Juuso Palmroth

VERKKOKARTTASOVELLUS

JavaScript/Openlayers-karttasovellus

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Huhtikuu 2020

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Huhtikuu 2020	**Tekijä/tekijät** Juuso Palmroth
Koulutusohjelma Tietotekniikka		
Työn nimi Verkkokarttasovellus JavaScript / Openlayers karttasovellus		
Työn ohjaaja Kauko Kolehmainen		**Sivumäärä** 22
Työelämäohjaaja Ville Hietala		

Opinnäytetyön päämääränä oli luoda asiakkaalle verkko-ohjelmistopino, jonka päälle rakennettiin verkkosovellus. Verkkosovelluksen tarkoituksena oli näyttää asiakkaan kuituverkon kattavuusalue kartalla verkkoselaimessa. Kuituverkon reittien sijainnit tuli hakea erillisestä tiedostosta ja päivittää sivun latauksen yhteydessä kartalle. Työssä käytössä oli Node.js-palvelinympäristö ja Express.js-reititysvälio-hjelma. Asiakasnäkymä luotiin käyttäen Pug-mallipohjaa.

Teoriaosuudessa käydään läpi Javascriptin, Node.js:än ja Openlayers-kirjaston teoriaa. Toteutuksessa käydään läpi sovelluksen teko vaiheittain ja avataan hieman eri vaiheiden tarpeita. Toteutus on kahdessa osassa, palvelinpuolen sovellus ja asiakaspuolen sovellus.

Opinnäytetyön tuloksena voitiin todeta asiakkaan vaatimusten mukaisen sovelluksen teon onnistuvan kohtuullisella vaivalla. Opinnäytetyön aikana selvisi nykyisen XML-tiedoston sisältävän paljon turhaa tietoa, joka mahdollisesti moninkertaistuessaan hidastaa sijaintien läpikäyntiä.

Asiasanat
JavaScript, Node.js, Express.js, Openlayers, HTTP, verkkosovellus, full stack

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Centria University of Applied Sciences	**Date** April 2020	**Author** Juuso Palmroth
Degree programme Information technology		
Name of thesis Web mapping application JavaScript / Openlayers map application		
Instructor Kauko Kolehmainen		**Pages** 22
Supervisor Ville Hietala		

The aim of this thesis was to create a web application stack and a web application on top of it. The aim of the web application was to show the optical fiber routes on the map on a web browser. Locations of the cables were read from an external file and drawn on to the map by every web page landing. This project required the use of Node.js as server environment and routed with Express.js middleware. Web pages were rendered from a Pug template.

The first part of the thesis covers the basic theory of Javascript, Node.js and Openlayers library. The second part reports the implementation of the web application step-by-step and explains the needs for the used methods. Implementation is divided into two parts, client-side application and server-side application.

As the conclusion of the thesis, web mapping application with the requirements of the client is doable with a reasonable effort. The amount of unnecessary data in XML-file could cause a performance issues if data is multiplied.

Key words
JavaScript, Node.js, Express.js, Openlayers, HTTP, web application, full stack

KÄSITTEIDEN MÄÄRITTELY

Funktio	Toimintalohko, jonka toiminnot suoritetaan vain sitä kutsutussa sijainnissa
Muuttuja	Alue, jonka tietotyypin perusteella kääntäjä varaa muistista tarpeeksi tilaa arvon säilytykseen
Parametri	Funktion kutsujan funktiolle siirtämää muuttuja
Kirjasto	Kokoelma muuttujia, luokkia ja funktioita, jotka voidaan lisätä osaksi projektia
Rajapinta	Valikoidut funktiot, joilla voidaan rajatusti käyttää sovelluksen ulkopuolista ohjelmaa
Ohjelmistopino	Joukko sovelluksia, joilla rakennetaan projektin vaatima suoritusympäristö
Objekti	Luokka-mallista luotu tietuetyyppi, joka on kokoelma muuttujia ja funktioita
Renderöinti	Kuvan piirtäminen kuvauksen datan pohjalta
Komentosarjakieli	Tulkattu ohjelmointikieli, joka yleensä suorittaa esim. Käyttöjärjestelmän valmiita toiminnollisuuksia.

TIIVISTELMÄ
ABSTRACT
KÄSITTEIDEN MÄÄRITTELY
SISÄLLYS

KUVIOT

KUVAT

1 JOHDANTO

Kokkolan alueella toimivalla Internet-palveluntarjoajalla Keskikaistalla on tarve saada kuituverkkokartta Keskikaistan verkkosivuille, josta asiakkaat voivat käydä katsomassa yrityksen nykyisen kuituverkon kattavuusalueen. Tämän opinnäytetyön tarkoitus on testata mahdollisuutta sovellukselle, joka pystyy lukemaan reittien sijainnit Extensible Markup Langue (XML)-tiedostosta ja päivittämään reitit kartalle. Tarkoituksena on voida jatkossa päivittää pelkästään XML-tiedostoa, ja reittien päivitys tapahtuu ilman ylimääräistä työtä. Tarkoituksena ei ole luoda asiakkaan käyttöön toimintoja, kartan käsittelyä lukuun ottamatta.

Aikaisempi kokemukseni Javascript- tai verkkokehityksestä on hyvin vähäinen. Koska nykyään erityisesti Javascript on valtavassa suosiossa ja kokonaisia ohjelmistopinoja voidaan luoda Javascriptillä, on se varsin mielenkiintoinen aihealue opinnäytetyöksi. Myös JavaScriptin syntaktinen yhtäläisyys C ja C++-kieliin saa sen tuntumaan tutummalta.

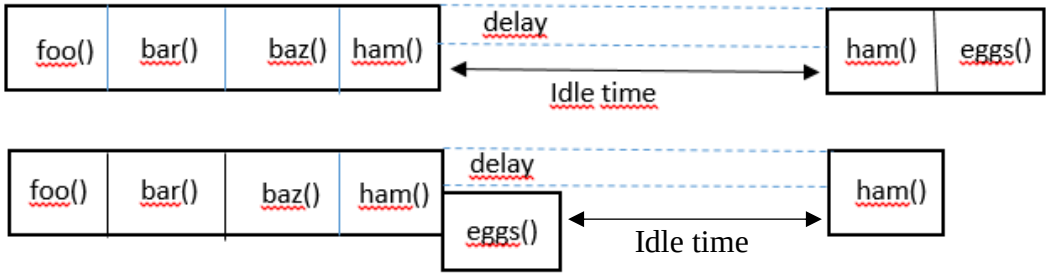
Ensimmäisessä osassa paneudutaan Javascriptin toimintaan ja sen tuomiin mahdollisuuksiin. Tarkoitukseni on kasata teknistä teoriaa Javascriptin toiminnasta, sen vahvuuksista ja heikkouksista. Opinnäytetyössä käydään lyhyesti läpi myös muita verkkokehityksen teknologioita, joista voi olla hyötyä opinnäytetyön kannalta. Myös Openlayers-karttakirjaston teoria on osana opinnäytetyötä, sillä se vaikuttaa olevan suureessa suosiossa maailmalla. Toisessa osassa on tarkoitus toteuttaa koko ohjelmapino, hyödyntäen tutkittuja teknologioita.

