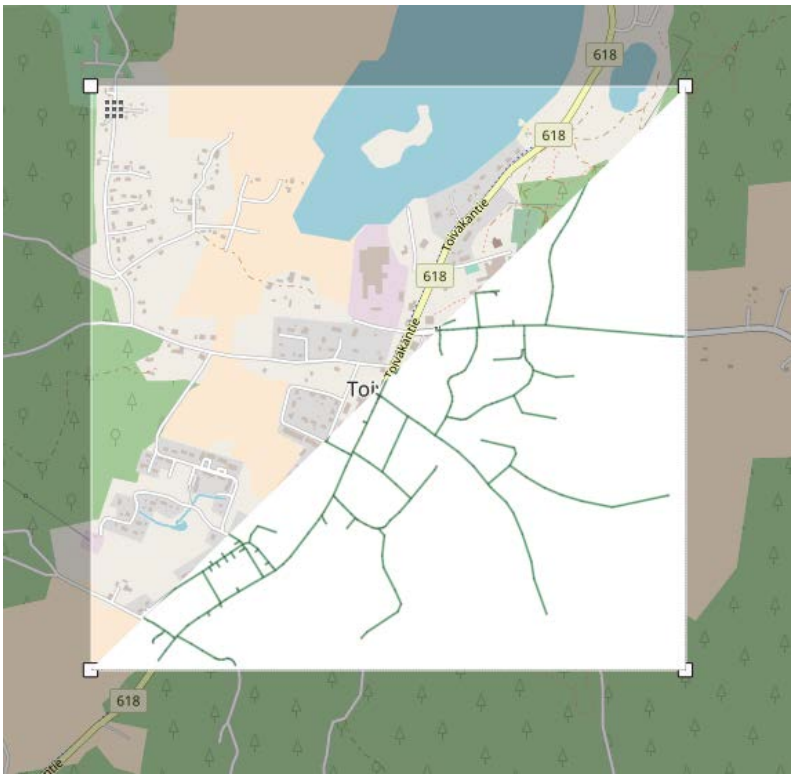
Jussi-Pekka Järvelin

OpenStreetMap-tiedoston lukeminen ja muokkaaminen selainympäristössä



Insinööri (AMK)

Tieto- ja viestintätekniikka

Kevät 2018

KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tiivistelmä

Tekijä: Järvelin Jussi-Pekka

Työn nimi: OpenStreetMap-tiedoston lukeminen ja muokkaaminen selainympäristössä

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: Web-ohjelmointi, OpenStreetMap, XML, SVG

Työn tilaajana oli Magister Solutions. Magister Solutions on Jyväskylässä sijaitseva ohjelmistoyritys, jonka toiminta keskittyy telekommunikaatioratkaisuihin. Magister Solutions toimii myös kansainvälisillä markkinoilla.

Työn tavoitteena oli toteuttaa verkkoselaimessa toimiva ohjelma, jolla pystyy muokkaamaan OpenStreetMapista vietyä karttatietoa. Karttatieto annetaan ohjelmalle OpenStreetMapin käyttämässä OSM XML -formaatissa. Ohjelman täytyisi parsia sille annettu tiedosto ja muokata tiedostosta saatua dataa piirrettävään muotoon. Käyttäjän täytyisi pystyä muokkaamaan piirrettyä tietoa editointityökalulla. Työssä keskityttiin teiden käsittelyyn ja muokkaamiseen.

Työn kannalta tärkeimmät aiheet ovat web-ohjelmoinnin perusteet, XML-metakieli, OSM XML -formaatti ja SVG-vektorigrafiikka. Lisäksi työssä käsitellään uuden projektin luominen käyttämällä Angular-ohjelmistokehystä.

Työtä lähdettiin kehittämään ennalta toteutetun tietorakenteen päälle. Ensin toteutettiin OSM XML -tiedoston lukeminen ja käsittely. Tiedostosta saadulle tiedolle tehtiin tarvittavat koordinaattimuunnokset, jonka jälkeen tieto lisättiin tietokantaan. Tiet piirrettiin käyttämällä SVG-vektorigrafiikkaa tietokannasta saadun tiedon perusteella. Lopuksi toteutettiin metodit teiden valitsemiselle ja poistamiselle.

Lopputuloksena on verkkoselaimessa toimiva ohjelma, jota käyttämällä pystyy rajoitetusti muokkaamaan OpenStreetMapin karttatietoa. Tiet piirtyvät oikein ja valitut tiet pystytään poistamaan sekä ohjelmasta että tietokannasta.

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Abstract

Author(s): Järvelin Jussi-Pekka

Title of the Publication: Reading and Editing an OpenStreetMap File in Web Environment

Degree Title: Bachelor of Engineering, Information and Communication Technology

Keywords: Web programming, OpenStreetMap, XML, SVG

This thesis was ordered by Magister Solutions. Magister Solutions is a software company located in Jyväskylä. Magister Solutions' main expertise is telecommunication solutions and they work also world-wide.

The goal of the thesis was to develop a web program for editing map data exported from OpenStreetMap. Map data is given to the program in OSM XML format which is used by OpenStreetMap. The given file would be parsed and edited to the format that can be drawn. The user should be able to edit the drawn data by using the editing tool. Handling and editing the roads were the focus of the thesis.

Most important topics for the thesis were the basics of web programming, XML metalanguage, OSM XML format and SVG vector graphics. Thesis will also go through how to start up a new project by using Angular software framework.

Before-made data structure was used as a base of the program. The first task was to create a way to read and handle the OSM XML file. The data had to go through coordinate transforms before adding to the data structure. The data of the data structure was used to display the roads with SVG vector graphics. The last task was to create methods to select and delete roads.

The result was a web program which has limited features for editing OpenStreetMap map data. Roads are displayed correctly, and selected roads are deleted both from the program and from the data structure.

Alkusanat

Kiitos Magister Solutionsille mielenkiintoisesta toimeksiannosta, sekä työkavereille, jotka auttoivat projektin toteuttamisessa.

Tässä työssä käytetään OpenStreetMapin omaisuutta OpenStreetMap-lisenssin rajoissa.

“© *OpenStreetMap contributors*”

Sisällys

Symboliluettelo

Avainsana (engl. keyword) = Ohjelmointikielessä käytettävä, kielen toiminnallisuuteen varattu sana, jota ei voi käyttää esimerkiksi muuttujien nimenä.

Itäkoordinaatti (engl. easting) = UTM-projektiossa käytetty koordinaatti, joka määrittää kuinka monta metriä itään liikutaan kaistan nollapisteestä.

Komentorivi (engl. command line interface, CLI) = Tekstikomentoihin pohjautuva tapa kommunikoida ihmisen ja tietokoneen välillä.

Käyttöliittymä (engl. interface) = Ohjelmiston osa, jolla käyttäjä käyttää ohjelmistoa.

Latitudi = Englannin kielestä peräisin oleva nimitys leveyspiirille.

Longitudi = Englannin kielestä peräisin oleva nimitys pituuspiirille.

Merkintäkieli tai **kuvauskieli** (engl. markup language) = Kieli tekstin rakenteen tai esitystavan kuvailuun metainformaation avulla.

Ohjelmistokehys (engl. software framework) = Ohjelmistotuote, jonka tarkoitus on nopeuttaa helpottaa ohjelman kehitystä esim. valmiilla komponenteilla.

Pohjoiskoordinaatti (engl. northing) = UTM-projektiossa käytetty koordinaatti, joka määrittää, kuinka monta metriä pohjoiseen liikutaan kaistan nollapisteestä.

Tuonti (engl. import) = Tuodaan esim. ulkopuolista dataa ohjelmaan tai järjestelmään.

Tunniste (engl. tag) = Elementin nimi, jolla merkintäkielen elementit erotetaan toisistaan. Tässä työssä tunnisteet ja elementit erotetaan muusta tekstistä kulmasuluilla, esim. <tunniste> tai <elementti>.

Vienti (engl. export) = Viedään dataa ulos ohjelmasta tai järjestelmästä ulkoiseen käyttöön.

1 Johdanto

Työn aihe on ”OpenStreetMap-tiedoston lukeminen ja muokkaaminen selainympäristössä”. Työn tavoitteena oli toteuttaa ohjelma, jolla pystytään muokkaamaan OpenStreetMapista haettua karttatietoa verkkoselaimessa toimivalla graafisella työkalulla.

Työn toimeksiantajana toimi Magister Solutions, ja työ toteutettiin harjoittelujakson aikana. Magister Solutions on Jyväskylässä sijaitseva telekommunikaatioratkaisuihin painottuva ohjelmistokehitysyritys. Magister Solutions toimii myös kansainvälisillä markkinoilla.

Karttatiedon hyödyntämiselle ja muokkaamiselle on paljon käyttötapoja erilaisissa sovelluksissa, esim. tonttien käytön suunnittelussa, pelimaailmojen generoinnissa tai simulaatioissa. Pelkästään karttatiedon lukeminen ja käsittely haluttuun muotoon nopeuttaa simulaatioympäristöjen luomista huomattavasti.

Nykyään yhä useampi sovellus toimii selainpohjaisena helpon saatavuuden ja jakamisen ansiosta. Selaimen suorituskyky kuitenkin asetti omat haasteensa paljon dataa piirtävälle ohjelmalle. Ohjelman täytyi täten olla sekä helppokäyttöinen että suorituskyvyltään tehokas.

2 Työssä käytetty teoria

2.1 Web-ohjelmoinnissa käytettävät ohjelmointikielet

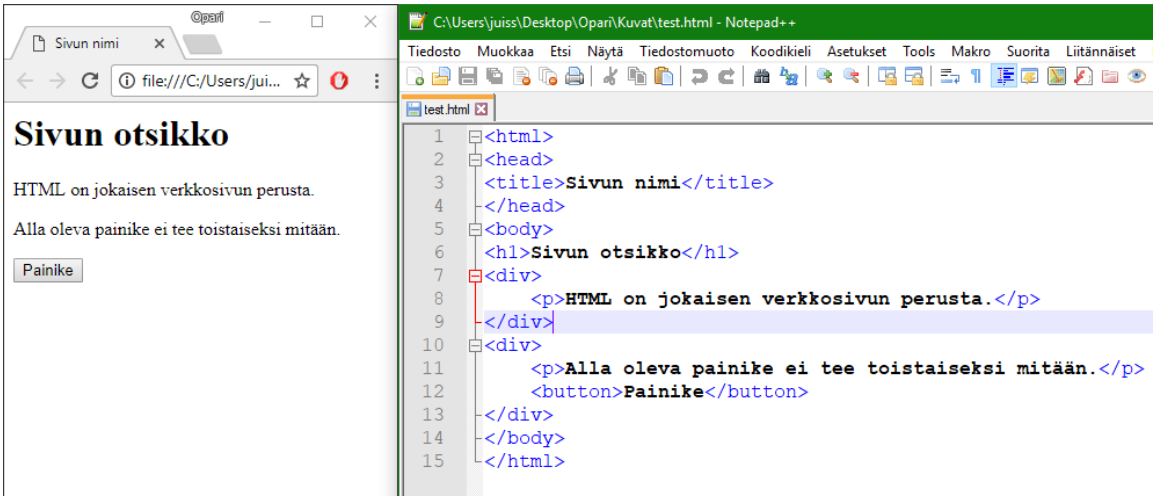
Web-ohjelmoinnin kolme ”peruspilaria” ovat HTML, CSS ja JavaScript. HTML määrittää verkkosivun rakenteen, CSS määrittää sivun muotoilun ja JavaScriptillä toteutetaan sivun toiminnallisuus.

2.1.1 HTML

HTML (Hypertext Markup Language) on merkintäkieli, jolla määritellään verkkosivuston sisältö ja rakenne. Alun perin HTML-dokumentit olivat toisiinsa hyperlinkeillä linkitettyjä tekstitiedostoja, mutta nykyään HTML mahdollistaa paljon monipuolisempien verkkosivujen luomisen. Selain (esim. Google Chrome) tulkitsee HTML-koodia ja luo tarkasteltavan verkkosivun sen perusteella. [1, s. 4]

Tämän hetken W3C suosittelema HTML-standardi on HTML 5.2. [2]

Kuvassa 1 on yksinkertainen esimerkki HTML-kielestä ja sen käyttämisestä.



Kuva 1. Esimerkki HTML:llä luodusta verkkosivusta.

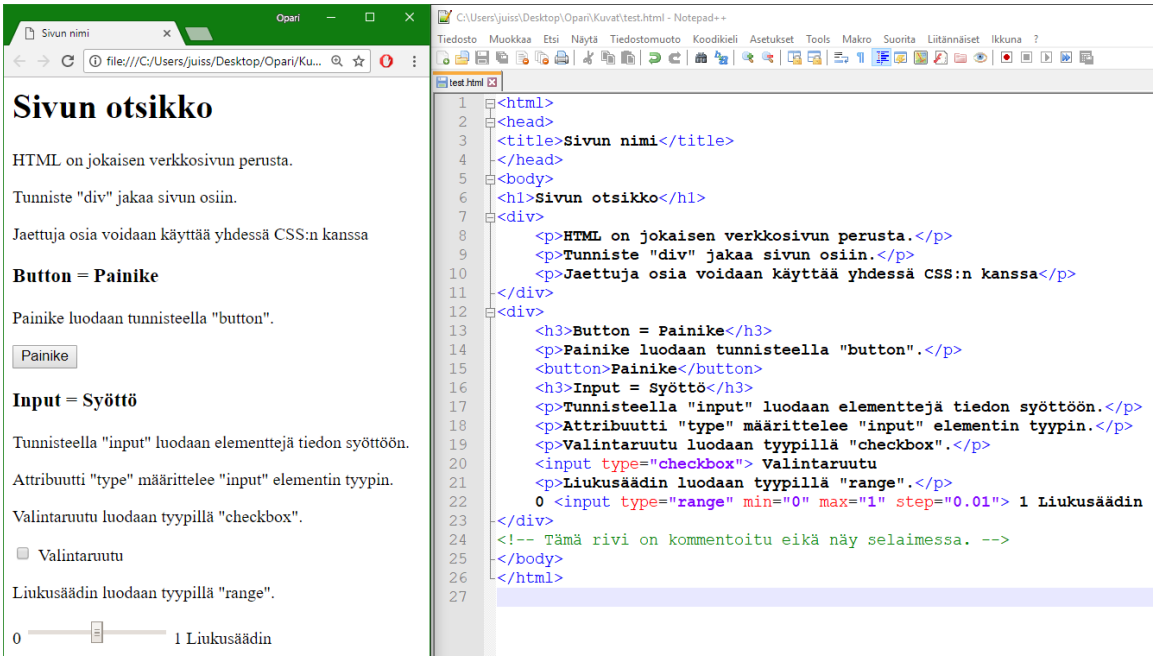
HTML-dokumentti koostuu elementeistä, joita merkitään kulmasulkujen sisällä olevilla tunnisteilla. HTML-sivun tulee sisältää minimissään tunnisteet <html>, <head> ja <body>. [1, s. 12]

HTML-dokumentti alkaa ja päättyy juurielementtiin, jonka tunniste on <html>. Elementit aloitetaan aloitustunnisteella, esim. <html>, ja suljetaan lopetustunnisteeseen, esim. </html>. Kaikkien muiden elementtien täytyy olla juurielementin sisällä. [1, s. 12]

Elementti <head> sisältää metatietoja, tyylitietoja ja skriptejä. Elementti <head> on pakollinen ja sen täytyy loppua juuri ennen elementin <body> alkamista. Tärkein elementti HEAD-tunnisteen sisällä on otsikon määrittävä <title>. [1, s. 12]

Elementti <body> sisältää sivun kaiken sisällön. HTML-dokumentin toiseksi viimeinen tunniste täytyy olla </body>. [1, s. 12]

Kuva 2 esittelee työssä käytettyjä HTML-elementtejä. On syytä mainita, että työssä käytetty Angular-ohjelmistokehys ei tarvitse kaikkia normaalin HTML-dokumentin vaatimia elementtejä toimiakseen.



Kuva 2. Projektissa käytettyjä HTML-tunnisteita.

Tunnisteet <h1>, <h2>, <h3>, <h4>, <h5> ja <h6> ovat otsikoille tarkoitettuja tunnisteita. <h1> on sivun pääotsikko ja <h2> on pääotsikkoa alemman tason otsikko. <h6> on alimman tason otsikko.

