Jaakko Hyry

# VERKKOSOVELLUKSEN SUUNNITTELU JA TOTEUTUS VISION AI -RATKAISUUN

# VERKKOSOVELLUKSEN SUUNNITTELU JA TOTEUTUS VISION AI -RATKAISUUN

Jaakko Hyry
Opinnäytetyö
Kevät 2020
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

# TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

---

Tekijä: Jaakko Hyry
Opinnäytetyön nimi suomeksi: Verkkosovelluksen suunnittelu ja toteutus Vision AI -ratkaisuun
Opinnäytetyön nimi englanniksi: Web application design and implementation for Vision AI solution
Työn ohjaaja: Pertti Heikkilä
Työn valmistumislukukausi ja -vuosi: kevät 2020
Sivumäärä: 34

---

Työ käsittelee verkkosovelluksen suunnittelun ja toteutuksen Symbio Finlandin Vision AI -ratkaisuun, jonka täytyi pystyä havainnollistamaan karttapohjalle ihmisten kulkemia reittejä murtoviivoina sekä visualisoimaan käyttöaste lämpökarttana erillisen liikenteen seurantasovelluksen havainnoimasta anonyymistä datasta. Lisäksi työssä käsitellään projektiivisen geometrian käsite homografia, joka oli keskeisimpiä asioita kamerakuvasta havaittujen kuvakoordinaattien muuntamisessa karttapohjaan. Opinnäytetyössä käydään myös läpi verkko-ohjelmointiin liittyvät komponentit ja taustasovelluksen sekä asiakasohjelman välinen tiedonvaihto REST-rajapinnan yli.

Verkkosovellus kehitettiin JavaScript-ohjelmointikielellä käyttäen OpenLayers-karttakirjastoa, joka tarjosi työkalut interaktiivisen karttapohjan kehitykseen. Kehitys alkoi teknologioiden kartuttamisella ja valmiina olevien Vision AI -komponentteihin tutustumisella, jonka jälkeen alkoi verkkosovelluksen suunnittelu ja toteutus.

Tuloksena oli toimiva verkkosovellus, joka pystyi hakemaan taustaohjelmalta projektiivisella muunnoksella muunnettua reittitietoa kuvakoordinaateista, jotka piirtyivät karttapohjaan murtoviivoina ja lämpökarttana.

---

Asiasanat: verkkosovellukset, verkko-ohjelmointi, taustaohjelmistot, rajapinnat, palvelimet

# ALKULAUSE

Opinnäytetyön tekeminen alkoi marraskuussa 2019 ja valmistui keväällä 2020. Haluaisin kiittää Symbiolta Director / Embedded Leadia Matti Lehtistä ja Software Architect Teemu Stenhammaria tuesta projektissa sekä Tampereen kaupunkia opinnäytetyön mahdollistamisesta.

Oulussa 5.4.2020

Jaakko Hyry

# SISÄLLYS

# SANASTO

**Algoritmi**

Jonkin ongelman ratkaisemiseen tehty toimintaohje.

**API**

Tulee englannin kielen sanoista Application Programming Interface, jolla tarkoitetaan ohjelmointirajapintaa

**Backend**

Yleisimmin käytetty englanninkielinen sana taustaohjelmistosta.

**Frontend**

Käyttäjälle näkyvä osa ohjelmistossa, esimerkiksi käyttöliittymä.

**HTTP**

Hypertext Transfer Protocol. Selainten ja www-palvelimien tiedonsiirtoprotokolla.

**IDE**

Tulee englannin kielen sanoista Integrated Development Environment, jolla tarkoitetaan sovelluksen kehitykseen käytettyä ohjelmakokonaisuutta.

**JSON**

Standardisoitu tietomalli, joka tulee englannin kielen sanoista JavaScript Object Notation.

**Ohjelmakirjasto**

Ohjelmointikieleen sisällytettävä koodimoduuli.

**Ohjelmistokehys**

Ohjelmakirjastoa laajempi kokonaisuus, joka tarjoaa valmiita osia sovelluksen kehitykseen (eng. framework).

**Pseudokoodi**

Ohjelmointikielen tapaista tekstiä, jonka tarkoituksena on havainnollistaa koodin toiminta ilman oikeaa ohjelmointikielen syntaksia.

**REST**

Tulee englannin kielen sanoista Representational State Transfer, jolla tarkoitetaan tietynlaista HTTP-protokollaan perustuvaa ohjelmointirajapintamallia.

# 1 JOHDANTO

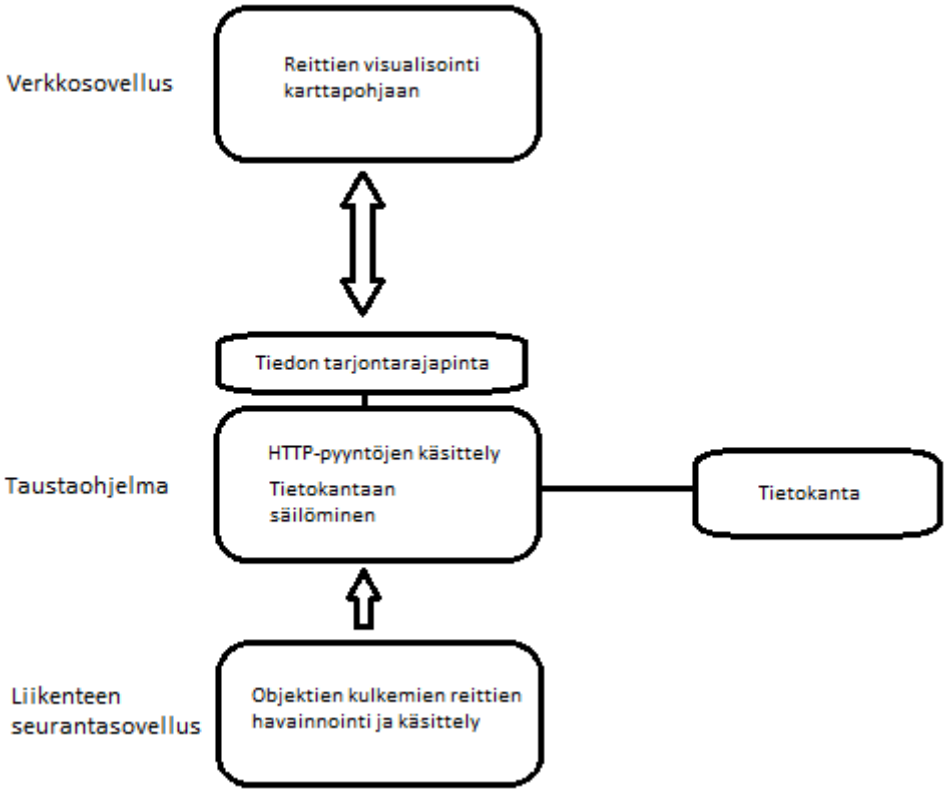
Joulukuussa 2018 aloitin ensimmäisen yritysprojektin tekemisen Symbio Finland Oy:ssä. Jatkoin yrityksessä harjoittelijana ja sain tehtyä jokaisen yritysprojektin sekä työharjoittelun hyvin arvosanoin. Marraskuussa 2019 ilmeni mahdollisuus tehdä opinnäytetyö Tampereen kaupungin tilaamasta jatkokehityksestä Vision AI -ratkaisuun.

