Teemu Tirkkonen

Pystygeometriasovelluksen kehittäminen raideliikenteeseen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

27.5.2024

Tiivistelmä

Tekijä:	Teemu Tirkkonen
Otsikko:	Pystygeometriasovelluksen kehittäminen raideliikenteeseen
Sivumäärä:	33 sivua
Aika:	27.5.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mobile Solutions
Ohjaajat:	Tutkijaopettaja Hannu Markkanen

Opinnäytetyön tarkoituksena oli tehdä sovellus junankuljettajille. Sovelluksella nähdään muun muassa raiteen ala- ja ylämäet, ratakilometrit sekä junaan liittyviä tietoja. Opinnäytetyössä avattiin myös pystygeometrian näkyvyyttä raideliikenteessä. Tietoja käytettiin myöhemmin tukena sovelluksen kehityksessä. Sovellus tehtiin raideliikenteessä toimivalle yritykselle noin kolmen kuukauden aikana. Tavoitteena oli saada sovellus testaukseen asiakkaalle. Tässä onnistuttiin ja suurin osa asiakkaan toiveista toteutettiin. Lisäksi sovelluksen tuli pystyä korvaamaan asiakkaan nykyinen järjestelmä. Sovelluksen kehitykseen käytettiin pääasiassa JavaScript-ohjelmointikieltä ja React-kirjastoa.

Projektin aikana kohdattiin muutamia ongelmia. Alkuvaiheessa ongelmana oli ratakilometrien pituus- ja leveyspiirien arvojen saaminen front-endin kuvaajaan. Asia kuitenkin ratkaistiin back-endin puolella. Osasta ratakilometrejä puuttui myös korkeusarvot, jotka ratkaistiin interpoloinnin avulla. Seuraavaksi selvitettiin, miten junan sijainti, ratakilometrit ja pituuskaltevuudet saatiin kuvaajalle. Ratakilometrit sijoitettiin vaaka-akselille. Kuvaajasta löytyneen elementin avulla pituuskaltevuudet sijoitettiin vastaavan ratakilometrin kohdalle. Projektin aikana mikään ongelma ei jäänyt ratkaisematttomaksi ja vaaditut ominaisuudet saatiin toteutettua.

Johtopäätöksenä todettiin, että tämänkaltaisen sovelluksen tekeminen vaatii hyvää taustatutkimusta ja suunnittelua. Joidenkin tietojen hyödyntäminen oli monimutkaisempaa kuin ajateltiin, ja projektin edetessä haluttuja ominaisuuksia muutettiin tarpeiden mukaan. Kuvaajaan tehtiin muun muassa 1,5 km:n ja 10 km:n näkymät, joita ei oltu alun perin suunniteltu. Projekti tuotti toimivan sovelluksen, joka toimitettiin asiakkaalle. Testaus jäi kuitenkin vähälle, joten laajempien testien jälkeen sovelluksen kehitystä voidaan jatkaa.

Avainsanat: raideliikenne, pystygeometria, ratageometria

Abstract

Author:	Teemu Tirkkonen
Title:	Development of vertical geometry application to rail traffic
Number of Pages:	33 pages
Date:	27 May 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and communication technology
Professional Major:	Mobile Solutions
Supervisors:	Hannu Markkanen, Researching Lecturer

The purpose of the final year project was to make an application for train drivers which they can utilize to see ascending and descending hills on a specific track, line-kilometers and information regarding the train in question. Another research topic was how vertical geometry is used in rail traffic. Information found from these studies were later used in the project. The goal of the project was to get the application ready for testing and develop features that were given by the customer.

The application was developed with JavaScript and React. In addition, several libraries were used during development. There were only a few bigger problems during development. However, in the end the project was finished without any major problems. Demands were also mostly met, and the application was ready for testing after 3 months.

The project conclusion was that making an application that utilizes this kind of information requires good planning and thorough research. Some features needed to be modified because they could not be developed exactly as planned. However, in the end a working application was delivered to the customer, but testing was limited. After running more extensive tests, the development of the application could continue.

Keywords: rail traffic, vertical geometry, track geometry

Sisällys

1 Johdanto

Opinnäytetyön tavoitteena on antaa kuva siitä, miten pystygeometria näkyy raideliikenteessä ja mihin sitä käytetään. Tavoitteena on selittää, miten pystygeometriasta opittuja tietoja käytetään sovelluskehityksessä. Lisäksi selitetään Suomessa käytössä oleva ratakilometrijärjestelmä, jota hyödynnetään sovelluksen kehityksessä. Sovellus toteutettiin Proxionille ja asiakkaana oli raideliikenteessä vaikuttava yritys. Työ tilattiin, koska asiakas halusi korvata nykyisen käytössä olevan järjestelmän uudella sovelluksella. Sovelluksesta pitäisi näkyä helposti junan sijainti radalla, ratakilometrit, ala- ja ylämäet sekä junaan liittyvät tiedot. Kehitystyössä käytettiin muun muassa JavaScriptiä ja Reactia.

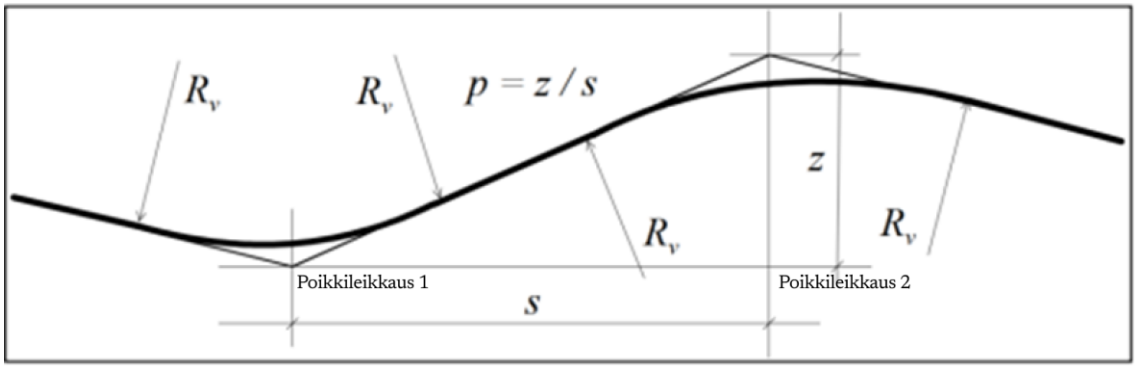
Pystygeometrialla tarkoitetaan radan profiilia, ja pystygeometrialla määrätään raiteen korkeusvaihtelut ja pituuskaltevuus. Pystygeometria vaihtelee rataosuuksien mukaan ja voi muun muassa tarkoittaa eroja pituuskaltevuuden maksimiraja-arvojen välillä sekä ala- ja ylämäkien määrää. Suomessa käytössä oleva ratakilometrijärjestelmä määrää ratakilometrien paikat. Niillä esitetään junan sijainti raideliikenteessä. Ratakilometri on yleensä kilometrin pituinen osio radalla.

Työ pohjautuu projektiin, joka tehtiin raideliikenteessä toimivalle yritykselle kolmen kuukauden aikana. Projektilla saatiin asiakkaalle uusi järjestelmä, jossa on aikaisempaan järjestelmään verrattuna uusia tarpeellisia ominaisuuksia. Työtehtäviin kuului front-endin kehittäminen sovellukseen ja tätä kautta myös aiheeseen liittyvien asioiden opiskeleminen, jotta sovellus saatiin toteutettua mahdollisimman hyvin.

2 Pystygeometria

2.1 Pystygeometria yleisesti

Pystygeometrialla tarkoitetaan radan profiilia. Sillä määrätään raiteen sijainti korkeussuunnassa ja määritellään pituuskaltevuus taitepisteiden avulla. Korkeus muodostuu kahdesta eri elementistä. Nämä ovat ympyränkaaren muotoiset kaltevuustaitteet ja suorat kaltevuusjaksot. (1, s. 17.) Kaltevuustaite on taitepisteiden välinen pyöristetty kaari, jolle lasketaan pyöristyskaarresäde. Kaltevuusjakso on vaakasuora välimatka kahden taitepisteen välillä. Jaksojen taitepisteiden avulla lasketaan myös pituuskaltevuus radalle. Raiteen laskennallinen ja todellinen pituuskaltevuus on sama raiteen kaltevuusjakson suorassa osassa (kuva 1). (1, s. 17; 3, s. 11.) Siirtymäkaaria käytetään raiteen kallistetuissa kaarteissa. Siirtymäkaaria on kahta eri tyyppiä, tasoituskaarisarja ja klotoidi. Klotoidi on käyrä, joka mahdollistaa pidemmät siirtymäkaaret ja raiteen kallistukset tasoituskaarisarjaan verrattuna. Tasoituskaarisarjat voidaan taivuttaa Suomessa lyhentäen kiskojen toimitusaikoja klotoidiin verrattuna. Lisäksi tasoituskaarisarjan maksiminopeus on pienempi kuin klotoidin. (2.)



Kuva 1. Raiteen pituuskaltevuus (1, s. 17).

Kuvassa 1 p on pituuskaltevuuden kaava, z on taitepisteiden korkeusero metreissä, s on taitepisteiden välimatka raidetta pitkin vaakatasossa metreissä ja R_v on pyöristyskaarresäde metreinä. Usealle raiteelle voi syntyä yhteinen korkeustaso, jos niillä on sama korkeusasema. Korkeusasemalla tarkoitetaan raiteen korkeusviivan korkeutta. Korkeusviivalla määritellään kiskon korkeus.

Pituuskaltevuus voi kuitenkin olla poikkeava. Näin tapahtuu, jos erisuuntaisilla raiteilla poikkileikkausten välinen matka on erilainen. (1, s. 17.)