Elementillä verkkosivun voi jakaa ja ryhmitellä sivun sisältöä lohkoihin. -elementin muodostama lohko alkaa aina uudelta riviltä [3, s. 187] ja lohkot kasaantuvat päällekkäin. -elementtiä käytetään usein CSS-koodin kanssa sivuston tyylin määrittelyyn, mutta ei itsessään tee mitään ulkoasuun liittyviä muutoksia. -elementillä

voi olla myös attribuuttina *id* tai *class*, joiden avulla CSS-koodi voidaan kohdistaa haluttuun lohkoon [3, s. 187]. [1, s. 12-14]

Tunniste <button> luo painikkeen, jolle voi määritellä *tapahtumia* (engl. *event*) [4] ja niiden aktivoimia metodeja. Esim. *<button onclick="painikePainettu()">Painike</button>* luo painikkeen tekstillä ”Painike” ja painiketta klikatessa suoritetaan metodi *painikePainettu*(). <button>-tunnisteeseen pystyy lisäämään CSS-koodia painikkeen ulkoasun muokkaamista varten. [5]

Elementti <input> on tarkoitettu sivuston toiminnan kontrollien toteuttamiseen. <input>-elementillä on attribuutti *type*, jonka avulla määritellään <input>-elementin tyyppi. Käsitellään esimerkkinä kaksi hyödyllistä <input>-tyyppiä: *checkbox* ja *range*. [3, s. 152]

Tyyppi *checkbox* luo valintaruudun ja käyttäjä pystyy joko valitsemaan ruudun tai poistamaan valinnan ruudusta. Valintaruutu muuttaa attribuutin *value* arvoa. Valittuna *value* on *true* (*tosi*) ja ei valittuna *false* (*epätosi*). [3, s. 156]

Tyyppi *range* luo liukuvalitsimen, jonka avulla käyttäjä pystyy muokkaamaan elementille asetettua arvoa liikuttelemalla valitsinta. Liukuvalitsimen muokkaama muuttuja määritellään attribuutilla *value*. Valitsimen minimi- ja maksimiarvot määritellään attribuuteilla *min* ja *max*. Lisäksi liukuvalitsimen *rakeisuutta* (engl. *granularity*) attribuutilla *step*. Vakiona rakeisuus on 1, ja haluttaessa arvo esim. kahden desimaalin tarkkuudella täytyy asettaa: *step="0.01"*. [6]

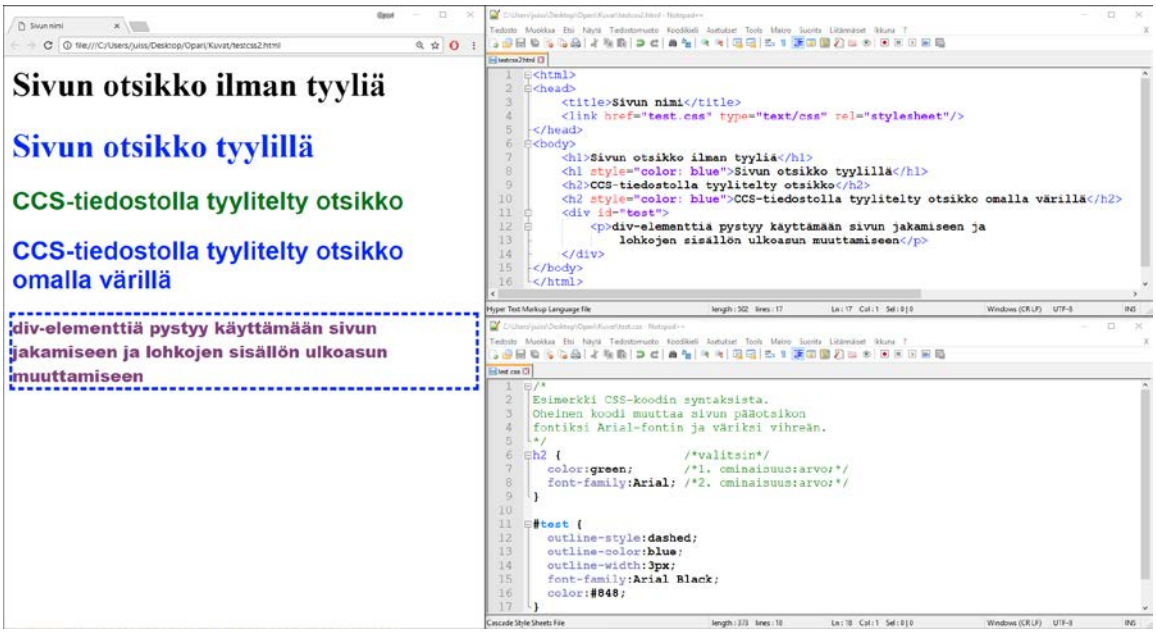
2.1.2 CSS

CSS (Cascading Style Sheets) on W3C:n ylläpitämä standardi esim. verkkosivujen ulkoasun ja esitystavan määrittelyyn [7]. CSS-kieli luotiin erottamaan ulkoasulliset tekijät HTML-dokumentin sisällöllisestä tekijöistä [8, s. 2].

Tyylin erottamista CCS-tiedostoon pidetään yleisesti hyvänä käytäntönä. Erillinen tiedosto tyylien hallintaan yksinkertaistaa HTML-dokumenttia sekä helpottaa tyylien ylläpitoa. CSS-tiedostoa käytettäessä tyyliä ei tarvitse kirjoittaa montaa kertaan samaan HTML:n elementtiin, koska tyylin määrittely kerran elementtiä kohden CSS-tiedostossa riittää. Lisäksi usea HTML-dokumentti pystyy käyttämään samaa CSS-tiedostoa. Tällöin vain yhden tiedoston muokkaaminen riittää koko sivuston tyylin muuttamiseen. Erillisen CSS-

tiedoston käyttäminen nopeuttaa myös verkkosivuston lataamista, koska CSS-tiedostoa ei tarvitse ladata kuin kerran ja se lyhentää HTML-dokumentteja. [3 s. 241]

Kuvassa 3 on yksinkertainen esimerkki CSS-koodista yhdessä HTML-koodin kanssa ja erikseen omana CSS-dokumenttina.



Kuva 3. Esimerkki CSS-koodin käyttämisestä. Oikealla ylhäällä HTML-koodi, oikealla alhaalla CSS-koodi ja vasemmalla kumpaakin tiedostoa käyttävä verkkosivu.

CSS-koodin syntaksi koostuu yksinkertaisimmillaan valitsimista sekä ominaisuuksista ja niiden arvoista: *valitsin {ominaisuus1:arvo; ominaisuus2:arvo;}*. Ominaisuudet (ja niiden arvot) erotetaan toisistaan puolipisteellä. CSS-koodissa välilyönneillä tai rivinvaihdoilla ei ole merkitystä. Valitsimen eteen tulee *risuaita* #, jos valitsin on viittaus HTML-dokumentin elementin attribuuttiin *id*. [8, s. 2]

Käytettäessä CSS-koodia ulkoisesti CSS-tiedosto linkitetään HTML-dokumenttiin elementillä <link>. Ilman linkitystä HTML-dokumentti ei pääse käsiksi CSS-tiedostoon. Elementillä <link> on kolme attribuuttia: *href*, *type* ja *rel*. Attribuutti *href* määrittelee tiedostopolun CSS-tiedostoon. Tiedostopoluksi riittää pelkkä tiedoston nimi, jos CSS-tiedosto on samassa hakemistossa kuin HTML-dokumentti. Attribuutilla *type* tarkennetaan käytettävä tiedostomuoto. CSS-tiedostoa käytettäessä *type*-attribuutin arvo on *text/css*. Attribuutti rel määrittää HTML-dokumentin ja siihen linkitettävän tiedoston *suhteen* (engl. *relation*). Linkittäessä CSS-tiedostoon *rel*-attribuutti on *stylesheet*. [3, s. 235]

CSS-koodin käyttäminen sisäisesti esim. HTML-dokumentissa onnistuu antamalla attribuutti *style* elementille, jonka kanssa CSS-koodia halutaan käyttää. Elementissä käytetty CSS-koodi päällekirjoittaa erillisestä CSS-tiedostosta haettujen tyylien päälle. [3, s. 236]

Työssä CSS-koodia käytetään yhdessä SVG:n kanssa, ei niinkään sivun muotoiluun. CSS-koodin käyttäminen SVG:n kanssa onnistuu pitkälti samalla tavalla kuin HTML:n kanssa. Myös SVG-elementtien tyylisäännöt voi kirjoittaa erilliseen CSS-tiedostoon, mutta työssä tyylit annetaan suoraan SVG-elementeille. SVG:n toimintaan ja tyylittelyyn perehdytään paremmin omassa luvussaan. [9, s. 39]

2.1.3 JavaScript

JavaScript (lyhennetään JS) on Netscapen kehittämä oliopohjainen ohjelmointikieli, joka on tarkoitettu dynaamisten toimintojen lisäämiseen verkkosivustoihin [10, s. 2].

JavaScript ei ole yhteydessä Java-ohjelmointikieleen, vaikka nimi siihen viittaakin. JavaScriptin alkuperäinen nimi oli LiveScript, mutta markkinointia varten kielen nimi muutettiin JavaScriptiksi [10, s. 2]. JavaScript tunnetaan myös nimellä ECMAScript, joka on ECMA:n (European Computer Manufacturer’s Association) luoma standardi JavaScriptistä. [10, s. 5]

JavaScript on skriptikieli. Skriptikieliä käytetään usein ohjelmistojen toiminnallisuuden laajentamiseen. JavaScript on muiden skriptikielien tapaan tulkattava kieli ja JavaScriptiä käyttävissä sovelluksissa (esim. verkkoselain) on JavaScript-tulkki. Tulkattava kieli suoritetaan rivi riviltä ajon aikana toisin kuin käännettävät kielet, kuten C++, jotka käännetään ennen suorittamista. Ajon aikainen tulkkaaminen tekee JavaScript-ohjelman muokkaamisesta helppoa ja nopeaa. Muokattu sisältö näkyy heti ladattaessa JavaScript-koodi uudestaan selaimeen. [10, s. 6]

JavaScript on olio-ohjelmointikieli ja JavaScriptin käyttäminen perustuu pitkälti kielen sisäisien olioiden käyttämiseen. Koko HTML-dokumentti käsitellään yhtenä oliona. JavaScriptissä pystyy myös käyttämään funktioita, jotka ovat funktio-olioita, joille voi antaa omia metodejaan. [10, s. 17] [10, s. 42]

JavaScriptin syntaksi on hyvin samankaltaista kuin esim. C-kielessä. Suurin osa JavaScriptin avainsanoista on periytynyt kielistä C ja Java. JavaScript-sovellus rakentuu lauseista, operaattoreista ja lausekkeista. Huomionarvoista on mainita, että C-kielestä

poiketen JavaScriptin lauseen ei tarvitse päättyä puolipisteeseen, jos samalla rivillä on vain yksi lause. Lauseet voi myös jakautua monelle riville. [10, s. 19–20]

JavaScriptissä muuttujat määritellään avainsanalla *var*. Muuttujilla ei ole JavaScript-kielessä tyyppiä ja samaan muuttujaan voi sijoittaa sekä numeroita että merkkijonoja ilman erillistä tyypin määrittelyä. Muuttujien tyypit täytyy kuitenkin tietää niitä käyttäessä tai voi joutua tekemisiin muuttujien tyyppeihin liittyvien virheiden kanssa. Avainsanan *var* käyttäminen ei ole pakollista, mutta sillä voidaan estää paikallisen ja globaalin muuttujan sekaantuminen. Paikallisen muuttujan pystyy määrittämään avainsanalla *let*. Lokaalia muuttujaa ei pysty käyttämään lohkonsa ulkopuolelta [11]. [10, s. 28]

JavaScript käyttää Javasta ja C:stä tuttuja ehtolauseita ja silmukoita: *if*, *switch*, *while* ja *for*. [10. s. 37]

2.2 XML

XML (Extensible Markup Language) on rakenteellisten merkintäkielien määrittelyyn käytetty metakieli. XML-kieltä käytetään luomaan merkintäkieliä, eikä ole olemassa varsinaista XML-dokumenttia. XML-dokumentista puhuttaessa tarkoitetaan XML:n avulla määritellyllä kielellä kirjoitettuja dokumentteja. [12]

XML-dokumentti rakentuu elementeistä, joita merkitään kulmasulkujen sisällä olevilla *tunnisteilla* (engl. *tag*). Elementit aloitetaan aloitustunnisteella, esim. <tag>, ja suljetaan lopetustunnisteeseen, esim. </tag>. Elementit voidaan myös avata ja sulkea saman tunnisteen sisällä, esim. <tag value=”1”/>. Ensimmäisellä tunnisteella merkitään juuritason elementtiä, joita on jokaisessa XML-dokumentissa vain yksi ja muut elementit ovat juurielementin sisällä. Elementtien sisältönä voi olla muita elementtejä. Elementillä voi olla attribuutteja, jotka antavat lisätietoa elementin sisällöstä. Attribuutit merkitään tunnisteen jälkeen kulmasulkujen sisään, esim. <tag vuosi=”2018”>. Attribuuttien arvot täytyy aina tulla *lainausmerkkien* ” ” sisään. [12]

Kuva 4 antaa esimerkin XML:llä tehdystä dokumentista.

```
<pelitietokanta>
    <peli>
        <nimi>Space Invaders</nimi>
        <alusta>Arcade</alusta>
        <julkaisuvuosi>1978</julkaisuvuosi>
    </peli>
    <peli>
        <nimi>Super Mario Bros. 3</nimi>
        <alusta>Nintendo Entertainment System</alusta>
        <julkaisuvuosi alue="JP">1988</julkaisuvuosi>
        <julkaisuvuosi alue="PAL">1991</julkaisuvuosi>
    </peli>
</pelitietokanta>
```

Kuva 4. Esimerkki XML-kielellä toteutetusta pelitietokantaa kuvaavasta dokumentista.

Esimerkissä juurielementti on <pelitietokanta>. Pelitietokannan sisällä on kaksi <peli>-elementtiä, joilla on kolme elementtiä: <nimi>, <alusta> ja <julkaisuvuosi>. Elementillä <julkaisuvuosi> voi olla attribuutti ”*alue*”, jolla merkataan, minkä alueen julkaisuvuodesta on kyse.

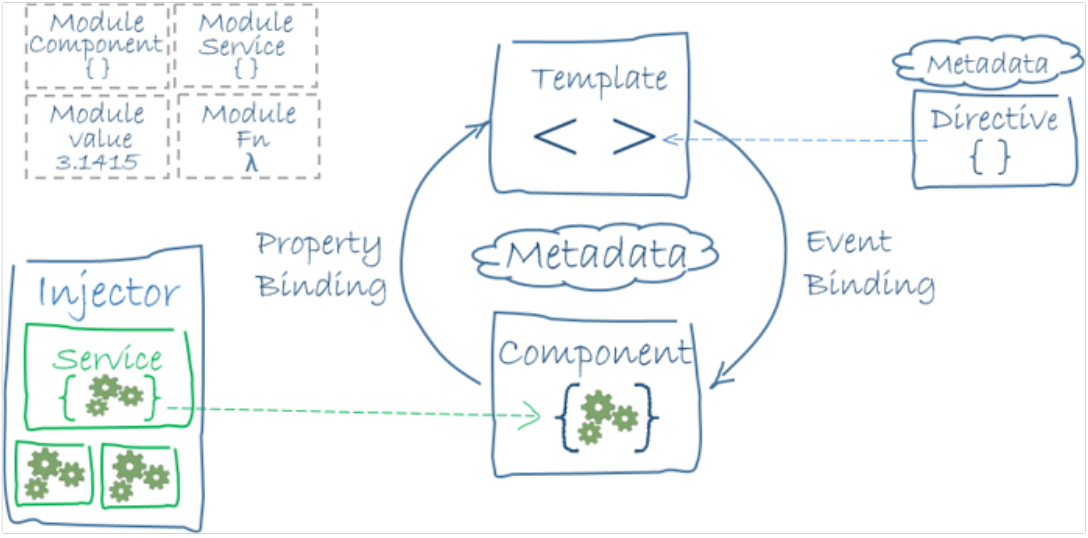
XML ja HTML vaikuttavat syntaksinsa puolesta samankaltaisilta. Tämä johtuu siitä, että HTML on merkintäkieli, kun taas XML:llä luodaan merkintäkieliä [12]. HTML pohjautuu SGML-metakieleen, joka on XML:n edeltäjä [12]. Tässä työssä käytettyjä XML pohjaisia merkintäkieliä ovat OSM XML ja SVG.

2.3 Angular

Angular (kutsutaan myös nimillä *Angular 2* tai *Angular 2+*) on Googlen ylläpitämä avoimen lähdekoodin ohjelmistokehys, joka pohjautuu TypeScript-ohjelmointikieleen. Se on tarkoitettu sovellusten kehittämiseen HTML:llä ja joko JavaScriptillä tai ohjelmointikielellä, joka kääntyy JavaScriptiksi, kuten TypeScript [13].