Työ on osa Symbio Finland Oy:n Vision AI -ratkaisua, johon oli tavoitteena tuottaa verkkosovellus, joka yhdistää käyttöliittymän sekä taustaohjelman kommunikoinnin. Käyttöliittymän täytyi mahdollistaa ihmisten kulkemien reittien visualisointi karttapohjassa lämpökarttana sekä murtoviivoina kamerakuvaa tutkivan erillisen Vision AI -komponentin havainnoista. Tietojen keräys tapahtui anonyymisti, eikä niihin tallentunut henkilötietoja.

Opinnäytetyössä käsitellään sovellusta koskeva suunnittelu- ja toteutusvaihe sekä käydään läpi olennaisimmat asiat koskien verkkopalvelun kokonaisuutta. Työn tilaajana toimi Symbio Finland Oy.

# 2 VISION AI -RATKAISU

Vision AI on Symbio Finland Oy:n kehittämä digitaalinen ohjelmistokokonaisuus automaattiseen liikenteenseurantaan.




*KUVA 1. Vision AI -ratkaisun komponentit sekä niiden toiminnot*

Järjestelmä koostuu kahdesta komponentista, jotka ovat liikenteen seurantasovellus ja taustaohjelmisto (kuva 1). Nämä komponentit tarjoavat pohjan sovellusten kehittämiselle, jotka käyttävät kokonaisuuden tarjoaman rajapinnan kuvakoordinaattitietoa, joista opinnäytetyössä kehitetty verkkosovellus oli yksi tällainen. Samaa teknologiaa käytettiin myös vuonna 2018 Tampereen kaupungille toteutetussa raitiovaunun täyttöasteen mittaamiseen käytetyssä ratkaisussa. (1.)

## 2.2 Liikenteen seurantasovellus

Yksi keskeisimmistä Vision AI -ratkaisun komponenteista on videokuvaa käsittelevä tekoälysovellus. Konenäkökirjastoja hyödyntämällä sillä voidaan

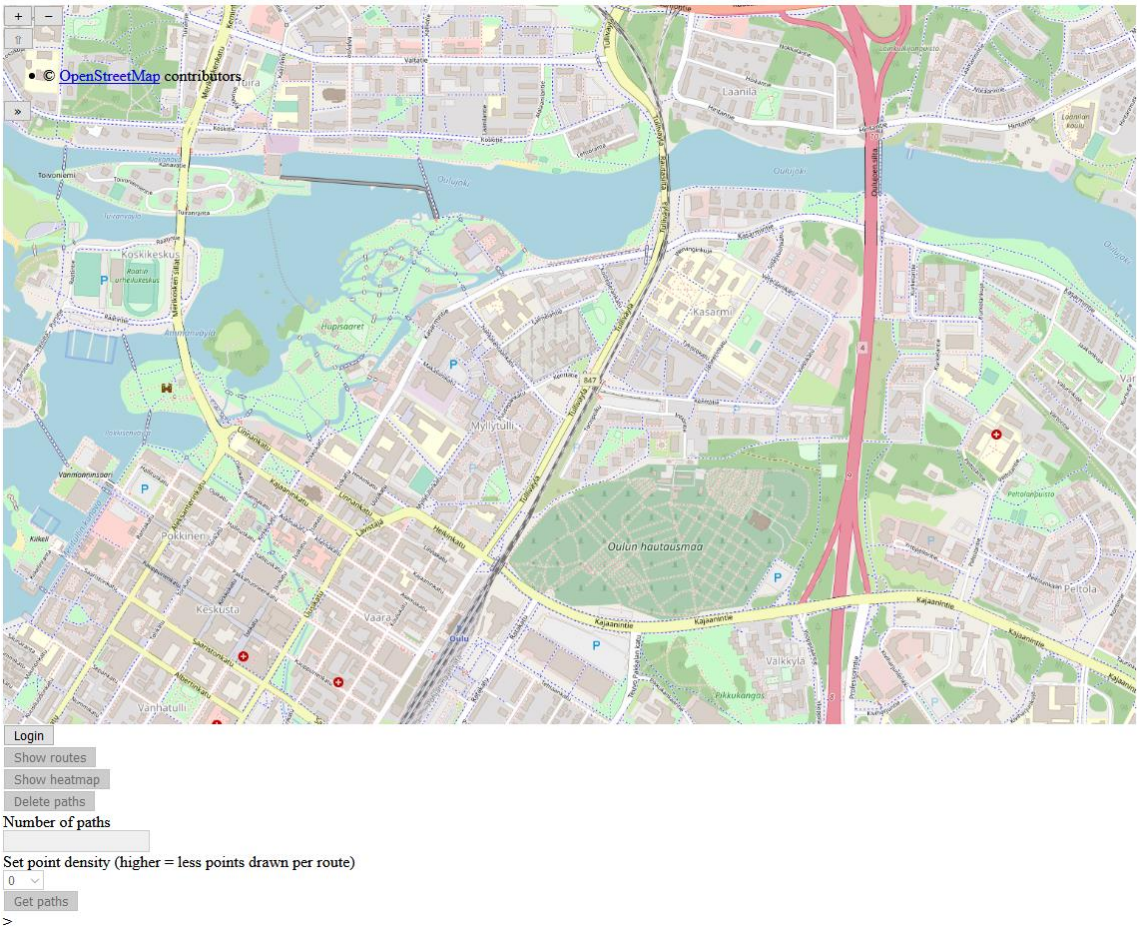
analysoida videokuvassa esiintyviä objekteja kuva kuvalta joko videotalletteesta tai livekuvasta. Sen toiminta perustuu siihen, että kuvissa esiintyville objekteille asetetaan ympäröivä nelikulmio, josta ohjelma laskee keskipisteen. Keskipisteelle lasketaan arvo analysoitavan kuvan pituudesta sekä leveydestä, joista saadaan x- ja y-akselin kuvakoordinaatti väliltä 0–1. Origo (0,0) sijaitsee vasemmassa yläreunassa. X-akseli kasvaa kuvan oikeaa reunaa kohden ja Y-akseli kasvaa kuvan alareunaa kohden, kuten kuvassa 2 on havainnollistettu.


*KUVA 2. Esimerkki kuvakoordinaatistosta*

Kerätty anonyymi tieto lähetetään taustaohjelmistolle, joka tallentaa kuvakoordinaatit tietokantaan. Pelkistä kuvakoordinaateista ei suoraan voida juontaa karttapohjaan vastaavia koordinaatteja, joten liikenteen seurantasovellukseen täytyi kehittää tapa, jolla voitiin muuntaa kuvakoordinaatit vastaamaan karttapohjan koordinaatteja.

## 2.3 Opinnäytetyössä kehitetty verkkosovellus

Sovellus mahdollisti Vision AI -ratkaisuun kuuluvan liikenteen seurantasovelluksen havainnoimat reittitietojen hakemisen tietokannasta ja niiden piirtämisen karttapohjaan murtoviivoina sekä lämpökarttana. Jokainen reittitieto muutettiin käyttöliittymän koodissa JSON-objektiksi, jonka parsituista x ja y koordinaateista sovellus pystyi muodostamaan murtoviivoja lämpökarttapisteiden välille. Käyttöliittymä sisälsi mahdollisuuden lämpökartan ja reittien piilottamiseen, haettujen reittien määrään sekä pisteiden tiheyteen. **Get paths** -painiketta klikkaamalla taustaohjelmistolle lähetettiin pyyntö reittitiedoista REST-rajapinnan yli GET-metodilla (kuva 3).