2 JAVASCRIPT

Javascript on pääasiassa verkkoympäristön kehitykseen kehitetty komentosarjakieli. Se on alustariippumaton, joten lähdekoodia voidaan muokkaamatta suorittaa jokaisessa Javascript-tuetussa selaimessa. Javascriptin suoritus tapahtuu joko tulkattuna tai nykyään yleisemmin ”Just In Time”-käännettynä (JIT), mikä kuitenkin aina vaatii suoritukseen isäntäympäristön, esimerkiksi verkkoselaimen. Alun perin Javascriptiä käytettiin luomaan vuorovaikutteisia verkkosivuja mutta viime vuosina Javascriptin käyttö myös palvelinpuolen ohjelmissa javerkkoympäristön ulkopuolella, kuten mobiiliapplikaatioissa, on ollut mahdollista Node.js-ajoympäristön ja React-ohjelmistokehyksen ansiosta. (Mozilla 2019a.)

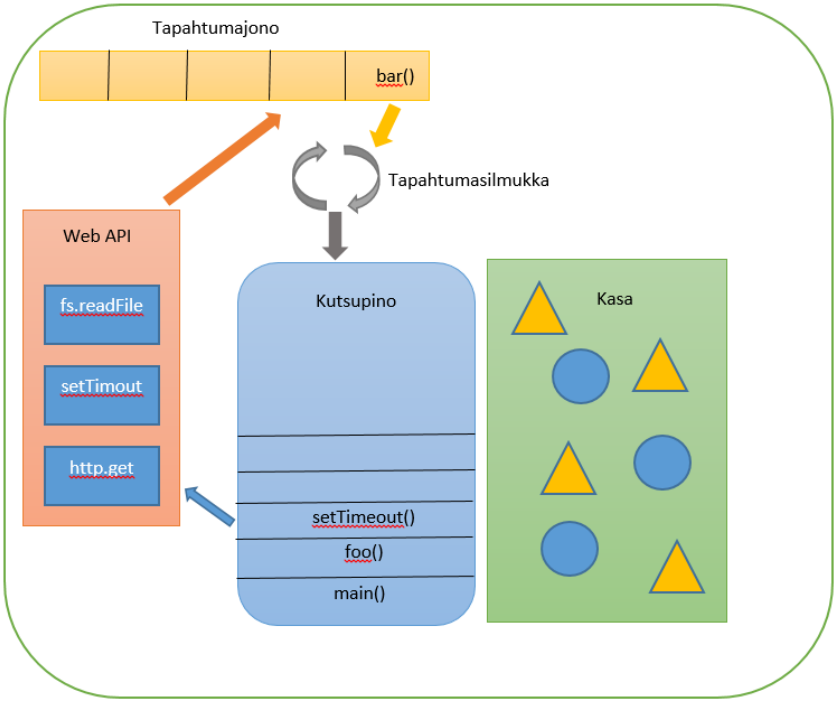
Javascript on heikosti tyypitetty kieli, joten siinä ei voida määrittää muuttujien tyyppejä. Muuttujat esitellään käyttämällä var-, let- tai const- avainsanaa ja kääntäjä valitsee sopivan tyypin. Var-avainsanalla muuttujan esittely funktion näkyvyysalueen ulkopuolella luo globaalin muuttujan, jolloin muuttujaa voidaan käyttää toisessa funktiossa vahingossa tai tarkoituksellisesti, ja muuttujaa voidaan myös käyttää ennen määritystä. Tämä voidaan estää käyttämällä Let-avainsanaa, jolloin muuttujan näkyvyys rajautuu vain näkyvyysalueeseen, jossa se on luotu ja muuttujaa voidaan käyttää vain määrittelyn jälkeen. Const-avainsanalla luodaan vain luku-muuttujia, joita ei siis voida muokata ohjelman suorituksen aikana. Javascript käyttää Javasta ja C-kielestä tuttuja toiminnankulkulausekkeita, kuten ”if...else” ja ”for”. (Mozilla 2019b; Peltomäki 2017, luku ”Tietotyypit”; Shute 2019, luku 1.)

Javascript on yksisäikeinen, asynkroninen ja tapahtumapohjainen kieli. Yksisäikeisyys rajoittaa ohjelman suorittamista yhteen prosessiin kerrallaan, ja tämän seurauksesta ohjelma pysähtyisi täysin jokaisen funktion suorittamisen ajaksi. Javascript toteuttaa saumattoman ohjelmankulun asynkronisuudella. Kuvasta 1 nähdään, kuinka asynkronisella suorittamisella voidaan luoda näennäinen rinnakkaisajo, vaikka todellisuudessa Javascript-moottori priorisoi funktioita niiden määrittelyjen mukaisesti ja suorittaa ne yksitellen. Jos moottori havaitsee asynkronisen funktion, siirretään se kasaan odottamaan valmistumista ja suoritetaan seuraava funktio kutsupinosta. Näin saadaan vähennettyä prosessorin joutokäyntiaikaa ja luotua käyttäjän kokema saumaton ohjelman suoritus. Asynkronisuutta käytetään erityisesti käsiteltäessä dataa kovalevyllä ja verkon rajapintakutsuissa, näiden hitauden vuoksi. (Shute 2019, luku 2.)



KUVA 1. Asynkronisen ja synkronisen ajon kaaviot (mukaillen Shute 2019, luku 2)

JavaScript-moottori käyttää kahta muistiosiota, kutsupinoa ja kasaa. Tämän lisäksi vielä isäntäympäristö mahdollistaa tapahtumajonon käytön. Kutsupinossa säilötään aktiivisten funktioiden ikkunat. Se on järjestelmällinen osio, joka toteuttaa Last In First Out-periaatetta, eli ensiksi muistiin lisätty funktio poistuu vasta, kun sitä seuraavat funktiot ovat poistuneet. Kasa on jäsentelemätön muistiosio, johon sijoitettuihin funktioihin viitataan muistiosoitteen perusteella. Kasaan sijoitetaan asynkroniset funktiot suorituksen ajaksi. Tapahtumajono toimii First In First Out-periaatteella, ja sinne siirretään valmiit asynkroniset funktiot odottamaan kutsupinon tyhjenemistä, jolloin tapahtumajonosta haetaan ensimmäiseksi valmistunut funktio. Tapahtumasilmukka tarkkailee kutsupinon tyhjenemistä ja tapahtumajonoon saapuvia valmistuneita funktiota ja kutsupinon ollessa tyhjänä kutsuu asynkronisen funktion takaisinkutsu-funktiota, joka lisätään kutsupinoon suoritettavaksi. (Roberts, 2014; Shute 2019, luku 2)



KUVA 2. Suorituksen tärkeimmät komponentit (mukaillen Roberts, 2014)