Angular on modulaarinen ohjelmistokehys. *Moduuli* (engl. *module*) sisältää ominaisuuksia, joita sovellus käyttää. Sovelluksen pohjana toimii juurimoduuli, joka on yleensä nimeltään *AppModule*. Sovelluksessa voi olla enemmän kuin yksi moduuli ja Angularin asennuksen mukana tulee Angularin *ydinmoduulin* (engl. *core*) lisäksi muutama muu moduuli. [13]

Angular-sovellus koostuu *komponenteista* (engl. *component*), *malleista* (engl. *template*) ja *palveluista* (engl. *service*). Komponentit ovat sovelluksen osia, joilla määritellään toiminnallisuutta. Mallit ovat HTML-dokumentteja, joilla määritellään komponentin rakenne. Palvelut antavat tietoa muilla sovelluksen osille. Kuva 5 havainnollistaa Angularin toiminnan pääpiirteittäin. [13]



Kuva 5. Angular-sovelluksen rakenne. [13]

2.3.1 TypeScript

TypeScript on Microsoftin kehittämä ja ylläpitämä, vahvasti tyypitetty käännettävä ohjelmointikieli. TypeScript on JavaScriptin *supersetti*, joka käännetään JavaScriptiksi. TypeScript tuo JavaScriptiin lisäominaisuuksia, kuten vahvan tyypittämisen. [14, s. 1]

TypeScript on pohjimmiltaan tavallista JavaScriptiä ja tästä johtuen TypeScriptin perussyntaksi on samanlaista kuin JavaScriptissä. TypeScript tukee JavaScript-pohjaisia kirjastoja ja ohjelmointikehyksiä, ja sitä pystyy ajamaan kaikissa JavaScriptiä tukevissa ympäristöissä. [14, s. 1]

TypeScript ei ole tulkattava kieli, kuten JavaScript, vaan se käännetään JavaScriptiksi ennen ajamista. Tämän vuoksi TypeScript-koodin testaaminen ei ole yhtä nopeaa kuin JavaScript-koodin testaaminen. Kääntäjä voi kuitenkin antaa koodia käännettäessä virheilmoituksia, jotka saattaisivat jäädä tavallisessa JavaScript-ohjelmassa huomaamatta [14, s. 2]. TypeScriptin vahva tyypittäminen auttaa myös huomaamaan tyyppivirheet helposti [14, s. 2].

2.3.2 Angular CLI

Angular CLI (Angular Command Line Interface) on komentorivi Angularille. Sitä käytetään Angular projektien hallintaan ja kehittämiseen tekstikomentoja käyttämällä. Komennot nopeuttavat ohjelman luomista, koska esim. uutta komponenttia luodessa kaikkea ei tarvitse kirjoittaa joka kerta uudelleen. [15]

Listaus hyödyllisistä ja tässä työssä käytetyistä komennoista:

- *ng new*: luo uuden projektiin, joka kääntyy ilman muokkauksia [15].
- *ng generate component [nimi]*: luo uuden komponentin ja lisää tarvittavat referenssit automaattisesti tiedostoon app.module.ts [15] [16].
- *ng serve*: kääntää projektin ja käynnistää serverin selaimelle [17].

2.3.3 Node.js

Node.js on avoimen lähdekoodin JavaScript runtime-ympäristö, jota käytetään JavaScript-koodin ajamiseen palvelimen puolella. Node.js mahdollistaa koodin suorittamisen palvelimella ennen kuin verkkosivu lähetetään käyttäjän selaimeen. [18]

2.3.4 npm

npm (node package manager) on Node.js:n käyttämä JavaScript-pakettien hallintatyökalu, sekä maailman suurin ohjelmistorekisteri. npm:ää käyttämällä voi käyttää toisten käyttäjien tekemiä paketteja ja jakaa itse kehittämiään paketteja. [19]

Pakettien asentaminen on erittäin helppoa käyttämällä komentoriviä:

- *npm install <paketin-nimi>*: asentaa paketin lokaalisti [20].
- *npm install -g <paketin-nimi>*: asentaa paketin globaalisti [21].

Pakettien asentaminen lokaalisti tai globaalisti riippuu paketin käyttötarkoituksesta. Käyttäjän halutessa käyttää pakettia komentorivityökaluna paketti täytyy asentaa globaalisti. Tällöin käyttäjä pääsee käsiksi mistä tahansa hakemistosta. Paketin ollessa osa vain yhtä

projektia kannattaa paketti asentaa lokaalisti, jolloin se asennetaan sen hetkiseen hakemistoon eikä siihen ole pääsyä muista hakemistoista. [21]

2.4 OpenStreetMap

OpenStreetMap on ilmainen ja vapaasti muokattava koko maailman kattava kartasto. Se koostuu vapaaehtoisten kartoittamasta tiedosta ja vapaan lisenssin lähteistä. [22]

OpenStreetMap-kartaston tietoa pystyy käyttämään omissa projekteissaan esim. viemällä halutun alueen karttatiedon OSM XML -tiedostomuodossa käyttämällä vientityökalua OpenStreetMapin kotisivulla. OSM XML on XML-pohjainen tiedostomuoto, joka sisältää OpenStreetMap-kartan käyttämän tiedon [23].

2.4.1 OpenStreetMap-lisenssi

OpenStreetMapista saatavan tiedon käyttäminen on ilmaista OpenStreetMap-lisenssin myötä, mutta käyttäessä kartan dataa täytyy mainita käytetyn tiedon kuuluvan OpenStreetMapille. OpenStreetMapin tekijänoikeuksien maininta tehdään liittämällä teksti “© *OpenStreetMap contributors*” karttaan tai ohjelmaan, jossa OpenStreetMapin tarjoamaa tietoa käytetään. [24]

2.4.2 OSM XML

OSM XML -tiedosto määrittelee kaiken tiedon kartalle rajatulta alueelta. OSM XML -tiedosto koostuu pääelementeistä <bounds>, <node>, <way> ja <relation>. <bounds>-elementtiä lukuun ottamatta muilla pääelementeillä voi olla elementti <tag> antamassa lisätietoa kyseisestä elementistä. [23]

OpenStreetMapin wikisivulta löytyy esimerkki OSM XML -tiedostosta (kuva 6). [23]

Kuva 6. Esimerkki OSM XML -tiedostosta. [23]

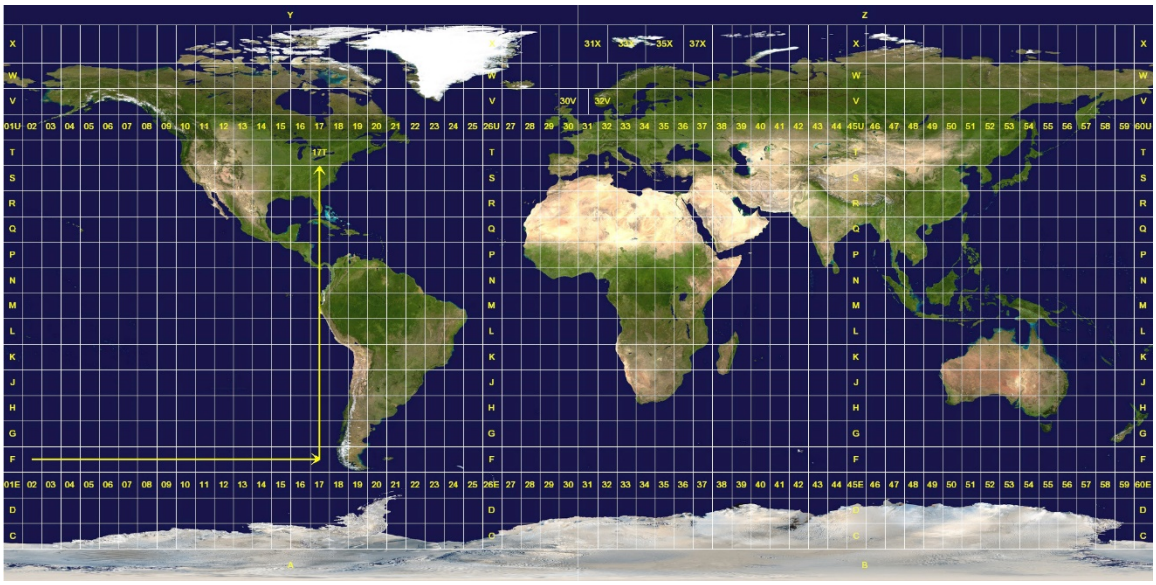
<bounds> määrittelee alueen, jolle kartan tieto sijoittuu. Alue määritellään attribuuteilla *minlat* (*minimi latitudi*), *minlon* (*minimi longitudi*), *maxlat* (*maksimi latitudi*) ja *maxlon* (*maksimi longitudi*). <node>-elementit ovat pisteitä, joilla on id-numero ja sijainti sekä tietoa pisteen lisääjästä, lisäysajasta yms. Pisteistä muodostetaan <way>-elementit, jotka pitävät sisällään kaikki muodot viivoista polygoneihin. <way>-elementillä on elementti <nd>, jonka attribuuttilla *ref* (*reference* = *viite*) viitataan <node>-elementin attribuuttiin *id*. <tag>-elementillä on attribuutit *k* (*key* = *avain*) ja *v* (*value* = *arvo*), joilla asetetaan elementille haluttu lisätieto tai -ominaisuus. [23]

<tag>-elementtien käsittely on työssä kriittisessä roolissa, koska niiden avulla saadaan tietää, mitkä <way>-elementeistä muodostavat tien. Jokaisella tiellä on <tag>-elementin avainattribuuttina *highway* ja arvoattribuutti vaihtelee tien tyypin mukaan [23].

OSM XML -tiedostoissa on yleensä todella paljon tietoa ja jopa pienen alueen kartan tiedoston pituus voi olla tuhansia rivejä. Esimerkkinä käytetyn Toivakan kartan pituus on lähellä 16 000 riviä alueen ollessa leveydeltään 1,96 km ja pituudeltaan 1,87 km. Muistia Toivakan esimerkkikartta vie noin 1 MB verran.

2.5 UTM-projektio

UTM-projektio (UTM = Universal Transverse Mercator) on karttaprojektio, joka pohjautuu Gauss-Krüger-projektioon. Projektiossa maapallo jaetaan UTM-*alueisiin* (engl. *zone*), joiden koordinaatit ilmoitetaan metreinä. Kuva 7 havainnollistaa UTM-alueiden jakautumisen maapallolle. [25]



Kuva 7. Maapallo jaettuna UTM-alueisiin. [26]

Leveyssuunnassa maapallo jaetaan kuuden asteen suikaleisiin, jolloin suikaleita on yhteensä 60 kappaletta. Suikaleet numeroidaan päivämäärärajasta alkaen 1–60. Pituussuunnassa maapallo jaetaan kahdeksan asteen suikaleisiin, jolloin suikaleita on yhteensä 20 kappaletta. Suikaleet merkitään kirjaimilla C-kirjaimesta X-kirjaimeen. Kirjaimia I ja O ei käytetä sekaannuksien välttämiseksi. Kirjaimet A, B, Z ja Y on varattu napa-alueille. [25]

UTM-alue muodostuu aluenumerosta ja aluekirjaimesta. UTM-koordinaatti saadaan, kun UTM-alueen perään lisätään itäkoordinaatti ja pohjoiskoordinaatti, joista kumpikin ilmoitetaan metreinä. Esimerkiksi Helsingin rautatieaseman sijainti UTM-koordinaatistossa on 41R 214247mE (*East* = *Itä*) ja 2761440mN (*North* = *Pohjoinen*). [25]

Työssä UTM-projektiota käytetään muuntamaan OSM XML -tiedoston käyttämät leveys- ja pituuspiirit metreiksi itä- ja pohjoiskoordinaattiin. Muunnoksen tekemiseen käytetään Node.js pakettia utm-latlng. Yksinkertaisuudessaan metodille annetaan koordinaatit muodossa (latitudi ja longitudi) ja metodi palauttaa objektin, jolla on koordinaatit muodossa (itäkoordinaatti, pohjoiskoordinaatti, aluenumero ja aluekirjain). [27]

2.6 SVG

SVG (Scalable Vector Graphics, suomeksi Skaalattava Vektori Grafiikka) on W3C:n kehittämä standardi kaksiulotteiselle vektorigrafiikalle. SVG koostuu kahdesta osasta: XML-pohjaisesta tiedostomuodosta ja graafisille sovelluksille tarkoitetusta ohjelmointirajapinnasta. SVG-kuvia pystyy luomaan ja muokkaamaan ohjelmointikielien avulla. [28] [9, s. 12]

SVG:tä käytetään monenlaisiin käyttötarkoituksiin, kuten selaimessa esitettävään grafiikkaan, käyttöliittymiin ja tulostuksiin. SVG on vahvasti teollisuuden tukema ja SVG:n määritelmää on ollut tekemässä mm. Adobe, Apple, Microsoft ja Nokia. [28]

SVG on skaalautuvuutensa takia erittäin suosittu kuvaformaatti verkkosivujen kehityksessä. Vektorigrafiikka esitetään abstraktien muotojen avulla ja suurennettaessa kuvaa tarkkuus ei huonone, kuten bittikarttagrafiikkaan pohjautuvissa kuvissa. SVG-kuva näkyy aina tarkkana laitteesta tai näytöstä riippumatta ja skaalautuu oikean kokoiseksi. Tällöin vältetään useiden kuvatiedostomuotojen kanssa esiintyvät ongelmat. [9, s. 12]

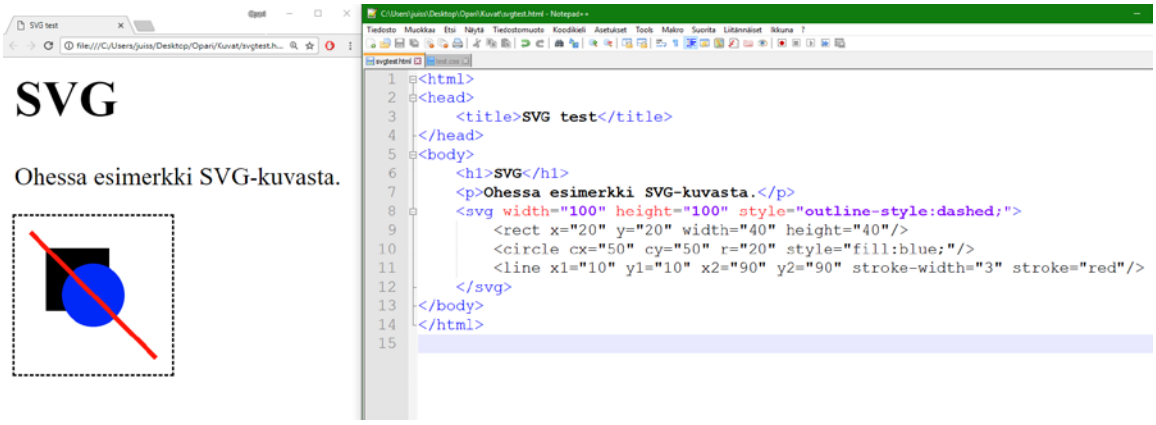
Skaalautumisen jälkeen SVG:n suurimpia etuja ovat tekstipohjaisuuden tuomat monipuoliset graafiset ominaisuudet sekä pieni tiedostokoko. SVG-kuva luodaan XML-pohjaisella SVG-merkintäkielellä. SVG-kuvia voi tuottaa pelkällä tekstieditorilla tai SVG-kuvia tukevalla ohjelmalla. SVG-kuvan tyylittelyyn voidaan käyttää CSS-kieltä ja SVG-kieli toimii suoraan HTML:ssä. SVG-kuvat ovat muistin puolesta suhteellisen pieniä tiedostoja, koska ne luodaan geometristen muotojen perusteella. SVG-kuvan koko ei vaikuta muistin viemiseen, vaan SVG-kuvatiedoston koko määräytyy yksityiskohtien perusteella. [9, s. 12]

Vektorigrafiikan haittapuolena on sen kovemmat laitevaatimukset verrattuna bittikarttagrafiikkaan. SVG-kuvan tulostaminen vaatii laitteelta paljon tehoja, jos kuvassa on paljon yksityiskohtia ja tehovaatimukset voivat karsia käyttökohteita. [9, s. 12]

2.6.1 SVG ohjelmoinnissa

SVG-kuva on XML-dokumentti, joten SVG-kuva koostuu elementeistä. SVG-kuvan juurielementti on <svg> ja juurielementille voi antaa mm. attribuutit *width*, *height* ja *viewBox*. <svg> -elementin sisälle tulee kaikki SVG-kuvan sisältö eli muodot ja tehosteet. [9, s. 17]

SVG-kuvan ei ole pakko olla aina oma tiedostonsa. SVG-koodia voi myös käyttää suoraan HTML-dokumentissa, ja koodilla luodun SVG-kuvan ulkomuotoa voi muokata käyttämällä CSS-koodia. Kuva 8 havainnollistaa yksinkertaisella esimerkillä, kuinka SVG toimii HTML:n ja CSS:n kanssa. [9, s. 38–39]



Kuva 8. Esimerkki SVG-koodista. Huomaa piirtojärjestyksen vaikutus SVG-kuvaan.