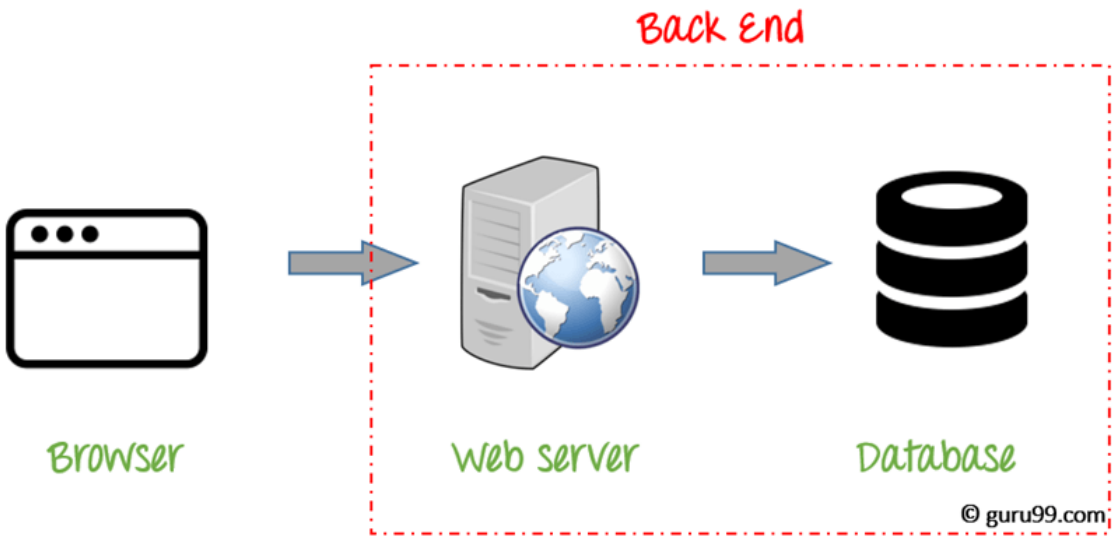

*KUVA 3. Päänäkymä verkkosovelluksen käyttöliittymästä*

# 3 VERKKOSOVELLUKSEN KEHITYKSESSÄ KÄYTETTYJÄ TEKNOLOGIOITA

Verkkosovellus koostuu tyypillisesti palvelimella ajossa olevasta taustaohjelmasta ja selaimella ajettavasta sovelluksesta, joissa kummassakin käytetään toisistaan eroavia teknologioita ja ohjelmointikieliä.

## 3.1 Taustaohjelma

Taustaohjelma eli backend sisältää tulevien pyyntöjen käsittelyyn ja niiden vastaamiseen vaaditun teknologian, joka koostuu tyypillisesti kolmesta osasta, jotka ovat palvelin, palvelinsovellus ja tietokanta. Taustaohjelman palvelinsovellus on vastuussa asiakasohjelman tekemien pyyntöjen käsittelystä ja niihin vastaamisesta, joita ovat usein tietokannasta tiedon hakeminen tai säilöminen. Asiakasohjelmat ovat ohjelmia, jotka lähettävät pyyntöjä palvelimelle. Yksi yleisimpiä asiakasohjelmia on verkkoselain, mutta se voi myös olla esimerkiksi mobiilisovellus tai sovellus, joka pyörii toisella palvelimella. (2.)




*KUVA 4. Tyypillinen taustaohjelma verkkosovelluksessa (3)*

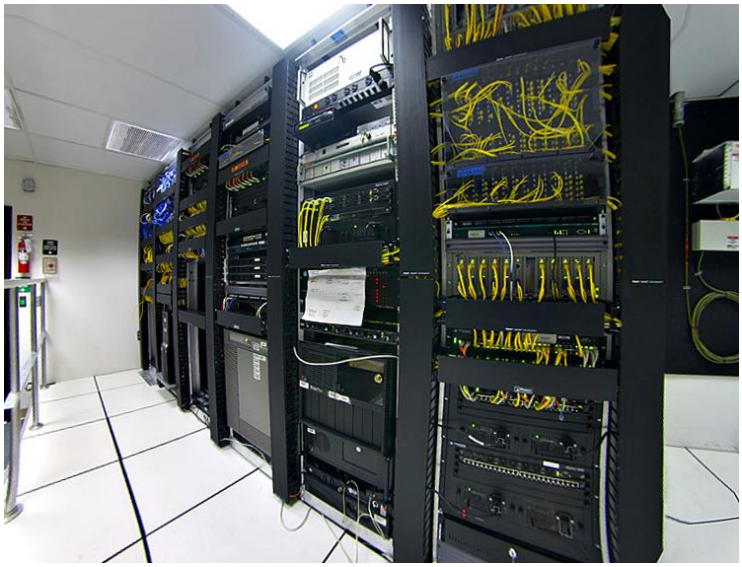
Kun käyttäjä lataa selaimellaan verkkosivun, taustaohjelmalle lähetään pyyntö halutusta resurssista. Palvelimelle toteutettu sovellus käsittelee pyynnön ja

palauttaa halutun resurssin käyttäjälle selaimeen. Tässä luvussa käsitellään asiakas-palvelin-palvelinmallia.

### 3.2 Palvelin

Palvelimen (joskus myös serveri) tarkoitus on vastata asiakkaan, joka voi olla palvelinta käyttävä sovellus tai tietokone, tiedonvaihdosta tietokoneverkon yli. Palvelin itse on useimmiten jatkuvasti päällä oleva tietokone tai muu laitteisto, johon on toteutettu palvelusta vastaava sovellus. Palvelimet ovat normaalisti jatkuvasti päällä, jotta esimerkiksi verkkosivu pysyy käyttäjille avoinna. Palvelintyyppejä on useita eri käyttötarkoituksiin. Yksi niistä on verkkopalvelin, joka vastaa verkkosovelluksen resurssien toimituksesta. Mahdollisten virtakatkosten tai muiden palvelinlaitteiston kaatumisen tai tietohäviön aiheuttamien tapahtumien varalta palvelin on hyvä rakentaa virheensietokykyiseksi. (4.)

Usein palvelin sisältää sensitiivistä tietoa, joten on tarpeen, että vain varmennetuilla henkilöillä on niihin pääsy. Yritysten tilojen oma palvelin saattaa sijaita erillisessä lukitussa palvelinhuoneessa. Suurien palveluntarjoajien palvelimet sijaitsevat useimmiten kuvan 5 mukaisissa tarkkaan valvotuissa palvelinkeskuksissa, joihin asiakas voi ottaa etäyhteyden. (4.)



*KUVA 5. Palvelinlaitteistoa palvelinkeskuksessa (4)*

Palvelimen nopeus perustuu sen suorituskykyyn ja siihen vaikuttavat monet asiat. Heikolla prosessorilla ja kovalevyllä varustetun palvelimen vastausajat ovat pidempiä kuin paremmin varustetulla. Esimerkiksi Intelin Xeon-prosessorisarja on palvelimille suunniteltu prosessori, joka sisältää normaalia prosessoria enemmän ytimiä ja tukee enemmän keskus- ja välimuistia. Palvelimen vasteajalla tarkoitetaan aikaa, joka menee vastauksen toimitukseen asiakasohjelmalle. Googlen mukaan 200 millisekunnin ja sen alle menevät vastaukset luokitellaan nopeiksi. Yli 200 millisekunnin vasteaika kertoo yleensä siitä, että palvelimen säädöissä tai siihen kuuluvassa sovelluksessa on ohjelmoitu jotain väärin. Palvelimen vasteaikaan vaikuttaa myös sen fyysinen sijainti, sillä mitä lähempänä se sijaitsee, sitä vähempien verkkojen ja lyhyemmän matkan täytyy signaalin kulkea. (5.)

