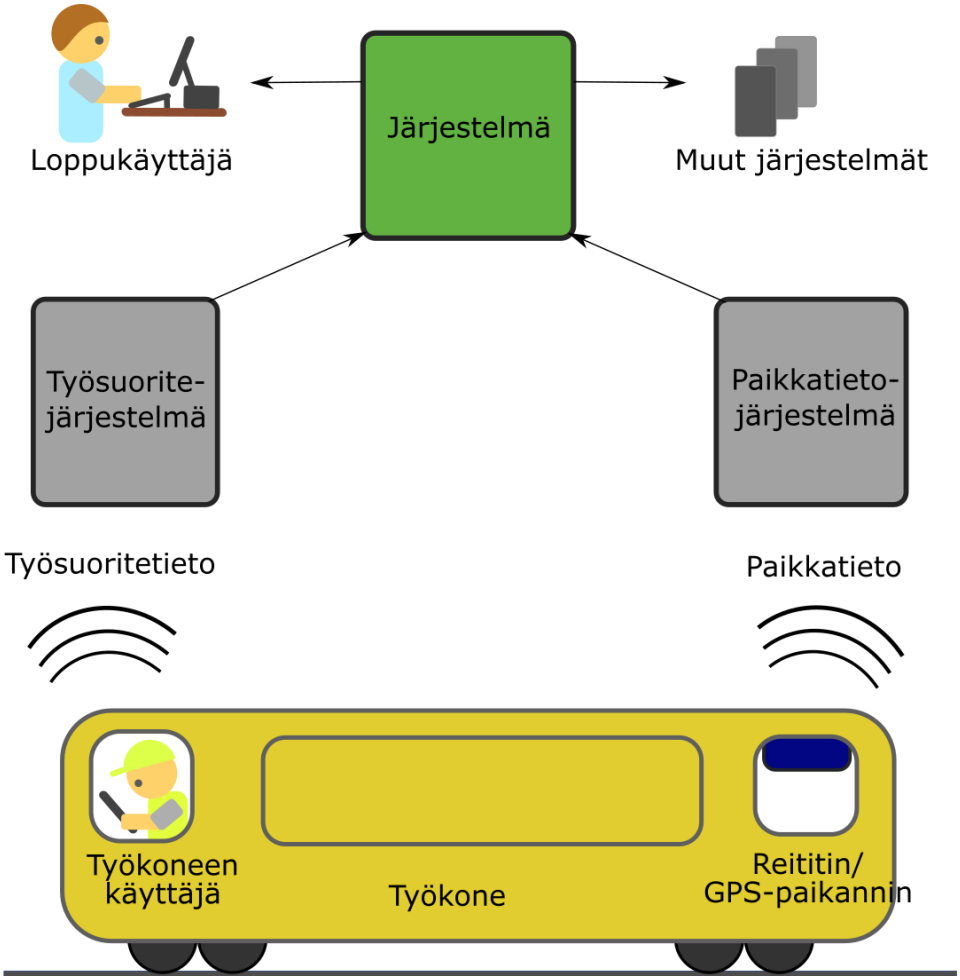
Tilaajan mukaan (1, 2) järjestelmä helpottaa tehtyjen työsuoritteiden seurantaa ja tarkistusta. Järjestelmästä saatua tietoa voidaan käyttää työkoneiden siirtojen parempaan suunnitteluun. Järjestelmästä saatuja tietoja voidaan hyödyntää myös töiden parempana raportointina tilaajan asiakkaiden suuntaan. Tilaaja voi toteuttaa toimintaa ohjaavan operaatiokeskuksen järjestelmän avulla.

Olin edellisenä projektinani tehnyt tilaajalle selainpohjaisen karttasovelluksen, jolla voi seurata junakaluston liikkeitä ja tarkastella erilaisia rataelementtejä. Tämän järjestelmän kehittäminen tuntui siten luontevalta seuraavaksi projektiksi. Työ toteutettiin käyttäen tilaajalla käytössä ollutta DevOps-toimintamallia.

2 Järjestelmän kuvaus

2.1 Yleiskuvaus

Järjestelmän tehtävänä on tallentaa työkoneiden sijaintitietoa ja esittää sijainti- sekä työsuoritehistoriaa kartalla. Järjestelmä hyödyntää toiminnassaan työkoneiden paikkatietoa ja työkoneiden tekemien suoritteiden kirjaustietoja. Paikkatieto kerätään työkoneissa olevista reitittimistä. Reitittimissä on paikannin, joka lähettää sijaintitietoa tasaisin väliajoin, kun työkone on käynnissä. Sijaintitiedot lähetetään ulkoisen toimittajan palvelimelle. Toimittajan palvelin tarjoaa vain koneiden uusimman sijaintitiedon. Isommissa työkoneissa on paikannin molemmissa päissä ja pienemmissä paikannin on vain koneen yhdessä ohjaamossa. Kuvassa 1 on esitetty järjestelmän yleiskuvaus.



Kuva 1. Järjestelmän yleiskuvaus. Työkoneesta saatu paikkatieto lähetetään ulkoiseen paikkatietojärjestelmään, ja koneen käyttäjän kirjaama työsuorite työsuoritejärjestelmään.

Työsuoritetiedot kerätään sovelluksesta, jonne työkoneiden käyttäjät kirjaavat tehdyt työt. Sovelluksessa on rajapinta, jonka avulla tiedot haetaan. Rajapinta tarjoaa työsuoritetietohistoriatietoja ja reaaliaikaista työsuoritteiden seurantaa. Työsuoritesovellus ja rajapinta ovat toisen ulkoisen toimittajan tekemiä.

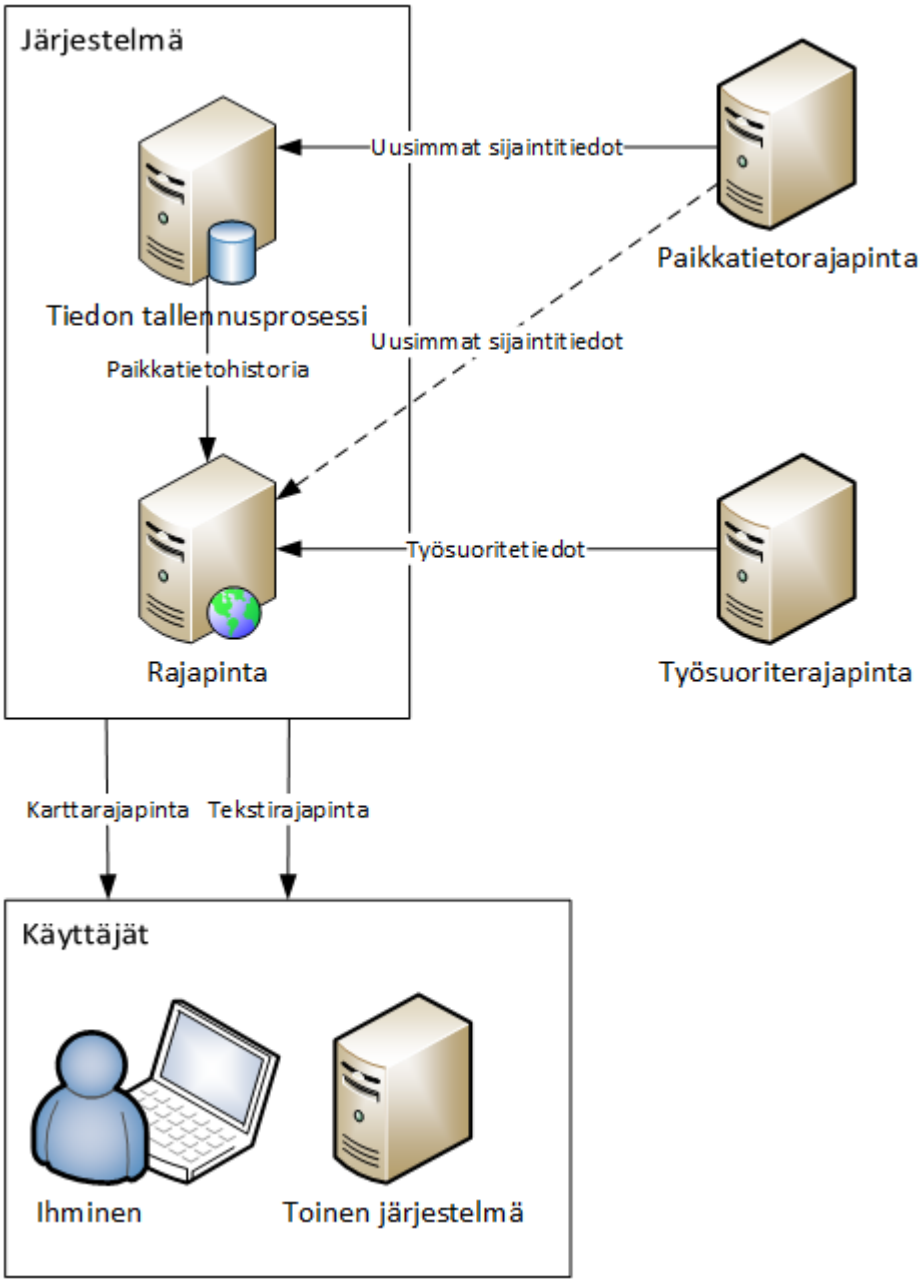
2.2 Toimintaperiaate

Järjestelmä on jaettu kahteen osaan: tiedon tallennukseen ja rajapintaan. Tiedon tallennuksen hoitaa Node.js-komentosarja. Komentosarja hakee uusimman paikkatiedon ulkoisesta järjestelmästä, erottelee vastauksen osiin ja tallentaa osat tietokantaan. Tämä toistuu kymmenen sekunnin välein.

Rajapinta on oma prosessinsa. Se on WWW-palvelin, joka tarjoaa tietokannasta haettua dataa teksti- ja verkkosivumuodossa. Rajapintaan lähetetään Hypertext Transfer Protocol Secure (HTTPS) -kysely joko verkkoselaimella tai ohjelmallisesti. Kyselyyn liitetään mukaan kyselyparametrit, joissa tarkennetaan, mitä tietoa halutaan hakea. Palvelimella kyselyparametrit erotellaan ja niiden mukaan koostetaan kysely tietokantaan. Tietokantakyselyn tulokset lähetetään joko sellaisenaan tai muokkausten kera takaisin käyttäjälle. Rajapintaprosessi hoitaa myös suoritteetta liikkuvien koneiden tarkkailun.

2.3 Yhteydet muihin järjestelmiin

Järjestelmä ei itse tuota dataa, vaan käyttää toiminnassaan kahden ulkoisen rajapinnan tietoja. Työkoneiden paikkatietohistoriaa varten ulkoisen paikkatietorajapinnan tiedot tallennetaan järjestelmään. Työsuoriterajapinnan tiedot taas haetaan tarvittaessa ja yhdistetään tallennettuun paikkatietoon. Järjestelmien väliset yhteydet on näytetty kuvassa 2.