SVG-kuva alustetaan tunnisteella <svg>, ja esimerkissä kuvalle annetaan sekä leveydeksi että pituudeksi 100 (pikseliä). SVG käyttää oletusmittayksikkönä pikseleitä, mutta se tukee myös muita mittayksiköitä, kuten senttimetrejä. Kuvan koon lisäksi <svg>-elementille annetaan CCS-koodia attribuutilla *style*. <svg>-elementin sisällä on kolme muuta elementtiä: <rect>, <circle> ja <line>.

Elementillä <rect> luodaan *suorakulmio* (engl. *rectangle*). <rect> saa attribuuttina *aloituspisteen koordinaatit* (*x* ja *y*), sekä *leveyden* (*width*) ja *korkeuden* (*height*). SVG-kuvan nollapiste on vasen yläkulma ja suorakulmioiden piirtäminen alkaa myös vasemmasta yläkulmasta. [9, s. 77]

Elementillä <circle> luodaan *ympyrä* (engl. *circle*). <circle> saa attribuuttina *ympyrän keskipisteen koordinaatit* (*cx* ja *cy*), sekä ympyrän *säteen* (*r*). Lisäksi esimerkissä ympyrä *täytetään* (engl. *fill*) sinisellä, ja täyttämiseen käytetään CSS-koodia. [9, s. 79]

Elementillä <line> luodaan suora *viiva* (engl. *line*). <line> saa attribuuttina *aloituspisteen koordinaatit* (*x1* ja *y1*) ja *lopetuspisteen koordinaatit* (*x2* ja *y2*). Lisäksi esimerkissä viivalle määritellään *vedon leveys* ja *vedon väri* käyttämällä CSS-kielen ominaisuuksia *stroke-width* ja *stroke*. [9, s. 81]

2.6.2 SVG ja Angular

SVG-kieltä pystyy käyttämään myös Angularin kanssa, mutta tällöin on otettava muutama asia huomioon. Yksinkertaisia kuvioita piirrettäessä SVG-koodin voi kirjoittaa tavallisesti. Voi kuitenkin tulla tilanteita, jolloin Angularin täytyy saada lisää tietoa Angularin mallin elementeistä.

Käytettäessä SVG-koodia Angularin mallissa on hyvä käytäntö käyttää SVG:n elementtien edessä etuliitettä *svg:*, esim. <svg:rect>. Etuliitettä käyttämällä Angular tietää, että sen täytyy luoda SVG-elementtejä eikä HTML-elementtejä. Angular pystyy yleensä päättelemään oikean elementin käyttämisen, jos kaikki SVG-elementit ovat juurielementin sisällä. Koodin jakaantuessa moneen komponenttiin SVG-elementtien hahmottaminen tulee Angularille vaikeammaksi, jolloin etuliitettä *svg:* käyttäminen on pakollista. [29]

Luodessa dynaamista SVG-grafiikkaa elementtien attribuutit täytyy sitoa sovelluksen muuttujiin. Normaalisti dynaamiset sitomiset tapahtuu ”*sitomalla omaisuus*” (engl. *property binding*) eli asettamalla attribuutin nimi *hakasulkujen* [] sisään, esim <svg:circle [cx]=”20”/>. SVG-koodin kanssa tämä ei toimikaan, vaan ohjelma antaa virheilmoituksen: ”*Ei pystytä sitomaan ’cx’:ään, koska se ei ’svg:circle’:n tunnettu omaisuus.*” Ongelma johtuu siitä, että *cx* on attribuutti eikä omaisuus. Ratkaisu tähän ongelmaan on attribuutin sitominen, omaisuuden sitomisen sijaan. Lisäämällä etuliite *attr* sidottavan attribuutin eteen Angular osaa käyttää attribuutin sitomista. Aiempi esimerkki muuttuu muotoon: <svg:circle [attr.cx]=”20”/>. [29]

3 Ennalta tehty työ

Työ lähdettiin toteuttamaan ennalta tehdylle Angular-projektin pohjalle. Projektiin oli toteutettu valmiiksi luokat eri kartan elementeille sekä tietorakenne karttatiedon säilömiseen ja hallintaan. Tässä luvussa kuvataan projektin rakenne ja kuinka tietorakenne rakentuu.

Projektin rakenne on hyvin yksinkertainen. Projektissa on vain yksi moduuli ja sovelluksen pääkomponentti AppComponent pitää sisällään vain yhden komponentin: KarttaEditori.

```
<div> <kartta-editori></kartta-editori> </div>
```

Koodi 1. AppComponentin mali.

KarttaEditori on projektin näkyvin komponentti. KarttaEditorin kautta hoidetaan sekä kartan piirtäminen ruudulle että editointiominaisuuksien hallinta. KarttaEditori käyttää karttatietojen lukemiseen, käsittelyyn ja hallintaan lukuisia palveluita. KarttaEditori käsittelee palveluilta saadun tiedon piirrettävään muotoon, jonka jälkeen tiet piirretään ruudulle. Teiden piirtäminen käsitellään luvussa 4.

Sovelluksen käynnistyessä KarttaEditori alustaa ensin karttatietoa sisältävät palvelut, joita ovat AluePalvelu, PistePalvelu ja TiePalvelu. Vasta näiden alustuksien jälkeen voidaan lukea TiedostoPalvelulle annettu tiedosto.

TiedostoPalvelu lataa sille annetun tiedoston ja välittää sen käsiteltäväksi XMLKasittelijaPalvelulle. XMLKasittelijaPalvelu hoitaa tiedoston parsimisen ja jakaa tiedot muille palveluille jatkokäsittelyyn. XMLKasittelijaPalvelu käsitellään myöhemmin omassa luvussaan.

AluePalvelu omistaa tiedon kartan rajoista. Sillä on attribuutteina x- ja y-koordinaattien minimi- ja maksimiarvot. AluePalvelulta voi hakea alueen leveyden ja korkeuden niille tarkoitetuilla metodeilla.

PistePalvelu omistaa jokaisen pisteen kartan rajojen sisältä. Pisteet ovat Piste-objekteja, joilla on yksilökohtainen id-numero sekä x- ja y-koordinaatit. Pisteet säilötään Map-rakenteeseen id-numeron perusteella. PistePalvelulta löytyy metodit pisteiden hakemiseen, lisäämiseen ja poistamiseen.

TiePalvelu omistaa jokaisen tien, jonka loppu- ja alkupiste on kartan rajojen sisällä. Tiet ovat Tie-objekteja, joilla on alku- ja loppupisteiden id-numerot, suunta, leveys, tyyppi ja valinta. TiePalvelulla on metodit yhden tai usean tien tietojen saamiseen. Lisäksi siltä löytyy metodit teiden lisäämiseen ja poistamiseen.

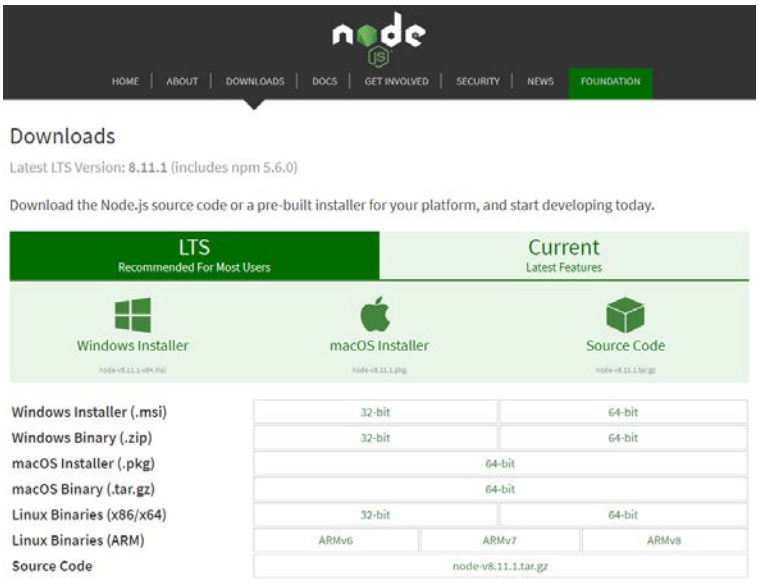
4 Työn toteuttaminen

4.1 Projektin luominen

Projektin tekeminen alkaa aina uuden projektin luomisesta. Ennen kuin päästään luomaan uusi Angular-projekti, täytyy asentaa Node.js ja npm. Node.js asennetaan perinteisesti asennustiedostolla, joka ladataan Node.js:n kotisivulta [30]. npm asennetaan Node.js:n asennuksen yhteydessä, mutta npm:n päivittäminen saattaa olla tarpeellista [31]. Angular (sekä Angular CLI) asennetaan npm:n kautta komentoriviä käyttämällä. Tarvittavien asennuksien jälkeen luodaan uusi Angular-projekti.

4.1.1 Node.js:n asentaminen

Node.js asennetaan kotisivuilta ladattavaa asennustiedostoa käyttämällä. Node.js:n kotisivuilta löytyy sivu ”Downloads” (kuva 9), jolta löytyy kaikki tämän hetkiset asennustiedostot Windowsille ja macOS:lle. Asennettava versio, 32-bittinen tai 64-bittinen, valitaan käyttöjärjestelmän mukaan. Sivulta löytyy myös binaarit Linuxille sekä lähdekoodi. [30]



Kuva 9. Downloads-sivulta löytyvät Node.js-asennustiedostot. [30]

Halutun asennustiedoston lataamisen jälkeen Node.js asennetaan seuraamalla asennuksen ohjeita. Onnistuneen asentamisen jälkeen avataan komentorivi ja siirrytään seuraavaan vaiheeseen.

4.1.2 npm:n asentaminen ja päivittäminen

npm asentuu tietokoneelle Node.js:n asennuksen mukana. Node.js ja npm ovat kuitenkin erillisiä projekteja, joista npm päivittyy huomattavasti useammin. Tästä johtuen npm on syytä päivittää komentoriviä käyttämällä. [31]

npm:n viimeisimmän version asentaminen tai siihen päivittäminen globaalisti tehdään komennolla: [31]

npm install npm@latest -g.

4.1.3 Angularin asentaminen ja projektin luominen

Angular asennetaan npm:n avulla asentamalla Angular CLI globaalisti komennolla: [16]

npm install -g @angular/cli

Angular CLI -komennot alkavat kirjainlyhenteellä *ng*. Uusi Angular projekti nimeltä *Testi* luodaan tämänhetkiseen hakemistoon komennolla: [16]

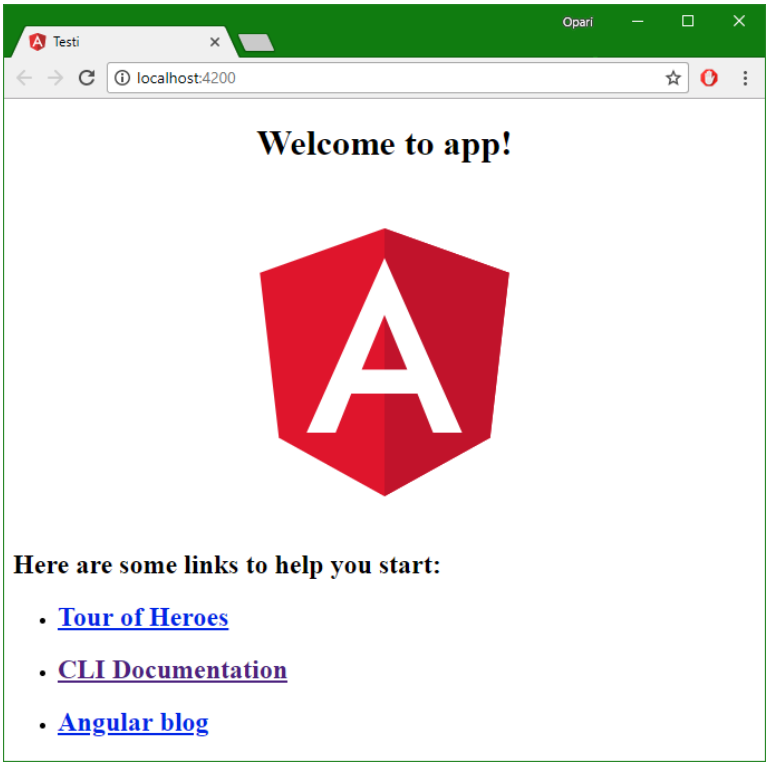
ng new Testi

Uusi projekti on nyt luotu. Projektin hakemistoon siirrytään komennolla *cd*. Projekti ajetaan kehitysserverillä komennolla *serve.* Ajamisen yhteydessä selain saadaan avattua serverin osoitteessa lisäkomennolla *--open*: [16]

cd Testi

ng serve --open

Projektille luodaan kehitysserveri, joka kääntää ohjelman automaattisesti muutoksien jälkeen. Kuvan 10 mukaisen projektin pitäisi nyt näkyä avatussa selaimessa.



Kuva 10. Uusi Angular-projekti.

Uuden Angular-projektin luominen kestää jonkin aikaa ja projektin kääntäminen kestää ensimmäisellä kerralla normaalia pidempään. Jatkossa muutokset päivittyvät kehitysserverille huomattavasti nopeammin ja ohjelman testaaminen on todella helppoa ja nopeaa.

4.2 OpenStreetMap-tiedoston lukeminen

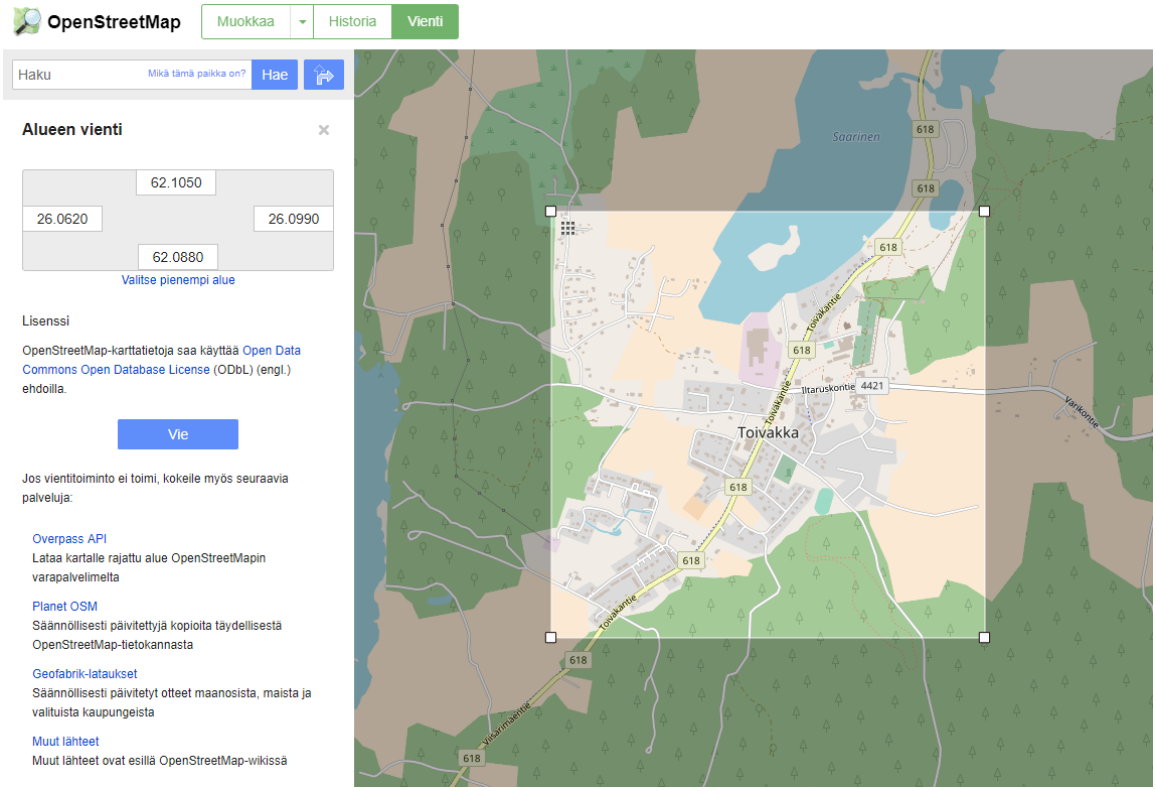
Projektin luomisen jälkeen toteutetaan työkalut OSM XML -tiedoston lukemiseen ja luetun tiedon piirtämiseen. Aluksi OSM XML -tiedosto täytyy viedä OpenStreetMapin kotisivulta. Viemisen jälkeen tiedosto parsitaan ohjelmalle käsiteltävään muotoon. Parsimisen yhteydessä valitaan halutut OSM XML -tunnisteet ja muunnetaan kaikki koordinaatit asteista metreiksi UTM-projektiota käyttämällä. Lopuksi parsitut tiet piirretään SVG-koodilla.