### 3.3 Tietokanta

Tietokanta on palvelimen kovalevyllä tai pilvessä sijaitseva strukturoitu tietojen kooste. Tietokanta koostuu tauluista ja sarakkeista, joista tietoa voidaan hakea tai muuttaa SQL-komennoilla. (6.)

Jos käyttäjä haluaisi valita taulukon 1 mukaisesta tietokannasta banaanin ja jokaisen siihen liittyvän sarakkeen tiedot käyttämällä hakuparametreja tuotteen id-numeroa, tapahtuisi se seuraavalla komennolla.

*SELECT * FROM hedelmat WHERE id = 1;*

*TAULUKKO 1. Esimerkki tietokantataulusta*

| | hedelmat | |
|---|---|---|
| id | tuote | hinta |
| 1 | banaani | 1 |
| 2 | tomaatti | 2 |

SELECT-komennolla tarkoitetaan sitä, että käyttäjä haluaa hakea tietoa taulusta. Tähti-symboli on korvausmerkki, jolla tarkoitetaan taulun sarakkeen jokaista arvoa. FROM-avainsanalla viitataan tauluun ”hedelmat” ja lopuksi WHERE tarkoittaa hakuparametria. Kysely lopetetaan aina puolipisteeseen. Vaikka SQL-syntaksi ei ole merkkikokoriippuvainen, on tavan mukaista kirjoittaa komennot isoilla kirjaimilla, mikä myös selventää komentojen tulkintaa. Jos käyttäjä haluaa pyytää pelkän banaanin hinnan käyttämällä hakuparametrina tuotteen nimeä, tapahtuisi se komennolla:

*SELECT hinta FROM hedelmat WHERE tuote = banaani;*

SQL-kielen keskeisimpiä syntakseja taulun tietojen kyselyyn sekä päivittämiseen ovat SELECT, INSERT INTO, DELETE.

Käsitellessään asiakasohjelman pyyntöä palvelinsovellus suorittaa tietokantakyselyitä ohjelmointirajapinnan kautta ja palauttaa halutun resurssin.

### 3.4 Palvelinohjelmointi

Staattiset verkkosivut palauttavat kovakoodattua sisältöä halutusta resurssista. Käyttäjän navigoidessa sivulta toiselle selain lähettää palvelimelle GET-metodin, jonka se käsittelee ja palauttaa selaimen käyttöliittymään. Dynaamisessa verkkosivussa vastauksen sisältö muuttuu käyttäjän mieltymyksien ja tapojen mukaisesti. Ennalta määritettyihin HTML-elementteihin voidaan upottaa tietokannasta haettua tietoa muuttujina paikkamerkkien tilalle. Saman palautettavan elementin sisältö saattaa muuttua käyttäjän käyttäessä sivua. Dynaamisen verkkosivun tukemiseksi suurin osa koodista on ajettava palvelimella ja sen kehitystä kutsutaan palvelinohjelmoinniksi. (7.)

Palvelin- ja asiakasohjelmoinnilla on eri tarkoitukset ja huolenaiheet, eivätkä ne käytä yleensä samaa ohjelmointikieltä JavaScriptiä lukuunottamatta. Ne myös pyörivät erilaisissa käyttöjärjestelmäympäristöissä. Frontend eli selaimessa käyttäjälle näkyvää sivua tai sovellusta kutsutaan asiakasohjelman koodiksi, joka vastaa sivuston ulkonäöstä ja toiminnallisuudesta. JavaScript on asiakasohjelman toteutuksessa usein käytetty verkko-ohjelmointikieli, jota käytetään yleensä HTML- ja CSS -tyyliohjeiden kanssa. Sovellus ajetaan

selaimessa, eikä sillä ole juurikaan pääsyä käyttöjärjestelmään. Backend eli palvelimella pyörivä sovellus sen sijaan vastaa selaimelle oikean sisällön toimituksesta vastauksena pyyntöihin. (7.)

Palvelinpuolen ohjelmisto voidaan toteuttaa usealla eri ohjelmointikielellä. Yleisiä näistä ovat esimerkiksi PHP, Python, Ruby, C# ja JavaScript (NodeJS). Palvelinpuolen koodilla on täysi pääsy palvelimen käyttöjärjestelmään. Tyypillisesti kehityksen nopeuttamiseksi kehittäjä käyttää jotain verkko-ohjelmistokehystä. Ne ovat kokoelma funktioita, objekteja ja ohjeita suunniteltu yleisten ongelmien ratkomiseksi ja erilaisten tehtävien yksinkertaistamiseksi. (7.)

### 3.5 REST-ohjelmointirajapinta

Ohjelmointirajapinnalla eli API:lla (Application Programming Interface) tarkoitetaan määritelmien sekä protokollien joukkoa ohjelmiston rakentamiseen ja sen integroimiseen.

REST on ohjelmointirajapintamalli, joka tulee englannin kielen sanoista Representational State Transfer. Sillä tarkoitetaan tiettyä verkkosovellusten tiedonvaihtoarkkitehtuuria tai verkko-ohjelmointirajapintaa, jossa tieto liikkuu palvelimen ja käyttöliittymän välillä yleensä JSON tai XML-muodossa HTTP-protokollan yli. REST-arkkitehtuuri erottaa asiakasohjelman palvelimesta ja tietovarastosta, joka parantaa rajapinnan siirrettävyyttä muille alustoille ja mahdollistaa kehitysvaiheessa eri komponenttien itsenäisen kehityksen. (8.)

Tyypillistä REST-pohjaiselle palvelulle on, että kyselyt asiakasohjelmalta tehdään niin sanotuille URI (Uniform Resource Identifier) -päätepisteille, joilla määritetään tiedon osoite. Taustaohjelmasovellukseen voidaan määrittää HTTP-metodin käsittely ja päätepiste. Esimerkiksi, jos kirjastopalvelimelta haluttaisiin kaikki A:lla alkavat kauhukirjat, voisi se kysely tapahtua asiakasohjelmalta seuraavasti:

GET-pyyntö URI-päätepisteeseen /kirjat/kategoria/kauhu/?ekaKirjain=a

URI-päätepisteeseen voi sisällyttää kyselyparametreja, jotka merkitään kysymysmerkin jälkeen. Yllä olevassa kyselyssä säätöparametri *ekaKirjain* käsitellään kyseisen URI-päätepisteen käsittelyyn tehdyssä funktiossa.

Kuvassa 7 on havainnollistettu pseudokoodattu esimerkki, kuinka ASP.NET-ohjelmistokehyksellä toteutettu taustaohjelmasovellus voisi yksinkertaisesti käsitellä kyselyn. ASP.NET -sovelluksessa niin sanotulle kontrollereille voidaan asettaa attribuutteja. Attribuutti [Route] määrittää URI-päätepisteen ja [HttpGet] käsiteltävän metodin. Kuvitteelliseen kirjatietokantaan lähetetään kysely string-muodossa, johon parametri **ekaKirjain** upotetaan. Näin voidaan käyttää kyselyparametreja asiakasohjelman GET-pyynnössä.

```
[Route("kirjat/kategoria/kauhu")]
[HttpGet]
0 references
public List<Kirja> kirjat(string ekaKirjain)
{
    List<Kirja> kirjaLista =  new List<Kirja>();
    string sqlQuery = "SELECT * from kirjat where kategoria=kauhu AND title LIKE " + "'" + ekaKirjain + "%'";
    dbQuery(sqlQuery);
    // -> Luodaan kirja-objektit tuloksista
    // -> Lisätään kirjaListaan ja serialisoidaan se haluttuun muotoon
    // -> Palautetaan kirjalista

    return kirjaLista;
}
```

*KUVA 6. GET-metodin käsittely ASP.NET palvelinsovelluksessa*

Mikäli haluttaisiin säilöä kauhukirja, voisi se tapahtua seuraavasti:

POST-pyyntö URI-päätepisteeseen /kirjat/kategoria/kauhu

POST-pyyntö tarvitsee mukaan tietopaketin (body) sisältäen tiedon ja otsikkotietoja (header), joissa voi olla esimerkiksi todentamiseen käytettävä poletti JSON Web Token. Tietopaketti voi olla esimerkiksi JSON-muodossa.