Kuva 2. Järjestelmien väliset yhteydet

Järjestelmän rajapinta käyttää suoraan ulkoista paikkatietorajapintaa tarjotessaan tiedon koneiden uusimmista sijainneista. Muissa tapauksissa paikkatiedot haetaan järjestelmän omasta rajapinnasta.

3 Käytetyt menetelmät

Projekti tehtiin käyttäen DevOps-toimintamallia ja siihen kuuluvaa jatkuvan integraation menetelmää ja työkaluja. Tässä luvussa keskitytään jatkuvan integraation toimintamalleihin.

3.1 Jatkuva integraatio

Jatkuvan integraation tapa tehdä sovelluksia on integroida tehtyjä muutoksia jatkuvasti versionhallintaan ja testata muutokset virheiden varalta. Työprosessi on syklinen. Projektin tapauksessa paikallisesti koodiin tehdyt muutokset yksikkötestataan ensin. Jos testit menevät läpi, muutokset viedään versionhallintajärjestelmään. Jatkuvan integraation tehtävänä on löytää lähdekoodin virheet nopeammin, lyhentää sovelluksen testaukseen käytettävää aikaa ja parantaa sovelluksen laatua. Jatkuvan integraation mallia käytettäessä tavoitteena on vakaa järjestelmä, jossa virheet nousevat nopeasti esille yksikkötestien ansiosta (3; 4 s. 2.)

3.2 Yksikkötestaus

Yksikkötestauksessa koodin eri osia testataan ennalta määrätyllä syötteellä, ja verrataan toteutunutta ulostuloa odotettuun ulostuloon. Yksikkötestausta käytetään yksittäisten aliohjelmien, eli yksiköiden, toimintaa muuttavien virheiden havaitsemiseen. Yksikkötestauksella varmistetaan myös, ettei jo kerran korjattu virhe pääse livahtamaan uudestaan lähdekoodia muutettaessa (5, s. 25–26.)

Projektissa käytin yksikkötestejä paitsi testaamaan järjestelmän aliohjelmien oikeaa toimintaa myös todentamaan rajapinnan ulostulon oikeellisuutta. Tämä helpotti ja nopeutti järjestelmän kehittämistä huomattavasti. Jokaisen muutoksen jälkeen pystyin automaattisesti tarkistamaan, ettei rajapinnan ulostulo muuttunut. Ilman testausta tarkistus olisi pitänyt tehdä manuaalisesti joka ikinen kerta, mikä olisi ollut hidasta ja virhealtista.

4 Käytetyt teknologiat

Insinöörityön teossa käytetyt teknologiat määräytyivät pitkälti tilaajan käytössä olevien teknologioiden mukaan. Järjestelmä koostuu Node.js-prosesseista ja käyttää PostgreSQL-tietokantaa.

4.1 PostgreSQL

Järjestelmä käyttää PostgreSQL-tietokantaa paikkatietojen säilytykseen. PostgreSQL on avoimen lähdekoodin olio-relaatiotietokantapalvelin. PostgreSQL yhdistää perinteisten ja uusien tietokannan hallintajärjestelmien ominaisuudet. Perinteisiä ominaisuuksia ovat esimeriksi transaktiot ja triggerit. Käyttäjän määrittelemät tyypit ja periytyminen taas ovat ominaisuuksia, joita ei perinteisestä tietokannan hallintajärjestelmästä löydy. PostgreSQL tukee nykyaikaisia Unix-pohjaisia alustoja sekä Windows NT -pohjaisia käyttöjärjestelmiä (6.)

4.2 JavaScript

JavaScript on oliopohjainen, heikosti tyypitetty alustariippumaton komentosarjakieli. JavaScript on yksi kolmesta verkkosivujen tekoon tarvittavasta kielestä. Siinä missä HTML (Hypertext Markup Language) määrittää verkkosivujen sisällön ja CSS (Cascading Style Sheets) niiden ulkoasun, JavaScript tekee sivuista interaktiiviset ja muokkaa sivun käyttäytymistä. Selaimen lisäksi JavaScriptiä käytetään myös palvelinpuolella. JavaScript-verkkopalvelin käsittelee HTTPS-pyyntöjä ja -vastauksia objekteina, joita muokkaamalla voidaan luoda dynaamisia verkkosivuja (7; 8.) JavaScriptin standardia ECMAScriptiä (ES) ylläpitää ECMA International (9, s. iii).

Järjestelmän teossa käytettiin ainoastaan JavaScriptiä. Palvelinpuolella käytössä on Node.js, ja selainpuolella karttanäkymä luodaan erillisellä JavaScript-kirjastolla. Kirjastosta on kerrottu tarkemmin luvussa 6.3.2.

4.3 Node.js

Node.js on asynkroninen JavaScript-kielen ajoympäristö. Se sai alkunsa, kun JavaScriptin kehittäjät laajensivat kielen käyttömahdollisuuksia pelkästä selainohjelmasta omaksi itsenäiseksi ohjelmaksi (10.) Node.jsn etuna onkin, että samaa JavaScriptiä voidaan käyttää sekä palvelin- että selainpuolella (11, s. xxiii). Sen avainominaisuuksia ovat hyvä skaalautuvuus, asynkroninen sisään- ja ulostulon käsittely ja yksisäikeinen tapahtumaohjattu arkkitehtuuri. Node.jsää voi käyttää esimerkiksi verkkosovelluksiin, sovelluspalvelimiin, verkkopalvelimena tai asiakasohjelmana (12, kappale 1.)

Node Package Manager (NPM) on Node.js-pakettienhallintajärjestelmä. Node.js:n mukana tulee asennettaessa vain perusominaisuudet. Perusominaisuuksia voi laajentaa kolmannen osapuolen avoimen lähdekoodin paketeilla. Paketeista löytyy toiminnallisuuksia laidasta laitaan: aikavyöhykkeiden muunnoksista esineiden internetin kanssa toimimiseen. Suosituimpia paketteja kirjoitushetkellä ovat esimerkiksi HTTP-pyyntöjen reititykseen ja käsittelyyn tarkoitettu express sekä käyttöliittymien rakentamiseen tarkoitettu react. Node.jsn oman pakettienhallintajärjestelmän avulla pakettien asennus on haluttu tehdä helpoksi (11 s. 65; 13.)

Projektin toiminnallisuus toteutettiin kokonaan Node.js:n avulla. Laaja pakettivalikoima helpotti järjestelmän kehitystä. Valikoimasta löytyivät paketit niin tietokannan kanssa asiointiin kuin verkkopalvelimen pystytykseen ja rajapintapyyntöjen käsittelyyn. Rajapinta pyörii Node.js-verkkopalvelimella, ja tietojen tallennus tietokantaan hoituu Node.js-komentosarjan avulla.

4.4 Git-versionhallinta

Versionhallintajärjestelmää käytetään tiedostojen muutosten seuraamiseen eli tiedoston eri versioiden hallintaan. Versionhallinnasta on hyötyä esimerkiksi tiedoston kadotessa tai virhetilanteessa, jolloin aiempi, toimiva versio tiedostosta voidaan palauttaa. Projektin versionhallinta toteutettiin Gitillä.

Git on avoimeen lähdekoodiin perustuva hajautettu versionhallintajärjestelmä. Git sai alkunsa vuonna 2005, kun Linuxin kehitysyhteisössä nousi tarve kehittää oma

versionhallintajärjestelmä Linuxin ytimen versionhallintaa varten. Gitin kehityksessä tavoitteena oli kehittää nopea ja yksinkertainen kokonaan hajautettu versionhallintajärjestelmä, joka pystyisi käsittelemään Linuxin ytimen kaltaisia isoja projekteja tehokkaasti.

Perinteisessä, keskitetyssä versionhallinnassa tiedostojen muutoksia seurataan yhdellä palvelimella. Palvelimen rikkoutuessa versionhallintaa ei kuitenkaan pysty käyttämään, ja varmuuskopioiden puuttuessa versionhallinnan kaikki tiedot on menetetty. Git sen sijaan käyttää hajautettua versionhallintaa. Hajautetussa versionhallinnassa kopio koko kuvauskannasta (repository) on palvelimen lisäksi kaikilla kuvauskannan asiakkailla. Palvelimen rikkoutuessa asiakkaan kopiota voidaan käyttää versionhallinnan tietojen palautukseen. Näin versionhallinnan tiedot ovat varmemmassa tallessa (14, s. 9–13.)

5 Järjestelmän toteutus

Järjestelmän toteutuksesta kerrotaan yleisluontoisesti luvussa 5.1. Toteutusprosessin aikana esiin nousseita yksittäisiä kehityskohteita ja haasteita on kuvattu tarkemmin luvuissa 5.2–5.4. Luvussa 5.3 esitetyt esimerkkikoodit eivät ole suoria otteita järjestelmässä käytetystä koodista.

5.1 Järjestelmän toteutuksen yleiskuvaus

Järjestelmän toteuttaminen kesti puoli vuotta. Aiemman kokemuksen perusteella valitsin Node.js-ympäristön käytettäväksi myös tässä projektissa. Ympäristö oli jo ennestään tuttu, joten sain projektin nopeasti käyntiin.

Aloitin projektin teon tutustumalla ja tutkimalla eri tietokantavaihtoehtoihin. Lopulta päädyin valitsemaan PostgreSQL:n. Valintaan vaikuttivat tilaajan tarjoamat tietokantavaihtoehdot, PostgreSQL:n antama suora ulostulo objektimuodossa ja laaja käyttäjäkunta sekä kattava dokumentaatio.

Tietokannan valinnan jälkeen kehitin yksinkertaisimman toimivan tuotteen. Se oli käytännössä oikean tietokantatietueen palauttava nettisivu, kun kyselyparametrina oli tietueen tunniste. Siitä ryhdyin pala palalta laajentamaan toiminnallisuuksia, kunnes

perustoiminnot olivat kasassa: yhden koneen sijaintitietojen haku annetulla aikavälillä, yksittäisen koneen yksittäisen työsuoritteen yhdistäminen paikkatietoon, yhden koneen viimeisimmän paikkatiedon haku annetulle ajanhetkelle. Opettelin myös luomaan yksikkötestejä koodin testaamiseen. Tämän jälkeen rupesin rakentamaan karttarajapintaa. Karttarajapintaa rakentaessa opin tekemään palvelinpuolelle JavaScript-moduuleita, jotta pystyin käyttämään samoja tietokantahaun aliohjelmia molempiin rajapintoihin. Karttarajapinnan valmistuttua rupesin viilailemaan rajapinnan ominaisuuksia. Lisäsin moneen toimintoon mahdollisuuden hakea tietoa useasta koneesta samaan aikaan ja toiminnon näyttää monta työsuoritetta kartalla. Lisäykset vaativat laajalti koodin uudelleenjärjestelyä. Uudelleenjärjestelyn aikana huomasin, kuinka tärkeää koodin automaattinen yksikkötestaus on.

Aivan lopuksi loin toiminnallisuuden suoritteetta liikkuvien työkoneiden havaitsemista varten. Toiminnallisuus hyödyntää rajapinnan käytössä olevia sisäisiä funktioita, joten koin luonnolliseksi sisällyttää sen rajapintaprosessin sisään oman prosessin sijasta.