4.2.1 OSM XML -tiedoston vieminen

OpenStreetMapin karttatiedon vienti tehdään kotisivun vientityökalulla. Haluttu alue valitaan rajaamalla alue kartasta tai kirjoittamalla haluttujen rajojen pituus- ja leveyspiirit niille

varattuihin laatikkoihin. Pienet tiedostot on mahdollista viedä OSM XML -tiedostomuodossa suoraan painamalla painiketta *Vie* (engl. *Export*) tai painamalla Enter-näppäintä. Oletuksena tiedosto latautuu nimellä *map.osm* ja tiedostopääte *osm* viittaa OpenStreetMapin käyttämään OSM XML -tiedostoformaattiin. Isommat alueet täytyy viedä valitsemalla Overpass API, joka lataa kartalle rajatun alueen OpenStreetMapin varapalvelimelta. Overpass API:a käyttäessä ladattu tiedosto on nimeltään *map* ilman mitään tiedostopäätteitä, mutta tieto on silti OSM XML -formaatissa. [32]

Kuva 11 on kuvakaappaus OpenStreetMapin vientityökalusta. Kuvassa on rajattu vientiä varten esimerkkikarttana käytetty alue Toivakasta.



Kuva 11. OpenStreetMap-sivuston karttatiedoston vientityökalu. Kuvassa näkyy työssä käytetyn esimerkkikartan rajat. [32]

4.2.2 OSM XML -tiedoston parsiminen

OSM XML -tiedosto täytyy parsia ohjelmalle käsiteltävään muotoon. Parsiminen tehdään XMLKasittelijaPalvelussa, joka saa TiedostoPalvelulta parsittavan tiedoston. Parsimiseen käytetään node-xml2js-työkalua, joka on helppokäyttöinen työkalu XML-tiedoston kääntämiseen JavaScript-objektiksi [33]. On huomionarvoista mainita, että node-xml2js on

kohtalaisen raskas työkalu eikä se sovi todella isojen XML-tiedostojen parsimiseen. Työkalu on kuitenkin todella helppokäyttöinen ja ohjelman tarvitsee parsia tiedosto vain käynnistyessään, jolloin mahdollinen pitkä latausaika esiintyy vain kerran.

node-xml2js:n ja kaikki sen tarvitsemat paketit saadaan asennettua helposti käyttämällä npm:n komentoa: [33]

npm install xml2js

OSM XML -tiedoston parsiminen työkalua käyttämällä on todella helppoa. Ensin luodaan xml2js-parseri *parsitaanString*, jolle annetaan *attribuuttiavain* (*attrkey*) *attributes* ja *merkkiavain* (*charkey*) *text*.

```
parsitaanString = new xml2js.Parser({
  attrkey: 'attributes', charkey: 'text' }).parseString;
```

Koodi 2. xml2js-parserin luominen.

Parserin luomisen jälkeen tiedoston parsiminen tapahtuu metodissa *tiedostonKasittely()*, jolle annetaan parametrinä käsiteltävä tiedosto *xmlTiedosto*. Parserille *parsitaanString* annetaan parametreinä *xmlTiedosto*, sekä virheenkäsittely *err* ja tulos *result*. Kaikki aaltosulkujen sisällä oleva koodi liittyy parsitun tiedoston käsittelyyn ja varsinainen parsinta (ohjelman raskain vaihe) on jo tehty.

```
tiedostonKasittely(xmlTiedosto: string): void {
  this.parsitaanString(xmlTiedosto, (err, result) => {
    //Parsitun tiedoston käsittely
  })
}
```

Koodi 3. Metodi *tiedostonKasittely*.

Parsinnan tulos *result* on JavaScript-objekti, joka omistaa kaikki parsitun XML-tiedoston elementit. Parseri palauttaa virheen *err*, jos parsinta epäonnistuu. Ennen OSM XML -tiedoston käsittelyä täytyy tarkistaa, onko annettu tiedosto OSM XML -formaatissa. Parsitun tiedoston elementtejä kysellään ja tarkastellaan if-lauseita käyttämällä. Parsitulle OSM XML -tiedostolle luodaan muuttuja *osm*, jolle asetetaan parsimisen tulos *result.osm*, jos parsitun tiedosto on OSM XML -tiedosto. Muussa tapauksessa *osm* jää arvoon *null*, jolloin tiedetään parsittavan XML-tiedoston olevan jokin muu kuin OSM XML -formaatin tiedosto.

```
this.parsitaanString(xmlTiedosto, (err, result) => {
  //Parsitun tiedoston käsittely
  let osm = null;
  if (err) { return; } //Virheen käsittely
  if (result.osm) { osm = result.osm; }
  if (osm) { /*OSM XML -tiedoston käsittely*/ }
})
```

Koodi 4. Parsitun tiedoston alkutarkistukset.

Tiedoston tarkistuksien jälkeen OSM XML -tiedosto voidaan käsitellä. Käsittely tapahtuu lauseen *if (osm) {/*käsittely*/}* sisällä, eli käsittely tehdään vain, jos muuttujalla *osm* on jotain sisältöä.

4.2.3 Parsitun tiedon käsittely

OSM XML -tiedoston elementtien käsittely tehdään käyttämällä if-lauseita ja for-silmukoita. Parsitun tiedon käsittely jakautuu kolmeen isompaan kokonaisuuteen, joissa käsitellään OSM XML -tiedoston tärkeimmät elementit <bounds>, <node> ja <way>. Jokaisen elementin käsittely alkaa tarkistamalla if-lauseella, onko muuttujalla *osm* kyseistä elementtiä attribuuttina. Tarkistuksen jälkeen luodaan muuttuja, jolle annetaan kyseinen elementti. Muuttuja saa taulukon elementeistä, jos samaa elementtiä on tiedostossa monta kappaletta, kuten elementtejä <node> ja <way>. Yleensä luotu muuttuja annetaan jollekin palvelulle käsiteltäväksi, mutta tarvittaessa muuttujan elementit voidaan käydä läpi for-silmukalla jatkokäsittelyä varten.

Elementeille <bounds> ja <node> luodut muuttujat annetaan suoraan AluePalvelun ja PistePalvelun metodeille, joilla haluttu tieto asetetaan tietokantaan. Metodien toiminta käsitellään omissa luvuissaan.

```
if (osm.bounds) {
  let bounds = osm.bounds;
  this.aluePalvelu.parsitaanRajatOSM(bounds);
}
if (osm.node) {
  let nodes = osm.node;
  this.pistePalvelu.parsitaanPisteetOSM(nodes);
}
```

Koodi 5. Elementtien <bounds> ja <node> käsittely.

Elementti <way> koostuu kahdesta elementistä, <nd> ja <tag>, joita tarvitaan teiden luomiseen. Tämän takia <way>-elementit täytyy käydä läpi kahdella for-silmukalla ennen elementin antamista TiePalvelulle.

Ensimmäisellä silmukalla käydään läpi jokainen <way>-elementti. Ensimmäisen silmukan jälkeen otetaan talteen taulukko elementeistä <nd>, jotka ovat referenssejä <node>-elementteihin. Referenssiä käytetään paikantamaan pisteet, joista tie koostuu. Tiellä ei siis itsellään ole tietoa sijainnista, mutta se voidaan selvittää alku- tai loppupisteen id-referenssin avulla. <nd>-elementit annetaan muuttujalle *nds*, joka annetaan myöhemmin parametrinä tiet luovalle metodille.

Toisella silmukalla käydään läpi <way>-elementin tunnisteet <tag>, joiden avulla karsitaan halutut tietyypit. Tietyyppien karsinta käsitellään omassa luvussaan.

```
if (osm.way) {
  let ways = osm.way;
  for (let way of ways) {
    if (way.nd) {
      let nds = way.nd;
      if (way.tag) {
        let tags = way.tag;
        for (let tag of tags) { /*Tietyyppien tarkistaminen*/ }
      }
    }
  }
}
```

Koodi 6. Elementin <way> käsittely.

4.2.4 Rajojen lisääminen tietokantaan

Rajojen koordinaattien lisääminen tietokantaan tehdään AluePalvelun metodilla *parsitaanRajatOSM()*. Metodille annetaan parametrinä OSM XML -tiedoston elementti <bounds>. Elementiltä <bounds> parsitaan rajojen minimi- ja maksimiarvojen koordinaatit pituus- ja leveysasteina.

Ennen kuin rajojen koordinaatteja pystytään käyttämään, ne täytyy muuttaa metreiksi käyttämällä UTM-projektiota. Muunnos toteutetaan työkalulla utm-latlng, joka alustetaan seuraavasti:

```
var utmObj = require('utm-latlng');
var utm = new utmObj();
```

Koodi 7. utm-latlng-työkalun alustaminen [27].

Luodulla objektilla *utm* löytyy metodit koordinaattien UTM-muunnoksiin metrien ja asteiden välillä. Työssä käytetään vain metodia *convertLatLngToUtm()*, jolla asteet muunnetaan metreiksi. Metodille annetaan muunnettavat asteet sekä muunnoksen tarkkuus desimaaleina.

Alustuksen jälkeen luodaan muuttujat *minimiRajat* ja *maksimiRajat*. Muuttujat saavat työkalulla metreiksi muutetut koordinaatit attribuutteina *Easting* ja *Northing*. Nämä attribuutit asetetaan AluePalvelun muuttujiin *minimiX*, *maksimiX*, *minimiY* ja *maksimiY*.

```
var minimiRajat = utm.convertLatLngToUtm(this.minlat, this.minlon, 10);
this.minimiX = minimiRajat.Easting;
this.minimiY = minimiRajat.Northing;

var maksimiRajat = utm.convertLatLngToUtm(this.maxlat, this.maxlon, 10);
this.maksimiX = maksimiRajat.Easting;
this.maksimiY = maksimiRajat.Northing;
```

Koodi 8. Rajojen koordinaattien muuttaminen asteista metreiksi.

Alueen rajojen oikeat koordinaatit ovat hyvin tärkeässä roolissa työn kannalta. Monet metodit käyttävät jollain tapaa tietoa alueen rajojen koordinaateista tai alueen koosta.

4.2.5 Pisteiden lisääminen tietokantaan

Pisteet lisätään tietokantaan PistePalvelun metodilla *parsitaanPisteetOSM()*. Metodille annetaan parametrinä muuttuja *nodes*, joka on taulukko <node>-elementeistä. Jokainen <node>-elementti käydään läpi for-silmukalla ja elementeiltä parsitaan pisteen id-numero ja pisteen koordinaatit pituus- ja leveysasteina.

Pisteiden koordinaateille tehdään sama koordinaattimuunnos kuin alueen rajoille. utm-latlng-työkalun alustuksen jälkeen luodaan muuttuja *muunnettuPiste* koordinaattimuunnosta varten. Metreiksi muunnetut koordinaatit asetetaan muuttujille *x* ja *y*.

```
var utmObj = require('utm-latlng');
var utm = new utmObj();

let muunnettuPiste = utm.convertLatLngToUtm(lat, lon, 10)
let x: number = muunnettuPiste.Easting;
let y: number = muunnettuPiste.Northing;
```

Koodi 9. Pisteen x- ja y-koordinaattien muuttaminen asteista metreiksi.

Koordinaattien muuntamisen jälkeen tarkistetaan, mitkä pisteet ovat AluePalvelulle määriteltyjen rajojen sisällä. OSM XML -tiedostosta löytyy esim. teille kuuluvia pisteitä, jotka menevät yli määriteltyjen rajojen. Erityisesti piirtovaiheessa rajojen ulkopuolella olevat pisteet voivat aiheuttaa erikoisia virheitä, joten ylimääräiset pisteet on karsittava pois. Karsinta tehdään tiedustelemalla rajat AluePalvelulta ja vertaamalla rajoja pisteen sijaintiin.

```
if (this.aluePalvelu.minimiX < x && x < this.aluePalvelu.maksimiX &&
    this.aluePalvelu.minimiY < y && y < this.aluePalvelu.maksimiY) {
  this.lisaaPiste(x, y, id);
}
```

Koodi 10. Piste lisätään tietokantaan vain sen ollessa kartan rajojen sisällä.

Lopuksi piste lisätään tietokantaan PistePalvelun metodilla *lisaaPiste()*, jolle annetaan parametreinä lisättävän pisteen koordinaatit x ja y, ja pisteen id-numero. Pisteet säilötään Map-objektiin *pisteet* id-numeron perusteella.

```
pisteet: Map<number, Point> = new Map();
```

Koodi 11. Map-rakenteen *pisteet* luominen.

Map-objektin etu taulukkoon verrattuna on sen aukottomuus. Aukottomuus on tärkeää valitessa ja poistettaessa pisteitä. Taulukkoon tulee helposti aukkoja, jos pisteiden id-numerot eivät ole järjestyksessä olevia juoksevia numeroita. OSM XML -tiedoston dataa käsiteltäessä tämä voi johtaa miljooniin tyhjiin kohtiin taulukossa, koska pisteiden id-numerot ovat usein todella suuria. Haluttujen pisteiden etsiminen taulukosta kestäisi tällaisissa tapauksissa tunteja, joten tarvitaan Mapin kaltainen aukoton tapa säilöä pisteet.

4.2.6 Haluttujen tietyyppien karsinta

Elementti <way> voi muodostaa minkä tahansa kartan muodon, jonka tyyppi määritellään tunniste-elementissä <tag>. Tiet merkitään <tag>-elementin avainattribuutilla *highway*. *highway* kattaa kaikki kuljettavat reitit moottoriteistä metsäpolkuihin. Elementillä <tag> on

avainattribuutin lisäksi arvoattribuutti, jolla määritetään tien tyyppi. Työssä käytetään OpenStreetMapin tärkeimmäksi määrittelemiä tietyyppejä [34], joista Suomessa esiintyy:

- motorway: moottoritie (ei esiinny Toivakassa).
- secondary: kaupunkien välinen tie.
- tertiary: pienien kaupunkien ja kylien välinen tie.
- unclassified: vähiten tärkeä paikkojen välinen tie valtion tieverkostossa.
- residential: asutusalueille vievä tie.
- service: palvelutiet, esim. kauppojen pihat ja parkkipaikat.

Tietyyppien karsinta aloitetaan asettamalla kaikki tiedoston <tag>-elementit muuttujaan *tags*. Tämän jälkeen jokainen elementti käydään läpi for-silmukalla, jonka sisällä etsitään elementit, joilta löytyy avainattribuutti *highway*. Näin saadaan tietoon jokainen tien muodostava <way>-elementti.

```
if (way.tag) {
  //Teiden etsiminen
  let tags = way.tag;
  for (let tag of tags) {
    if (tag.attributes.k == "highway") {/*Tietyyppien tarkistaminen*/}
  }
}
```

Koodi 12. Elementtien <tag> käsittely.

Seuraavaksi etsitään halutut tietyypit <tag>-elementin arvoattribuutin avulla. Jos arvoattribuutti täsmää halutun tietyypin kanssa, lisätään kyseinen tie tietokantaan käyttämällä tiePalvelun metodia *parsitaanTietOSM()*. Metodi saa parametreinä aiemmin talteen otetut pisteet *nds* sekä tietyypin *tag.attributes.v*.

```
if (tag.attributes.k == "highway") {
  //Tietyyppien tarkistaminen
  if (tag.attributes.v == "motorway") {
    this.tiePalvelu.parsitaanTietOSM(nds, tag.attributes.v);
  }
  if (tag.attributes.v == "secondary") {
    this.tiePalvelu.parsitaanTietOSM(nds, tag.attributes.v);
  }
  if (tag.attributes.v == "tertiary") {
    this.tiePalvelu.parsitaanTietOSM(nds, tag.attributes.v);
  }
  if (tag.attributes.v == "residential") {
    this.tiePalvelu.parsitaanTietOSM(nds, tag.attributes.v);
  }
  if (tag.attributes.v == "unclassified") {
    this.tiePalvelu.parsitaanTietOSM(nds, tag.attributes.v);
  }
  if (tag.attributes.v == "service") {
    this.tiePalvelu.parsitaanTietOSM(nds, tag.attributes.v);
  }
}
```

Koodi 13. Tietyyppien karsinta vertailemalla arvoattribuutteja.

Haluttujen tietyyppien valitsemisen jälkeen tiet voidaan lisätä tietokantaan.

4.2.7 Teiden lisääminen tietokantaan

Tiet lisätään tietokantaan TiePalvelun metodilla *parsitaanTietOSM()*. Metodille annetaan parametreinä <nd>-elementit sisältävä taulukko *nds* ja tien tyyppi *tag*. Tiet koostuvat viidestä attribuutista: *alkuPiste*, *loppuPiste*, *suunta*, *leveys* ja *tyyppi*.