### 3.5.1 HTTP-protokolla

HTTP on protokolla ohjelmatasossa, jolla voidaan siirtää hypermediadokumentteja kuten HTML. Verkkoselainten ja palvelimien väliseen kommunikointiin tarkoitettu protokolla noudattaa klassista asiakas-palvelin-palvelinmallia, jossa asiakas, usein verkkoselain avaa yhteyden, tekee pyynnön ja odottaa vastausta. HTTP:n yli palvelimelle menevistä pyynnöistä ei jää mitään tietoa, mikä tekee siitä tilattoman. Yksinkertaisestettuna käyttäjän avatessa verkkosivun, selain tekee HTTP-pyynnön käyttäen GET-metodia, johon palvelin vastaa palauttamalla HTML-hypermediadokumentin. HTTP-protokollan

yleisimpiä metodeja ovat GET ja POST, mutta se sisältää myös vähemmän käytettyjä kuten DELETE tai PUT. (9.)

- GET-komennolla tarkoitetaan vain resurssin hakemista. Palvelinsovellus tekee tietokantakyselyn haluttuun resurssiin ja palauttaa tiedon asiakasohjelmalle yleensä JSON-muodossa, jossa tieto voidaan helposti parsia käyttämällä käytettävän ohjelmointikielen JSON:in parsintaan tarkoitettua kirjastoa.
- POST-komennolla tarkoitetaan resurssin lisäämistä.
- PUT-komennolla tarkoitetaan resurssin lisäämistä tai jo olemassa olevan resurssin muokkaamista.
- DELETE-komennolla tarkoitetaan resurssin poistamista.

**HTTP-tilakoodit**

Palvelinsovelluksen käsitellessä pyyntöä se palauttaa vastauksen mukana myös kolminumeroisen numerosarjan viitaten pyynnön käsittelyn onnistumiseen. Nämä tilakoodit (HTTP status code) on kategorisoitu ensimmäisen numeron perusteella. (10.)

- 1xx tarkoittaa, että pyyntö tavoitettiin ja prosessointi jatkuu. Nämä ovat niin sanottuja informatiivisia vastauksia.
- 2xx tarkoittaa pyynnön onnistumista.
- 3xx tarkoittaa pyynnön uudelleenohjausta.
- 4xx tarkoittaa asiakasohjelman virhettä. Pyynnön sisältäessä virheellistä syntaksia pyyntöä ei voida toteuttaa.
- 5xx tarkoittaa palvelimen sisäistä virhettä, eikä pyyntöä voida toteuttaa.

### 3.5.2 JSON

JSON (JavaScript Object Notation) on standardisoitu tietomuoto, jota käytännössä kaikki nykyaikaiset ohjelmointikielet tukevat, ja jota yleisesti käytetään verkko-ohjelmoinnin tiedonsiirrossa. Sen ominaisuuksia ovat helppo luettavuus, keveys ja ohjelmointikielestä riippumattomuus. (10.)

JSON-objekti koostuu avain-arvopareista, joiden määrittely alkaa aukinaisella aaltosululla ja loppuu kiinni olevalla aaltosululla. Objetkin arvot määritellään näiden merkkien sisällä.  Kuvassa 8 on JSON-objekti, joka pitää sisällään listan toisia JSON-objekteja.


*KUVA 7. JSON-objektiesimerkki*

Mikäli edellämainitusta objektista haluttaisiin parsia ”nimi”-kenttä tai ”arvo3”-avaimen alla olevan taulukon indeksin 0 arvon ”arrayobjekti1”, tapahtuisi se Python-ohjelmointikielellä kuvan 9 mukaisesti.

```
2   obj = {
3     "nimi": "json objekti",
4     "arvo1": "tekstia",
5     "arvo2": 1,
6     "arvo3":
7     [
8       {
9         "arrayobjekti1": "yksi"
10      },
11      {
12        "arrayobjekti2":"kaksi"
13      }
14    ]
15  }
16
17  # parsii "nimi"-avaimen arvon:
18  nimi = obj["nimi"]
19  # parsii "arvo3"-avaimen olevan arrayn indeksin 0 kohdassa olevan "arrayobjekti1"-avaimen arvon
20  arrayobjekti1 = obj["arvo3"][0]["arrayobjekti1"]
```

*KUVA 8. JSON-objektin parsiminen Python-ohjelmointikielellä*

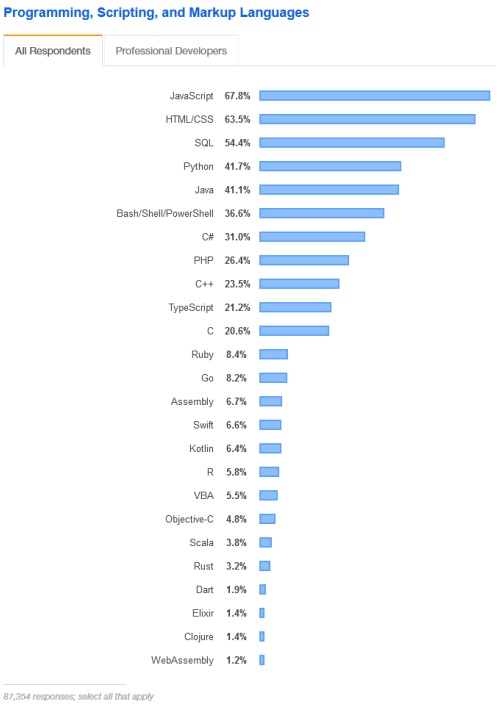
Ohjelmointikieli sisältää yleensä valmiin kirjaston JSONin manipulointiin, mutta on myös mahdollista, että ohjelmoija käyttää kolmannen osapuolen tekemiä kirjastoja. Yksi näistä on Java-ohjelmointikielelle käännetty Googlen Gson-kirjasto.

# 4 VERKKOSOVELLUKSEN SUUNNITTELU JA TOTEUTUS

Projektin kehitys tapahtui Visual Studio Code -koodieditorilla Linux-koneella NodeJS:n tarjoamalla paikallisella palvelimella. Paikallisympäristössä taustaohjelmiston sekä verkkosovelluksen kehitys tapahtui käytössä olevalla tietokoneella, joka toimi palvelimena, ja jossa taustaohjelmistoa ajettiin. Tämä mahdollisti sovelluksen muokkauksen ja testauksen nopeasti ilman, että sovellus täytyisi aina erikseen lähettää erilliselle palvelimelle. Paikallisympäristö myös eristää kehityksen pitäen sen piilossa muilta. Kun sovelluksen lopullinen toiminta oli todennettu ja testattu paikallisympäristössä, oli se turvallista ladata palvelimelle.

## 4.1 Ohjelmointikielen sekä muiden teknologioiden valinta

Koska kyseessä oli verkkokäyttöliittymä, täytyi kyseessä olla frontendin kehittämiseen tarkoitettu verkko-ohjelmointikieli. Päädyin käyttämään JavaScript-ohjelmointikieltä sen yleisyyden ja tuettavuuden takia. Vuonna 2019 StackOverflow’n kyselyssä 67,8 % kertoi käyttävänsä JavaScriptiä (kuva 11). JavaScriptin tueksi valitsin myös React-ohjelmistokehyksen, joka eristää käyttöliittymän elementit omiin komponentteihin.