Pystygeometrian suunnittelua ohjaavat erilaiset EU:n komission direktiivit ja ohjeistukset. (3, s. 14.) Suunnittelussa pitää ottaa muun muassa huomioon taloudelliset perusteet, geotekniikka, maasto, sillat ja rakenteelliset vaatimukset. Parhaimmillaan pystygeometria on selkeää ja yksinkertaista. Pituuskaltevuuden raja-arvot määräytyvät liikenteen mukaan. On vältettävä alle 600 metrin pituisia erisuuntaisia pituuskaltevuuksia ja yli 2 000 metrin pituisia kaltevuusjaksoja. Näin turvallisuus saadaan taattua ja pysytään maksimipituuskaltevuusarvojen alapuolella. Suunnittelussa on otettava huomioon myös liikenteen tarpeet muun muassa jarrutus- ja kiihdytysalueiden sopivan suuntaisilla kaltevuuksilla. Kaltevuuksia on kahta eri tyyppiä, jotka ovat nouseva ja laskeva. Kaltevuus voi siis olla pituussuunnassa ylöspäin tai alaspäin menevää. (1, s. 19–20.)

2.2 Pystygeometria linjaraiteilla

Pystygeometria on hieman erilaista jokaisella rataosuudella. Linjaraiteella tarkoitetaan raidetta, jota käytetään muun muassa matkustajaliikenteessä. Esimerkiksi Helsingin ja Tikkurilan välinen raide on linjaraide. Pituuskaltevuuden maksimiarvot vaihtelevat muun muassa sekaliikenneradan, matkustajaliikenneradan ja tavaraliikenneradan välillä taulukon 1 mukaan.

Taulukko 1. Pituuskaltevuuden raja-arvot suoralla radalla (1, s. 20).

RATA	**PITUUSKALTEVUUS [‰]**		
	Suositeltava	**Maksimiarvo**	**Lupa-arvo**
Sekaliikenneradat	≤ 10	12,5	25
Matkustajaliikenneradat	≤ 10	15	40
Tavaraliikenneradat	≤ 10	12,5	25

Pituuskaltevuuden maksimiarvon ylittyessä on pohdittava ylittymisen vaikutuksia ja tehtävä lupahakemus rataverkon haltijalle. Syitä maksimiarvon ylittämiselle voi olla useita. Yleisin on nykyisen pituuskaltevuuden muuttaminen. (1, s. 20.)

Ainoastaan uudet radat ja raiteet voivat ylittää raja-arvot. Maksimiarvon ylittyessä maksimiarvo ei saa enää huonontua. Kaarteessa raja-arvoa tulee pienentää ominaiskaarrevastuksen arvolla W_r (kaava 1). Huomionarvoista on myös junan nopeuden ja raiteen kallistuksen vaikutus todelliseen vastukseen. (1, s. 20.)

$$w_r = \frac{650}{\mathrm{R}-55}\,[‰] \qquad (1)$$

Kaavalla 1 lasketaan ominaiskaarrevastuksen arvo W_r ja R on kaavassa kaarresäde metreissä.

2.3 Raiteen kallistus

Kallistus on raiteen kiskojen välinen korkeusero. Päätavoite on halutun nopeuden saavuttaminen ja raiteen geometrian kaarien aiheuttaman poikittaiskiihtyvyyden vähentäminen. Tasapainokallistus eli teoreettinen kallistus on teknisesti ihanteellinen todelliselle junanopeudelle (kaava 2). (1, s. 26.) Tällöin raiteen

tasossa olevaa poikittaiskiihtyvyyttä ei kohdistu junaan. Poikittaiskiihtyvyys on keskeisvoiman aiheuttama vaakasuuntainen kiihtyvyys. (4, s. 18.)

$$D_{EQ} = \frac{12{,}5V^2}{\mathrm{R}} \quad (2)$$

Kaavassa 2 raiteen tasapainokallistus millimetreissä esitetään kirjainyhdistelmällä D_{EQ}, V on nopeus kaarteessa km/h ja R on kaarresäde metreissä. Kallistusta suunniteltaessa on otettava huomioon junien eri nopeudet. Kallistuksen täytyy sopia kaiken liikenteen tarpeisiin. Valittu kallistus ei siis välttämättä ole yksittäiselle junalle ihanteellinen. Nopeuden täytyy olla lähellä junapainoilla painotettua junien keskinopeutta. Usein raiteen normaalikallistuksen avulla lasketaan raiteen kallistus. (1, s. 26.) Normaalikallistuksessa junaan kohdistuu vähän poikittaiskiihtyvyyttä, jonka matkustajat tuntevat junan kulkiessa kaarteessa. Tasapainokallistuksen ja normaalikallistuksen ero on poikittaiskiihtyvyyden määrässä. Tasapainokallistuksessa sitä ei ole ollenkaan ja normaalikallistuksessa sitä on vähän. Raiteen maksimikallistus lasketaan kaavan 3 mukaan.

$$D_{lim} = (R - 50) \times 0.7\ mm \quad (3)$$

Kaavassa 3 maksimikallistus millimetreissä esitetään kirjainyhdistelmällä D_{lim} ja R on kaarresäde metreissä. Maksimikallistuksella tarkoitetaan raiteen suurinta sallittua kallistusta, kun kaarresäde R < 320 m. (1, s. 28.)

Kallistusta mitoitettaessa on otettava huomioon muutamia asioita. Junat kulkevat määrätyillä nopeuksilla eri raideosuuksilla. Junan tulee siis kulkea eri nopeuksilla ilman ongelmia ja matkustusmukavuuden tulee olla mahdollisimman hyvä. Turvallisuus täytyy myös ottaa huomioon, ja nopeus ei saa vaarantaa sitä. Kiskojen kulumisen tulee olla myös mahdollisimman tasaista. Hitaammat junat on huomioitava, ja raiteen kallistus säädettävä niiden mukaan.

Normaalikallistusta pienemmän kallistuksen käyttämiseen on myös muita syitä. Kallistusviiste ja siirtymäkaaren pituus on otettava huomioon, ja niiden pohjalta raide kallistetaan raiteen tavoitenopeuden perusteella. Kallistusviisteellä tarkoitetaan raiteen osuutta, jossa raiteen kallistus muuttuu. (1, s. 27; 4, s. 18.)

3 Ratakilometrijärjestelmä

3.1 Järjestelmän toiminta

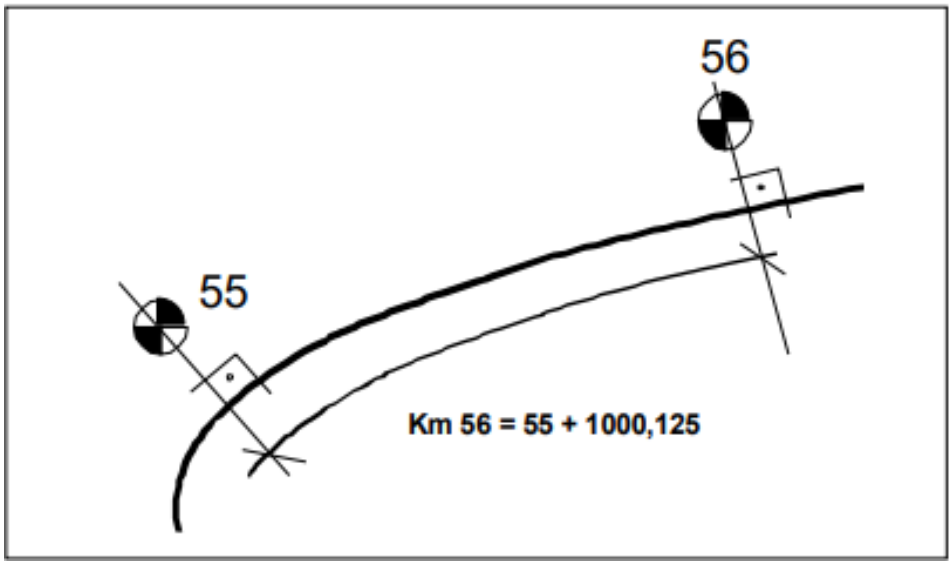
Suomessa on käytössä ratakilometrijärjestelmä, jota käytetään sijainnin esittämiseen raideliikenteessä. Järjestelmä koostuu ratatiedon ylläpidossa olevista ratanumeroista ja ratakilometreistä siten, että järjestelmä muodostaa rataosoitteen. Ratakilometri on tietyn mittainen osa radalla. Lisäksi järjestelmässä käytetään sijainnin määrittämiseen tasakilometripisteitä. Pisteet saadaan tasokoordinaattijärjestelmästä koordinaatteina. Järjestelmä näkyy matkustajille radanvarressa olevilla kilometritolpilla (kuva 2). (1, s. 67; 5, s. 6.)

Kuva 2. Ratakilometritolppa 168 Heinolan ratapihalla (7).

Ratakilometrin tunnus määräytyy tasakilometripisteen mukaan, joka on numeroarvoltaan pienemmän tai aakkosjärjestyksessä aiemman tunnuksen mukainen. (1, s. 67.) Ratakilometrit lasketaan yleisesti ottaen Helsingistä poispäin, ja ne kuvaavat etäisyyttä Helsinkiin. Ratakilometri 0 oli siis Helsingin vanhan rautatieaseman pääraiteen alkupisteen kohdalla. Nykyinen Helsingin päärautatieasema on kuitenkin hieman vanhaa asemaa pohjoisempana. (5, s. 6–7.) Näin ollen

Helsingin päärautatieaseman ratakilometri on 0+159. Rautatieasema sijaitsee siis ratakilometrillä 0 ja on 159 metriä alkupisteestä eteenpäin. Ratakilometri muodostuu ensisijaisesti etäisyydestä Helsinkiin eli esimerkiksi ratakilometri 168 on myös sen etäisyys Helsinkiin. Aakkosia käytetään, kun rataa lyhennetään tai pidennetään ja syntyy useita samoja ratakilometrilukemia. Tällöin saman numeroiset uudet ja vanhat ratakilometrit erotetaan aakkosilla toisistaan. (1, s. 69; 5, s. 6–7.)