Alku- ja loppupiste ovat referenssejä PistePalvelun pisteisiin. Tämän takia ennen tien lisäämistä tietokantaan tarkistetaan, onko kumpikin tien pisteistä tietokannassa. Tien pisteet pystyvät sijaitsemaan myös alueen rajojen ulkopuolella, jolloin niiden pisteitä ei ole lisätty tietokantaan. Tarkistusta varten luodaan taulukko *ndrefs*. <nd>-elementit sisältävä taulukko *nds* käydään läpi for-silmukalla, jossa jokaisella pistereferenssillä etsitään piste PistePalvelulta. Pisteen löytyessä piste parsitaan numeroksi ja annetaan muuttujalle *ndTemp*. Lopuksi *ndTemp* lisätään taulukkoon *nds*.

```
parsitaanTietOSM(nds, tag) {
  let ndrefs: number[] = new Array();
  for (let nd of nds) {
    if (this.pistePalvelu.saaPiste(nd.attributes.ref)) {
      let ndtemp = Number.parseInt(nd.attributes.ref);
      ndrefs.push(ndtemp);
    }
  }
  //Tien lisääminen
}
```

Koodi 14. Tarkistetaan, löytyykö tien alku- tai loppupistettä tietokannasta.

Ylimenevien pistereferenssien karsimisen jälkeen tiet lisätään tietokantaan. Tiet koostuvat monesta pienestä pätkästä. Kokonaiset tiet luodaan for-silmukan avulla. Taulukko *ndrefs* käydään läpi toiseksi viimeiseen pisteeseen asti. Tien pätkän alkupiste on sen hetkinen piste ja loppupiste on taulukon seuraava piste. Toiseksi viimeisen pisteen kohdalla loppupiste on taulukon viimeinen piste, joten koko tie on saatu luotua. Pisteiden lisäksi tielle annetaan suunta, leveys ja tyyppi. Suunta saa vakioarvona arvon 0, jolla merkitään työssä kaksisuuntaisia teitä. Leveys saa vakioarvona arvon 3. Tyyppi on parametrinä saatu *tag*. Lopuksi luodaan *Tie*-tyyppinen muuttuja *uusiTie* parsituilla arvoilla ja lisätään *uusiTie* tietokantaan metodilla *lisaaTie()*.

```
//Tien lisääminen
for (let i = 0; i < ndrefs.length - 1; i++) {
  let alkuPiste: number = ndrefs[i];
  let loppuPiste: number = ndrefs[i + 1];
  let suunta = 0;
  let leveys = 3;
  let tyyppi = tag;
  let uusiTie = new Tie(alkuPiste, loppuPiste, suunta, leveys, tyyppi);
  this.lisaaTie(uusiTie);
}
```

Koodi 15. Lisätään tie tietokantaan.

Teiden lisäämisen myötä OSM XML -tiedoston käsittely on saatu tehtyä. Seuraavaksi piirretään tiet saadun tiedon perusteella.

4.3 Teiden piirtäminen

Tiet piirretään KarttaEditori-komponentin mallissa ja piirtäminen tehdään SVG-koodilla. Piirrossa täytyy ottaa huomioon eroavaisuudet karttatiedon ja SVG-kuvan käyttämissä

koordinaatistoissa. Työn käyttämässä karttatiedossa pienimmät koordinaatit ovat vasemmassa alakulmassa ja SVG-kuvan nollakohta eli origo on vasemmassa yläkulmassa. On myös syytä muistaa, että kartan pienin koordinaatti ei yleensä ole nolla. Näistä seikoista johtuen karttatietoa täytyy muokata ennen piirtämistä.

4.3.1 Koordinaatiston muutokset

Karttatieto, tässä tapauksessa pisteiden koordinaatit, muokataan KarttaEditorin metodeissa *saaPisteX()* ja *saaPisteY()*. Metodit etsivät PistePalvelusta pisteen id-numeron perusteella, tekevät pisteen x- tai y-koordinaatille tarvittavat muutokset ja palauttavat koordinaatin halutussa muodossa. Metodien toiminnallisuus on y-koordinaatin kääntämistä lukuun ottamatta sama.

Ensin tarkistetaan, löytyykö PistePalvelulta yhtään pistettä id-numerolla *pisteRef*. Metodi palauttaa nollan, jos pistettä ei löydy. Tarkistuksen jälkeen pisteen koordinaatit muutetaan selaimelle sopivaksi sekä skaalataan. Skaalaamisen yhteydessä tehdään myös y-koordinaatin kääntäminen.

```
saaPisteY(pisteRef): number {
  let piste = this.pistePalvelu.saaPiste(pisteRef);
  if (!piste) {
    return 0;
  }
  //Koordinaatin muutokset
  //Skaalaus (ja y-koordinaatin kääntäminen)
}
```

Koodi 16. Metodin *saaPisteY()* alkualustus ja pisteen olemassa olon tarkistus.

Koordinaattimuutos aloitetaan luomalla lokaali muuttuja *y*, jolle asetetaan pisteen y-koordinaatti metodilla *saaY()*. Muutoksia ei saa tehdä suoraan tietokantaan, koska alkuperäisiä koordinaatteja voidaan haluta käyttää jossain muualla. Seuraavaksi pisteen koordinaatista vähennetään alueen minimirajan koordinaatti, jos pisteen koordinaatti on suurempi tai yhtä suuri kuin minimirajan koordinaatti. Metodi palauttaa arvon nolla, jos pisteen koordinaatti on alueen minimirajaa pienempi.

```
//Koordinaatin muutokset
let y: number = piste.saaY();
if (piste.saaY() >= this.aluePalvelu.minimiY) {
  y -= this.aluePalvelu.minimiY;
}
if (piste.saaY() < this.aluePalvelu.minimiY) {
  return 0;
}
```

Koodi 17. Muutetaan pisteen y-koordinaattia vastaamaan alueen rajoja.

Kun pisteen koordinaatit on muunnettu selaimelle ymmärrettävään muotoon, pisteet skaalataan halutulla tavalla. Skaalaus on syytä tehdä erityisesti käytettäessä todella isojen alueiden karttoja. Y-koodinaatille tehdään samalla origon kääntäminen vähentämällä *y* alueen korkeudesta. Erotus kerrotaan skaalauskertoimella ja saadaan lopullinen y-koordinaatti, joka palautetaan SVG-koodin käyttöön.

```
//Skaalaus (ja y-koordinaatin kääntäminen)
y = this.saaYSkaalaus() * (this.saaKorkeus() - y);
//x = this.saaXSkaalaus() * x;
return y;
```

Koodi 18. Skaalataan ja käännetään y-koordinaatti.

4.3.2 Teiden piirtäminen SVG-koodilla

Seuraavaksi siirrytään KarttaEditorin mallin puolelle. SVG-koodille tehdään oma palkki, jota tarvitaan myöhemmin editoinnin toteutuksessa. Kaikki teiden piirtämiseen tarvittava SVG-koodi tulee <svg>-elementin sisälle. <svg>-elementissä täytyy alustaa SVG-kuvan korkeus ja leveys sekä elementin tyyli CSS-koodina. SVG-kuvan leveys ja korkeus on alueiden leveys ja korkeus kerrottuna x- ja y-koordinaattien skaalauskertoimilla. CSS-koodin avulla piirretään kuvan reunoille musta katkoviiva erottamaan kuva muusta sivusta. <svg>-tunnisteelle annetaan myös teiden editointiin käytettäviä metodeja, jotka käsitellään omissa luvuissaan.

```
<div #kartta>
  <svg [attr.korkeus.px]="saaYSkaalaus()*saaKorkeus()"
      [attr.leveys.px]="saaXSkaalaus()*saaLeveys()"
      style="outline-style:dashed; outline-color:black; outline-width:1px;"
      <!-- Metodit teiden editointiin --> >
    <!-- SVG koodi -->
  </svg>
</div>
```

Koodi 19. SVG-kuvalle asetetaan korkeus ja leveys skaalattuna.

SVG-kuvaan piirretään kaikki tietokannan tiet käyttämällä Angularin for-silmukkaa **ngFor*. Yksi tie koostuu kahdesta <svg:line>-elementistä eli SVG-viivasta. Ensin piirretään leveämpi viiva, jonka leveys on tien leveys kaksinkertaisena. Leveän viivan päälle piirretään yhden pikselin paksuinen viiva merkitsemään tien keskikohtaa. Kummallakin tiellä on samat alku- ja loppupisteet, jotka haetaan tien metodeilla *saaPisteX()* ja *saaPisteY()*. Metodit saavat parametreinä pisteiden id-numeron. Tyylillä määritellään viivojen värit RGB-formaatissa. RGB-formaatissa ilmoitettu väri koostuu kolmesta numeroarvosta, jotka ovat järjestyksessä punainen, vihreä ja sininen. Tien väri on normaalisti vihreä ja valittuna sininen. Teiden valitseminen käsitellään omassa luvussaan.

```
<svg *ngFor="let tie of TiePalvelu.saaTiet()">
 <svg:line *ngIf="!tie.onValittu" <!-- Käsitellään myöhemmin -->
   [attr.x1]="saaPisteX(tie.alkuPiste)" [attr.x2]="saaPisteX(tie.loppuPiste)”
   [attr.y1]="saaPisteY(tie.alkuPiste)" [attr.y2]="saaPisteY(tie.loppuPiste)"
   [style.stroke-width]="tie.leveys*2" style="stroke:#596;"/>
 <svg:line *ngIf="!tie.onValittu" <!-- Käsitellään myöhemmin -->
   [attr.x1]="saaPisteX(tie.alkuPiste)" [attr.x2]="saaPisteX(tie.loppuPiste)"
   [attr.y1]="saaPisteY(tie.alkuPiste)" [attr.y2]="saaPisteY(tie.loppuPiste)"
   [style.stroke-width]="1" style="stroke:#254;"/>
</svg>
```

Koodi 20. Normaalin tien ja valitun tien SVG-koodi.

Teiden alku- ja loppupisteet piirretään elementillä <svg:circle> eli SVG-ympyrällä. Ympyröiden keskipisteen koordinaatit *cx* ja *cy* haetaan samoilla metodeilla kuin teiden pisteet aiemmin. Pisteiden säteeksi *r* ja piirron leveydeksi asetetaan yksi pikseli. Ympyrät täytetään mustalla värillä.

```
 <svg:circle
   [attr.cx]="saaPisteX(tie.alkuPiste)"
   [attr.cy]="SaaPisteY(tie.alkuPiste)"
   r="1" stroke="black" stroke-width="1" fill="black" />
 <svg:circle
   [attr.cx]="saaPisteX(tie.loppuPiste)"
   [attr.cy]="saaPisteY(tie.loppuPiste)"
   r="1" stroke="black" stroke-width="1" fill="black"/>
```

Koodi 21. Pisteiden piirtäminen.

Nyt selaimeen piirtyy viivoista ja pisteistä koostuva SVG-kuva Toivakan teistä (kuva 12). Lähellä kartan rajoja olevat tiet saattavat loppua kesken jo hyvin kaukana rajasta. Tämä johtuu rajojen ulkopuolella olevien pisteiden karsimisesta.

Kuva 12. SVG-koodilla piirretty kartta.

Teiden piirtämisen jälkeen toteutetaan editointityökalu, jolla saadaan ohjelmaan toiminnallisuutta.

4.4 Editointityökalun toteuttaminen

Työn viimeinen vaihe on toteuttaa editointityökalu teiden hallintaan. Editointityökalun tulee sisältää seuraavat ominaisuudet: yhden tai useamman tien valinta ja teiden poistaminen. Tiet voi valita joko klikkaamalla yksittäistä tietä tai maalaamalla alueen kartasta, jolloin kaikki maalatun alueen sisällä olevat tiet valitaan. Valitut tiet poistetaan klikkaamalla painiketta *Poista Tavarat* (engl. *Delete Items*).

4.4.1 Yhden tien valitseminen

Valinnan toteuttaminen aloitetaan lisäämällä Tie-luokalle muuttuja *onValittu*, johon säilötään tieto, onko tie valittu vai ei. Muuttujan avulla valitut tiet pystytään etsimään tietokannasta helposti.

Yksittäinen tie valitaan klikkaamalla tien <svg>-elementtiä, joten elementille täytyy antaa metodi valitsemisen käsittelyyn. Elementille annetaan klikkauksella aktivoituva tapahtuma *(click)* ja tapahtumalle annetaan metodi *valitaanTie()*. Metodi saa parametrinä piirrettävän tien. *(click)* aktivoituu vain, jos klikkaus alkaa ja loppuu samassa kohdassa.

```
<svg:line <!-- SVG koodi --> (click)="valitaanTie(tie)"/>
```

Koodi 22. Metodi *valitaanTie()* annetaan SVG-viivoille.

Metodissa *valitaanTie()* etsitään klikattu tie tiePalvelulta ja asetetaan tie valituksi. Tie asetetaan takaisin normaaliksi, jos klikattu tie on jo valittu.

```
valitaanTie(klikattuTie: Tie) {
  let onJoValittu = false;
  for (let tie of this.tiePalvelu.saaTiet()) {
    if (tie.id == klikattuTie.id) {
      if (tie.onValittu) { onjoValittu = true; }
      if (!onJoValittu)  { tie.onValittu = true; }
      if (onJoValittu)   { tie.onValittu = false; }
    }
  }
}
```

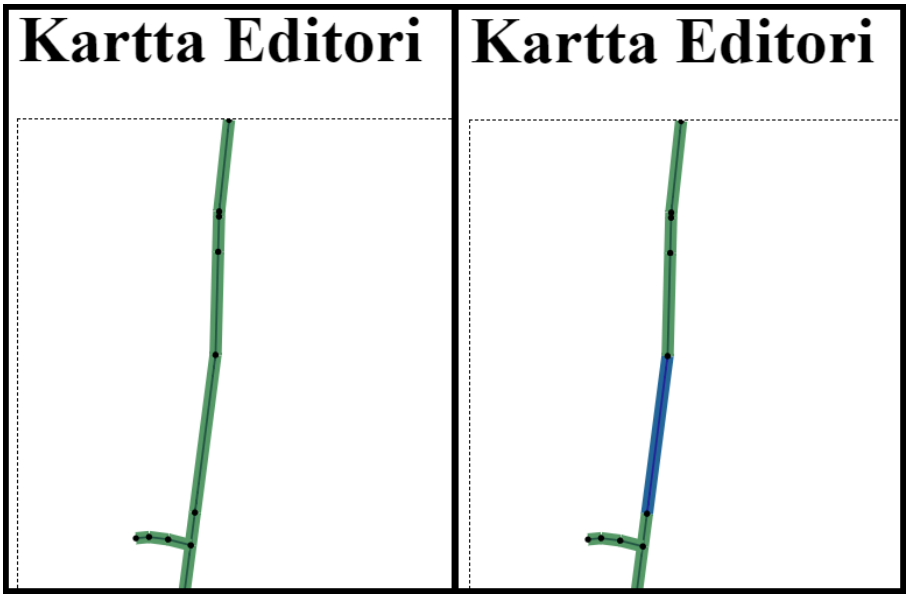
Koodi 23. Tien valinnan tarkistaminen ja asettaminen.

Lopuksi piirretään tie eri väreillä valinnan mukaan. Valinnan määrittelyyn käytetään Angularin if-lausetta **ngIf*. Normaalisti tiet piirretään vihreänä ja valitut tiet piirretään sinisenä. Väriä lukuun ottamatta teiden SVG-koodi on sama.

```
<svg:line *ngIf="!tie.onValittu"
  <!-- SVG koodi normaalille tielle (vihreä) -->
  (click)="valitaanTie(tie)"/>
<svg:line *ngIf="tie.onValittu"
  <!-- SVG koodi valitulle tielle (sininen) -->
  (click)="valitaanTie(tie)"/>
```

Koodi 24. SVG-viivat valinnan tunnistamisella ja tien valinnalla.

Kuvasta 13 nähdään, kuinka tie valitaan sitä klikkaamalla. Valinnan myötä tien väri vaihtuu siniseksi, kuten on tarkoituskin.



Kuva 13. Yksittäisen valinta klikkaamalla SVG-viivaa.

Yhden tien valitseminen luo hyvän pohjan monen tien valitsemiselle.

4.4.2 Monen tien valitseminen

Monen tien yhtäaikaista valitsemista varten tarvitaan keino rajata haluttu alue helposti hiirtä käyttämällä. Alueen rajaamista varten luodaan keino maalata haluttu alue kartasta. Aluetta maalatessa luodaan neljä pistettä, joiden sisällä olevat tiet valitaan.