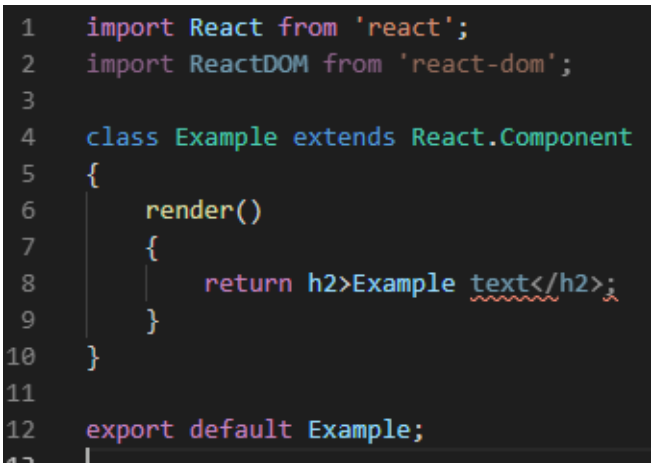



*KUVA 9. Käytetyimmät ohjelmointikielet vuonna 2019 StackOverflow'n tekemässä kyselyssä (12)*

## 4.2 Kehityksessä käytetyt ohjelmistot

### Visual Studio Code

Visual Studio Code on avoimeen lähdekoodiin perustuva tekstieditori, joka ei ole kuitenkaan täydellinen ohjelmointiympäristö, mutta tarjoaa tuen mm. virheenkorjaukselle sekä syntaksin korostukselle. Visual Studio Coden tukemia ohjelmointikieliä voi laajentaa lataamalla laajennuksia ohjelman omalla integroidulla laajennusten hallintatyökalulla.




*KUVA 10. Virheellistä syntaksia Visual Studio Code -tekstieditorissa*

Kuvasta 12 voidaan nähdä Visual Studio Coden virheellisen sekä toimivan syntaksin korostus.

**Postman**

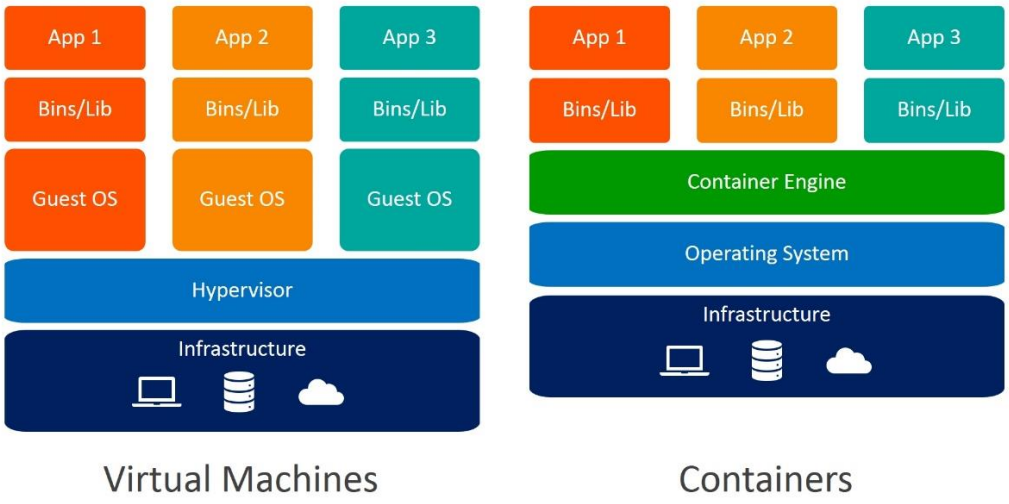
Postman on rajapintakehitykseen tehty sovellus, jolla voidaan helposti tehdä rajapintakutsuja palvelimelle sen käyttölittymästä. Käytin projektissa Postmania taustaohjelmiston REST-rajapinnan testaukseen ja kehitykseen.

**Ubuntu 18.x**

Ubuntu on Linux-käyttöjärjestelmän jakelupaketti (myös jakelu tai distro).
Projektin kehitys tapahtui Linux-ympäristössä.

**Docker**

Docker tarjoaa käyttöjärjestelmätason virtualisaation, joka perustuu kontitusteknologiaan. Sillä voidaan eristää sovellus ja sen riippuvuudet omaan konttiin, joka helpottaa ohjelman kehitystä ja ajamista eri ympäristöissä (kuva 13).




*KUVA 11. Docker-kontin ero virtuaalikoneeseen verrattuna (13)*

Virtuaalikone ja Docker tarjoavat molemmat tavan eristää sovellus laitteistosta riippumattomaan, eristettyyn ympäristöön. Docker kuitenkin eroaa siinä mielessä, että ajossa olevat kontit jakavat saman laitteiston kernelin, kun taas virtuaalikone on oma instanssi, jolla on oma käyttöjärjestelmä (kuva 13). Virtuaalikoneen luomiseen laitteistolta allokoidaan haluttu määrä resursseja instanssia kohden, kuten vaikka keskusmuistin tai tallennustilan määrä, kun taas kontit asettuvat suoraan käyttöjärjestelmän päälle saaden käyttöön suoraan laitteiston resurssit. Kontitusteknologian hyötyjä ovat mm. sen käynnistysaika ja keveys. Kontit eivät sisällä omaa käyttöjärjestelmää, eli niiden koko jää usein pieneksi, yleensä 5–100 megatavua. Virtuaalikoneesen sisällytetty oma käyttöjärjestelmä voi nostaa sen kokoa gigatavuilla. (13.)

### 4.3 Rajapintakutsut JavaScriptin Fetch-kirjastoa käyttäen

Jotta käyttöliittymään saatiin tieto kuljetuista reiteistä, täytyi taustaohjelman palvelimeen päästä käsiksi ohjelmallisesti. Käyttöliittymään tehtiin luokka, joka vastasi kyselyistä palvelimelle HTTP-pyyntöjä REST-rajapinnan yli. Luokan staattiset metodit tarjosivat helpon tavan pyytää esimerkiksi tietty reitti sen tunnistenumeroa käyttäen, johon palvelin vastasi palauttamalla tiedot JSON-objektina. JavaScriptin Fetch-kirjasto tarjosi nykyaikasen sekä monipuolisen tavan noutaa tietoa palvelimelta. Yksinkertaisimmillaan syntaksi on kuvan 14 tapainen. (14.)

```
let promise = fetch(url, [options])
```

*KUVA 12. JavaScriptin Fetch-kirjaston käyttöesimerkki*

Kuvassa 15 parametri *url* tarkoittaa haettavaa sivua ja *options* pyyntöä koskevia asetuksia, esimerkiksi HTTP-metodia tai otsikkotietoja. Funktiota voi myös kutsua ilman *options*-parametria, jolloin se palauttaa sivun sisällön käyttäen GET-metodia. (14).

```
1   let response = await fetch(url);
2   // Tarkastetaan, että vastauksen tilakoodi on alueella 200-299
3   if (response.ok)
4   {
5     // Säilötään tieto muuttujaan JSON-muodossa
6     let json = await response.json();
7   }
8   // Pyynnön epäonnistuessa tuodaan käyttöliittymään virheilmoitus
9   else
10  {
11    alert("HTTP-Error: " + response.status);
12  }
```

*KUVA 13. Fetch-kirjastolla haetun tiedon virheenkäsittely*

GET-metodilla tehty pyyntö muunnetaan JSON-muotoon jatkokäsittelyä varten. Vastaukselle on hyvä tehdä yleinen virheenkäsittely, jolla tarkastetaan onko HTTP-vastauksen tilakoodi arvoalueella 200-299 (kuva 15).