5.2 Karttarajapinnan käyttäjäkokemus

Karttarajapinnan käyttäjäkokemuksesta on pyritty tekemään intuitiivinen ja helppo alusta alkaen. Kaikissa kyselyissä kartta mukauttaa automaattisesti näkymänsä niin, että kaikki kohteet näkyvät kartalla. Työsuoritenäkymässä eri suoritteet erottuvat toisistaan väreillä, joten käyttäjä voi yhdellä vilkaisulla saada yleiskuvan tehdyistä töistä. Suurien työsuoritekyselyiden suorittaminen voi olla hidasta. Mahdollisen hitauden vuoksi sivun vielä latautuessa näytetään latausanimaatiota, jotta käyttäjä kuitenkin tietää sivun latautuvan. Hahmottamisen helpottamiseksi sekä työ- että sijaintihistorianäkymässä sijainnin alku ja loppu on merkitty lipulla. Käyttäjäkokemusten perusteella sijaintihistorianäkymään lisättiin väriskaala, joka auttaa myös hahmottamaan koneen liikkeitä tarkemmin.

Jotta kartan vasteaika olisi mahdollisimman lyhyt, näytettyjen sijaintipisteiden määrä säätyy kartan suurennustason mukaan. Mitä lähemmäs suurennetaan, sitä enemmän näytetään pisteitä. Vasteajan pienentämiseksi muutin myös karttatasojen tyypin vektorista kuvaksi.

Karttarajapinnan toimintaa kuvataan tarkemmin luvussa 6.3, jossa esitetään myös kuvia rajapinnan palauttamista karttanäkymistä.

5.3 Haasteet tiedon yhdistämisessä eri rajapinnoista

Yksinkertaisissa kyselyissä, kuten koneen yksittäistä paikkatietoa tai viimeisintä paikkatietoa hakiessa, tietokannan oma ulostulo on riittävä. Monimutkaisemmissa kyselyissä, kuten työsuoritteiden paikkatietoa hakiessa, täytyy tietokannan ulostuloa jalostaa. Työsuoritetiedot haetaan ulkoisesta rajapinnasta, josta ne tulevat työvuoroittain. Jokaisesta työvuorosta käydään läpi jokaisen työprojektin jokainen työsuorite. Yksittäisten työsuoritetietojen alku- ja loppuajankohdilla haetaan sisäisestä rajapinnasta sijaintihistoriatietoja. Ne yhdistetään työsuoritetiedon kanssa uudeksi objektiksi. Nämä yhdistelmäobjektit kootaan lopuksi yhteen objektilistaksi ja lähetetään selainpuolelle.

Haastavinta järjestelmän teossa oli sen tärkein ominaisuus: työsuoritteiden esitys kartalla. Haaste johtui pitkälti siitä, että olin alussa valinnut huonosti tarkoitukseen sopivan tavan yhdistää sijainti- ja suoriterajapinnan tietoja. Ongelman ymmärtämiseksi työvuoro-objektin rakenne on hyvä havainnollistaa. Esimerkkikoodissa 1 kuvataan pelkistetysti työvuoro-objektin rakenne.

```
{
  projects: [
        {
            tucs:[
                {
                    suorite: "Tukeminen"
                },
                {
                    suorite: "Siirto"
                }
            ]
        }
    ]
}
```

Esimerkkikoodi 1. Työvuoro-objektin pelkistetty rakenne.

Alkuperäisessä tavassa ensin haettiin työvuoro annetulta aikaväliltä. Työvuoron ensimmäisen työprojektin sisältämät suoritteet käytiin läpi for-silmukalla ja samalla haettiin jokaiselle suoritteelle sijaintitiedot. Työvuoron sisältämät suoritteet ja suoritteiden sijaintitiedot talletettiin kuitenkin eri objektilistoihin. Tuloksena tuli siis lista, jossa oli

kronologisessa järjestyksessä työsuoritteet, ja lista, jossa oli suoritteita vastaavat paikkatiedot. Työsuorite-listan ensimmäisen alkion sijaintitiedot löytyivät siis paikkatieto-listan ensimmäisestä alkiosta. Tämä toimi hyvin, kunnes tajusin, että samaan työvuoroon voi kuulua monta työprojektia ja haetulla aikavälillä voi olla monta työvuoroa. Vanhalla ratkaisulla olisi ollut haastavaa yrittää selaimella yhdistää kahta pitkää listaa, joissa alkioiden järjestys on ainoa yhdistävä tekijä.

Ongelman ratkaisi uudenlainen lähestymistapa sijaintitiedon ja suoritetiedon yhdistämiseen. Suoritetta ja sijaintia ei talleteta eri listoihin. Kun suoritteen sijainti on haettu, muodostetaan uusi objekti. Objektiin talletetaan työsuorite, paikkatieto ja suoritteeseen liittyvä työprojekti. Näin suorite ja paikkatieto kulkevat samassa paketissa yhdistettynä. Haetun aikavälin jokaisen työvuoron jokaisen työprojektin jokaisesta suoritteesta tehty uusi objekti talletetaan listaan. Kun kaikki suoritteet on käyty läpi, lista lähetetään selaimelle.

Suorite- ja paikkatietojen yhdistäminen

Toinen haaste suorite- ja paikkatietojen yhdistämisessä oli JavaScriptin asynkroninen luonne. Useimmissa ohjelmointikielissä, esimerkiksi Pythonissa ja C++:ssa, koodin suoritus tapahtuu synkronisesti järjestyksessä niin, että seuraavaa koodiriviä ei voi suorittaa ennen kuin edellinen rivi on ajettu ja valmis. JavaScriptissa asia ei aina ole niin. Jos edellisellä rivillä kutsutaan funktiota, joka hakee tietoa tietokannasta, seuraavan rivin suoritus alkaa, vaikka tietokantahaku ei olisi valmis. Tämä on ongelmallista, kun tietokannasta haettua tietoa halutaan käsitellä koodissa myöhemmin (15.)

```
function getLocation(starttime, endtime, machineidentifier, callback){
   /*
    Tässä välissä haetaan tietokannasta tiedot
    */
   return databaseresponse
}
var machinelocation= getLocation(start, end, machine);
console.log(machinelocation);
// machinelocation ei ole määritetty
```

Esimerkkikoodi 2. Tietokantahaku on asynkroninen, joten machinelocation-muuttujan arvoa ei ole määritetty.

Esimerkkikoodissa 2 on esitetty asynkronisen funktionkutsun ongelma muuttujaan tallennettaessa. Asynkronisuuden vuoksi `console.log(machinelocation)`

suoritetaan ennen kuin tietokantahaku `getLocation` on valmis, joten `machinelocation`-muuttujan arvoa ei ole määritetty.

Nopean ratkaisun ongelmaan toivat vastakutsufunktiot (callback). Vastakutsufunktio on funktio, jota kutsutaan, kun siihen liittyvä suoritettava funktio on valmis. Se annetaan suorittavalle funktiolle parametrinä (15.) Vastakutsufunktiolla on kaksi parametriä: virhe ja vastaus. Virheparametrin arvo on tyhjä ja vastausparametri sisältää funktion palauttaman vastauksen, jos funktion suoritus on onnistunut odotetusti. Virhetilanteessa virheparametri sisältää tiedon virheestä ja vastausparametrin arvo on tyhjä. Tietokantahakuun yhdistettynä vastakutsufunktiota kutsutaan siis haun valmistuttua, jolloin vastausparametri sisältää haun tulokset. Vastakutsufunktion avulla toteutettu tietokantahaku on esitetty koodiesimerkissä 3.

```
function getLocation(starttime, endtime, machineidentifier, callback){
    /*
    Tässä välissä haetaan tietokannasta tiedot
    */
   //Jos haun aikana tulee virhe:
   return callback("Tietoa ei voitu hakea",null);
   //Jos kaikki meni kuten pitikin:
   return callback(null, databaseresponse);
}
getLocation(start, end, machine, function (error, machinelocation){
    if(error){
        // Tietokantahaun aikana on käynyt virhe
    }else{
        // Machinelocation sisältää nyt tietokannan vastauksen
        console.log(machinelocation);
    }
});
```

Esimerkkikoodi 3. Tietokantahaku vastakutsufunktiolla. Haun tuloksia voidaan nyt käsitellä vastakutsufunktiossa.

Vastakutsufunktioiden käyttö toimi alussa, kun koodia ei ollut paljon. Kehityksen edetessä koodini alkoi kuitenkin sortumaan JavaScriptiä vaivaavaan ”callback hell” -ilmiöön, jossa vastakutsufunktioiden ketjuttaminen teki koodin rakenteen ja logiikan hahmottamisesta haasteellista (15.) Huomasin, etten itsekään enää pysynyt perässä koodin toimintalogiikasta. Hetken pähkäiltyäni löysin ratkaisun JavaScriptin uusista ominaisuuksista: lupauksista ja async/await-toiminnallisuudesta.

Lupaus on olio, joka kuvastaa tulevaisuudessa suoritettavan toiminnon arvoa. Toimintoa suoritettaessa lupaus joko täytetään tai hylätään. Kun lupausolio on luotu, sen perään

voidaan liittää then-funktio käsittelemään lupauksen palauttamaa arvoa. Lupausolion käyttö on kuvattu koodiesimerkissä 4.

```
function getLocation(starttime, endtime, machineidentifier, callback){
    return new Promise((resolve, reject) => {
        /*
        Otetaan yhteys tietokantaan
        */
       //Jos haun aikana tulee virhe:
       return reject("Tietoa ei voitu hakea");
       //Jos kaikki meni kuten pitikin:
       return resolve(databaseresponse);
    })
}
getLocation(start, end, machine)
//machinelocation sisältää tietokantahaun tulokset
.then(machinelocation => console.log(machinelocation))
.catch(error => console.log("Virhe: ",error))
```

Esimerkkikoodi 4. Lupausolion käyttö

Lupausten käytössä on etuja vastakutsufunktioiden käyttöön nähden, kuten then-funktioiden ketjuttaminen. Rajapinnan tapauksessa ketjuttaminen on hyödyllistä, kun tietokannasta haetaan työkonetietoja ja haettujen tietojen mukaan haetaan uusia tietoja. Ketjuttamisessa on myös se hyöty, että jokaiselle then-funktiolle ei tarvita omaa virheenkäsittelijää, vaan yhdellä catch-funktiolla voidaan käsitellä kaikkien then-funktioiden virheet (16.) Koodiesimerkissä 5 on havainnollistettu lupausten ketjuttaminen.

```
getLatestWorkshift(start, end, machine)
.then(workshift => getLocation(workshift.start, workshift.end, workshift.ma-
chine))
.then(locationsforworkshift=> console.log("Koneen sijainnit viimeisimmän
työvuoron aikana: ", locationsforworkshift))
.catch(error => console.log(error));
```

Esimerkkikoodi 5. Lupausten ketjuttaminen. ”getLatestWorkshift” on myös lupauksen palauttava funktio.