Maalaaminen toteutetaan kolmella metodilla, joita kutsutaan kolmella eri hiiren tapahtumalla. Metodi *maalausAlkaa()* aktivoituu hiiren painikkeen painuessa alas, jolloin maalaaminen alkaa. Metodia *maalausMenossa()* kutsutaan aina hiiren liikkuessa. Maalaaminen loppuu hiiren painikkeen vapautuessa metodiin *maalausLoppuu()*. Metodit *maalausAlkaa()* ja *maalausMenossa()* saavat parametrinä tapahtuman *$event*, jolla saadaan käyttöön hiiren sijainti.

```
<div #kartta >
  <svg <!-- SVG alustus -->
      (mousedown)="maalausAlkaa($event)"
      (mousemove)="maalausMenossa($event)"
      (mouseup)="maalausLoppuu()">
    <!-- SVG koodi -->
  </svg>
</div>
```

Koodi 25. SVG-kuvalle annetaan metodit maalaamisen toteuttamiseen.

Maalausalueen piirtoa varten luodaan neljä pistettä. Tärkeimmät pisteistä ovat *maalausPisteAlku* ja *maalausPisteLoppu*. Alkupiste pysyy koko maalauksen ajan samana ja loppupiste liikkuu hiiren mukana. Alku- ja loppupisteen perusteella lasketaan kahden lisäpisteen sijainnit, jotta saadaan suorakulmion muotoinen maalausalue luotua.

Metodi *maalausAlkaa()* aloittaa maalaamisen sekä kertoo KarttaPalvelulle maalaamisen olevan käynnissä muuttujalla *maalataan*. Lisäksi kaikki tiet palautetaan normaaleiksi.

```
maalausAlkaa(event: MouseEvent) {
  //Alkuasetukset
  this.maalataan = true;
  for (let tie of this.tiePalvelu.saaTiet()) {
    tie.onValittu = false;
  }
  //Pisteiden sijainnin asettaminen
}
```

Koodi 26. Metodin *maalausAlkaa()* alkuasetukset.

Alkuasetusten jälkeen jokaisen maalauspisteen alkuarvoksi asetetaan hiiren todellinen sijainti. Hiiren todellinen sijainti on hiiren sijainti elementillä (*layerX* ja *layerY*) vähennettynä elementin *etäisyydellä* (engl. *offset*) selaimessa (*karttaOffsetLeft* ja *karttaOffsetTop*). *layerX* ja *layerY* saadaan hiiren tapahtumalta *event*. Elementin etäisyyden saaminen on hiukan työläämpi operaatio.

```
maalausAlkaa(event: MouseEvent) {
  //Alkuasetukset

  //Pisteiden sijainnin asettaminen
  this.maalausPisteAlku.x = event.layerX - this.karttaOffsetLeft;
  this.maalausPisteAlku.y = event.layerY - this.karttaOffsetTop;
  this.maalausPisteLoppu.x = event.layerX - this.karttaOffsetLeft;
  this.maalausPisteLoppu.y = event.layerY - this.karttaOffsetTop;

  this.lisaPiste.x = this.maalausPisteAlku.x;
  this.lisaPiste.y = this.maalausPisteLoppu.y;
  this.lisaPiste2.x = this.maalausPisteLoppu.x;
  this.lisaPiste2.y = this.maalausPisteAlku.y;
}
```

Koodi 27. Maalauspisteiden sijainnin asettaminen maalaamisen alussa.

Maalauspisteiden koordinaateissa täytyy ottaa huomioon piirtoalueen sijainti selaimessa. <svg>-elementti asetetaan -palkkiin, jonka sijainti selaimessa saadaan tiedusteltua käyttämällä Angularin ominaisuutta *@ViewChild*. *@ViewChild* luo objektin parametrinä annetusta HTML:n elementistä. Elementille annetaan tunniste *#kartta*, jonka avulla

@ViewChild osaa luoda objektin -elementistä. Elementin sijainti saadaan hakemalla objektin *nativeElement* attribuutit *offsetTop* ja *offsetLeft*. Attribuutit asetetaan KarttaEditorin muuttujille *karttaOffsetTop* ja *karttaOffsetLeft* ohjelman käynnistyessä.

```
@ViewChild('kartta') kartta;
karttaOffsetLeft: number;
karttaOffsetTop: number;
ngOnInit() {
  this.karttaOffsetTop = this.kartta.nativeElement.offsetTop;
  this.karttaOffsetLeft = this.kartta.nativeElement.offsetLeft;
}
```

Koodi 28. Selvitetään kartan sijainti selaimessa.

Koordinaattien oikaisun jälkeen voidaan piirtää maalauspisteet sekä niiden väliset viivat KarttaEditorin mallissa. Sekä pisteet että viivat ovat mustia yhden pikselin paksuisia <svg>-elementtejä. Yhteensä piirretään neljä pistettä ja neljä viivaa. Kummatkin piirretään vain maalauksen aikana, mikä tarkistetaan <svg>-elementille annetulla Angularin if-lauseella **ngIf*.

```
<svg *ngIf="maalataan">
  <!-- maalausPisteAlku -->
  <svg:circle
    [attr.cx]="dragPointStart.x" [attr.cy]="dragPointStart.y"
    r="1" stroke="black" stroke-width="1" fill="black" />
  <!-- maalausPisteLoppu -->
  <!-- lisaPiste -->
  <!-- lisaPiste2 -->
  <!-- viiva maalausPisteAlku - lisaPiste -->
  <svg:line
    [attr.x1]="dragPointStart.x" [attr.x2]="dragPointExtra.x"
    [attr.y1]="dragPointStart.y" [attr.y2]="dragPointExtra.y"
    stroke="black" stroke-width="1" />
  <!-- viiva maalausPisteAlku – lisaPiste2 -->
  <!-- viiva lisaPiste - maalausPisteLoppu -->
  <!-- viiva lisaPiste2 - maalausPisteLoppu -->
</svg>
```

Koodi 29. Maalauspisteiden ja niiden välisten viivojen piirtäminen.

Seuraavaksi maalauspisteet pannaan liikkumaan hiiren mukana metodilla *maalausMenossa()*. Metodi saa hiiren tapahtuman *event* ja pisteiden sijainnit asetetaan samalla tavalla, kuten metodissa *maalausAlkaa()*. Ainoana erona pisteet päivitetään vain, jos maalaus on menossa. Ilman tarkistusta pisteiden sijainti päivitettäisiin aina hiiren liikkuessa, myös hiiren painikkeen ollessa ylhäällä. Alkupisteelle ei tehdä mitään.

```
maalausMenossa(event: MouseEvent) {
  if (this.maalataan == true) {
    this.maalausPisteLoppu.x = event.layerX - this.karttaOffsetLeft;
    this.maalausPisteLoppu.y = event.layerY - this.karttaOffsetTop;

    this.lisaPiste.x = this.maalausPisteAlku.x;
    this.lisaPiste.y = this.maalausPisteLoppu.y;
    this.lisaPiste2.x = this.maalausPisteLoppu.x;
    this.lisaPiste2.y = this.maalausPisteAlku.y;
  }
}
```

Koodi 30. Maalauspisteiden sijainnin asettaminen maalaamisen aikana.

Alueen maalaaminen päättyy metodiin *maalausLoppuu()*, joka ajetaan hiiren painikkeen noustessa. Ensin metodi asettaa maalauksen pois päältä ja luo taulukon *valiPisteet*, johon säilötään tietokannasta etsittävät pisteet. Alkuasetusten jälkeen maalauspisteiden paikkoja vaihdetaan tarvittaessa. Lopuksi etsitään maalauspisteiden sisällä olevat pisteet, joiden id-numeroita käyttämällä etsitään valitut tiet tietokannasta.

```
maalausLoppuu() {
  //Alkuasetukset
  this.maalataan = false;
  let valiPisteet: Point[] = [];

  //Maalauspisteiden paikan vaihtaminen tarvittaessa
  //Etsitään tietokannasta maalatulla alueella olevat pisteet
  //Etsitään tietokannasta valittavat tiet pisteiden perusteella
}
```

Koodi 31. Metodin *maalausLoppuu()* alkuasetukset.

Maalauspisteiden sijainteja täytyy vaihtaa, jos maalaaminen tapahtuu johonkin toiseen suuntaan kuin vasemmalta ylhäältä oikealle alas. Syy tähän on se, että maalausalueen sisällä olevat pisteet etsitään käyttämällä maalausalueen oikeaa yläkulmaa ja vasenta alakulmaa. Maalatessa alas oikealle käytettäisiin aloitus- ja lopetuspisteitä, mutta tarkastus ei toimi kyseisillä pisteillä maalauksen tapahtuessa muihin suuntiin. Asia korjataan luomalla väliaikaiset alku- ja loppupisteet *valiAlku* ja *valiLoppu*, joille annetaan oletuksena normaalit alku- ja loppupisteet. *valiAlku* tulee aina olla maalausalueen vasen yläkulma ja vastaavasti *valiLoppu* tulee aina olla maalausalueen oikea alakulma. Oikeat pisteet asetetaan kolmella tarkistuksella, joilla vertaillaan pisteiden x- ja y-koordinaatteja.

```
//Pisteiden paikan vaihtaminen tarvittaessa
let valiAlku: Piste = this.maalausPisteAlku;
let valiLoppu: Piste = this.maalausPisteLoppu;
if (this.maalausPisteAlku.x > this.maalausPisteLoppu.x &&
    this.maalausPisteAlku.y > this.maalausPisteLoppu.y) {
  valiAlku = this.maalausPisteLoppu;
  valiLoppu = this.maalausPisteAlku;
}
if (this.maalausPisteAlku.x > this. maalausPisteLoppu.x &&
    this.maalausPisteAlku.y < this. maalausPisteLoppu.y) {
  valiAlku = this.valiPiste2;
  valiLoppu = this.valiPiste;
}
if (this.maalausPisteAlku.x < this.maalausPisteLoppu.x &&
    this.maalausPisteAlku.y > this.maalausPisteLoppu.y) {
  valiAlku = this.valiPiste;
  valiLoppu = this.valiPiste2;
}
```

Koodi 32. Maalauspisteiden sijaintien vaihtaminen tarvittaessa.

Tarvittavien pisteiden vaihdosten jälkeen etsitään maalausalueen sisällä olevat pisteet. Etsintää varten käydään kaikki PistePalvelun pisteet läpi for-silmukalla. Map-rakenne täytyy muuttaa taulukoksi metodilla *Array.from()*, jolle annetaan Mapin *arvot* (engl. *values*) eli pisteet. Seuraavaksi tarkistetaan, ovatko pisteen x- ja y-koodinaatit isompia kuin alkupisteen koordinaatit ja pienempiä kuin loppupisteen koordinaatit. Kaikkien ehtojen toteutuessa piste on maalausalueen sisällä, joten se valitaan ja lisätään taulukkoon *valiPisteet*.

```
//Etsitään tietokannasta maalatulla alueella olevat pisteet
for (let piste of Array.from(this.pistePalvelu.saaPisteet().values()))
{
  if (valiAlku.x < this.saaPisteX(piste.id) &&
      this.saaPisteX(piste.id) < valiLoppu.x &&
      valiAlku.y < this.saaPisteY(piste.id) &&
      this.saaPisteY(piste.id) < valiLoppu.y ) {
    piste.onValittu = true;
    valiPisteet.push(piste);
  }
}
```

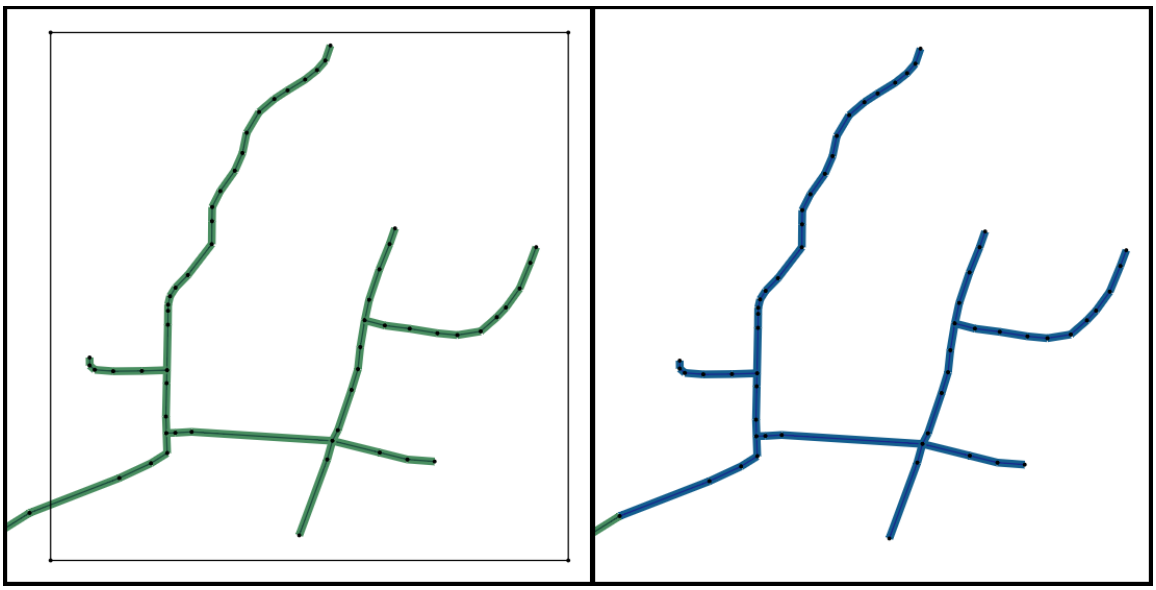
Koodi 33. Etsitään tietokannasta jokainen maalausalueen sisällä oleva piste.

Lopuksi etsitään TiePalvelulta kaikki tiet, joiden alku- tai loppupisteiden id-numerot täsmäävät taulukon *valiPisteet* pisteiden id-numeron kanssa. Tie valitaan, jos edes toinen tien pisteen id-numeroista täsmää valittujen pisteiden kanssa.

```
//Etsitään tietokannasta valittavat tiet pisteiden perusteella
for (let piste of valiPisteet) {
  for (let tie of this.tiePalvelu.saaTiet()) {
    if (tie.loppuPiste == piste.id || tie.AlkuPiste == piste.id) {
      tie.onValittu = true;
    }
  }
}
```

Koodi 34. Valitaan tiet, joiden alku- tai loppupiste on maalausalueen sisällä.

SVG-kuvan klikkaaminen luo nyt klikkauskohtaan maalauspisteet, jotka liikkuvat hiiren mukana. Hiiren painikkeen vapautuessa kaikki maalausalueen sisällä olevien pisteiden omistavat tiet valitaan kuvan 14 mukaisesti.



Kuva 14. Monen tien valinta maalaamalla alue hiirellä.

Teiden valitseminen on turhaa ilman mitään teille tehtäviä ominaisuuksia, joten seuraavaksi luodaan tapa poistaa valitut tiet.

4.4.3 Teiden poistaminen

Viimeinen käsiteltävä aihe on teiden poistaminen. Teiden poistamista varten luodaan TiePalvelulle metodi ”*poistaValitutTiet()*” ja luodaan ohjelmaan painike, jolla metodi saadaan ajettua.

Metodin *poistaValitutTiet()* toiminta on hyvin yksinkertainen. Luodaan taulukko *poistettavatId*, jolle annetaan jokaisen valitun tien id-numero. Valitut tiet etsitään for-silmukan

avulla. Valittujen teiden id-numeroiden löydyttyä *poistettavatId* käydään läpi for-silmukalla. Jokainen tie, jonka id-numero täsmää taulukosta löytyvän id-numeron kanssa, poistetaan TiePalvelun metodilla *poistaTie()*.

```
  poistaValitutTiet(): void {
    let poistettavatId: number[] = [];
    for (let tie of this.tiet) {
      if (tie.onValittu) {
        poistettavatId.push(tie.id);
      }
    }
    for (let poistettavaId of poistettavatId) {
      this.poistaTie(poistettavaId);
    }
  }
```

Koodi 35. Poistetaan valitut tiet tietokannasta.