### 4.4 Kuvakoordinaattien muuntaminen karttakoordinaateiksi

Yksi kehitettävän sovelluksen keskeisempiä ongelmia oli kuvasta havaittujen koordinaattien muuntaminen vastaamaan karttapohjan koordinaatteja. Projektiivisen geometrian käsitettä homografia eli projektiivista muunnosta pystyi käyttämään ongelman ratkaisemiseksi. Asiaa tutkittuani löysin konenäkökirjaston, joka tarjosi juuri oikeat metodit tämän ongelman ratkaisuun.

#### 4.4.1 OpenCv-kirjasto

OpenCV on avoimeen lähdekoodiin perustuva konenäkökirjasto, joka on käännetty mm. Pythonille, Javalle, C/C++:lle sekä osittain JavaScriptille.

Konenäkökirjaston tarjoama funktio *findHomography()* tarjoaa homografian laskemisen kahden pistetaulukon välille. *findHomography()*-funktion määritelmä OpenCV:n virallisesta Python-dokumentaatiosta on seuraavanlainen.

*findHomography(srcPoints, dstPoints[, method[, ransacReprojThreshold[, mask[, maxIters[, confidence]]]]])*

Pakolliset parametrit *srcPoints ja dstPoints* ovat double-tyyppisiä taulukoita, joista *srcPoints* pitää sisällään lähdekuvan koordinaatit ja vastaavasti *dstPoints* kohdekuvan koordinaatit. Muut parametrit eivät ole pakollisia, vaan niille asettuu vakioarvo niiden puuttuessa. Funktio palauttaa 3x3 float-tyyppisen moniulotteisen muunnosmatriisin. Kun kahden kuvan välinen homografia on selvitetty, OpenCV tarjoaa perspektiivimuunnosfunktion *perspectiveTransform().* (15.)

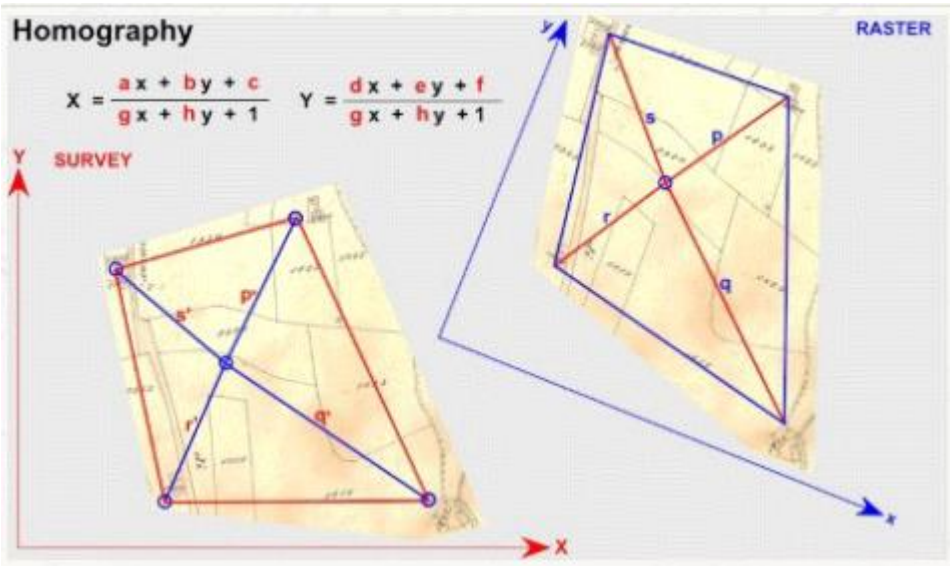
Virallinen dokumentin määritelmä tälle funktiolle on seuraavanlainen.

*perspectiveTransform(src, m[, dst])*

Käyttäen *findHomography()*-funktion paluuarvosta saatua muunnosmatriisia, voidaan se asettaa edellämainitun funktion parametriksi *m* ja parametri *src* on float-tyyppinen taulukko, johon halutut muunnospisteet säilötään. (15.)

### 4.4.2 Projektiivinen muunnos eli homografia

Projektiivisella muunnoksella 2D-avaruudessa olevan tason pisteet voidaan projisoida toiseen tasoon (kuva 16). Homografiassa pisteillä on yksi-yhteen -riippuvuus tasojen välillä, eli esimerkiksi karteesisessa koordinaatistossa jokaista reaaliluvuista muodostuvaa (x,y)-paria vastaa tietty koordinaattipari toisessa koordinaatistossa. Projektiivisilla muunnoksilla voidaan siirtyä koordinaatistosta toiseen yksiselitteisesti. (16.)




*KUVA 14. Projektiivinen muunnos ja sen laskukaava (16)*

Jotta homografia kuvan ja karttapohjan välillä saatiin selville, täytyi ensiksi tehdä kamerakuvan testikalibraatio, jossa selvitettiin maamerkkejä käyttämällä xy-koordinaatit seisomalla kamerakuvassa neljässä valitussa pisteessä, jotka muodostivat nelikulmion. Seuraavaksi täytyi todentaa nämä neljä vastaavaa pistettä karttapohjalla. Strategisesti valitut pisteet oli helppo löytää karttapohjaan maamerkkien avulla. Kalibraatio osoittautui yllättävänkin tarkaksi testausvaiheessa. Kyseistä tapaa käyttämällä saatiin *findHomography()*-funktioon oleelliset parametrit *srcPts* jotka olivat tässä tapauksessa kuvan neljä pistettä ja *srcPts*, jotka taas olivat karttakuvaan mallinnetut neljä pistettä. Nyt jokainen kuvakoordinaatistossa todettu koordinaatti muuttui kartassa vastaavaan pisteeseen.

### 4.5 OpenLayers -karttakirjaston käyttö JavaScript verkkosovelluksessa

Ohjelmaan täytyi löytää sopiva karttakirjasto, johon valitsin OpenLayersin. OpenLayers on avoimeen lähdekoodiin perustuva karttakirjasto, joka tarjoaa laajan valikoiman työkaluja interaktiivisen karttaohjelman rakentamiseen. Kirjaston käyttöönotto edellyttää kirjaston lataamista, joka tapahtui kätevästi Noden pakettihallintaan tarkoitetulla npm-työkalulla käyttäen ”npm install ol” -komentoa.

```
1   import Map from 'ol/Map';
2
3   var map = new Map({
4       target: 'map',
5       layers: [
6         new ol.layer.Tile({
7           source: new ol.source.OSM()
8         })
9       ],
10      view: new ol.View({
11        center: ol.proj.fromLonLat([37.41, 8.82]),
12        zoom: 4
13      })
14    });
```

*KUVA 15. OpenLayers-kirjaston sisällyttäminen ja karttaobjektin luominen*

Ensin karttakirjasto sisällytetään JavaScript-sovellukseen käyttämällä import-lauseketta (rivi 1). Sisällyttämisen jälkeen new-operaattorilla voidaan luoda objekti Map-luokasta, joka tarjoaa karttanäkymän ja luokan metodit (kuva 17). Map-objekti on kirjaston oleellisin komponentti, koska se tarjoaa karttanäkymän. Kartta koostuu Layereista (kerros) sekä View'stä (näkymä). Layereilla tarkoitetaan kartan päälle piirtyviä kerroksia, joista käytin OpenLayers:in tarjoamaa HeatMap-kerrosta, jolla karttaan saatiin lämpökarttanäkymä, ja Vector-kerrosta, joka mahdollisti reittien visualisoinnin murtoviivoina.