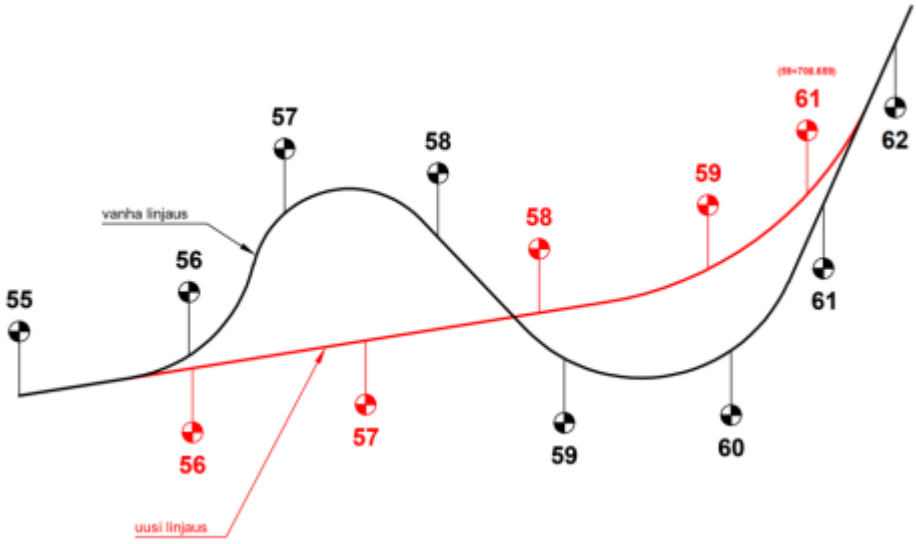
Uusi ratakilometri alkaa aina tasakilometripisteeltä ja päättyy seuraavaan pisteeseen tai pituusmittausraiteen päättymiskohtaan. Pituusmittausraiteella tarkoitetaan raidetta, jota pitkin mitataan ratakilometrin pituus. Pituus saadaan tasakilometrien pituusmittausraiteella olevista projektiopisteistä ja niiden välimatkasta vaakatasossa pituusmittausraiteen keskilinjaa pitkin (kuva 3). (1, s. 67.)



Kuva 3. Ratakilometrin pituus (1, s. 67).

Kuvasta 3 nähdään ratakilometrin pituuden muodostuminen pituusmittausraiteella projektiopisteiden avulla. Ratakilometri 56 on siis noin 1 000 metriä ratakilometri 55:stä eteenpäin. Korkeuden vaihtelua ei oteta huomioon, kun pituutta määritetään. Kuten nimestä voi päätellä, ratakilometrin pituus on yleisesti ottaen kilometri, mutta suuria poikkeuksiakin löytyy. Radan rataoikaisut tai raidegeometrian muuttaminen voivat muuttaa pituutta radikaalisti. Pituuteen vaikuttaa myös käytetty koordinaattijärjestelmä. (1, s. 67.)

Rataoikaisun ylettyessä ratakilometripisteen ohitse määritetään uusi sijainti vähintään yhdelle tasakilometripisteelle. Tällöin ratakilometrin pituuden on säilyttävä samana, jotta oikaisun jälkeiset paikalleen jäävät rataosoitteet eivät muutu. Ratakilometripisteen tunnus ja sijainti tulee määrittää niin, että tunnus säilyy entisenä. Oikaisusta syntyneet uudet ratakilometripisteet määritellään ensisijaisesti 1 000 metriä pitkiksi. Poikkeus on viimeistä edellinen ratakilometri, joka määritellään 500–1 500 metriä pitkäksi. Radan oikaisun jälkeen osa käytössä olleista ratakilometripisteistä voi jäädä pois uudelta radalta (kuva 4). (1, s. 69.)



Kuva 4. Ratakilometrit radan lyhentymisen jälkeen (1, s. 69).

Radan pidentyessä huomattavasti aikaisemmasta ratakilometrit mahdollisesti lisääntyvät. Tällöin uudet ratakilometrit nimetään aakkosilla yksilöiden ne toisistaan. Uudella radalla voi olla esimerkiksi ratakilometri 55 ja 55A. (1, s. 70.)

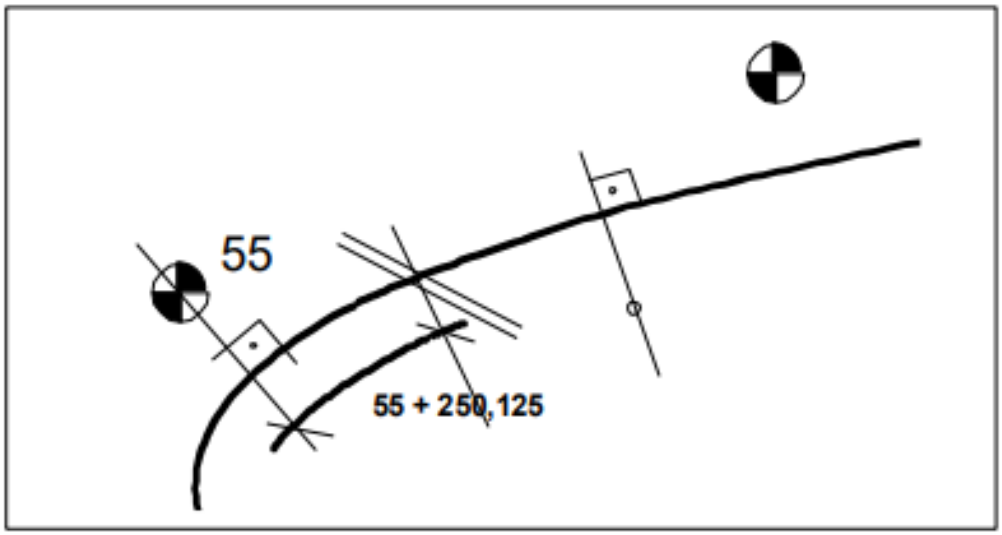
3.2 Pituusmittausraide

Ratanumeroilla on aina yksi pituusmittausraide. Useampiraiteisilla radoilla pituusmittausraiteeksi valitaan jokin näistä raiteista. Käytössä voi olla myös useita pituusmittausraiteita rinnakkain, jos ollaan muun muassa rataosien yhtymäkohdassa tai suurella rautatieliikennepaikalla. Rautatieliikennepaikka on rataverkolla oleva kohde, jota käytetään henkilö- tai tavaraliikenteen palvelupaikkana.

Ratakilometrisijainti määräytyy ratanumeron osoittaman pituusmittausraiteen mukaan. Suurimpien rautatieliikennepaikkojen pituusmittausraiteet, ratanumerot ja raiteiden kuuluminen ratanumeroihin esitetään pituusmittausraidekaavioissa. (1, s. 68.)

3.3 Ratakilometrisijainti

Ratakilometrisijainnilla tarkoitetaan kohteen projektiopisteen paikkaa pituusmittausraiteella. Kohteella tarkoitetaan esimerkiksi rautatieasemaa tai paikkaa, jossa lastataan ja kuormataan tavaraa. Ratakilometrisijainti ilmaistaan muodossa km + m, jossa ratakilometri on km. Kohteen sijainnin ja edeltävän tasakilometripisteen välimatka pituusmittausraiteen keskilinjaa pitkin on m. Kohteen leikatessa pituusmittausraiteen sijainti on keskilinjojen leikkauspisteen sijainti. (kuva 5).



Kuva 5. Kohteen ratakilometrisijainti (1, s. 69).

Kuvasta 5 nähdään kohteen ratakilometrisijainti raiteella. Kohde on siis ratakilometri 55:stä noin 259 metriä eteenpäin. Seuraava ratakilometri on 56, vaikka sitä ei ole kuvaan merkitty numerolla. (1, s. 68–69.) Välimatkoja pituusmittausraidetta pitkin voidaan laskea luotettavasti vain, jos käytetään ratakilometrien todellisia pituuksia ja ne otetaan laskelmissa huomioon. Ratakilometrisijainneista voidaan laskea luotettavasti välimatkoja vain pituusmittausraiteita pitkin.

3.4 Rautatieliikennepaikka

Rautatieliikennepaikalla tarkoitetaan rataverkolla olevaa kohdetta, jota käytetään henkilö- tai tavaraliikenteen palvelupaikkana. Paikkaa voidaan käyttää myös rautatieliikenteen ohjaukseen, mutta yleensä liikennepaikalla on useampia tehtäviä lueteltujen lisäksi. Liikennepaikka on laaja käsite, joka kattaa useamman liikenteen hoidossa ja asiakaspalvelussa käytettävän rakennuksen. Näihin lukeutuvat muun muassa asemarakennukset, ratapihat, opastimet ja veto- ja vaunukaluston huoltotilat. Liikennepaikka voi siis käsittää eri asioita radan mukaan. (6, s. 7–9.)