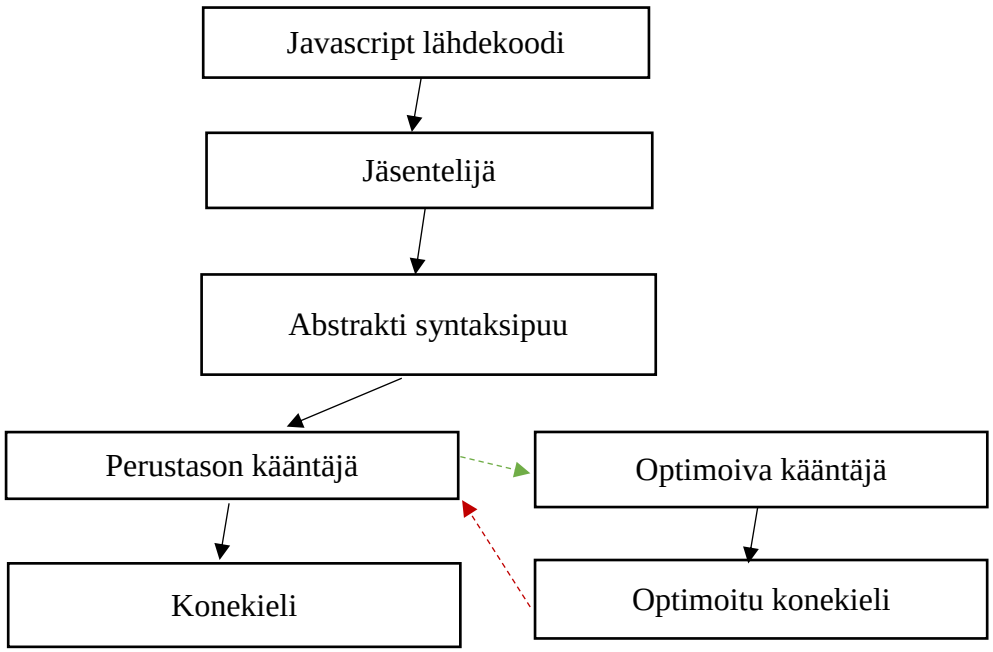
2.1 Javascript-moottori

Moottori on JavaScript-kielen tulkki, joka tulkkaa kielen konekielelle. Jokaisella verkkoselaimella on oma moottorinsa, ja jokainen moottori toteuttaa koodin tulkkauksen omalla tavallaan. (Hinkelmann JSConf EU 2017). Muutamia tunnettuja moottoreita ovat

- V8 (Google Chrome, node.js)
- SpiderMonkey (Mozilla Firefox)
- Chakra (Microsoft Edge)
- JavaScriptCore (Apple Safari)

Kaikki edellä listatut moottorit on kirjoitettu käyttäen C++-kieltä. (Chakra repository 2020; JavaScriptCore repository 2020; SpiderMonkey repository 2020; V8 repository 2020).

Kuvio 1 esittää, kuinka JavaScript-koodi käännetään konekielelle. Moottori aloittaa kääntämisen jäsentelemällä koodia rivi kerrallaan abstraktiksi syntaksipuuksi. Perustason kääntäjä kääntää syntaksipuusta konekieleksi, ja jos objektia käytetään usein, siirretään se optimoivan kääntäjän käännettäväksi. Jos taas objektin tyyppejä muutetaan, joutuu perustason kääntäjä peruuttamaan optimoinnin ja luomaan perustason konekieltä. Tämän takia on suorituksen kannalta edullista käyttää objekteja aina samoilla tietotyypeillä ja määrittää käyttämättömät tietotyypit undefined-arvoon, jolloin voidaan välttää matalan tason turhat hypyt vertailukomentojen aikana. (Hinkelmann, 2017; Hölttä, 2017.)



KUVIO 1. JavaScript-moottorin toimintakaavio (mukaillen Hinkelmann, 2017)

Moottori on osa isäntäympäristöä ja hallitsee ainoastaan kutsupinoa sekä kasaa. Myös muut komponentit, kuten tapahtumajono ja – käsittelijä, Document Object Model (DOM) ja ohjelmointirajapinnat ovat osa isäntäympäristöä. Yleisimmin isäntäympäristönä toimii selain, joka tarjoaa useita rajapintakirjastoja, muun muassa DOM- ja tiedostonkäsittelykirjastot. (Mozilla 2020c; Mozilla 2020d; Patel 2018.)

2.2 Node.JS

Node.js on alustariippumaton isäntäympäristö, joka on kehitetty suorittamaan Javascriptiä selaimen ulkopuolella, ja sen yleisin käyttökohde ovatkin palvelinohjelmat. Node on lähes täysin asynkroniseksi luotu isäntäympäristö, eikä se näin ollen käytä useaa prosessia suorituksen aikana. Node mahdollista koko ohjelmapinon ohjelmoinnin Javascriptillä. (Nodejs 2020a; Nodejs 2020b.)

Node rakentuu V8-kääntäjästä, tapahtumajonosta, tapahtumasilmukasta ja joukosta rajapintoja, jotka Node.js:än tapauksessa ovat Npm-moduuleja. Npm-moduulien käyttö poikkeaa hieman tavallisista Javascript-kirjastoista, sillä ne asennetaan osaksi projektia Npm-paketinhallintatyökalulla. Myös moduulien tuonti Javascript-tiedostoon poikkeaa siten, että ne tuodaan komennolla require, kun taas normaalisti Javascript-kirjastot tuodaan import-komennolla. Node.js poikkeaa selaimista sillä, että se tarjoaa mahdollisuuden matalan tason rajapintakutsuihin, kuten muistinallokointi-funktiot. (Nodejs 2020a; Nodejs 2020b; Nodejs 2020c.)

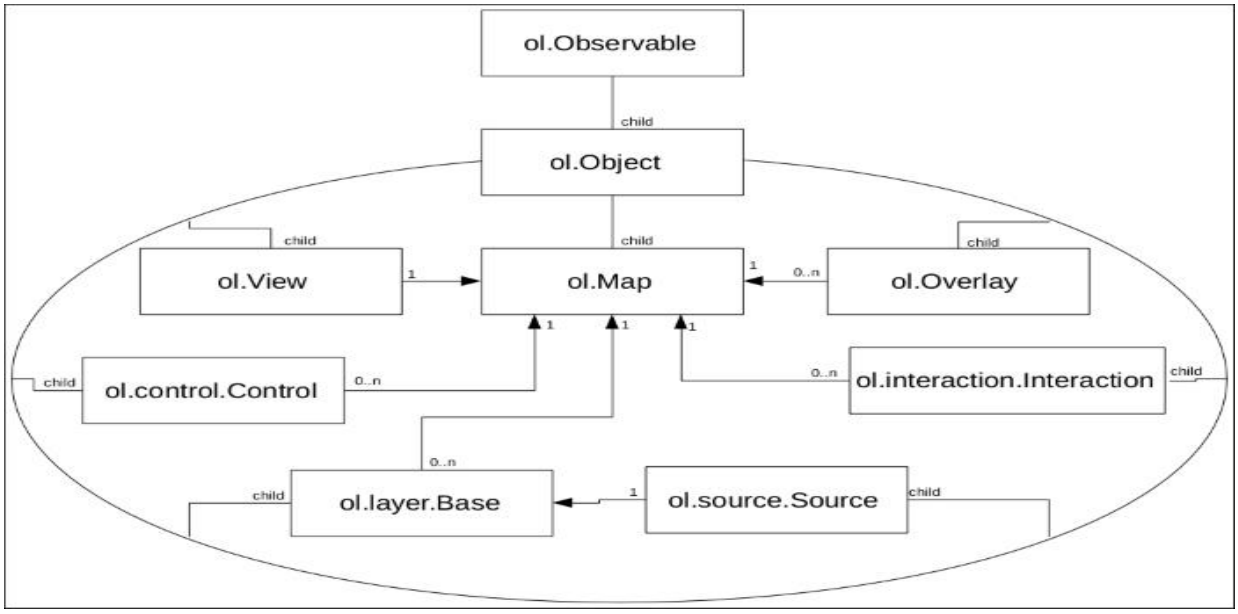
Node.js:än käyttö palvelimena vaatii jonkinlaisen verkkokehyksen, joka toimii reitittimenä tuleville ja lähteville pyynnöille. ExpressJS on reititys- ja väliohjelmistokehys, joka kuuntelee tulevia HyperText Transfer Protocol (HTTP)-pyyntöjä ja käsittelee ja ohjaa ne päätepisteille. (Expressjs 2020a; Expressjs 2020b.)