Ensin tietokannasta haetaan uusin työvuoro koneelle. Työvuoron tiedot palautetaan objektissa workshift. Workshift-objektista erotetut työvuoron alku- ja lopetusajat sekä koneen tunniste annetaan parametrinä seuraavalle funktiolle, joka hakee koneen sijaintitiedot annetulle aikavälille. Koneen sijaintihistoria työvuoron ajalta on täten haettu, ja koodi toimii, jos tarkoituksena on hakea vain yhden koneen sijaintitiedot.

Rajapinnan tapauksessa range- ja work-palveluiden pitää kuitenkin hakea peräkkäin monen koneen tiedot saman kyselyn aikana, sillä rajapintakyselyssä voi määritellä useita

koneita. Kysellyt koneet käydään läpi yksi kerrallaan for-silmukan sisässä, jolloin niille haetaan sijaintitietohistoria.

```
for (var machine in machinelist) {
     getLatestWorkshift(start, end, machine)
     .then(workshift => doSomethingWith(workshift)
     .catch(error => console.log(error));
}
```

Esimerkkikoodi 6. Funktion kutsu for-silmukan sisässä. Vaikka getLatestWorkshift on lupauksen palauttava funktio, ei silmukan suoritus pysähdy odottamaan funktion palauttamaa vastausta.

Vaikka koodiesimerkin 6 koodi vaikuttaakin synkroniselta, for-silmukka ehtii käydä kaikki iteraationsa läpi ennen kuin `getLatestWorkshift` on suorittanut toimintonsa ja palauttanut tietoa. Silmukka ei jää odottamaan, että sen sisällä kutsuttu funktio on valmis. Täten myöskään lupaukset eivät ehdi palauttamaan vastauksiaan, eli tietokantahaun tulokset eivät tallennu. Törmäsin siis samaan ongelman kuin alussa, mutta hieman eri kontekstissa. Onneksi ratkaisu oli lähellä.

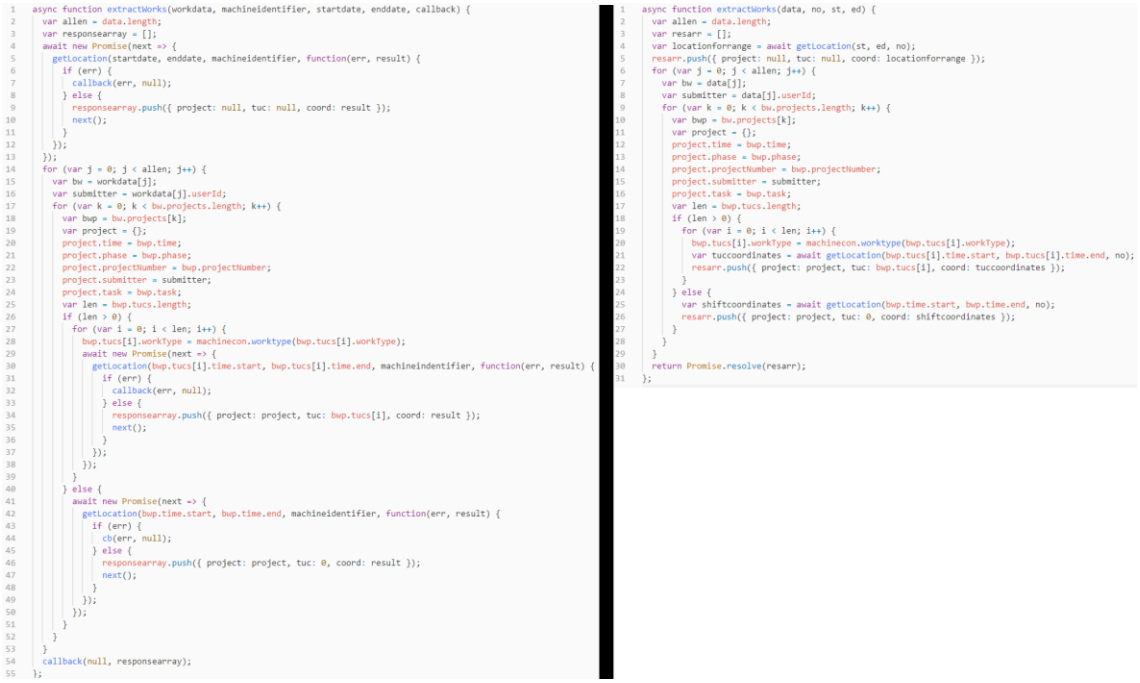
Yhdistämisen ratkaisuna async/await-ominaisuus

Javascript-standardin ES2017-julkaisussa mukaan lisättiin async/await-ominaisuus, joka helpottaa asynkronisten operaatioiden kanssa toimimista. Sanaparin ensimmäinen osa ”async” tarkoittaa asynkronista funktiota. Asynkroninen funktio luodaan kuten normaali funktio, mutta funktion alkuun lisätään ”async”. Lupausten ja vastakutsufunktioiden tapaan asynkroninen funktio ei keskeytä koodin suorittamista. Itseasiassa asynkroninen funktio palauttaa lupauksen. Jos funktiota suoritettaessa käy virhe, lupaus hylätään. Asynkronisen funktion hyödyt nousevat todella esiin, kun käyttöön otetaan sanaparin toinen puolisko, ”await”. Await-ilmaisu voidaan laittaa lupausta käyttävän funktiokutsun eteen. Asynkronisen funktion suoritus keskeytyy await-ilmaisun kohdalla odottamaan kutsutun funktion palauttamaa lupausta. Awaitin avulla asynkronisia operaatioita voi suorittaa synkronisesti. Awaitin avulla esimerkiksi tietokantahakua voi kutsua for-silmukan sisällä ja tallentaa tulokset muuttujaan. Silmukan suoritus keskeytyy, kunnes tietokantahaku on valmis (17.)

```
for (var machine in machinelist) {
      try{
            var workshift = await getLatestWorkshift(start, end, machine);
            doSomethingWith(workshift);
      }catch(error){
            console.log(error);
      }
}
```

Esimerkkikoodi 7. Funktion kutsu for-silmukan sisässä käyttäen hyödyksi await-ilmaisua. Silmukan suoritus keskeytyy, kunnes getLatestWorkshift on palauttanut vastauksen.

Kuten esimerkkikoodista 7 nähdään, async/await-paria käyttämällä koodin rakennetta voi huomattavasti yksinkertaistaa, ja sen toiminnan hahmottaminen on helpompaa. Await-ilmaisua käytettäessä virheenkäsittely tapahtuu try/catch-menetelmällä. Kuvassa 3 on esitetty rinnakkain työsuoritteiden ja sijaintitiedon yhdistävän funktion rakenne ennen ja jälkeen async/awaitin käyttöönottoa.



```
async function extractWorks(workdata, machineidentifier, startdate, enddate, callback) {
  var allen = data.length;
  var responsearray = [];
  await new Promise(next => {
    getLocation(startdate, enddate, machineidentifier, function(err, result) {
      if (err) {
        callback(err, null);
      } else {
        responsearray.push({ project: null, tuc: null, coord: result });
        next();
      }
    });
  });
  for (var j = 0; j < allen; j++) {
    var bw = workdata[j];
    var submitter = workdata[j].userId;
    for (var k = 0; k < bw.projects.length; k++) {
      var bwp = bw.projects[k];
      var project = {};
      project.time = bwp.time;
      project.phase = bwp.phase;
      project.projectNumber = bwp.projectNumber;
      project.submitter = submitter;
      project.task = bwp.task;
      var len = bwp.tucs.length;
      if (len > 0) {
        for (var i = 0; i < len; i++) {
          bwp.tucs[i].workType = machinecon.worktype(bwp.tucs[i].workType);
          await new Promise(next => {
            getLocation(bwp.tucs[i].time.start, bwp.tucs[i].time.end, machineidentifier, function(err, result) {
              if (err) {
                callback(err, null);
              } else {
                responsearray.push({ project: project, tuc: bwp.tucs[i], coord: result });
                next();
              }
            });
          });
        }
      } else {
        await new Promise(next => {
          getLocation(bwp.time.start, bwp.time.end, machineidentifier, function(err, result) {
            if (err) {
              cb(err, null);
            } else {
              responsearray.push({ project: project, tuc: 0, coord: result });
              next();
            }
          });
        });
      }
    }
  }
  callback(null, responsearray);
};
```

```
async function extractWorks(data, no, st, ed) {
  var allen = data.length;
  var resarr = [];
  var locationforrange = await getLocation(st, ed, no);
  resarr.push({ project: null, tuc: null, coord: locationforrange });
  for (var j = 0; j < allen; j++) {
    var bw = data[j];
    var submitter = data[j].userId;
    for (var k = 0; k < bw.projects.length; k++) {
      var bwp = bw.projects[k];
      var project = {};
      project.time = bwp.time;
      project.phase = bwp.phase;
      project.projectNumber = bwp.projectNumber;
      project.submitter = submitter;
      project.task = bwp.task;
      var len = bwp.tucs.length;
      if (len > 0) {
        for (var i = 0; i < len; i++) {
          bwp.tucs[i].workType = machinecon.worktype(bwp.tucs[i].workType);
          var tuccoordinates = await getLocation(bwp.tucs[i].time.start, bwp.tucs[i].time.end, no);
          resarr.push({ project: project, tuc: bwp.tucs[i], coord: tuccoordinates });
        }
      } else {
        var shiftcoordinates = await getLocation(bwp.time.start, bwp.time.end, no);
        resarr.push({ project: project, tuc: 0, coord: shiftcoordinates });
      }
    }
  }
  return Promise.resolve(resarr);
};
```

Kuva 3. Työsuoritteen ja sijaintitiedon yhdistävän funktion rakenne ennen async/awaitin käyttöönottoa (vasemmalla) ja käyttöönoton jälkeen (oikealla).

Pystyin async/awaitin käytön avulla lyhentämään rivimäärän lähes puoleen entisestä, ja funktion rakenne yksinkertaistui huomattavasti. Kuvassa vasemmalla olevan, funktion vanhan version kirjoittamisen aikaan aloin vasta opettelemaan lupauksien käyttöä, jonka vuoksi käytin niitä kuin vastakutsufunktioita. Tämä johti ”callback hell” -ilmiöön, vaikka käytinkin lupauksia. Ilmiölle on tyypillistä oikealle lisääntyvien sisennysten muodostama

kolmio, joka syntyy, kun funktioiden kutsuja kertyy sisäkkäin. Koodin rakenne on hankala hahmottaa. Uusi versio funktiosta oikealla on litteämpi sisennysten suhteen ja siistimpi sekä ulkoasultaan että logiikaltaan. Koodin toiminnallisuus on kuitenkin pysynyt samana.

5.4 Tietokannan indeksointi

Toteutuksen loppupuolella tutustuin tietokannan indeksointiin. Yleensä indeksointi nopeuttaa tietokantaa lukevia SELECT- ja WHERE-kyselyitä. Näin ei kuitenkaan välttämättä ole sellaisen tietokannan tapauksessa, jossa uutta dataa lisätään usein (18.)