Rautatieliikennepaikkojen ja radan rakenteisiin liittyvistä määräyksistä vastaa Traficom eli liikenne- ja viestintävirasto. Rataverkon haltijan tehtävänä on varmistaa, että rataverkolla seurataan alan eurooppalaista standardisointia. Rautatieliikennepaikat tulee esittää valtion rataverkon verkkoselostuksessa, ja niistä on oltava raiteistokaavio. Paikan sijainti on suunniteltava niin, että se toimii matkustaja-, tavaraliikenne- ja liikenteenohjaustehtävissä. Rautatieliikennepaikka käsittää eri asioita riippuen sen käyttötarkoituksesta. Se voi olla liikennepaikka, linjavaihde tai seisake. Liikennepaikka on alue, joka toimii tavara- tai matkustajaliikenteessä. Liikennepaikan määrittelee rataverkon haltija. Aiemmin myös rautatieasema laskettiin liikennepaikaksi, mutta tästä on sittemmin luovuttu. Rautatieasemalla tarkoitetaan tässä kontekstissa asemarakennuksesta ja sivuraiteesta koostuvaa paikkaa. (6, s. 7–9.)

Linjavaihteella tarkoitetaan rautatieliikennepaikkaa, joka hoitaa tavara- ja ratatyöliikenteen tarpeet. Sitä ei ole varustettu liikenteenohjauslaittein, kuten liikennepaikkaa. Vaihteella voi olla ratapiha, jossa tavara kuormataan ja lastataan. Yksinkertaisin linjavaihde käsittää vain yhden pääraiteen vaihteen. Seisake on tarkoitettu ainoastaan henkilöliikenteen käyttöön erottaen sen liikennepaikasta ja linjavaihteesta. Seisakkeella ei ole asemarakennuksia eikä liikenteenhoitoon liittyviä toimintoja, kuten ratapihaa. (4, s. 99; 6, s. 11–12.)

4 Sovelluksen kehitys

4.1 Vaatimusmäärittely

Sovellus kehitettiin Proxionilla raideliikenteessä toimivalle asiakkaalle. Sovelluksesta tulisi helposti nähdä muun muassa tietyn reitin korkeusvaihtelut, junan sijainti reitillä ja ratakilometrit. Perusvaatimus oli tehdä sovellus, jonka avulla junankuljettaja pystyy ennakoimaan ja näkemään reitin pystygeometrian sekä junaan liittyvät tiedot. Tavoitteena oli korvata asiakkaan nykyinen järjestelmä ja lisätä käyttöominaisuuksia. Sovelluksessa tulisi olla mahdollisuus kirjautumiseen, junan tietojen hakemiseen junanumeron perusteella ja raiteen pystygeometrian näkemiseen.

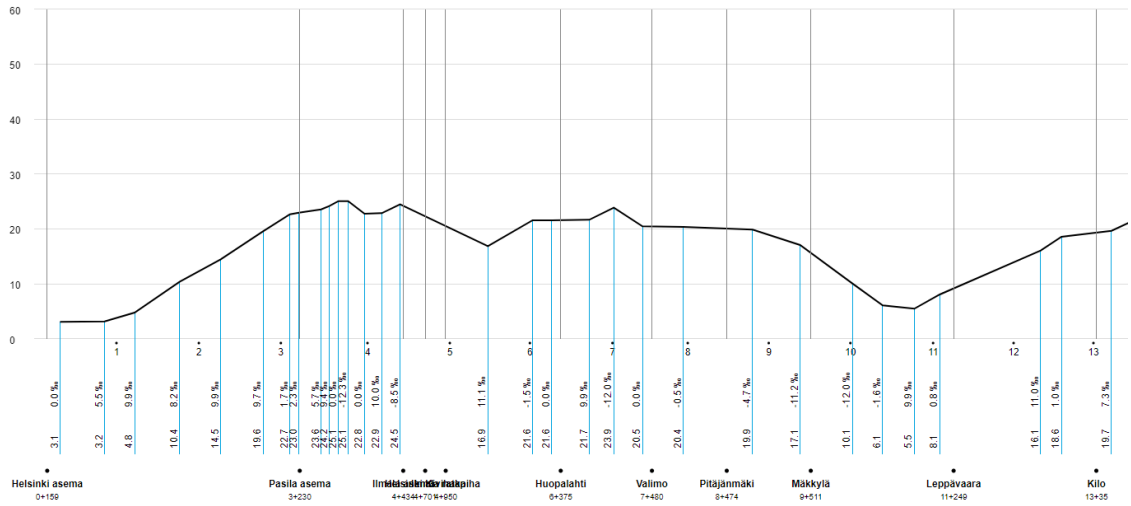
Pystygeometria tulisi näyttää viivadiagrammina, jossa pystyakselilla on korkeus merenpinnasta ja vaaka-akselilla ratakilometrit. Lisäksi diagrammissa tulisi näkyä mahdolliset rautatieasemat tai muut pysähdyspisteet sekä junan nykyinen sijainti ja pituus viivaa pitkin eri värillä. Asiakkaan toiveen mukaisesti junan sijainti tulisi näyttää punaisella, sijainnin ollessa saatavilla. Muissa tapauksissa vihreällä, kun sijainti ei ole saatavilla. Diagrammissa pitäisi olla kaksi eri näkymää, jotka näyttävät reittiä tietyn verran eteen- ja taaksepäin. Oletusnäkymän tulisi olla kymmenen kilometrin näkymä ja painiketta painamalla voidaan vaihtaa puolentoista kilometrin näkymään. Näkymissä juna tulisi keskittää keskelle diagrammia, jolloin nähdään eteen- ja taaksepäin. Kehityksen edetessä vaatimuksia tuli muutamia lisää. Suurin näistä oli mahdollisuus liikkua kilometrin verran eteen- ja taaksepäin näkymissä.

4.2 Kehityksen suunnittelu

Sovelluksen kehitys aloitettiin aloituspalaverilla, jossa tarkastettiin projektin aikataulu ja suunnittelun eri vaiheet. Tavoitteena oli saada projekti valmiiksi kolmen kuukauden aikana. Asiakkaan kanssa käydyn keskustelun pohjalta tehtiin käyttöliittymä-piirustukset, joiden pohjalta aloitettiin pohtimaan käyttöliittymän toteutusta.

Sovellus päätettiin kehittää JavaScriptillä, Reactilla ja Visual Studio Codella. Aluksi mietittiin myös TypeScriptiä, mutta edellä mainituista teknologioista oli enemmän kokemusta. Sovellus on web-sovellus, joka toimii myös puhelimilla ja skaalautuu eri näyttökokoihin. Projektissa käytettiin paljon erilaisia kirjastoja, jotka helpottavat sovelluksen kehityksessä. Projektinhallintaan käytettiin GitLabia, jolla toteutettiin myös jatkuva integraatio. Jatkuva integraatio tarkoittaa koodin automaattista tarkistusta aina, kun uusi versio tallennetaan GitLabiin. Tarkoituksena on löytää ohjelmointivirheet nopeammin ja parantaa ohjelmiston laatua. Versionhallintaan käytettiin Gitiä ja tehtävienhallintaan Jiraa. Projektin tehtävät jaettiin Jirassa ja tallennettiin GitLabiin tehtävän numeron mukaan niiden valmistuessa. Tällä tavalla pidettiin yllä järjestelmällisyyttä ja projektin etenemisen seuraaminen oli vaivatonta. Tehtävän valmistuttua tehtiin GitLabin sisällä merge request, jonka tarkisti kokeneempi sovelluskehittäjä. Tällä tarkoitetaan uuden version yhdistämistä vanhaan.

Projekti aloitettiin tutkimalla, mitä tietoa on saatavilla. Samalla pohdittiin asiakkaan tilaamien toimintojen toteutusta ja mahdollisia muutoksia sovelluksen ominaisuuksiin. Ensiksi tutkittiin Julia - Pystygeometria -sivustoa, josta näkee rataverkon pystygeometrian tietyn ratanumeron perusteella. Kuvaaja näyttää vaaka-akselilla ratakilometrit ja pystyakselilla korkeuden merenpinnasta metreinä (kuva 6).

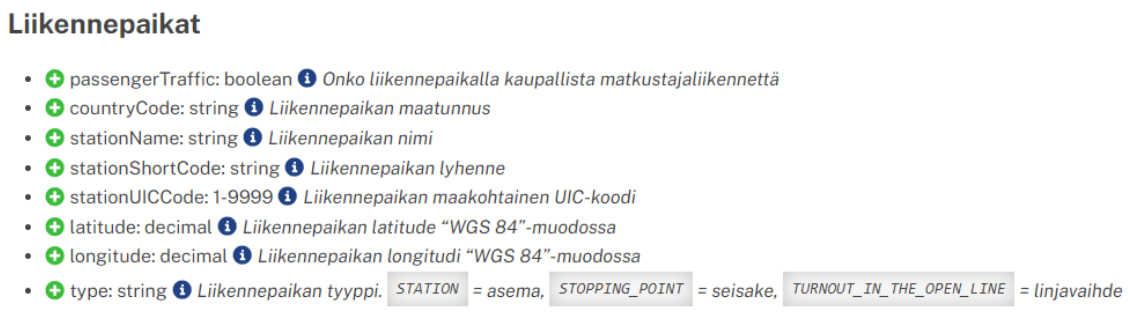


Kuva 6. Kuvakaappaus Julia - Pystygeometria -sivuston kuvaajasta, josta näkee tietyn ratanumeron pystygeometriatiedot (8).

Sivustolta selvisi, että tiedot haetaan Digitrafficin avoimesta datasta. Samaa dataa voidaan hyödyntää tässä projektissa. Digitrafficin sivustolla on kattavat dokumentaatiot sovellusrajapinnan käyttöön. Projektiin rakennettiin kuitenkin oma sovellusrajapinta hyödyntäen Digitrafficin sovellusrajapintaa. Sovellusrajapinta rakennettiin, koska Digitrafficin sovellusrajapinnassa oli muun muassa tarpeettomia tietoja. Sovellusrajapinnan rakensi Proxionin back-end-kehittäjä.