3 OPENLAYERS

Openlayers on JavaScript karttakirjasto, joka toimii loppukäyttäjän selaimessa. Openlayersilla luodut sovellukset toimivat lähes kaikissa selaimissa, myös mobiililaitteilla. Kirjasto on julkaistu Berkley Software Distribution (BSD) 2-ehdollisella lisenssillä, joten sen koodi on avoimesti nähtävissä ja muokattavissa ja ilmainen. Kirjasto ei myöskään ole sidoksissa yksityisiin kolmannen osapuolen kirjastoihin. (Gratier, Spencer & Hazzard 2015, luku 1.)

3.1 Luokat

Vaikka Javascriptissä ei varsinaisesti ole luokkiin perustuvaa periytymistä, käyttää Openlayers luokkia kuvan 3 mukaisesti, selkeyttämään lähdekoodin rakennetta. Luokilla ei ole toiminnallisesti suurta vaikutusta koodiin, sillä luokat ovat ainoastaan erikoisfunktioita. Luokka tulee kuitenkin esitellä ennen sen käyttöä, toisin kuin funktio, joka voidaan esitellä missä vaiheessa koodia tahansa. (Mozilla 2019e.)



KUVA 3. Openlayers-luokkien periytymiskaavio (Gratier, Spencer & Hazzard) ©Packt Publishing

Openlayersin pääluokkana toimii Observable-luokka, josta periytyy Object-luokka. Observable-luokkaa ei useimmiten käytetä itsessään, vaan lähes aina jonkin periytetyn luokan kautta. Observable-luokasta periytyy myös useita hyödyllisiä funktioita, kuten on-funktio jolla, voidaan suorittaa toimintoja esimerkiksi sivun latautumisen jälkeen. Object-luokka perii kaikki Observable-luokan metodit ja lisäksi tarjoaa joukon metodeita, joilla voidaan hallita periytyvien luokkien tietoja. Näitä ovat esimerkiksi get- ja set-

funktiot, joilla voidaan hakea tai asettaa arvoja. Myöskään Object-luokkaa ei usein käytetä suoraan. (Gratier ym. 2015, luku 2; Openlayers 2019a.)

Map-luokka on keskeisin käyttäjän vastuulla oleva luokka, ja se on pakollinen jokaisessa sovelluksessa. Map-luokalla on joko suora tai epäsuora suhde kaikkiin muihin periytettyihin instansseihin, ja ne yleensä annetaankin parametreina Map-objektin luonnissa. Map-luokalla ei ole omia metodeja, vaan kaikki metodit ovat perittyjä PluggableMap-luokalta ja ne ovat pääasiassa muiden luokkien lisäys- tai hakumetodeja. Map-luokan näkymiseen verkkosivuilla vaaditaan View- ja Layer-objektit sekä target-parametri, jolle annetaan arvoksi jokin HTML-elementti, johon kartta halutaan lisätä. Jollei muita instansseja määritetä parametreina Map-luokalle, käytetään oletusarvoja kyseisille instansseille. Map-luokka ei itse hallitse näkymää, vaan se jää View-objektin vastuulle. (Gratier ym. 2015, luku 2; Openlayers 2019b.)

View-objekti vastaa kartan näkymästä. Sen tärkeimpiä ominaisuuksia ovat muun muassa tarkennus, keskitys, resoluutio, kierto ja projektio. Sen vastuulla on myös määrittää, mihin käyttäjä katsoo kartalla, muttei mitä kyseisessä sijainnissa näkyy. Näkymästä vastaa Layers-objekti, joka voi esittää sekä bittikarttagrafiikkaa että vektorigrafiikkaa. Bittikarttagrafiikkaa yleensä käytetään, kun halutaan esittää kartta. Vektorigrafiikalla piirretään kuviot kartan päälle. Myös Overlay-objektilla voidaan luoda kartalla näkyviä esineitä, Overlaylla luodut esineet ovat HTML-elementtejä ja niitä voidaan käyttää esimerkiksi aukeavien info-ikkunoiden luontiin. (Gratier ym. 2015, luku 2; Openlayers 2020c.)

Kartan käyttäjä voi liikuttaa, tarkentaa tai pyörittää karttaa kahden luokan avulla. Ensimmäinen on Control-objekti, jonka avulla saadaan luotua esimerkiksi navigointi- ja tarkennusnäppäimet kartan reunaan, ja niistä käyttäjä voi ohjata kartan näkymää (Openlayers 2020d). Toinen on Interaction-objekti, jolla voidaan luoda hiiren, kosketuksen tai näppäimistön syötteeseen reagoivia navigointitapahtumia (Openlayers 2020e.)