Koska järjestelmä hakee ja lisää sijaintitietoa taulukkoon kymmenen sekunnin välein, ajattelin, ettei indeksointi sopisi järjestelmässä käytettyyn tietokantaan. Kokeilin kuitenkin indeksointia, ja tulokset olivat todella hyviä. Tietokantahakuihin kulunut aika pieneni yli puolella. Esimerkiksi testitapauksena käyttämääni viiden koneen työsuoritteiden hakuun vuorokauden ajalta kului ennen indeksointia 8–9 sekuntia, mutta indeksoinnin jälkeen vain noin 4 sekuntia. Hakujen nopeuttaminen on yksi avaintekijöistä rajapinnan käyttäjäkokemuksessa.

6 Rajapinnan kuvaus

6.1 Rajapinnan rakenne

Rajapinta jakautuu kahteen osaan: teksti- ja karttarajapintaan. Osiin perehdytään tarkemmin luvuissa 6.2 ja 6.3.

Rajapinta toimii verkkopalvelimella, joka vastaa sille lähetettyihin pyyntöihin. Tietoa haetaan pyyntöjen mukana toimitettavien kyselyparametrien mukaan. Rajapinnassa käytettyjä parametrejä ovat aloitus- ja lopetusaika sekä työkoneen yksilöllinen tunnus.

6.2 Tekstirajapinta

Tekstirajapinta tarjoaa paikkatietorajapinnan palvelut tekstimuotoisina vastauksina. Rajapinnan tiedot on tarkoitettu laitteiden väliseen viestintään tai jatkojalostukseen, ei niinkään suoraan ihmisten käytettäväksi. Taulukossa 1 on kuvattu tekstirajapinnan tarjoamat palvelut ja kyselyiden mukana toimitettavat parametrit.

Taulukko 1. Tekstirajapinnan tarjoamat palvelut.

Kyselyn nimi	**Parametrit**	**Kuvaus**
List	Koneen tunniste, palautettavan listan koko	Palauttaa listan paikkatieto-objekteja kysytyn työkoneen viimeisimmistä sijainneista. Listan kokoa voi säätää erikseen lähettämällä kyselyn mukana parametrin, jonka arvo on haluttu listan koko. Oletusarvoisesti palautetaan lista, jossa on kymmenen viimeisintä paikkatieto-objektia haetulle koneelle.
Latest		Palauttaa kaikkien saatavilla olevien koneiden viimeisimmän työskentelystatuksen ja paikkatiedon.
Time	Ajanhetki, koneen tunniste	Palauttaa kronologisesti seuraavan paikkatieto-objektin annetulle koneelle annettuna ajanhetkenä. Jos konetta ei ole määritetty, palautetaan kymmenen lähintä paikkatieto-objektia mille tahansa koneelle annettuna ajanhetkenä. Kyselyä käytetään latest-kyselyn tietojen muodostuksessa, kun haetaan koneen statusta lähinnä oleva sijaintitieto.
Range	Ajanjakso, koneen tunniste	Palauttaa listan paikkatieto-objekteista annetulle koneelle annetulla ajanjaksolla. Jos konetta ei ole määritetty, palautetaan kaikkien koneiden sijainnit annetulta ajanjaksolta
Work	Ajanjakso, koneen tunniste	Palauttaa annetun koneen sijaintihistorian tehtyjen työsuoritteiden kanssa annetulla ajanjaksolla. Jos koneita on useita, palauttaa kaikkien koneiden työsuoritteet ja paikkatieto-objektit

Vastauksen formaatti on rajapinnoissa yleisesti käytetty JSON, eli JavaScript Object Notation. Nimestään huolimatta JSON on ohjelmointikieliriippumaton. JSON on tarkoitettu nimenomaan tiedon vaihtoon eri järjestelmien välillä. JSON-formaattiin kuuluu kaksi eri rakennetta: objekti ja lista. Objekti muodostetaan nimi/arvopareista, joissa nimi ja arvo

erotetaan kaksoispisteellä, ja parit toisistaan pilkulla. Rajapintavastauksen perusformaatti on työkoneen paikkatieto-objekti, joka on esitetty koodiesimerkissä 8.

```
{
      "id":"2018-07-16 11:08:47Testikone",
      "datetime":"2018-07-16T08:08:47.000Z",
      "name":"Testikone",
      "speed":"0.00",
      "coordx":"60.86465643",
      "coordy":"26.69451697",
      "rotation":"0"
}
```

Esimerkkikoodi 8. Paikkatieto-objekti, jollaisista rajapinnan palauttamat vastaukset pääasiassa koostuvat.

Listalla tarkoitetaan listaa dataa, jossa jokainen listan jäsen on hakasulkeiden sisässä, ja jäsenet erotetaan toisistaan pilkulla. Objekti voi sisältää listan, ja lista voi sisältää objekteja (9, s. iii,1,3.) Rajapinnassa edellä mainitusta on suuri hyöty, sillä monimutkaisetkin tietorakenteet voidaan kuvata yhdistelmänä objekteja ja listoja. Koodiesimerkissä 9 on esimerkki tekstirajapinnan palauttamasta objektista, joka sisältää paikkatiedon ja työsuoritteen.

```
{
      "coord": [{},{},{}],
      "project":{
            "projectNumber":"1234",
            "phase": "1234"
      },
      "tuc":{
            "workType": 1,
            "workSubType": 0
      }
}
```

Esimerkkikoodi 9. Rajapinnan palauttama vastaus, jossa on kuvattu työsuorite ja siihen liittyvät paikkatieto-objektit. Paikkatieto-objektit esitetään tässä tyhjinä objekteina tilan säästön vuoksi, mutta niiden rakenne on esitetty koodiesimerkissä 8.

Esimerkissä nähdään kolme eri objektia: paikkatieto (coords), työprojekti (project) ja suoritteet (tucs). Coords-objekti sisältää listan, jonka jäsenet ovat paikkatieto-objekteja. Project-objekti sisältää tietoa työsuoritteeseen kuuluvasta työprojektista. Tuc-objekti sisältää tietoa tehdystä työsuoritteesta.

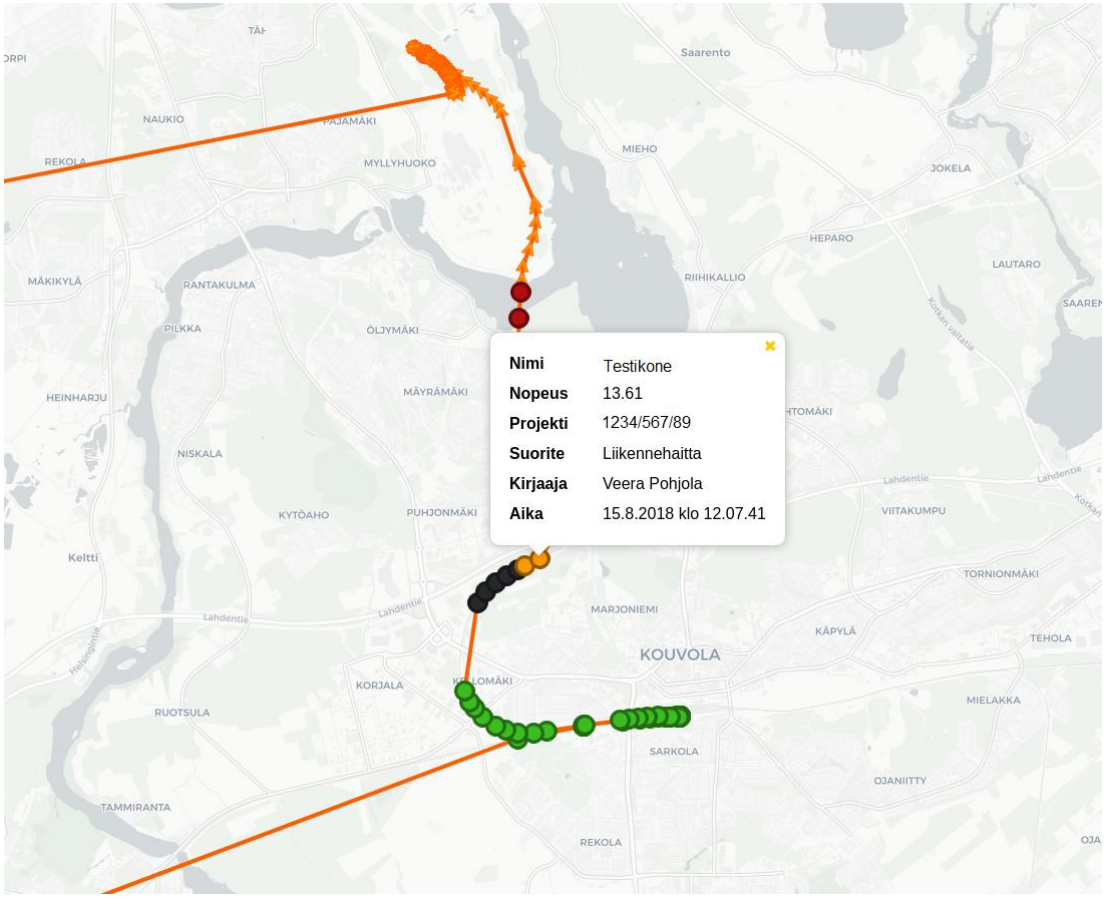
6.3 Karttarajapinta

Karttarajapinta on palvelinpuolella lähes identtinen tekstirajapinnan kanssa, lukuun ottamatta parametrien pakollisuuseroja ja muutamaa kyselyä. Rajapintojen ero nouseekin esiin selainpuolella. Siinä missä tekstirajapinnan ulostulo on lista tekstimuotoista dataa, tarjoaa karttarajapinta verkkosivun, jossa sama data on lisätty kartalle. Näin rajapinnan tietoja voidaan hyödyntää suoraan loppukäyttäjän ymmärtämässä muodossa.

Kartalla esitettävä tieto vaihtelee kyselyehdoista riippuen, mutta visualisointinäkökulmasta se voidaan jakaa kahteen eri luokkaan: ”pistemäiseen” ja ”yhdistettävään” muotoon. ”Pistemäinen” tieto tarkoittaa, että kartalla esiintyvät yksittäiset pisteet eivät ole suoraan yhteydessä toisiinsa. Esimerkiksi työkoneiden tämänhetkistä tilaa tai tiettyjen koneiden paikkatietoa tietyllä ajanhetkellä näyttävät kartat kuuluvat tähän, kuten on esitetty kuvassa 4.

Kuva 4. Toisistaan riippumatonta paikkatietoa. Esimerkkinä latest-kysely. Yksittäiset koneet on kuvattu valkoisella kuvakkeella.

Kartalla näytetään siis yksi tai useampi paikkatieto, jotka eivät suoraan riipu toisistaan. ”Yhdistettävä” tieto taas viittaa tapauksiin, jossa kartalla olevien paikkatietopisteiden järjestyksellä ja yhteydellä on merkitystä.



Kuva 5. Yhdistettävää paikkatietoa. Esimerkkinä work-kysely.