Sovelluksessa pitäisi pystyä siirtämään kuvaajaa eteenpäin, ja sen pitäisi päivittyä automaattisesti. Seuraavaksi etsittiin sopivaa kirjastoa tähän tarkoitukseen. Lopulta päädyttiin kirjastoon nimeltä Recharts laajan dokumentaation ja esimerkkien takia. Sivustolta löytyy muun muassa esimerkki kuvaajasta, jota voi kohdentaa ja loitontaa. Vaikutti myös olevan mahdollista, että kuvaajaa pystyy päivittämään automaattisesti lataamatta sivua uudelleen.

Seuraavaksi selvitettiin, miten junan sijainnin voisi sijoittaa oikeaan kohtaan kuvaajalla. Kuvaajasta pitäisi näkyä selvästi, missä kohtaa ollaan, jotta junankuljettaja pystyy ajamaan sopivalla nopeudella ylä- ja alamäkiin. Käyttäjän sijainnin kuvaajaan voisi saada esimerkiksi vertaamalla käyttäjän sijaintia liikennepaikan sijaintiin. Digitrafficin sivuilta selvisi liikennepaikkahaun sisältö (kuva 7).



Kuva 7. Kuvakaappaus Digitrafficin sivustolta, josta näkyy JSON:n rakenne (9).

Rakenteen perusteella voitiin olla melko varmoja, että junan sijainnin voi kohdentaa lähimpään liikennepaikkaan. Sivustolta ei kuitenkaan selvinnyt, saako ratakilometritietoja Digitrafficin rajapinnasta. Back-end-kehittäjä löysi ratkaisun asiaan rajapinnan kehittyessä. Näin ollen oman rajapinnan valmistuttua ratakilometreille saatiin myös tarvittavat koordinaatit.

Alkututkimusten jälkeen varsinainen projektin tekeminen voitiin aloittaa. Ensin luotiin perus React-projekti komennolla npx create-react-app. Komento luo valmiin Next.js full-stack -kehikon, jonka pohjalta on helppo alkaa rakentaa eri kokoisia React-sovelluksia. Seuraavaksi luotiin codestyle-tiedosto, joka pitää koodin samanlaisena koko projektissa. Lopuksi React-versio vaihdettiin vanhempaan versioon, koska haluttiin käyttää Material UI -kirjastoa. Kirjasto ei ollut yhteensopiva uusimman React-version kanssa projektia kehittäessä.

4.3 Käyttöliittymän kehitys

Käyttöliittymän kehitys aloitettiin asentamalla tarvittavat kirjastot projektiin sekä peruskäyttöliittymän implementoimisella. Käyttöliittymää alettiin kehittämään piirustusten perusteella. Sovelluksen avauksen jälkeen tulisi aueta kirjautumissivu, johon kirjaudutaan käyttäjätunnuksella ja salasanalla. Kirjautumisen onnistuessa päädytään kotisivulle. Kotisivulta voi hakea tietyn junan reitin päivämäärän mukaan. Haun onnistuessa annetuilla tiedoilla päädytään datan näyttävälle sivulle. Sivulla on muun muassa reitin korkeuserot, junan tiedot, nopeus ja junan sijainti reitillä.

Kirjautumissivuun käytettiin MUI-kirjastoa ja alkuun siitä tehtiin niin yksinkertainen kuin mahdollista. Navigointi sivustojen välillä toteutettiin käyttämällä React Router JavaScript -kehikkoa esimerkkikoodin 1 mukaisesti.

```
export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="login" element={<LoginPage />} />
           <Route path="login" element={<ChartPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}
```

Esimerkkikoodi 1. React Routerin luominen sovellukseen ja navigoinnin tekeminen.

React Router on JavaScript-kehikko, joka mahdollistaa asiakas- ja palvelinpuolen reitityksen Reactia käyttävissä sovelluksissa. Kehikon avulla voidaan tehdä internetsivu tai mobiilisovellus, joka ei päivity uudelleen sivua vaihdettaessa. React Router mahdollistaa myös selainhistorian toimintojen käytön säilyttäen oikean näkymän sovelluksessa. (10.)

Kirjautumisen jälkeen siirrytään kotisivulle, josta saa haettua tietyn junan reitin ja reittiin liittyvät tiedot. Tiedot haetaan rajapinnasta junanumeron ja päivämäärän perusteella. Haun voi tehdä vasta, kun kumpaankin kenttään on syötetty arvot. Jos tietoja ei löydy, annetaan virheilmoitus käyttäjälle ja pyydetään tarkistamaan, onko junanumero ja päivämäärä oikein. Junan sijainti haetaan KUPLAn (kuljettajan päälaite) avulla tai käytetään käyttäjän oman laitteen sijaintia.

KUPLA on järjestelmä, jolla Finntraffic sähköistää rautatieliikenteessä käytettyjä tiedonvälitysmenettelyitä. Järjestelmä on mahdollistanut paperisista aikatauluista luopumisen, koska kuljettaja pystyy välittämään tietoja sähköisesti liikenteenohjauksen sekä liikenteenhallintajärjestelmien kanssa. Järjestelmän myötä paperiset aikataulut jäävät varajärjestelmiksi, koska järjestelmän käyttäminen on pakollista raideliikenteessä valtion rataverkolla ja vaihtotyössä rautatieliikennepaikkojen välillä. (11.) Sijaintia pyydetään, kun junanumeron haku on tehty. Käyttäjän hylätessä oman laitteen sijainnin käytön kuvaaja luodaan käyttäen KUPLAn sijaintitietoja. Sijaintia pyydetään esimerkkikoodin 2 mukaisesti.

```
function handlePermission() {
     navigator.permissions.query({ name: "geolocation" }).then((result)
     => {
     if (result.state === "prompt") {
          report(result.state);
          navigator.geolocation.getCurrentPosition(
               revealPosition,
               positionDenied,
               geoSettings,
          );
     } else {
         report(result.state);
     }
     result.addEventListener("change", () => {
          report(result.state);
     });
 });
}

function report(state) {
  console.log(`Permission ${state}`);
}

handlePermission();
```

Esimerkkikoodi 2. Sijainnin pyytäminen ja vastauksen kirjaaminen.

Sijainnin pyytämiseen oli alun perin suunniteltu oma ilmoitus, mutta valitettavasti sijainnin pyytäminen tapahtuu automaattisesti selaimessa. Kyselystä saa kuitenkin selville, hylkäsikö käyttäjä laitteen sijainnin käytön vai ei. Jos käyttäjä estää oman laitteen sijainnin käytön tai se ei toimi jostain syystä, käytetään KUPLAn GPS-tietoja. KUPLAn GPS-tiedot tulevat kuitenkin vain 6 sekunnin välein. Junan sijainti kuvaajalla ei siis ole reaaliaikainen tätä vaihtoehtoa käytettäessä. Tämän takia suositaan käyttäjän oman laitteen GPS-sijainnin käyttöä, koska tällöin saadaan reaaliaikainen sijainti.

4.4 Junan tietojen ja kuvaajan luonti

Junan ja sen reittiin liittyviä tietoja näyttävän sivun kehittäminen aloitettiin tekemällä komponentti kuvaajan yläpuolelle. Sivun kehittäminen aloitettiin tästä, koska kuvaajaan tulevat tiedot ovat erillisiä komponentin tiedoista. Komponentin tekoon käytettiin MUI-kirjastosta saatavaa App Bar -komponenttia, jolla saadaan hyvä pohja omalle komponentille (kuva 8).

Kuva 8. MUI-kirjastosta saatava komponentti.

Sovelluksessa ei tarvittu navigointiin tarkoitettua painiketta vasemmassa reunassa eikä myöskään kirjautumispainiketta oikeassa reunassa. Nämä poistettiin ennen komponentin muokkaamista. Tekstin tilalle tuli allekkain junan numero, paino ja pituus. Näiden oikealle puolelle tehtiin Radio Group, jolla valitaan sijainnin käyttö. Vaihtoehdot ovat KUPLA tai oma laite. Käyttäjän antaessa vahingossa väärän junan numeron tarvittiin painike, jolla voi palata takaisin edelliselle sivulle. Painike tuli sijainnin valinnan viereen. Painikkeen viereen tehtiin vielä toinen painike, jolla pystyy vaihtamaan sovelluksen vaaleaan tai tummaan tilaan (kuva 9).

Kuva 9. Valmis yläpalkkikomponentti.

Junaan liittyvät tiedot saatiin alun perin JSON-tiedostosta, joka oli esimerkki siitä, miltä sovellusrajapinta tulee näyttämään. Rajapinnan valmistuessa siirryttiin JSON-tiedostosta rajapinnan käyttöön. Junan tiedot saadaan junakokoonpanoista, joissa on junan pituus, paino ja nopeus. Sovellusrajapintaa kutsuttiin Axios-nimisen JavaScript-kirjaston avulla.

Axios on lupauksiin perustuva http-asiakasohjelma node.js:ään ja selaimeen. Se on myös isomorfisminen, mikä tarkoittaa, että selain ja node.js toimivat samalla koodikannalla. Palvelinpuolella käytetään natiivia node.js http -moduulia ja selainpuolella XMLHttpRequest-moduulia. (12.) Tietoja lukiessa piti ottaa huomioon, että junan pituus ja paino voivat muuttua liikennepaikkojen välillä. Juna voi ottaa mukaan lisää lastia tai vaunuja matkan edetessä. Junakokoonpanojen lisäksi rajapinnassa on journeySection-objekteja. Objektien perusteella

tiedetään junan paino ja pituus tietyssä kohdassa raidetta. Oikea journeySection luettiin ratakilometrin perusteella eli junan sijainnin mennessä tietyn kohdan ohi. Sijainnin saamiseen käytetään automaattisesti käyttäjän omaa laitetta, jos käyttäjä hyväksyi paikantamisen käytön junan hakua tehtäessä. Sijainti saadaan leveys- ja pituuspiireissä esimerkkikoodi 3:n mukaisesti.

```
[object GeolocationPosition] {
  coords: [object GeolocationCoordinates] {
    accuracy: 13228.39334523453,
    altitude: null,
    altitudeAccuracy: null,
    heading: null,
    latitude: 60.1980928,
    longitude: 24.9430016,
    speed: null
  },
  timestamp: 1711544917819
}
```

Esimerkkikoodi 3. Käyttäjän laitteen sijaintitiedot.