KarttaEditorin malliin lisätään painike-elementti <button>, jolle annetaan klikkaustapahtuma *(click)*. Tapahtumalle annetaan metodi *painikePoistaTavarat()*.

```
<div>
  <button (click)="painikePoistaTavarat()">Delete Items</button>
</div>
```

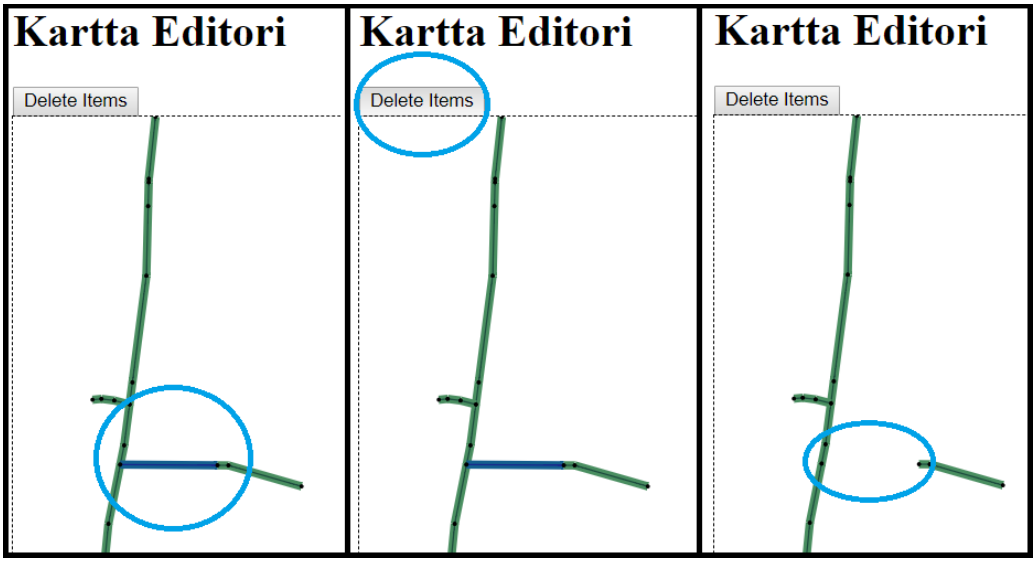
Koodi 36. Lisätään sovellukseen painike, jolla poistetaan valitut tiet.

Metodi *painikePoistaTavarat()* kutsuu TiePalvelun metodia *poistaValitutTiet()*. Metodiin *painikePoistaTavarat()* voi jatkossa lisätä muita metodeja, esim. pisteiden tai muiden karttaelementtien poistamiselle.

```
painikePoistaTavarat(): void {
    this.tiePalvelu.poistaValitutTiet();
  }
```

Koodi 37. Metodi *painikePoistaTavarat*() kutsuu TiePalvelua poistamaan valitut tiet.

Nyt valittuja teitä pystyy poistamaan. Kuvassa 15 valitaan yksittäinen tie klikkaamalla sitä ja poistetaan valittu tie painamalla painiketta *Delete Items*.



Kuva 15. Yksittäisen tien valitseminen ja poistaminen.

Kuvassa 16 valitaan monta tietä maalaamalla alue ja kaikki maalatun alueen tiet poistetaan painamalla painiketta *Delete Items*.

Kuva 16. Usean tien valitseminen ja poistaminen.

Teiden poistamisen myötä työn tavoitteet on saavutettu. Seuraavissa luvuissa käsitellään työn lopputulos, ideat jatkokehitykseen sekä yhteenveto työstä.

5 Lopputulos

Työn tuloksena on karttatiedon käsittelyyn tarkoitettu ohjelma, joka toimii OSM XML -tiedostojen kanssa. Ohjelma käsittelee toistaiseksi vain kartasta saatuja teitä. Tiet pystytään valitsemaan ja poistamaan.

Käsiteltävä tiedosto annetaan TiedostoPalvelulle, joka lisää kartan tiedot tietokantaan käyttämällä XMLKasittelijaPalvelua. XMLKasitelijaPalvelu etsii tiedostosta halutut elementit, jotka annetaan muille palveluille käsiteltäväksi. XMLKasittelijaPalvelussa määritellään myös halutut tietyypit.

Kartan rajojen ja pisteiden koordinaatit muutetaan asteista metreihin UTM-projektiota käyttäen ennen niiden lisäämistä tietokantaan. Selaimessa piirtämistä varten tietokannasta saataville koordinaateille tehdään lisämuutoksia KarttaPalvelussa.

Ohjelma piirtää OSM XML -tiedostosta saadut tiet käyttämällä SVG-kieltä. SVG-elementeille on annettu metodeja editointiominaisuuksia varten.

Yksittäinen tie valitaan klikkaamalla tietä. Usea tie valitaan maalaamalla hiirellä alue, jonka sisällä olevat tiet halutaan valita. Valitut tiet muuttavat piirretään sinisenä. Normaalisti tiet piirretään vihreänä. Valitut tiet pystytään poistamaan klikkaamalla poistamispainiketta.

5.1 Onnistumiset

Työn lopputulos on hyvä pohja kartan editointiin tarkoitetulle ohjelmalle. Ohjelmaa on helppo lähteä laajentamaan ja jatkokehittämään.

Koordinaattimuutoksista huolimatta tiet piirtyvät oikein ja selkeästi verrattuna alkuperäiseen karttaan. Esimerkkikartasta tunnistaa Toivakan keskustan helposti.

Ohjelman suorituskyvyn osalta päästiin tavoitteeseen ja ohjelma pyörii sulavasti selaimessa myös todella ison alueen karttaa piirrettäessä. Raskain vaihe on tiedoston parsiminen ohjelman käynnistyessä. Parsimisen hitaus johtuu xml2js-työkalusta, joten suorituskykyä ei pysty parantamaan käynnistymisen osalta ilman työkalun vaihtamista. Työkalun vaihtamiselle ei kuitenkaan nähty tarvetta, koska ohjelma toimii muuten riittävän nopeasti.

Teiden valitseminen on helppoa ja se toimii oikein. Poistaminen poistaa oikeat tiet niin SVG-kuvasta kuin myös tietokannasta. Teiden poistuminen myös tietokannasta on tärkeää, jos kartan muokattua tietoa halutaan käyttää toisessa ohjelmassa.

5.2 Parannettavaa

Työtä tehdessä aikaa meni seikkoihin, jotka alusta lähtien paremmin toteuttamalla olisivat voineet säästää paljon aikaa. Esimerkiksi tietyyppien nimien etsimiseen meni paljon aikaa ja lisäämällä kaikki tietyypit tietokantaan aikaa olisi säästynyt paljon. Projektin alussa aikaa meni myös yrittäessä saada lukuiset JavaScript-paketit toimimaan Angularin kanssa.

Ajan puutteen takia suunniteltuja ominaisuuksia jouduttiin karsimaan. Karsituista ominaisuuksista tärkeimmät olivat rakennusten lisääminen tietokantaan sekä uusien teiden ja rakennusten luominen ohjelmalla. Alkuperäinen suunnitelma oli todella laaja ja tehtäviä olisi pitänyt priorisoida sekä suunnitella enemmän.

Ohjelmassa myös ilmeni muutamia bugeja työn aikana. Klikkaustapahtumat eivät aina toimi keskenään ja tuntemattomasta syystä johtuen selaimen zoomaustaso vaikuttaa asiaan. Alun perin teiden mukana poistettiin myös valitut pisteet, mutta ominaisuus karsittiin, koska se saattoi aiheuttaa ongelmia SVG-koodin kanssa. Osalle teistä saattoi jäädä referenssi poistettuun pisteeseen, minkä takia tie piirtyi väärin. Uuden tien luomisen toteuttaminen aloitettiin, mutta uusien teiden pisteiden sijaintien kanssa tuli vastaavanlaisia ongelmia ja ominaisuus lopulta karsittiin.

5.3 Jatkokehitysideat

Ohjelman kehitystä voi jatkaa lisäämällä ominaisuuksia ja muokkaamalla ulkoasua. Tulevaisuuden ominaisuudet voivat liittyä esim. karttatiedostoihin, kartan elementteihin tai tavaroiden poistamiseen ja luomiseen. Ohessa ideoita ominaisuuksista, joita tämänkaltaiseen ohjelmaan voisi lisätä.

OSM XML -tiedostoja olisi mukavampi käsitellä, jos ne ladattaisiin ohjelmaan ja käyttäjä pystyisi valitsemaan halutun karttatiedoston listasta. Samaan listaan voisi tallentaa muutetut kartat ja karttoja voisi viedä ulos ohjelmasta halutussa formaatissa.

Aikarajoitteiden takia ohjelmassa ei toimi kuin tiet, mutta muiden karttaelementtien lisääminen on todella tärkeää. Rakennukset ja maantieteelliset elementit, kuten järvet ja metsät, tekisivät kartoista paljon helpommin hahmotettavia.

Tietyyppien valitseminen on toteutettu kankeasti ja sen ylläpito on erittäin työlästä. Kaikki tietyypit voisi lisätä kerralla tietokantaan ja editorista voisi tiedoston parsimisen jälkeen valita piirrettävät tiet, esim. valintaruutuja klikkailemalla. Tiedostoa viedessä ulos ohjelmasta otetaan huomioon vain piirretyt tiet. Tämän ominaisuuden pystyisi laajentamaan myös muihin karttaelementteihin, joilla on paljon tunnisteita.

Tavaroiden poistaminen täytyy laajentaa kattamaan kaikki kartan elementit. Turhat pisteet täytyy poistaa, koska siten saadaan vapautettua muistia. Pikanäppäimet tekisivät ohjelman käyttämisestä helpompaa ja tehokkaampaa, esim. tien poistaminen näppäimellä *DEL* on huomattavasti nopeampaa kuin klikkaamalla poistamiseen tarkoitettua painiketta.

Ohjelman todellinen potentiaali piilee uusien rakennusten ja teiden lisäämisessä karttaan sekä jo olemassa olevien elementtien muokkaamisessa ja liikuttelussa. Tällöin ohjelmaa pystyisi käyttämään pohjana monissa hyvin erilaisissa sovelluksissa.

6 Yhteenveto

Tavoitteena oli toteuttaa verkkoselaimessa toimiva sovellus, jolla voi muokata sille annetun OpenStreetMap-tiedoston dataa. Tavoitteena oli lisätä kartan tiet ohjelman tietokantaan ja toteuttaa ominaisuudet tietokannassa olevien teiden käsittelyyn. Ohjelmaan oli toteutettu ennalta tietorakenne kartan elementeille ja metodit elementtien hallintaan.

OpenStreetMapin kotisivuilta viety OSM XML -tiedosto parsittiin käyttämällä xml2js-työkalua. Tiedostosta parsittiin kartan rajat, pisteet ja tiet. Parsimisen yhteydessä valittiin halutut tietyypit käyttämällä teiden tunnisteita. Parsittu tieto annettiin kunkin elementin palvelulle jatkokäsittelyyn. Rajojen ja pisteiden koordinaateille tehtiin muunnos asteista metreiksi UTM-projektioon käyttämällä utm-latlng-työkalua. Tarvittavien muunnoksien jälkeen elementit lisättiin tietokantaan. Parsimiseen käytetty työkalu todettiin hitaaksi ison alueen karttaa käytettäessä, mutta hitaus ilmeni vain sovelluksen käynnistyessä eikä se haitannut sovelluksen käyttämistä. Helppouden takia kyseisen työkalun käyttämistä jatkettiin hitaudesta huolimatta.

Kartan piirtäminen toteutettiin käyttämällä SVG-vektorigrafiikkaa. Tiet koostuivat alku- ja loppupisteistä sekä pisteiden välisistä viivoista. SVG-kuvalle ja SVG-viivoille annettiin hiiren klikkauksista aktivoituvat tapahtumat teiden valitsemista varten. Tiet piirrettiin normaalisti vihreänä ja valittuna ollessaan sinisenä.

Editoriin toteutettiin ominaisuudet teiden valitsemiselle ja poistamiselle. Yksittäinen tie valittiin klikkaamalla SVG-viivaa. Usean tien valitseminen tehtiin maalaamalla alue, jonka sisällä olevat tiet valittiin. Valittujen teiden poistamista varten luotiin painike, jota klikkaamalla valitut tiet poistettiin sekä SVG-kuvasta että tietokannasta.

Lopputulos on hyvä pohja karttaa hyödyntäville sovelluksille. Ohjelmaa voidaan jatkokehittää lisäämällä tuki muille kartan elementeille, kuten rakennuksille, ja lisäämällä ominaisuuden lisätä ja muokata elementtejä.

Lähteet

[1] Sarva J. / Otavan Opisto. HTML:n perusteet. Oppimateriaali, 2012. Saatavilla: http://opinnot.internetix.fi/fi/muikku2materiaalit/muut/ammatillinen/web/html/html_perusteet.pdf [Haettu 9.3.2018]

[2] HTML 5.2 -dokumentaatio. Saatavilla: https://www.w3.org/TR/html/ [Haettu 9.3.2018]

[3] Duckett J. HTML&CSS – design and build websites. John Wiley & Sons, 2011.

[4] W3Schools -verkkosivusto, HTML Event Attributes. Saatavilla: https://www.w3schools.com/tags/ref_eventattributes.asp [Haettu 26.3.2018]

[5] W3Schools -verkkosivusto, HTML Tags. Saatavilla: https://www.w3schools.com/tags/tag_button.asp [Haettu 26.3.2018]

[6] MDN web docs, HTML -dokumentaatio, input, range. Saatavilla: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/range [Haettu 24.3.2018]

[7] W3C -kotisivu, Standards, CSS. Saatavilla: https://www.w3.org/standards/techs/css#w3c_all [Haettu 24.3.2018]

[8] Sarva J. ja Tarmia M. / Otavan Opisto. CSS-perusteet. Oppimateriaali, 2011. Saatavilla: https://avkymppi.net/css-perusteet.pdf [Haettu 9.3.2018]

[9] Nykänen O. SVG – Skaalautuva vektorigrafiikka. WSOY, 1. painos, tammikuu 2007.

[10] Peltomäki J. JavaScript. Teknolit, 1. painos, maaliskuu 2000.

[11] MDN web docs, JavaScript -dokumentaatio, let. Saatavilla: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let [Haettu 28.3.2018]

[12] Heikniemi J. Mikä on XML? Verkkojulkaisu, 19.2.2001. Saatavilla: http://www.heikniemi.fi/kirj/moxml.html [Haettu 8.3.2018]

[13] Angular -dokumentaatio, Guide, Architecture Overview. Saatavilla: https://angular.io/guide/architecture [Haettu 14.3.2018]

[14] Tutorials Point. TypeScript. E-kirja, 2016. Saatavilla: https://www.tutorialspoint.com/typescript/typescript_tutorial.pdf [Haettu 27.3.2018]

[15] Angular CLI -kotisivu. Saatavilla: https://cli.angular.io/ [Haettu 6.3.2018]

[16] Angular CLI:n GitHub -sivu. Saatavilla: https://github.com/angular/angular-cli#documentation [Haettu 6.3.2018]

[17] Angular CLI:n GitHub -dokumentaatio, serve. Saatavilla: https://github.com/angular/angular-cli/wiki/serve [Haettu 6.3.2018]

[18] Node.js -kotisivu. Saatavilla: https://nodejs.org/en/about/ [Haettu 7.3.2018]

[19] npm -kotisivu. Saatavilla: https://www.npmjs.com/ [Haettu 7.3.2018]

[20] npm -dokumentaatio, Getting Started 04. Saatavilla: https://docs.npmjs.com/getting-started/installing-npm-packages-locally [Haettu 7.3.2018]

[21] npm -dokumentaatio, Getting Started 08. Saatavilla: https://docs.npmjs.com/getting-started/installing-npm-packages-globally [Haettu 7.3.2018]

[22] OpenStreetMap Wiki, About OpenStreetMap. Saatavilla: https://wiki.openstreet-map.org/wiki/About_OpenStreetMap [Haettu 7.3.2018]

[23] OpenStreetMap Wiki, OSM XML. Saatavilla: https://wiki.openstreet-map.org/wiki/OSM_XML [Haettu 8.3.0218]

[24] OpenStreetMap -kotisivu, copyright. Saatavilla: https://www.openstreetmap.org/co-pyright/en [Haettu 7.3.2018]

[25] Geokov -verkkosivusto, Education, UTM. Saatavilla: http://geokov.com/educa-tion/utm.aspx [Haettu 27.3.2018]

[26] Wikipedia Commons. Saatavilla: https://upload.wikimedia.org/wikipedia/com-mons/e/ed/Utm-zones.jpg [Haettu 28.3.2018]

[27] utm-latlng GitHub -sivu. Saatavilla: https://github.com/shahid28/utm-latlng [Haettu 8.3.0218]

[28] SVG -kotisivu, About SVG. Saatavilla: https://www.w3.org/Graphics/SVG/About.html [Haettu 8.3.2018]

[29] Parviainen T. SVG and Canvas Graphics in Angular 2. Verkkojulkaisu, 12.12.2016. Saatavilla: https://teropa.info/blog/2016/12/12/graphics-in-angular-2.html [Haettu 19.3.2018]

[30] Node.js -kotisivu, Downloads. Saatavilla: https://nodejs.org/en/download/ [Haettu 8.4.2018]

[31] npm -kotisivu, Get npm. Saatavilla: https://www.npmjs.com/get-npm [Haettu 9.4.2018]

[32] OpenStreetMap -kotisivu. Saatavilla: https://www.openstreetmap.org/export#map=14/62.0935/26.1125 [Haettu 6.3.2018]

[33] node-xml2js:n GitHub -sivu. Saatavilla: https://github.com/Leonidas-from-XIV/node-xml2js [Haettu 8.3.0218]

[34] OpenStreetMap Wiki, Key:highway. Saatavilla: https://wiki.open-streetmap.org/wiki/Key:highway [Haettu 17.4.2018]