Tällaisia ovat työsuoritetieto, kuten kuvassa 5, ja sijainnin historiatieto. Molemmissa on oleellista, että kartalle tuleva paikkatieto on oikeassa (kronologisessa) järjestyksessä, sillä paikkatietopisteet yhdistetään toisiinsa.

Kaikissa karttarajapinnan vastauksissa kartalla näkyviin objekteihin lisätään metatietoa. Metatiedon saa esille objektia klikkaamalla. Metatieto sisältää tiedon ainakin työkoneen nimestä, nopeudesta ja sijainnin kellonajasta. Lisätietoja ovat työsuorite, työprojekti, koneen viimeisin tapahtuma ja työvuoron alkamisaika. Karttaobjektien väriä ja muotoa voidaan muokata metatietojen mukaan. Esimeriksi työsuoritetta näytettäessä eri tyyppiset työsuoritteet näytetään eri värein. Tämän ansiosta käyttäjä voi nopealla vilkaisulla saada yleiskuvan koneen tekemistä töistä. Lisäksi tehollinen työ on jaettu omalle tasolleen, jolloin käyttäjä voi halutessaan tarkastella vain tehtyä tehollista työtä tietyllä aikavälillä. Kuva 6 havainnollistaa metatiedon esitystä kartalla.

Kuva 6. Metatiedon esitys kartalla. Esimerkkinä latest-kysely.

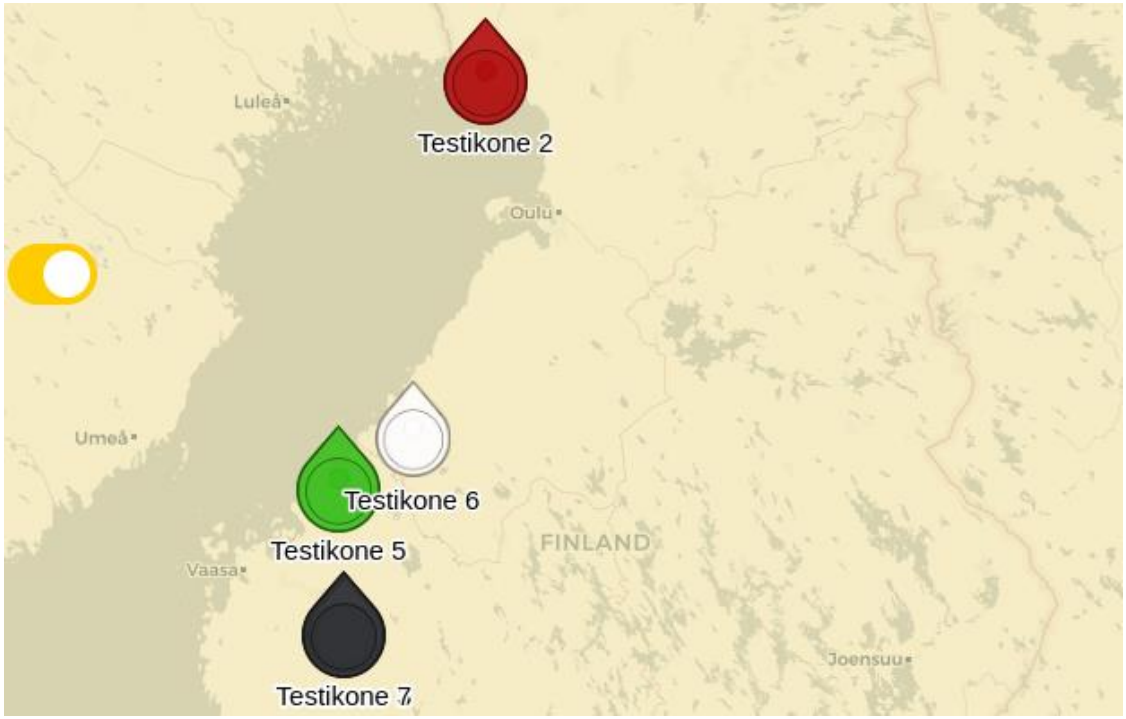
Metatieto lisätään kartalle lisättäviin karttaobjekteihin kartan luomisvaiheessa. Karttaa napauttaessa haetaan napautetun pikselin kohdalla oleva karttaobjekti. Karttaobjektin sisältämät metatiedot käydään kohta kohdalta läpi, ja niistä muodostetaan dynaamisesti HTML-taulukko. Taulukko syötetään tyhjään emoelementtiin, tässä tapauksessa kartalla näkyvään puhekuplaan. Jos kartalla näkyvää tietoa pitää päivittää sivun latautumisen jälkeen, kuten esimeriksi latest-sivun tapauksessa, korvataan vanhan tiedon sisältämät karttaobjektit uusilla. Kartan toimintaperiaatteesta kerrotaan lisää luvussa 6.3.1.

6.3.1 Karttarajapinnan tarjoamat palvelut

Karttarajapinta tarjoaa muutamia erilaisia kyselyitä halutun datan esittämiseksi kartalla. Kyselyparametrejä muuttamalla voidaan rajata tai suodattaa haluttua dataa.

Latest

Latest-kysely palauttaa kartan, jossa on kaikkien saatavilla olevien koneiden sijainti viimeisimmän työskentelystatuksen mukaan. Kyselyyn ei tarvita parametrejä. Kuvassa 7 on esitetty latest-kyselyn palauttama karttanäkymä.



Kuva 7. Ote latest-kyselyn palauttamasta karttanäkymästä.

Koneet, jotka eivät ole aktiivisina, näytetään valkoisina. Suoritteita tekevät koneet näytetään suoritteen mukaisella värillä. Sivulla on liukusäädin, jonka aktivoimalla kartta tallentaa työkoneiden liikkeitä ja aktiivisuutta sivun ollessa auki. Näin työkoneiden tekemiä suoritteita ja tilaa voidaan seurata reaaliajassa, kuten on näytetty kuvassa 8.

Kuva 8. Työkoneen tilan seuranta reaaliajassa latest-näkymän liukusäätimen avulla. Tilan vaihdos näkyy työkoneen paikkatietohistoriaa kuvaavien palloikonien värin vaihdoksena.

Työkoneen tämänhetkinen sijainti merkitään isohkolla ympyrällä, jossa on nokka. Työkoneen sijaintihistoria merkitään pienemmillä ympyröillä, joiden väri kuvastaa työkoneen tilaa sijaintihetkellä.

Time

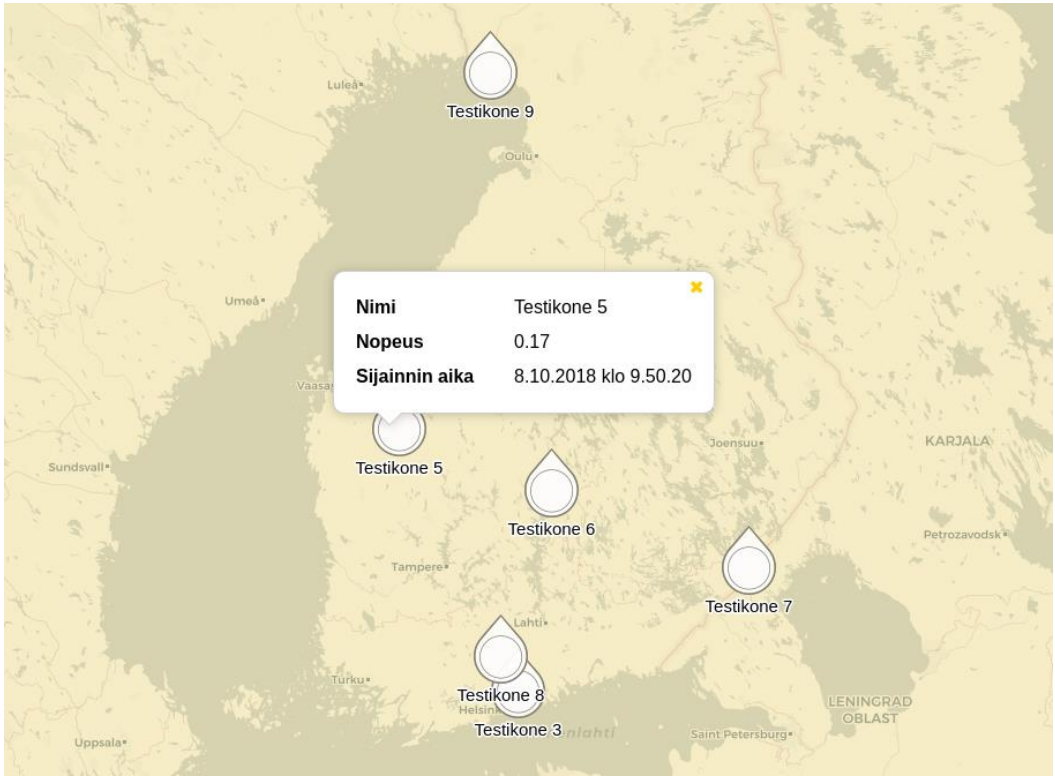
Time-kysely palauttaa kronologisesti seuraavan paikkatiedon annetulle koneelle annettuna ajanhetkenä. Tekstirajapinnasta poiketen sekä koneen nimi että ajanhetki ovat pakollisia parametrejä. Kuvassa 9 on esitetty time-kyselyn palautama karttanäkymä.

Kuva 9. Time-kyselyn palauttama karttanäkymä, kun testikoneelle on haettu lähintä sijaintia ajanhetkelle 15.7.2018 kello 11:35.

Kysely on tarkoitettu käytettäväksi, kun yksittäisen koneen sijainti lähellä tiettyä ajanhetkeä halutaan tietää. Rajapinnan käyttäjä ei voi tietää työkoneen sijainnin tarkkaa aikaleimaa, joten on oltava keino, jolla rajapinta palauttaa lähimmän sijaintitiedon annetulle hetkelle.

Newest

Newest-kysely palauttaa annettujen koneiden viimeisimmän sijainnin ilman työsuoritetietoja. Kuvassa 10 on esitetty newest-kyselyn palauttama karttanäkymä.