Jos sijaintia ei jostain syystä saada käyttäjän laitteen perusteella, voidaan käyttää KUPLAn sijaintitietoja. Käyttäjän pitää kuitenkin itse vaihtaa sijainti KUPLAn käyttöön sivuston yläpalkista. Vaihtamisen jälkeen sijaintitiedot haetaan rajapinnasta, josta saadaan junan sijainti leveys- ja pituuspiirien mukaan. Sijainti tulee JSON-muodossa esimerkkikoodi 4:n mukaan.

```
"geometry": {
        "type": "Point",
        "coordinates": [22.808656, 63.317011]
      },
```

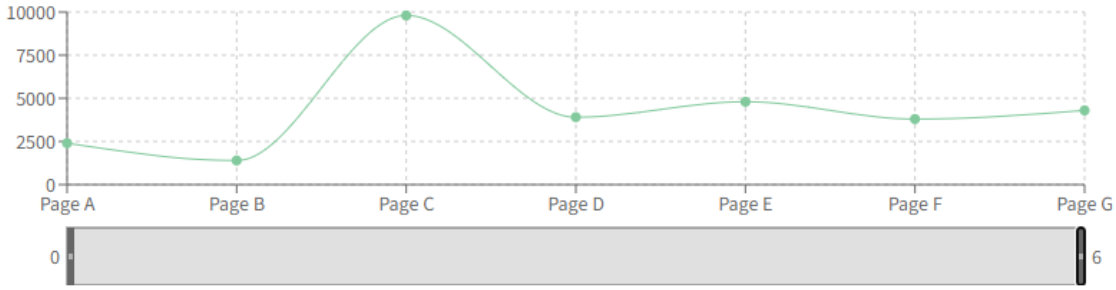
Esimerkkikoodi 4. Sijaintitiedot JSON-muodossa.

Erona oman laitteen sijaintiin on kuuden sekunnin viive sijaintitiedoissa. Sijaintitietojen saamiseen tarvitaan myös junanumero ja junan lähtöaika. KUPLAn sijaintitietojen käyttäminen on siis hieman monimutkaisempaa kuin oman laitteen käyttäminen.

Tumma- ja vaaleatila toteutettiin MUI:n teemojen avulla. Teema spesifioi muun muassa komponenttien värit, varjostuksen, elementtien sopivan läpinäkymättömyyden ja pintojen synkkyyden. Teema mahdollistaa myös yhdenmukaisen värityksen koko sovellukseen ilman, että jokaista komponenttia tarvitsee muuttaa

erikseen. MUI:n avulla on helppo tehdä omat teemat tummalle ja vaalealle tilalle.

Yläpalkkikomponentin valmistuttua alettiin työstämään kuvaajaa, joka on myös oma komponenttinsa. Kuvaajan tekoon käytettiin Recharts-kirjastoa, jolla saadaan tehtyä monipuolisia kuvaajia. Projektissa käytettiin pohjana kirjaston kohdennettavaa kuvaajaa. Kuvaajalla pystyy kohdentamaan tiettyyn alueeseen liukusäätimen avulla. (kuva 10).



Kuva 10. Kuvaaja, jolla voi kohdentaa tiettyyn alueeseen liukusäätimellä (13).

Kuvaajan tulisi näyttää pystyakselilla korkeus merenpinnasta ja vaaka-akselilla ratakilometrit. Kuvaajassa tulisi myös näkyä radan kaltevuudet pituussuuntaiseen vaakatasoon nähden, junan sijainti ja liikennepaikat. Tiedot saatiin suoraan rajapinnasta JSON-muodossa, josta ne tallennettiin React Reduxiin. React Redux on tilanhallinta-kirjasto. Kirjastolla voidaan tallentaa tietoja lokaalisti sovellukseen ja kutsua tietoja muihin tiedostoihin sovelluksen sisällä tekemättä uutta rajapintakyselyä. (14.)

Kuvaajan kehittäminen aloitettiin laittamalla korkeustiedot pystyakselille ja ratakilometrit vaaka-akselille. Ensimmäinen ongelma ilmeni jo ratakilometrien kohdalla. Kuvaajan tulisi olla lineaarinen, jotta junankuljettaja pystyy arvioimaan nopeuden ja ennakoimaan ylä- ja alamäkiin mahdollisimman hyvin. Rajapinnasta saatava data ei kuitenkaan anna ratakilometrejä lineaarisesti vaan datassa saattaa olla esimerkiksi kilometrin tai kahden kilometrin eroja ratakilometrien välillä. Toinen ongelma oli korkeusarvojen puuttuminen osasta ratakilometrejä. Ratakilometrien epätasaisuuteen löytyi kuitenkin ratkaisu back-end-puolella, jonka jälkeen rajapinnasta saatiin arvot lineaarisesti. Puuttuvat korkeusarvot piti

interpoloida front-endin-puolella. Interpolointi tarkoittaa tuntemattoman pisteen laskemista tunnettujen pisteiden arvojen avulla. Tässä tapauksessa siis laskettiin puuttuvat korkeusarvot muiden arvojen kautta. Laskemiseen käytettiin lineaarista interpolointia kaavan 4 mukaan.

$$(y) = y_1 + (x - x_1)\frac{(y_2 - y_1)}{(x_2 - x_1)} \quad (4)$$

Kaavassa 4 kirjain y on puuttuva korkeusarvo, y1 on ensimmäinen tunnettu korkeusarvo ja y2 toinen tunnettu arvo. Vastaavasti x1 on tunnettu ratakilometri ja x2 on toinen tunnettu arvo. x on ratakilometriarvo, jonka korkeutta yritetään selvittää.

Pystyakselin ja vaaka-akselin arvojen valmistuttua kuvaajaan mietittiin muiden tietojen toteutusta. Kuvaajaan tulisi saada näkyviin radan kaltevuudet, junan sijainti ja liikennepaikat. Mainittujen tietojen saamiseen kuvaajaan käytettiin foreignObject-nimistä SVG (Scalable Vector Graphics) -elementtiä. Elementtiä oli pakko käyttää, koska Recharts-kirjasto hyväksyy ainoastaan SVG-elementin kuvaajan lapsielementiksi. Tällä tavalla saadaan tehtyä omia muunneltavia elementtejä kuvaajalle. Asian selvittyä laitettiin radan kaltevuudet kuvaajalle. Kaltevuudet saatiin rajapinnasta desimaaleina, jotka muutettiin promilleiksi. Tähän päädyttiin, koska raideliikenteessä kaltevuudet ilmoitetaan yleisesti ottaen promilleina. Lopuksi kaltevuustietojen ympärille piirrettiin ääriviivat, jotta ne näkyisivät helpommin.

Kaltevuustietojen valmistumisen jälkeen yhdistettiin junan sijainti oikeaan ratakilometriin. Samalla lisättiin junan sijaintia havainnollistava veturilogo kuvaajalle ja junan pituus pitää myös esittää kuvaajalla. Logon saaminen kuvaajaan kävi vaivattomasti samalla periaatteella, kuin kaltevuuksien saaminen kuvaajalle. Sijainnin yhdistäminen ei kuitenkaan sujunut aivan vaivattomasti, koska piti löytää järkevä toteutus. Yksi mahdollisuus voisi olla laskea, mikä on lähin ratakilometri, joka vastaa käyttäjän laitteen tai KUPLAn antamaa sijaintia. Tuloksen perusteella junan sijainti voitaisiin laittaa kuvaajalle ratakilometrin kohdalle. Näin myös toimittiin, koska se vaikutti olevan yksinkertaisin tapa käyttämättä liikaa aikaa vaihtoehtoisen toteutuksen pohtimiseen. Lähimmän ratakilometrin

laskemiseen käytettiin Haversinen kaavaa (kaava 5). Haversineen löytyi myös kirjasto, jonka avulla laskeminen tapahtui nopeasti front-endissä.

$$haversine\left(\frac{d}{r}\right) = haversine(\Phi_2 - \Phi_1) + \cos(\Phi_1)\cos(\Phi_2)\, haversine(\lambda_2 - \lambda_1) \quad (5)$$

Kaavalla 5 voidaan laskea lyhyin kahden pisteen välillä oleva etäisyys pallon pintaa pitkin. Kaava on erityisen tärkeä navigoinnissa, mutta se ei ole täysin tarkka. Kaavassa oletetaan maapallon olevan täydellinen pallo, minkä takia etäisyys tulee olemaan hieman epätarkka. Kaavassa 5 kirjain r on maapallon säde ja d on kahden pisteen välinen etäisyys. Φ on pisteiden leveysasteet ja λ on pisteiden pituusasteet. Kaavan avulla saatiin laskettua sovelluksessa lähin ratakilometri vertaamalla sijainnista saatuja leveys- ja pituusasteita ratakilometrien asteisiin. Junan sijainti yhdistettiin ratakilometriin, jonka etäisyys oli pienin käyttäjän sijaintiin verrattuna.