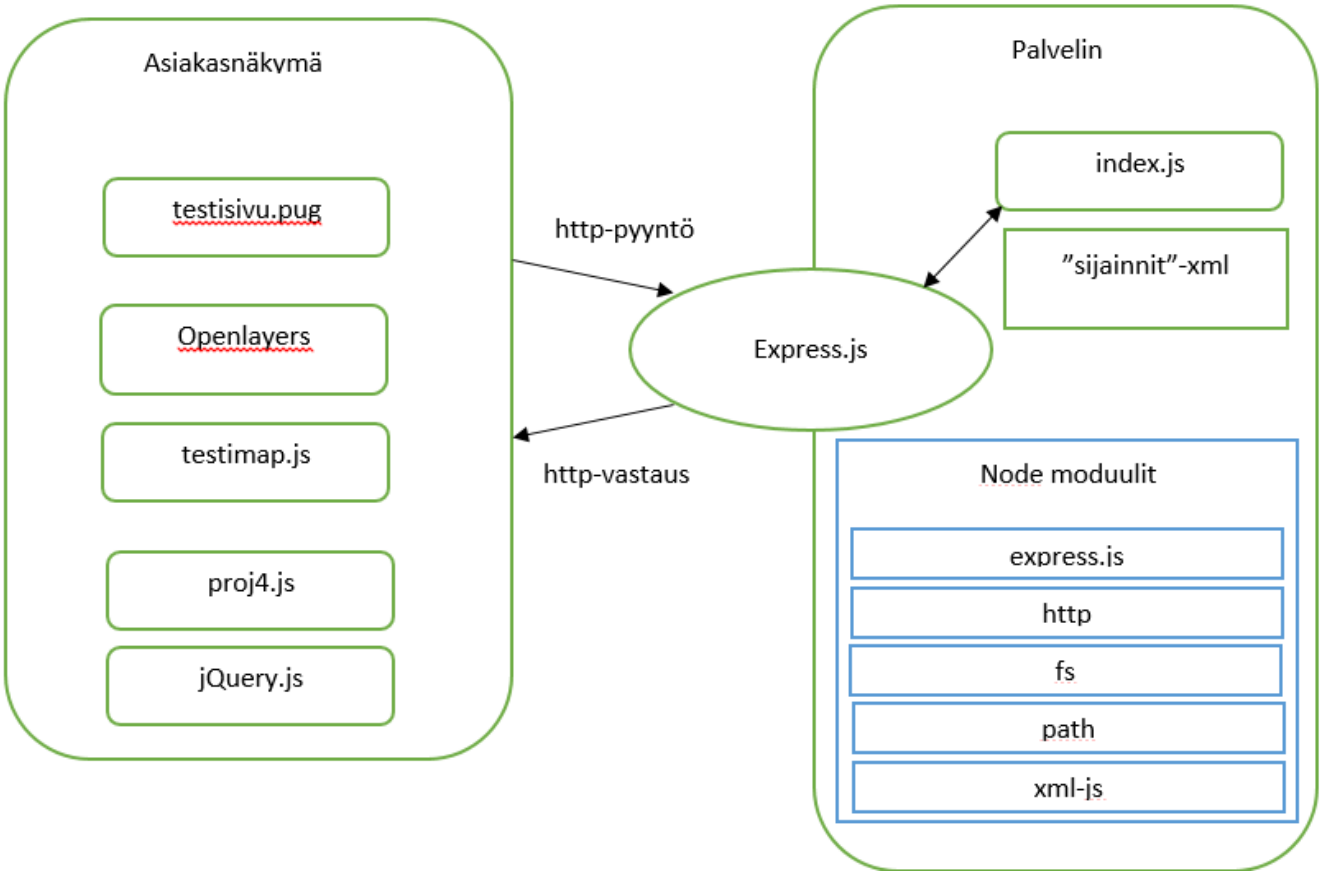
3.3 Kartan piirto

Openlayersissa on kolme tapaa renderöidä kuvaa. Oletuksena on aktiivinen kanvaasirenderöinti, jossa hyödynnetään HTML5:n tukemaa kanvaasia eli eräänlaista piirtoalustaa. Kanvaasiin on mahdollista luoda kuvioita ainoastaan Javascriptillä. Kanvaasin etuja ovat suorituskykyinen renderöinti ja tuki kaikilla tunnetuimmilla verkkoselaimilla. (Gratier ym. 2015, luku 3; W3School 2020a). DOM-renderöinnillä on mahdollista käyttää HTML-elementtejä piirron kohteena, kuten img- ja div-elementtejä. Sen etu

on sen tuki kanvaasirenderöintiä vanhemmille verkkoselaimille, mutta se kärsii ominaisuuksien puutteesta ja heikommasta suorituskyvystä. (Gratier ym. 2015, luku 3). Kolmas vaihtoehto, WebGL on näistä uusin, ja se mahdollistaa kolmiulotteisen kuvan piirron. Sen etuina ovat hyvä suorituskyky ja laaja ominaisuuksien tarjonta. Myös tuki kaikille uusimmille selaimille on, mukaan lukien Internet explorer 11. Sen huono puoli on tarve raudalle, pääasiassa näytönohjaimelle, joka tukee WebGL:ää. (Gratier ym. 2015, luku 3; Mozilla 2020f.)

4 TOTEUTUS

Opinnäytetyön verkkosovellus on demo, jossa hahmotellaan sen sopivuutta asiakkaan verkkosivustolle. Toteutukseen kuuluu verkkosivun, palvelimen ja reitityksen luonti. Verkkosovellus piirtää kartan ja sen päälle piirtokerrokset näyttämään kuituverkon sijainnit Kokkolan alueella. Sivujen luontiin käytän Pug-mallipohjamoottoria puhtaasti siistimmän syntaksin takia. Palvelin luodaan Node.js-ympäristöön REST-tyyliin, hyödyntäen apuna Express.js-moduulia. Yhteyspyynnöt ohjataan päätepisteille Index.js-ohjelman kautta. Koordinaatit kaapeleille luetaan XML-tiedostosta ja Xml-Js-moduulin avulla muutetaan JavaScript Object Notation (JSON)-muotoon. Kuvio 2 näyttää komponentit ja niiden sijainnit ohjelmistopinossa.



KUVIO 2. Ohjelmistopinon arkkitehtuuri

Sovelluksen toimintamalli on yksinkertainen, kaikki tapahtuu sivun latautumisen aikana. Käyttäjä voi ainoastaan tarkkailla kartalla näytettäviä asioita ja ohjata kartan näkymää. Käyttäjä lähettää HTTP GET -pyynnön palvelimelle, jossa Express-väliohjelma kuuntelee portille saapuvia pyyntöjä ja jos pyynnön

päätepiste on määritetty, se ohjaa käyttäjän kyseiselle sivulle. Kun käyttäjä on pyytänyt pääsyä testisivu.pug-tiedostoon, renderöidään testisivu, ja palvelin aloittaa XML-tiedoston muuntamisen JSON-muotoon. Kun käyttäjä on ladannut sivun, lähetetään uusi get-pyyntö, joka aloittaa palvelimella JSON-objektin iteroinnin. Kun kaikki sijaintitiedot on kerätty taulukkoon, palautetaan taulukko käyttäjälle. Seuraavaksi aloitetaan kartan määritys ja piirto, jonka jälkeen määritetään reittikerros ja piirretään kerrokset kartalle.

4.1 Back-end

Back-endissä on käytössä kaksi Express-reititystä, sivun haku ja sijaintien haku. Palvelimella myös suoritetaan sijaintien iterointi. Palvelimelle otetaan käyttöön Node.js-mooduulit require-funktiolla kuvan 4 tavalla.

```
3  var express = require('express'),
4      app = express();
5
6  var http = require('http');
7  var fs = require('fs');
8  var path = require('path');
9  var xmljs = require('xml-js');
```

KUVA 4. Moduulien tuonti projektiin

Express-moduuli tuodaan ensin, jonka jälkeen luodaan Express-objekti nimeltä app. Tämän lisäksi tuodaan http-moduuli, jotta voidaan luoda HTTP-palvelin. Fs-moduulilla voidaan käsitellä paikallisia tiedostoja. Path-moduulilla voidaan käsitellä tiedostopolkuja. Xml-js-moduuli mahdollistaa XML-tiedoston muunnon JSON-merkkijonoksi.

Kuvan 6 ensimmäisellä rivillä luodaan HTTP-palvelin, http-moduulin createServer-funktiolla, jolle annetaan parametriksi app-objekti. Tämän seurauksena app-objekti toimii tapahtumantarkkailijana palvelimella. Kahdella viimeisellä rivillä asetetaan HTTP-palvelin kuuntelemaan porttia listen-funktiolla. Listen-funktiolle annetaan parametriksi mikä tahansa vapaana oleva portti kuunneltavaksi kuvan 5 osoittamalla tavalla.

```
1  var httpServer = http.createServer(app);
2  var port = 5025;
3  httpServer.listen(port);
```

KUVA 5. Palvelimen luonti ja portin kuuntelun asetus

Koska käytössä on Pug-mallipohja, tarvitsee Expressille osoittaa käyttöön Pug-näkymämoottori set-funktiolla kuvan 6 ensimmäisen rivin tapaan. Funktiolle annetaan parametreiksi kohteen nimi, jota halutaan muokata, ja arvo, johon se halutaan asettaa. Lisäksi määritetään Express käyttämään juurikansiona staattista polkua käyttämällä use-funktiota, kuten kuvan toisella rivillä. Parametriksi annetaan haluttu polku.

```
16  app.set('view engine', 'pug');
17  app.use(express.static(__dirname + '/styles'));
```