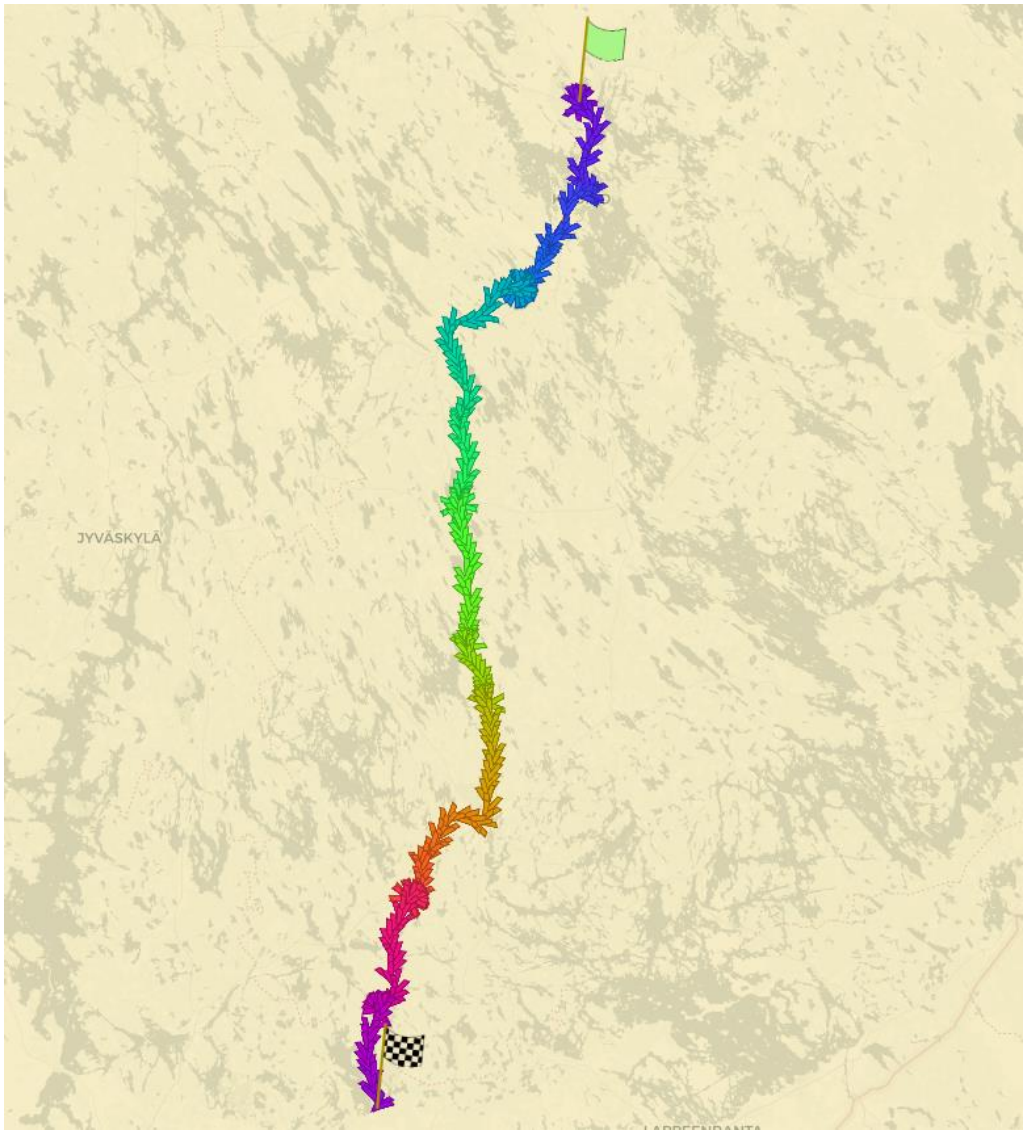


Kuva 10. Newest-kyselyn palauttama karttanäkymä.

Kyselyn avulla voidaan tarkastella haluttujen koneiden uusinta sijaintia. Kyselyä voidaan käyttää suodattimen tavoin, kun halutaan tietää, missä tietyt koneet sijaitsevat juuri nyt.

Range

Range-kysely palauttaa annetun koneen sijaintihistorian annetulla ajanjaksolla. Kuvassa 11 on esitetty range-kyselyn palauttama karttanäkymä.

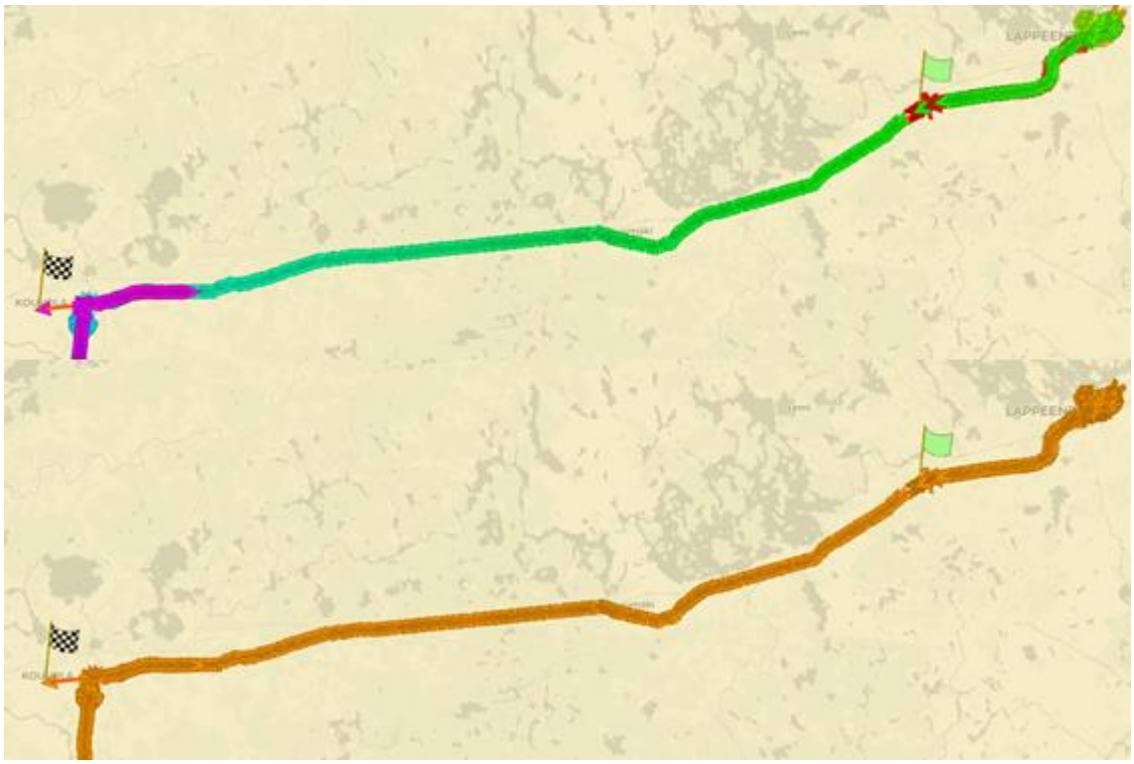


Kuva 11. Range-kyselyn palauttama karttanäkymä. Työkone on aloittanut matkansa vihreän lipun kohdalta ja päättänyt sen ruudulliseen lippuun.

Sijaintihistoria esitetään kartalla viivana, jossa on nuolia näyttämässä työkoneen kulkeman matkan suuntaa. Hahmottamisen helpottamiseksi matkan alku- ja loppupää on merkitty lipuin. Nuolta napauttamalla esitetään tietoa kyseisestä sijaintipisteestä.

Sijaintihistoria sisältää usein liikkeitä samoissa paikoissa. Nuolissa on liukuvärjäys, jotta samassa paikassa pyörineen koneen kulkema matka on helpompi hahmottaa. Kuvan 12 oikeassa yläreunassa näkyy, kuinka kone on ensin lähtenyt oikealle (punaiset nuolet), ja tullut lopulta samaa reittiä takaisin (vihreät nuolet). Samoin kuvan 12 vasemmassa

reunassa näkyy, kuinka kone on kääntynyt alas (siniset nuolet) ja palannut takaisin ylös (violetit nuolet).

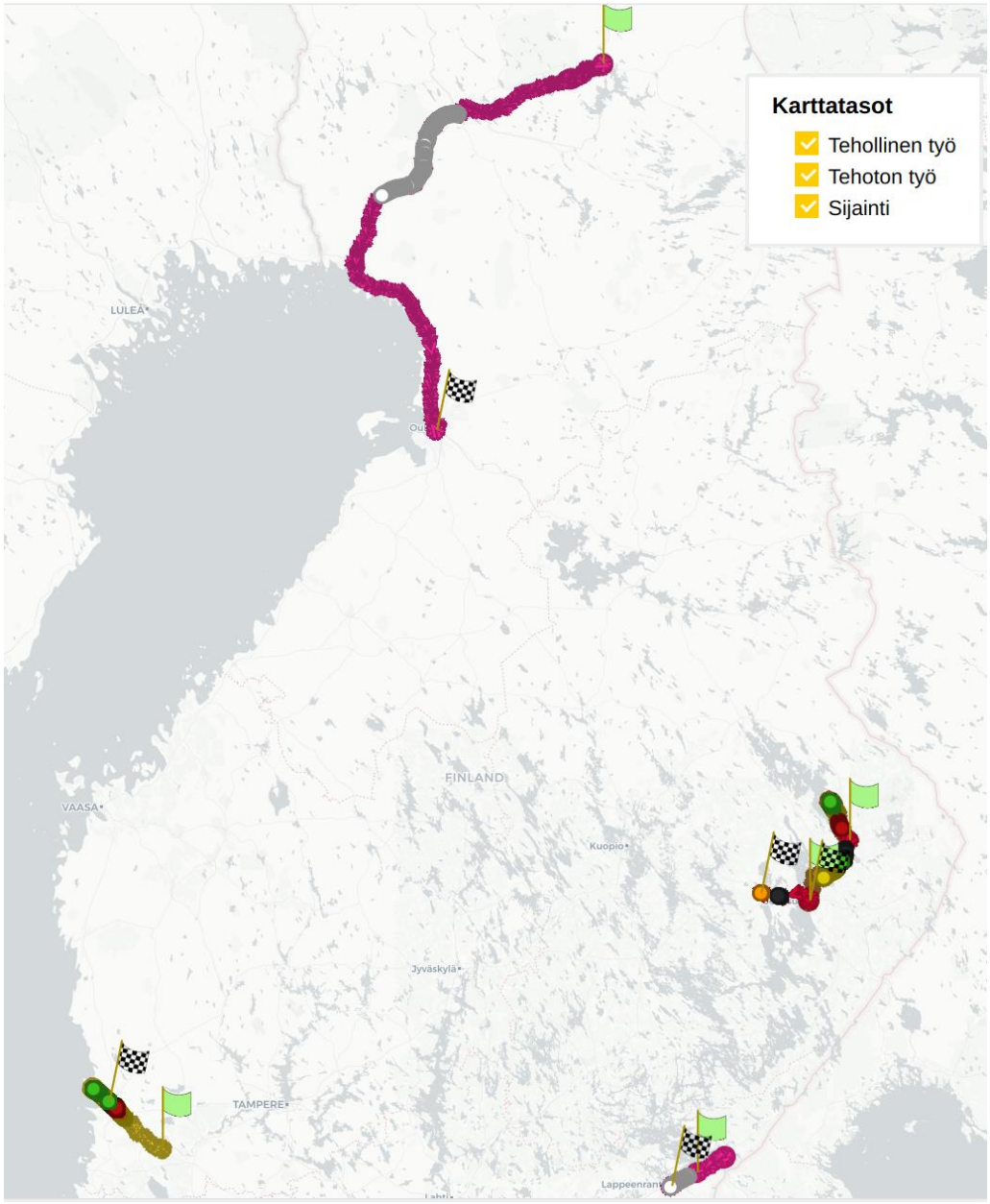


Kuva 12. Koneen kulkema matka värjättynä (yllä) ja yksivärisenä (alla). Liukuvärjäyksen avulla on helpompi havaita kohdat, joissa kone on liikkunut edestakaisin.