Haversine on trigonometrinen funktio, kuten sini, kosini ja tangentti. Haversine voidaan esittää esimerkiksi kaavalla 6. Tämä kaava on todennäköisesti ensimmäinen tarkasteltava yhtälö, kun halutaan ymmärtää etäisyyksien laskemista pallolla.

$$haversine(\theta) = \sin^2(\frac{\theta}{2}) \quad (6)$$

Haversinen kaava on uudelleen muotoiltu kosinien laista, mutta muotoilu haversinen suhteen on hyödyllisempää pienille kulmille ja etäisyyksille. Sana Haversine tulee myös kyseisestä yhtälöstä. Syy miksi haversinea ei ole vain sievennetty sineiksi ja kosineiksi liittyy historiaan. Funktio kehitettiin aikana, jolloin ei ollut laskimia eikä tehokkaita tietokoneita ja erilaiset kulmat sekä etäisyydet laskettiin lokitaulun avulla. (15.)

Asiakkaan kanssa käydyn keskustelun pohjalta junan pituutta aloitettiin kehittämään kuvaajalle. Junan pituuden pitäisi näkyä kuvaajalla punaisena, kun sijainti on saatavilla. Sijainnin puuttuessa pituuden tulisi näkyä vihreällä. Ratakilometrin kohdalla pitäisi näkyä veturilogo ja mahdolliset vaunut sen perässä. Veturilogo havainnollistaa junan sijaintia ja vaunut junan pituutta. Kuten ratakilometrien

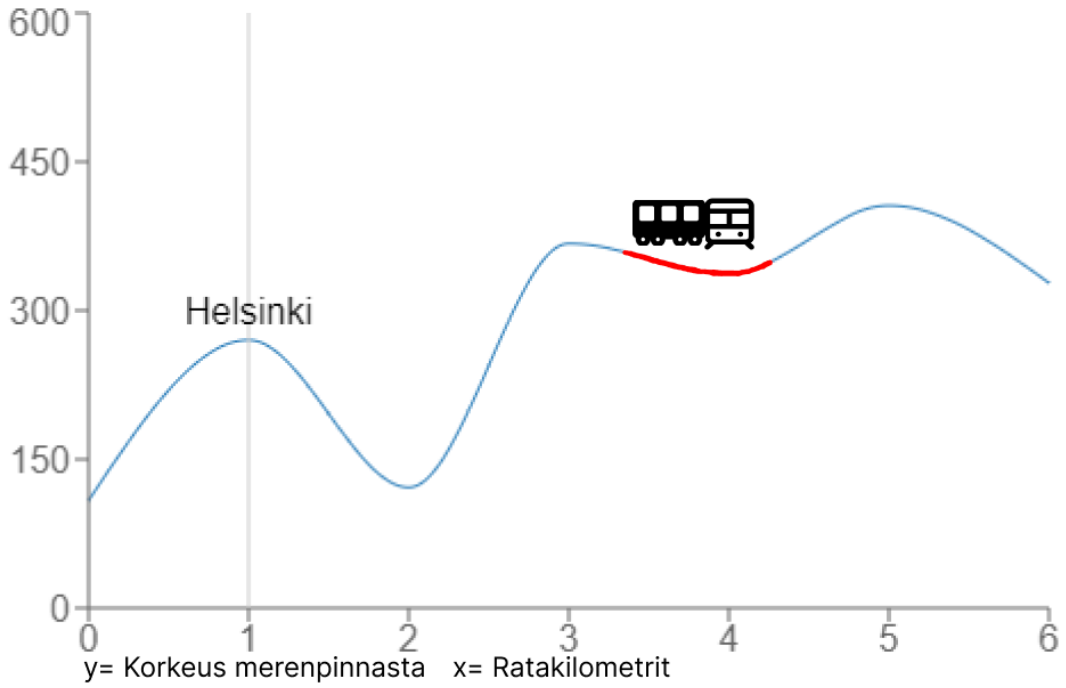
kohdalla, myös sijainnin kohdalla piti selvittää, miten asian voi toteuttaa Recharts-kirjastolla. Yksi tapa toteuttaa asia olisi luoda kaksi eri suoraa. Suorista ensimmäinen on junan sijainti ja toinen on varsinainen data eli ratakilometrit, liikennepaikat ja niin edelleen. Ongelmaksi kuitenkin muodostuisi suorien renderöinti päällekkäin ilman virheitä. Molempien suorien päivittäminen samaan aikaan olisi myös hankalaa, koska sijainnin pitäisi liikkua kuvaajassa. Näiden ongelmien vuoksi päädyttiin helpompaan ja vähemmän kyseenalaiseen ratkaisuun. Suorat ovat SVG-elementtejä, joten ne hyväksyvät viivaelementin. Kaikista helpoin tapa toteuttaa asia olisi muuttaa elementin väriä (esimerkkikoodi 5).

```
<Line dataKey="value" stroke={
    d => d > 50 ? "red" : "green"
} />
```

Esimerkkikoodi 5. Viivan värin vaihtaminen dataan perustuen.

Valitettavasti tämä ei ole mahdollista Recharts-kirjastolla tällä hetkellä. Viivan värin vaihtaminen toteutettiin vaihtoehtoisen elementin avulla. Suorat ovat SVG-elementtejä, joten viivaelementtinä voitiin käyttää LinearGradient-nimistä elementtiä. Tämä mahdollistaa värin muuttamisen muun muassa käyttäjän toimintojen perusteella. Voidaan siis muuttaa esimerkiksi suoran väriä, kun käyttäjä laittaa hiiren sen päälle. Asian selvittyä voitiin junan piirtämisen toteutus aloittaa. Lähimmän ratakilometrin perusteella laskettiin, mihin asti juna pitää piirtää. Junan pituus miinustettiin ensin sijaintia vastaavasta ratakilometristä. Tällöin saatiin piste, johon asti juna yltää. Piste ei todennäköisesti osu ratakilometriin, joten käytettiin lineaarista interpolointia. Pisteen interpoloinnin jälkeen junan pituus saatiin piirrettyä pisteen ja junan sijainnin välille.

Veturilogon ja vaunujen saaminen kuvaajaan toteutettiin ReferenceLine-komponentilla. Komponentin avulla voidaan luoda suoralle referenssi dataan pohjautuen. Komponentille annetaan joko x- tai y-arvo, jonka perusteella luodaan referenssiviiva suoralle. Toinen vaihtoehto on antaa komponentille useampi piste x- ja y-arvoilla, jolloin viiva piirtyy niiden välille. Tavoista jälkimmäistä hyödynnettiin projektissa (kuva 11).



Kuva 11. Kuvaaja, jossa veturilogo ja vaunut havainnollistavat junan sijaintia ja pituutta.

Kuvassa 11 liikennepaikka on luotu antamalla komponentille yksi arvo, jolloin se asettuu ratakilometrin kohdalle. Veturilogo ja vaunut sen sijaan asettuvat kuvaajalle kahden arvon avulla. Veturi on junan varsinainen sijainti ja vaunut havainnollistavat pituutta. Liikennepaikkojen saamiseen kuvaajalle käytettiin ReferenceLine-komponenttia ja foreignObject-elementtiä, kuten radan kaltevuuksienkin kanssa. Liikennepaikat saatiin rajapinnasta samassa JSON-objektissa kuin ratakilometrit. Kaikilla ratakilometreillä ei kuitenkaan ole liikennepaikkoja. Liikennepaikka voitiin siis helposti sijoittaa kuvaajalle, koska ne ovat tietyissä ratakilometreissä. Liikennepaikan tausta värjättiin ja tehtiin ääriviivat sen ympärille näkyvyyden parantamiseksi.

4.5 Kuvaajan eri näkymät

Kuvaajaan perustoimintojen valmistuttua asiakkaalta kysyttiin, mitä muuta kuvaajaan vielä haluttaisiin. Päädyttiin siihen, että kuvaajaan tehdään puolentoista kilometrin näkymä ja kymmenen kilometrin näkymä. Näkymistä kymmenen kilometrin näkymä on oletusnäkymä. Näkymään tehtiin myös painikkeet, jolla voi

liikkua kuvaajassa kilometrin edespäin tai taaksepäin. Painikkeiden avulla kuljettaja voi katsoa reittiä pidemmälle kuin kymmenen kilometriä, jos sille on tarvetta. Puolentoista kilometrin näkymässä junan sijainti laitetaan keskelle kuvaajaa eli näytetään 750 m eteen- ja taaksepäin. Näkymä on siis yhteensä 1,5 km. Kymmenen kilometrin näkymässä näytetään 10 km junan sijainnista eteenpäin. Asiakkaan toiveesta kuvaajaa muutettiin niin, että kaltevuustiedot näkyvät vain 1,5 km:n näkymässä. Vastaavasti 10 km:n näkymässä näytetään ainoastaan liikennepaikat ja junan sijainti.

Ratakilometrit tulivat rajapinnasta koko matkalle, joten näkymiä varten dataa piti suodattaa. Kymmenen kilometrin näkymässä datasta suodatettiin pois kaikki ratakilometrit kymmenen kilometrin jälkeen. Datan suodatuksen jälkeen data laitettiin Redux-taulukkoon. Puolentoista kilometrin näkymä piti toteuttaa käyttämällä lineaarista interpolointia, koska tarvittiin pisteet 750 m junan sijainnista taakse- ja eteenpäin. Tämä toteutettiin lisäämällä junan sijaintia vastaavaan ratakilometriin 750 m ja interpoloimalla korkeusarvot pisteelle. Sama prosessi tehtiin myös toiselle pisteelle eli vähennettiin 750 m junan sijaintia vastaavasta ratakilometristä ja interpoloitiin korkeusarvot. Interpoloinnin jälkeen pisteet lisättiin Reduxissa olevaan taulukkoon ja luotiin 1,5 km:n näkymä käyttäjän vaihtaessa siihen.