KUVA 6. Express-objektin määritykset

Expressin reititykseen tarvitaan funktiota get, joka vastaa käyttäjän lähettämään get-pyyntöön. Funktiolle annetaan parametreiksi haettava resurssi ja takaisinkutsufunktio, jossa suoritetaan tarvittavat toimet pyynnön vastaukseen. Takaisinkutsufunktiolle annetaan parametreiksi req ja res, jotka ovat request- ja response-objektit. Nämä objektit sisältävät tarpeelliset ominaisuudet, kuten tiedostomuoto tiedostoille, joita palvelimelta halutaan hakea. Kuvan 7 pyyntö vastaa testisivu-resurssin kyselyyn ja kutsuu takaisinkutsufunktiota, eli response-objektin funktiolla render renderöidään HTML-sivu Pug-mallipohjasta. Tämän jälkeen luodaan rivillä 28 tapahtumantarkkailija, joka tarkkailee pyynnön valmistumista ja aktivoituu, kun pyyntö on valmis. Takaisinkutsufunktio määrittää XML-tiedoston osaksi polkua join-funktiolla, jolle annetaan parametriksi alkuperäinen polku ja liitettävä tiedosto. Seuraavaksi kutsutaan fs-moduulin readFile-funktiota, jolle annetaan parametreiksi tiedostopolku, merkistöstandardi ja takaisinkutsufunktio. Takaisinkutsufunktiolle annetaan parametreiksi err ja data, joista err sisältää virhekoodit ja data tiedostosta luettavan sisällön. Yksinkertainen virheentarkistus suoritetaan rivillä 32, ja jos virhekoodi ei ole 0, heitetään virheilmoitus ja funktion suoritus lopetetaan. Jos virhekoodi on 0, luetaan data tiedostosta ja muunnetaan se JSON-muotoon.

```
24  app.get('/testisivu', function(req,res, next){
25      var ip = req.connection.remoteAddress;
26      console.log(ip+ " requested Get page");
27      res.render('testisivu');
28      req.on('end', function () {
29          var filePath = path.join(__dirname, 'verkkogml_export.xml');
30
31          fs.readFile(filePath, 'utf8', function(err, data){
32              if (err) {throw err;}
33              jsonStr = xmljs.xml2json(data);
34          });
35      });
36  });
```

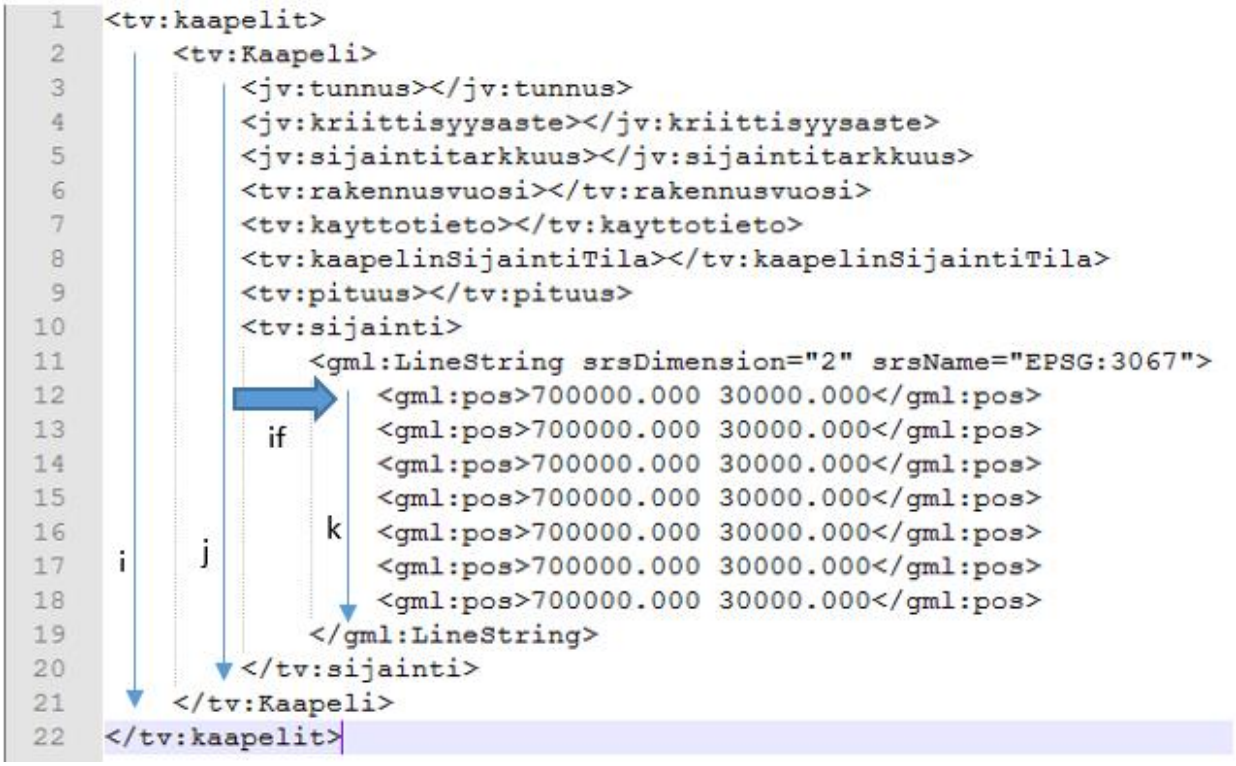
KUVA 7. HTTP GET testisivu-kyselyn toiminnot

Kun käyttäjä on ladannut aloitussivun, lähetetään käyttäjältä uusi kysely, jolla haetaan sijaintitaulukon sisältävä JSON-objekti. Kuvassa 8 näkyy sijaintien iterointifunktion kutsu, jonka jälkeen lähetetään käyttäjälle vastaus JSON-muodossa. Sijaintien saaminen vaatii uuden pyynnön, sillä ensimmäinen get testisivu-pyynnön sisältö on muotoa text/html, kun taas get getGml-pyynnön sisältö on muotoa application/json. Tämän seurauksena olisi ristiriita get-pyyntöön palvelimelta lähtevän vastausviestin sisältötyypissä.

```
38  app.get('/getGml', function(req,res){
39      var locationData;
40      locationData = iterateLocations(jsonStr);
41      res.json(locationData);
42  });
```

KUVA 8. HTTP GET getGml-kyselyn toiminnot

Sijaintietojen iterointi nykyisestä XML-tiedostosta on työläs prosessi, sillä tiedosto sisältää kartan kannalta paljon tarpeetonta tietoa. Iterointia hankaloittaa se, ettei jokainen kaapelit-elementti sisällä samaa määrää elementtejä ja että ylimääräinen elementti lisätään listan keskelle. Tämän takia sijainti-elementti ei ole aina samassa indeksissä. Kuvassa 9 on esimerkki XML-tiedoston rakenteesta ja iterointitarpeesta.



KUVA 9. XML-tiedoston rakenne ja iteroinnin tarve

Koska Kaapeli-elementtejä on useita ja mahdollisesti tulossa vielä useampi lisää, ei voida asettaa minkäänlaisia arvoja rajaamaan kaapeleiden määrää, vaan ne joudutaan dynaamisesti iteroimaan. Tähän käytetään for-silmukkaa ja muuttujaa i kuvaamaan kaapeli-elementin indeksiä. Kaapelin tv:sijainti-elementti joudutaan myös hakemaan iteroimalla sen yksilökohtaisen indeksin takia. Tämä toteutetaan sisennetyllä for-silmukalla ja se toistetaan jokaiselle kaapeli-elementille. Sen muuttujana toimii j kuvaamaan kaapelin elementtien indeksiä. Kun tv:sijainti-elementti löytyy, hypätään jälleen sisennettyyn for-silmukkaan, jossa iteroidaan kaikki sijaintitiedot. Apuna käytetään muuttujaa k kuvaamaan positio-elementtien indeksiä.