Kuvan 12 alaosassa esitetyssä yksivärisessä reitissä yllä mainittuja liikkeitä on huomattavasti vaikeampi hahmottaa.

Work

Work-kysely palauttaa karttanäkymän, jossa on esitetty annettujen koneiden sijaintihistoria tehtyjen työsuoritteiden kanssa annetulla ajanjaksolla. Näkymää voi suodattaa karttatasovalikolla. Work-kyselyn karttanäkymä on esitetty kuvassa 13.



Kuva 13. Work-kyselyn palauttama karttanäkymä.

Koneiden kulkemat kokonaismatkat on esitetty range-kyselyn tapaan nuoliviivoilla, joiden päissä on liput. Koneiden tekemät suoritteet on merkitty nuoliviivojen päälle pistein. Pisteiden väri kuvastaa tehtyä suoritetta. Suoritepistettä tai sijaintinuolta napauttamalla saa esiin joko suoritteeseen tai sijaintiin liittyviä tietoja.

6.3.2 OpenLayers-kirjasto

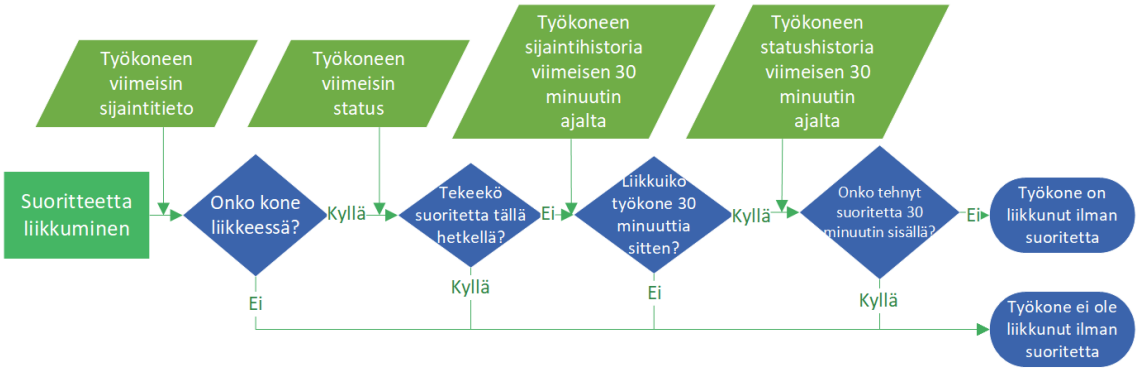
Karttarajapinta käyttää OpenLayers-kirjastoa tekstitiedon näyttämiseksi kartalla. OpenLayers on Open Source Geospatial Foundationin ylläpitämä avoimen lähdekoodin projekti (19). Kirjaston ominaisuuksia ovat tuki uudelle HTML5-kielelle, rasteritaustakartoille ja vektoritasoille (20). OpenLayersin avulla voi koostaa interaktiivisen kartan verkkosivulle. Kirjastoa voi projektin verkkosivujen (21,22) mukaan käyttää joko palvelinpuolella Node.js-pakettina tai perinteiseen tapaan selainpuolella JavaScript-kirjastona.

Kirjastoa käytettäessä kartta jaetaan objekteiksi. Jokaisella kartalla on karttaobjekti. Karttaobjekti sisältää vähintään kohde-elementin, karttatason ja näkymän. Kohde-elementillä tarkoitetaan verkkosivulta löytyvää HTML-elementtiä, jossa kartta halutaan näyttää. Karttatasoja on eri tyyppisiä, mutta minimissään ainakin taustakarttataso tarvitaan. Karttatasoilla on aina lähdeobjekti. Lähdeobjekti sisältää tasolla näytettävän datan, ja karttataso määrittää, miten data näytetään. Näkymä määrittelee myös kartan ohjaamiseen liittyvät asiat, kuten kartan projektion (23.)

7 Suoritteetta liikkuvien koneiden havaitseminen

Yksi järjestelmälle asetetuista tehtävistä on suoritteetta liikkuvien koneiden havaitseminen. Ilman suoritetta liikkuvasta koneesta on luotava hälytys. Tilaajan määritelmä suoritteetta liikkuvasta koneesta on kone, joka on ollut liikkeessä ainakin puoli tuntia, eikä ole tänä aikana tehnyt yhtään suoritetta. Hälytysehdoissa on siis kaksi osaa, joista molempien pitää täyttyä: aika ja nopeus. Ilman aikarajaa hälytys tulisi aina koneen lähtiessä liikkeelle. Nopeusrajoituksen puuttuessa hälytyksiä satelisi ratapihalla paikkaa vaihtavista koneista.

Suoritteetta liikkuvien koneiden havaitseminen on yksinkertainen ja lineaarinen prosessi, jonka toimintalogiikka on kuvattu kuvassa 14.



Kuva 14. Suoritteetta liikkuvan työkoneen havaitsemislogiikka.

Työkoneiden viimeisimmistä sijaintitiedoista erotellaan ne, jotka ovat liikkeessä. Liikkeessä olevien koneiden viimeisin tila, eli status, tarkistetaan. Jos tilasta käy ilmi, että kone ei tee tällä hetkellä suoritetta, haetaan koneen sijaintitiedot puolen tunnin ajalta. Jos kone on ollut liikkeessä puoli tuntia aiemmin, haetaan koneen tilahistoria puolen tunnin ajalta. Tilahistoriasta tarkastellaan, onko kone tehnyt puolen tunnin aikana suoritteen. Jos tilahistoriasta ei löydy suoritteita, on kone liikkunut ilman suoritetta. Koneesta luodaan hälytys.

8 Yhteenveto

Insinöörityötä tehdessä opin uutta ja syvensin osaamistani tutuissa osa-alueissa. Päällimmäisenä mieleen jäivät tietokantojen käyttö, yksikkötestaus ja syvempi ymmärrys JavaScriptin toiminnasta. Tietokantojen kanssa aloitin aivan alusta, mutta taidot kasvoivat nopeasti sovelluksen toteutusta tehdessä. Yksikkötestauksesta olin kuullut aiemmin, mutten ollut kokeillut sitä. Projektin edetessä en katunut kertaakaan yksikkötestauksen käyttöä, sillä se säästi monelta harmilta. Tärkeimpänä oppina koen kuitenkin syventyneen ymmärryksen JavaScriptistä. Edellisessä projektissa olin kyllä käyttänyt JavaScriptiä, mutta tätä projektia tehdessä löysin ja opettelin paljon minulle uusia ominaisuuksia, kuten lupausten käyttöä, joita ilman projektin toiminnallisuuksien teko ei olisi ollut mahdollista.

Työkoneiden sijaintihistoriatiedon tallentaminen onnistui, samoin tietoa eteenpäin jakava rajapinta. Työsuoritteen tiedot oli mahdollista yhdistää koneen senhetkiseen sijaintiin, ja luoda yhdistetystä tiedosta visuaalinen esitys kartalla. Tehdyn työn seuraaminen kartalla ei aiemmin ollut mahdollista.

Nykyisellään järjestelmä on täysin toimiva kokonaisuus, mutta myös jatkokehitettävää löytyy. Olennaisin jatkokehityskohde on tietokantaan talletettujen tietojen arkistointi. Jos sijaintihistoriatietoja halutaan säilyttää monta vuotta, sijaintihistoriataulukon koko kasvaa ja haut hidastuvat. Vanha data kannattaa siirtää arkistointiin tarkoitettuun palveluun, jolloin useimmiten haettu tuore data saadaan pidettyä kohtalaisen kokoisessa taulukossa.

Lähteet

1 Laitinen, Janne. 2018. Tuotantopäällikkö, VR Track Oy, Helsinki. Keskustelu 8.10.2018.

2 Pöyhönen, Saku. 2018. Resurssipäällikkö, VR Track Oy, Helsinki. Keskustelu 8.10.2018.

3 Fowler, Martin. 2006. Continous Integration. Verkkoaineisto. <https://www.martin-fowler.com/articles/continuousIntegration.html>. 1.5.2006. Luettu 25.9.2018.

4 Khan, Asif; Prudnikov Mikhail; Shen Xiang & Stacy David. 2017. Practicing Continuous Integration and Continuous Delivery on AWS. E-kirja. Amazon Web Services Inc.

5 Saleh, Hazem. 2013. JavaScript Unit Testing. E-kirja. Packt Publishing.

6 FAQ. Verkkoaineisto. PostgreSQL Wiki. <https://wiki.postgresql.org/wiki/FAQ>. Päivitetty 30.5.2018. Luettu 16.9.2018.

7 What is JavaScript? Verkkoaineisto. MDN web docs. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript>. Päivitetty 24.7.2018. Luettu 16.9.2018.

8 JavaScript Tutorial. Verkkoaineisto. W3schools. <https://www.w3schools.com/js/>. Luettu 16.9.2018.

9 The JSON Data Interchange Syntax. 2017. ECMA-404 2nd Edition / December 2017. Ecma International.

10 Patel, Priyesh. 2018. What exactly is Node.js? Verkkoaineisto. <https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5>. 18.4.2018. Luettu 9.9.2018.

11 Syed, Basarat. 2014. Beginning Node.js. E-kirja. Apress.

12 Herron, David. 2016. Node.js Web Development. E-kirja. Packt Publishing.

13 NPM. Verkkoaineisto. <https://www.npmjs.com/>. Luettu 7.10.

14 Chacon, Scott & Straub, Ben. 2018. Pro Git Version 2.1.87, 2018-09-25. E-kirja. Apress.

15 Olson, Peter. 2017. Introduction to asynchronous JavaScript. Verkkoaineisto. <https://www.pluralsight.com/guides/introduction-to-asynchronous-javascript>. 8.7.2017. Luettu 17.9.

16 Using promises. Verkkoaineisto. MDN web docs. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises>. Päivitetty 11.9.2018. Luettu 17.9.2018.

17 Sharma, Tarun. 2018. Writing neat asynchronous Node JS code with Promises & Async/Await. Verkkoaineisto. 22.4.2018. Luettu 18.9.2018.

18 PostgreSQL - INDEXES. Verkkoaineisto. Tutorialspoint. <https://www.tutorialspoint.com/postgresql/postgresql_indexes.htm>. Luettu 26.9.2018.

19 OpenLayers - OSGeo. Verkkoaineisto. The Open Source Geospatial Foundation. <https://www.osgeo.org/projects/openlayers/>. Luettu 8.9.2018.

20 OpenLayers - Welcome. Verkkoaineisto. <http://openlayers.org/>. Luettu 8.9.2018.

21 OpenLayers. Verkkoaineisto. <https://www.npmjs.com/package/ol>. Luettu 8.9.2018.

22 Quick Start. Verkkoaineisto. <http://openlayers.org/en/v4.6.5/doc/quickstart.html>. Luettu 8.9.2018.

23 Basic Concepts. Verkkoaineisto. <http://openlayers.org/en/latest/doc/tutorials/concepts.html>. Luettu 8.9.2018.