Kaltevuustiedot haluttiin ainoastaan 1,5 km:n näkymään, joten ne poistettiin ensin 10 km:n näkymästä. Rajapinnasta tulee kaltevuustiedoille kaksi arvoa, joista pitää valita oikea radan ylä- ja alamäkiin perustuen. Ylämäkeen mentäessä kaltevuustiedon tulisi olla positiivinen ja alamäessä negatiivinen. Oikea kaltevuusarvo valittiin vertaamalla peräkkäisten ratakilometrien korkeuseroja. Korkeuden laskiessa mennään alamäkeen, jolloin valitaan laskeva kaltevuusarvo. Vastaavasti korkeuden noustessa kuljetaan ylämäkeen, jolloin valitaan nouseva kaltevuusarvo.

Kymmenen kilometrin näkymän toteutus onnistui vaivattomasti suodattamalla datasta arvot junan sijaintiin perustuen. Junan sijaintiin siis lisättiin kymmenen kilometriä ja suodatettiin loput arvot pois. Oletusnäkymänä käytettiin 10 km:n näkymää. Puolentoista kilometrin näkymään päästiin painamalla painiketta.

Kaltevuustiedot piilotettiin käyttämällä ehdollista renderöintiä. Ehdollinen renderöinti tarkoittaa, että luodaan tietty komponentti perustuen eri ehtoihin. Tässä tapauksessa katsottiin, onko kuvaaja oletusnäkymässä vai 1,5 km:n näkymässä.

Näkymien toteuttamisen jälkeen tehtiin painikkeet eteen- ja taaksepäin liikkumiseen. Painikkeilla voi liikkua kuvaajassa yhden kilometrin eteenpäin tai taaksepäin. Painikkeet toteutettiin samalla tavalla kuin kaltevuudet eli käyttämällä Recharts-kirjaston foreignObject-elementtiä. Painikkeet sijoitettiin kuvaajan kumpaankin päätyyn eli oikealla olevasta painikkeesta pääsi eteenpäin ja vasemmalla taaksepäin. Painikkeiden funktionaalisuus saatiin toteutettua helposti muutamalla taulukkoa, josta tiedot saadaan. Mentäessä kymmenen kilometrin näkymässä eteenpäin taulukosta otettiin pois ensimmäinen alkio ja loppuun lisättiin seuraava alkio. Taulukon muutoksen jälkeen kuvaaja päivitettiin uudelleen, jolloin liikuttiin 1 km eteenpäin kuvaajalla.

Kuvaajan valmistuttua huomattiin, että näkyvyys pienillä näytöillä on huono. Kuvaaja leikkautui näytön yläreunasta, jolloin kaikkia tietoja ei ollut mahdollista lukea. Sovellukseen tehtiin komponentti, jolla kehotetaan käyttäjää kääntämään puhelin vaakatasoon junatietojen haun jälkeen. Tällä tavalla vältyttiin lisämuutoksilta kuvaajaan ja eri näyttökokojen sovituksiin, koska vaakatasossa kuvaaja näkyi kaikilla näyttökoilla hyvin.

4.6 Junan nopeuden näyttäminen

Kuvaajan piirtämisen jälkeen tarvittiin vielä junan nopeustiedot. Tietoja varten tehtiin oma komponentti, joka laitettiin kuvaajan alle. Junan nopeustiedot saadaan rajapinnasta, ja ne perustuvat Digitrafficin tietoihin. Junan nopeustiedot saadaan junakokoonpanoista. Nopeustietoja lukiessa pitää valita oikea journeySection-objekti. Junan pituuden tai painon lisääntyessä myös junan nopeus tulee muuttumaan. JourneySection luetaan ratakilometrin perusteella eli junan mennessä tietyn kohdan ohi. Objektista saadaan myös junan nykyinen nopeus ja keskinopeus. Nopeudet saadaan muodossa km/t, joten niille ei tarvitse tehdä mitään muutoksia. Junan reaaliaikainen nopeus sijoitettiin komponenttiin

päällimmäiseksi ja keskinopeus sen alle. Lopuksi komponenttiin tehtiin liukusäädin, jolla ominaisuuden voi kytkeä päälle tai pois.

5 Yhteenveto

Opinnäytetyössä selitettiin pystygeometrian käyttöä raideliikenteessä ja asioiden yhteys toisiinsa. Lisäksi kerrottiin Suomessa käytössä olevasta ratakilometrijärjestelmästä ja siihen kuuluvista asioista kuten rautatieliikennepaikoista. Näitä tietoja käytettiin sovelluksen luonnissa, ja ne olivat erityisen tärkeitä, koska aihealue ei ollut ennestään tuttu.

Työssä opittiin, että pystygeometrian hyödyntäminen sovelluskehityksessä ei ole kaikkein yksinkertaisin tehtävä. Kehityksen aikana pidettiin monia palavereja, joissa mietittiin, mikä olisi paras tapa toteuttaa tietty ominaisuus. Projektin alussa kohdattiin muun muassa ongelma ratakilometrien kanssa, mikä saatiin ratkaistua tiimityöllä. Asiakkaan kanssa pidettiin säännöllisin väliajoin palavereja, joissa keskusteltiin projektin kulusta ja mahdollisista ongelmista. Palaverit olivat tärkeitä, koska joidenkin ominaisuuksien toteutusta jouduttiin hieman muuttamaan. Projektin kehityksen aikana tuli myös uusia ideoita, joita asiakas ei alun perin pyytänyt. Näihin lukeutui esimerkiksi kuvaajan eri näkymät.

Opinnäytetyössä selostettiin selainpohjaisen sovelluksen tekeminen JavaScriptillä ja Reactilla. Teknologiat valittiin aikaisemman kokemuksen perusteella ja sovellus tehtiin raideliikenteessä toimivalle yritykselle. Projekti kesti kolmisen kuukautta, jonka aikana sovellus saatiin testaukseen asiakkaalle. Sovelluksella näkee junalla ajettavan reitin pystygeometrian. Sovelluksella siis näkee reitin korkeuserot, ratakilometrit, liikennepaikat ja radan kaltevuus. Lisäksi näytettiin junan paino, pituus ja nopeus. Junankuljettajat hyödyntävät sovellusta raideliikenteessä. Sovelluksen avulla junankuljettajat voivat muun muassa ennakoida nopeuden ala- ja ylämäkiin, nähdä junan sijainnin ja nykyisen ratakilometrin. Asiakkaan toiveet saavutettiin, mutta sovelluksen testaus jäi melko vähälle. Projektia voi jatkaa vielä pidemmälle laajempien testien jälkeen. Vähäisen testauksen takia varsinkin sovelluksen aliohjelmia ja toiminnallisuutta voi parantaa huomattavasti.

Lähteet

1 Seppälä, Henri; Myllymäki, Tuija; Niemi, Petri; Lindholm, Henri; Tast, Maunu; Martikainen, Juha-Pekka; Moilanen, Markus; Sipiläinen, Antti & Hölttä, Pasi. 2021. Ratatekniset ohjeet (RATO) osa 2 Radan geometria. Helsinki: Väylävirasto.

2 Ratasuunnittelu. Verkkoaineisto. Helsingin Kaupunki. <https://raitiotie-ohje.fi/ratasuunnittelu/#raiteen_vaakageometria>. Luettu 29.3.2024.

3 Lohilahti, Lasse. 2019. Ratageometria mittausvastaavan työkaluna. Opinnäytetyö. Lapin Ammattikorkeakoulu. Theseus-tietokanta.

4 Taimela, Reijo. 2011. Raidegeometrian suunnittelu. Helsinki: Liikennevirasto.

5 Sundström, Saara. 2012. Raidegeometrian laadunvarmistus KoneGIS-järjestelmässä. Opinnäytetyö. Metropolia Ammattikorkeakoulu. Theseus-tietokanta.

6 Niemi, Petri; Lindholm, Henri; Tast, Maunu; Martikainen, Juha-Pekka; Moilanen, Markus; Sipiläinen, Antti & Hölttä, Pasi. 2021. Ratatekniset ohjeet (RATO) osa 7 Rautatieliikennepaikat. Helsinki: Väylävirasto.

7 File:Heinola km168.jpg. Verkkoaineisto. Wikimedia Commons. <https://commons.wikimedia.org/wiki/File:Heinola_km168.jpg>. Päivitetty 18.2.2023. Luettu 20.5.2024.

8 Sirkiä Teemu. 2021. Julia – Pystygeometria. Verkkoaineisto. < https://juliadata.fi/pystygeometria/>. Päivitetty 28.12.2021. Luettu 8.4.2024.

9 Rautatieliikenne. Verkkoaineisto. <https://www.digitraffic.fi/rautatieliikenne/#raideosuudet>. Luettu 8.4.2024.

10 React Router. Verkkoaineisto. W3 schools. <https://www.w3schools.com/react/react_router.asp>. Luettu 29.3.2024.

11 KUPLA. Verkkoaineisto. Fintraffic. <https://www.fintraffic.fi/fi/raide/kupla>. Luettu 29.3.2024.

12 What is Axios? Verkkoaineisto. Axios. <https://axios-http.com/docs/intro>. Luettu 29.3.2024.

13 SynchronizedLineChart. Verkkoaineisto. Recharts. <https://recharts.org/en-US/examples/SynchronizedLineChart>. Luettu 8.4.2024.

14 Getting Started with React Redux. Verkkoaineisto. React Redux. <https://react-redux.js.org/introduction/getting-started>. Luettu 29.3.2024.

15 Haversine formula to find distance between two points on a sphere. Verkkoaineisto. GeeksForGeeks. <https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/>. Päivitetty 5.9.2022. Luettu 29.3.2024.