Kuva 10 näyttää iterointifunktion toteutuksen. Funktio ottaa parametriksi JSON-merkkijonon, joka muutetaan JSON.parse-funktiolla objektiksi. Rivillä 46 rajataan objekti pelkästään tv:kaapelit-elementtiin, sillä samalla tasolla on useita tarpeettomia elementtejä. Rivillä 48 aloitetaan tv:Kaapeli-elementtien iterointi esittelemällä muuttuja i for-silmukan sisällä ja määrittämällä i:n raja-arvo pienemmäksi kuin tv:kaapelit-elementin pituus. Seuraavaksi jokaisella ”i”:n arvolla suoritetaan sisennetty for-silmukka, jossa esitellään muuttuja j, jonka raja-arvoksi määritetään pienemmäksi kuin tv:Kaapeli-elementin pituus. Rivillä 50 suoritetaan jokaiselle j:n arvolle if-vertailu, ja jos elementin nimi on tv:sijainti, aloitetaan viimeinen sisennetty for-silmukka muuttujalla k, jonka raja-arvoksi määritetään pienempi kuin tv:sijainti-elementin pituus. Rivillä 53 haetaan jokaisen positio-elementin arvo. Koska arvo on merkkijono,

jossa on sekä leveys- että pituusaste ja koska Openlayers käyttää päinvastaista järjestystä, joudutaan merkkijono pilkkomaan ja järjestämään uusiksi. strArr.split-funktiolla voidaan pilkkoa merkkijono, parametriksi se ottaa merkin, josta jono katkaistaan, tässä tapauksessa välilyönti. Sen jälkeen riveillä 55-57 järjestetään arvot uudelleen taulukkoon ja muutetaan ne float-muotoon. Rivillä 60 työnnetään uusi lineArray-taulukko osaksi array-taulukkoa, joka funktion lopuksi palautetaan kutsujalle.

```
44  var iterateLocations = function(LocationsJSON) {
45      var parsedLocations = JSON.parse(LocationsJSON);
46      var json = parsedLocations.elements[0].elements[8];
47      var array = [];
48      for(var i = 0; i < json.elements.length; i++){
49          for(var j = 0; j < json.elements[i].elements.length; j++){
50              if(json.elements[i].elements[j].name == "tv:sijainti"){
51                  var lineArray = [];
52                  for(var k = 0; k < json.elements[i].elements[j].elements[0].elements.length; k++){
53                      var strArr = json.elements[i].elements[j].elements[0].elements[k].elements[0].text;
54                      var pos = strArr.split(' ');
55                      var swapArr = parseFloat(pos[0]);
56                          pos[0] = parseFloat(pos[1]);
57                          pos[1] = swapArr;
58                          lineArray.push(pos);
59                  }
60                  array.push(lineArray);
61              }
62          }
63      }
64      return array;
65  }
```

KUVA 10. Iterointifunktio

4.2 Front-end

Verkkosivu, joka renderöidään käyttäjälle, on hyvin yksinkertainen. Siinä on HTML-sivulle välttämättömät elementit, riippuvuudet ja div-elementti. Div-elementtiin upotetaan kanvaasi, johon Openlayers piirtää kartan ja reitit. Kuva 11:sta näkyy Pug-muodon kirjoitustapa, elementeissä ei käytetä HTML:stä tuttuja symboleita, eikä elementtejä tarvitse sulkea. Siinä myös sisennyksillä on merkitystä, sillä sisennetty elementti kuuluu osaksi ulompaa elementtiä.

```
1  html
2
3      head
4          base(href="/")
5          link(rel="stylesheet", href="/bootstrap/css/bootstrap.css" type="text/css")
6          link(rel="stylesheet", href="/keskikaista.css" type="text/css")
7          link(rel="stylesheet", href="/javascript/openlayers/ol.css" type="text/css")
8
9
10         script(src="/javascript/jquery-3.2.1.min.js")
11         script(src="/javascript/openlayers/ol.js")
12         script(src="/javascript/proj4.js")
13         script(src="/javascript/testimap.js")
14
15         meta(name="viewport", content="user-scalable=yes, width=device-width, initial-scale=1.0")
16         title Testisivu
17
18     body(id="testibody")
19
20     div(id="testiTausta", class="map")
```

KUVA 11. Verkkosivun Pug-mallipohja

Testimap.js-tiedostossa luodaan Openlayers-objektit ja funktiot ja käsitellään asiakaspuolen tapahtumatarkkailijan funktiot. Kuva 12 kuvaa jQuery-funktiota, jonka ready-tapahtuma aktivoituu, kun testisivun lataus on valmis. Sen seurauksena lähetetään getJson-pyyntö palvelimelle ja takaisinkutsufunktiossa parametrina on palvelimelta palautunut sijaintiobjekti. Takaisinkutsufunktiossa kutsutaan myös drawMap ja drawCables-funktioita. DrawMap-funktio on parametriton ja drawCables-funktio vaatii parametriksi palautettua sijaintiobjektia. JSON-merkkijonoa ei tarvitse erikseen muuntaa objektiksi, sillä getJSON-funktio tekee sen automaattisesti.

```
4   //Get request for JSON on page readystate, parse string to javascript object and call draw functions.
5   $('testisivu').ready(function(){
6       $.getJSON('/getGML', function(gmlLocations){
7           drawMap();
8           drawCables(gmlLocations);
9       });
10  });
```

KUVA 12. jQuery-tapahtumakäsittelijät

DrawMap-funktiossa luodaan karttakerros, näkymä sekä haetaan karttalähde. Kuvassa 13, rivillä 15 määritetään kartan projektio. Alkuperäinen projektio on Web Mercator-projektio EPSG:3587, mutta XML-tiedoston koordinaatit ovat TM35FIN-projektio EPSG:3067. Riveillä 17–20 luodaan kerrosobjekti ol.layer.Tile, jonka ominaisuudelle annetaan arvoksi OpenStreetMap-kartan lähde ol.sourceOSM. Riveillä 22–26 luodaan näkymäobjekti, jonka ominaisuudet zoom, center ja projection määritellään niin, että oletusnäkymä osoittaa Kokkolan alueelle. Riveillä 28–34 luodaan ja määritetään karttaobjekti ol.Map, sen lisäyskohde on Pug-tiedoston testiTausta-elementti. Kerrosominaisuuteen lisätään aiemmin ylempänä luotu karttakerrosobjekti ja näkymäksi näkymäobjekti. Projektioksi määritetään ylempänä lisätty proj4-määritys.

```
13  var drawMap = function (){
14
15  proj4.defs('EPSG:3067', '+proj=utm +zone=35 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs');
16
17      var mapLayer = new ol.layer.Tile({
18          source: new ol.source.OSM({
19          })
20      });
21
22      var view = new ol.View({
23          zoom: 13,
24          center: [ 324829.387, 7092957.03 ],
25          projection: 'EPSG:3067'
26      });
27
28      map = new ol.Map({
29          target: 'testiTausta',
30          layers: [mapLayer],
31          view: view,
32          projection: 'EPSG:3067'
33      });
34  }
```

KUVA 13. DrawMap-funktio

Kuvassa 14 esitetty drawCables-funktio ottaa vastaan sijaintiobjektin parametrina. Rivillä 46 luodaan vektoriobjekti ol.source.Vector. Riveillä 47–53 iteroidaan sijaintiobjekti ja jokaisesta sijainnista luodaan piste, joka lisätään luotuun ominaisuusobjektiin ol.Feature. Objektille määritetään ominaisuudet id ja geometry. Id on tunniste, jolle luodaan nimeksi indeksin arvolla varustettu cable. Geometry on geometriaominaisuus, jolle luodaan arvoksi uusi objekti ol.geom.LineString jonka parametrina on piste. Tämän jälkeen kutsutaan lisäysfunktiota addFeature, jonka parametrina on ominaisuusobjekti. Tämä for-silmukka lisää kaikki pisteet cableSource-objektiin. Riveillä 59–68 luodaan vektorikerrosobjekti, jonka source- ominaisuuden arvo määritetään cableSource-objektiksi ja tyyliominaisuus style, jolle luodaan tyyliobjekti ol.style.Style. Tyyliobjektin ominaisuudeksi luodaan ol.style.Stroke-objekti, jossa määritetään piirretyn viivan tyyli. Lopuksi lisätään kerros karttaobjektiin. Kuvassa 15 näkyy lopullinen kartta reitteineen verkkoselaimessa.

```
var drawCables = function(gmlLocations){

    //Cables Source
    cablesSource = new ol.source.Vector({});
    for(var i = 0; i < gmlLocations.length; i++){
        var points = gmlLocations[i];

        var cableFeatures = new ol.Feature({
            id: "cable" + i,
            geometry: new ol.geom.LineString(points)
        });

        cablesSource.addFeature(cableFeatures);
    };

    //Cables Layer
    var cablesLayer = new ol.layer.Vector({
        source: cablesSource,
        style: new ol.style.Style({
            stroke: new ol.style.Stroke({
                color: '#00AAFF',//'#00FF45',
                width: 7
            })
        })
    });
    map.addLayer(cablesLayer);
}
```

KUVA 14. Piirtokerroksen määrittely

KUVA 15. Karttasovellus verkkoselaimessa

5 POHDINTA

Vaikka lähes kaikki opinnäytetyössä vaadittavat teknologiat olivat minulle tuntemattomia, omasta mielestäni sovelluksen toteutus sujui hyvin ja oli varsin mielenkiintoinen vaihe. Node.js-palvelimen dokumentaatio oli mielestäni kattava ja helposti saatavilla, kuten myös Openlayers-kirjaston. Suurimmat ongelmat olivat erityisesti Javascript-teorian kirjoituksessa, sillä lähes jokainen lähde keskittyi opastamaan esimerkkien kautta, jolloin teorian kirjoittaminen vaikeutui ja vaati valtavasti ponnisteluja saattaa loppuun. Vaikka esimerkit voivat hyvin auttaa oppimaan asioita, tykkään kuitenkin itse ymmärtää, miten asiat toimivat.

Toteutuksessa olisin toivonut siistittyä Xml-tiedostoa, jonka läpikäyminen olisi ollut suoraviivaisempi ja tarkoitukseen suunniteltu, koska erityisesti alussa oli vaikeuksia saada vain asiakkaan käyttöön tarkoitettu tieto siirtymään palvelimelta asiakkaalle. Tietenkin tällainen on arkipäivää työelämässä, joten siihenkin täytyy asennoitua.

Sovellusta voisi helposti kehittää vielä lisäämällä käyttäjälle toimintoja ja käyttämällä uudempaa Openlayers 5-versiota, joka on siirtynyt palvelinpuolella ajettavaksi, jolloin käyttäjä lataisi vain karttaobjektin sivulle saapuessaan. Näin vältyttäisiin ylimääräiseltä tietojen lähetykseltä, eikä asiakaan tietokoneen suorituskyky vaikuttaisi yhtä suuresti kartan toimintaan.

Mielenkiintoni erityisesti palvelinpuolen kehitykseen lisääntyi opinnäytetyön seurauksena. JavaScriptin suhteen olen edelleen hieman epäileväinen, juuri sen dokumentoimattomuuden takia. Itsekriittisyys ja täydellisyyden tavoittelu johtivat motivaation laskuun, kun lähteiden löytäminen osoittautui huomattavasti haastavammaksi kuin olin suunnitellut.

LÄHTEET

Chakra repository 2020. ChakraCore. Saatavissa https://github.com/microsoft/ChakraCore. Viitattu 15.3.2020

Expressjs 2020a. Routing. Saatavissa https://expressjs.com/en/guide/routing.html. Viitattu 5.4.2020

Expressjs 2020b. Using Middleware. Saatavissa https://expressjs.com/en/guide/using-middleware.html. Viitattu 5.4.2020

Gratier, T., Spencer, P. & Hazzard, E. 2015. OpenLayers 3: Beginner's Guide. Packt Publishing. Saatavissa http://shop.oreilly.com/product/9781782162360.do. Viitattu 2.6.2019

Hinkelmann, F. 2017. JavaScript engines – How do they even? | JSConf EU. Saatavissa https://www.youtube.com/watch?v=p-iiEDtpy6I

Hölttä, M. 2017. Parsing JavaScript – Better lazy than eager? | JSConf EU 2017 Saatavissa https://www.youtube.com/watch?v=Fg7niTmNNLg

JavaScriptCore repository 2020. JavaScriptCore. Saatavissa https://svn.webkit.org/repository/webkit/trunk/Source/JavaScriptCore/. Viitattu 15.3.2020

Mozilla 2019a. Introduction. Saatavissa https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction. Viitattu 16.11.2019

Mozilla 2019b. Grammar and Types. Saatavissa https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_Types. Viitattu 17.11.2019

Mozilla 2020c.About JavaScript. Saatavissa https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. Viitattu 28.3.2020

Mozilla 2020d. Web APIs. Saatavissa https://developer.mozilla.org/en-US/docs/Web/API. Viitattu 28.3.2020

Mozilla 2019e. Classes. Saatavissa https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes. Viitattu 31.7.2019

Mozilla 2020f. WebGL: 2D and 3D graphics for the web. Saatavissa https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API. Viitattu 14.4.2020

Nodejs 2020a. Introduction to Node.js. Saatavissa https://nodejs.dev/introduction-to-nodejs. Viitattu 19.3.2020

Nodejs 2020b. Differences between Node.js and the Browser. Saatavissa https://nodejs.dev/differences-between-nodejs-and-the-browser. Viitattu 19.3.2020

Nodejs 2020c. An introduction to the npm package manager. Saatavissa https://nodejs.dev/an-introduction-to-the-npm-package-manager. Viitattu 29.3.2020

OpenLayers 2019a. BaseObject. Saatavissa https://openlayers.org/en/latest/apidoc/module-ol_Object-BaseObject.html. Viitattu 31.7.2019

OpenLayers 2019b. Map. Saatavissa https://openlayers.org/en/latest/apidoc/module-ol_Map-Map.html. Viitattu 31.7.2019

OpenLayers 2020c. View. Saatavissa https://openlayers.org/en/latest/apidoc/module-ol_View-View.html. Viitattu 13.4.2020

OpenLayers 2020d. Control. Saatavissa https://openlayers.org/en/latest/apidoc/module-ol_control_Control-Control.html. Viitattu 13.4.2020

OpenLayers 2020e. Interaction. Saatavissa https://openlayers.org/en/latest/apidoc/module-ol_interaction_Interaction-Interaction.html. Viitattu 14.4.2020

Patel, P. What exactly is Node.js 2018, Saatavissa https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/. Viitattu 29.3.2020

Peltomäki, J. JavaScript-kieli 2017, Saatavissa https://books.google.fi/books/about/JavaScript_kieli.html?id=0xY4DwAAQBAJ&redir_esc=y. Viitattu 17.11.2019

Roberts, P. 2014. What the heck is the event loop anyway? | Phillip Roberts | JSConf EU. Saatavissa https://www.youtube.com/watch?v=8aGhZQkoFbQ

Shute, Z. 2019. Advanced JavaScript. Saatavissa https://www.oreilly.com/library/view/advanced-javascript/9781789800104/. Viitattu 17.11.2019

SpiderMonkey repository 2020, https://hg.mozilla.org/mozilla-central/file. Viitattu 15.3.2020

V8 repository 2020. V8. https://github.com/v8/v8. Viitattu 15.3.2020

W3Schools 2020a. HTML Canvas Grapchics. Saatavissa https://www.w3schools.com/html/html5_canvas.asp. Viitattu 14.4.2020