OpenLayersissä tieto on enkapsuloitu jossa lopullisella kartassa olevan visuaalisen elementin takana on monta objektikerrosta. Lämpökarttapisteen piirtämiseen määritellään ensiksi Feature-objekti (kuva 18).

```
19  var heatMapFeature = new Feature({
20    geometry: new Point([[232323.4],[221142.5]]),
21    name: 'HeatMap point'
22  });
```

*KUVA 16. Feature-objektin luonti*

Feature-objekti pitää sisällään ominaisuuden (property) geometry, jolla määritellään piirrettävän elementin maantieteellinen arvo. Tässä tapauksessa sille on annettu arvoksi Point-objekti, jolle voidaan antaa argumentteina pisteen

koordinaatit. Luotu Feature-objekti sisällytetään HeatMap-kerrokseen ja kerros lisätään Map-objektiin (kuva 19).

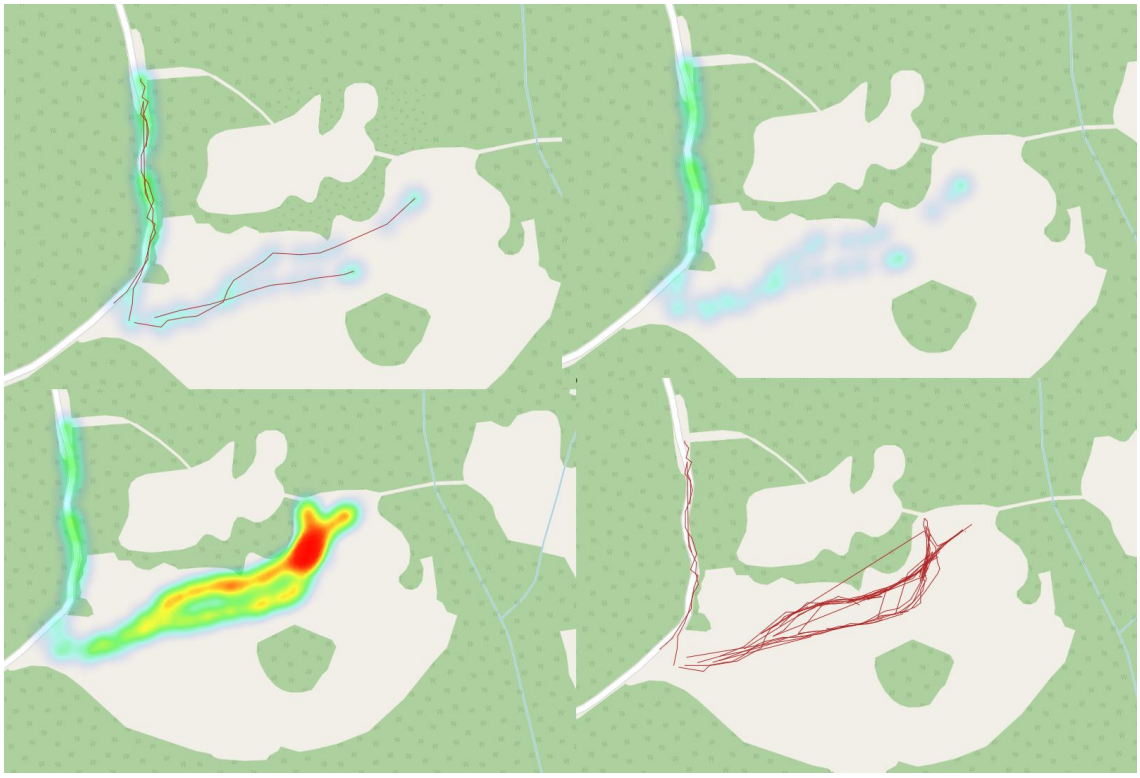
```
24  var heatMapLayer = new Heatmap({
25      features: heatMapFeature
26  })
27  // vaihtoehtoisesti
28  heatMapLayer.getSource().addFeature(heatMapFeature)
29
30  map.addLayer
```

*KUVA 17. Feature-objektin lisääminen karttaobjektiin*

Sovellus käsitteli taustaohjelmalta haetut JSON-objektit parsien niistä valmiiksi karttapohjaan sopivaksi muunnetut koordinaatit. Parsitut koordinaatit pystyi nyt asettamaan Point-objektiin, jolla saatiin lämpökarttapisteet piirtymään karttapohjaan. Vastaavasti reittien piirtoon käytettiin OpenLayersin MultiLineString-objektia, joka yhdisti yhden reitin lämpökarttapisteet murtoviivalla.

Tietomallia täytyi keventää, jotta reittien piirto saatiin nopeammaksi. Kuvan analysointisovellukseen kehitettiin myös logiikka reittien kaksoiskappaleiden poistamiseen ja ohjelmallisesti asetettava raja, jonka yli edellisen koordinaatin x tai y arvon täytyi mennä, ennen kuin se lisätään pisteitä sisältävään listaan, mikä vähensi huomattavasti päällekkäisiä tai toisiaan liian lähellä olevia pisteitä. Taustaohjelmaan kehitettiin tapa säilöä ja hakea kalibraatiot REST-rajapinnan yli. Liikenteen seurantasovellukseen pystyi alun perinkin syöttämään ajoparametreja sen käynnistyksen yhteydessä, joihin lisäsin mahdollisuuden syöttää kalibraation numerotunnisteen. Sovelluksen käynnistyessä se teki kyselyn taustapalvelimelle syötetyllä numerotunnisteella ja haki sitä vastaavan kalibraation käyttöönsä.

Esimerkkitiedolla tuotetuista reiteistä voidaan havainnollistaa lämpökartan värin syveneminen risteyskohdissa (kuva 20).



*KUVA 18. Reittien ja lämpökartan kehittyminen OpenLayers-karttapohjassa*

### 4.6 Sovelluksen käyttöönotto

Kun sovelluksen toiminta oli todennettu paikallispalvelimella, se pystyttiin siirtämään verkkopalvelimelle ja sitä kautta myös asiakkaan käyttöön. Sovellus otettiin käyttöön Google Cloud Platform -alustalle, joka on Googlen samaa sisäistä infrastruktuuria käyttävä pilvipalvelualusta. Se sisältää modulaarisia pilvipalveluja aina tallennustilasta data-analyysiin ja verkkopalvelimeen. (17.)

### 4.7 Työn tulokset

Verkkokäyttöliittymän toiminnallisuusvaatimukset toteutuivat. REST-rajapinta mahdollisti helpon ja kätevän tavan tiedonsiirtoon käyttöliittymän ja palvelimen välillä. Kamerakuvassa havaitut liikeradat saatiin muutettua murtoviivoiksi ja lämpökartaksi karttapohjaan hyvällä tarkkuudella.

# 5 YHTEENVETO

Työn tarkoituksena oli kehittää verkkosovellus jolla pystyi visualisoimaan karttapohjalla anonyymisti kerättyä tietoa kuljetuista reiteistä yhdistämällä käyttöliittymä jo olemassa olevaan taustaohjelmaan ja kehittää toimintalogiikka tiedonsiirtoon niiden välille hyödyntämällä REST-rajapinta-arkkitehtuuria. Myös valmiina olevaan liikenteen seurantaan kehitettyyn sovellukseen täytyi kehittää tapa, jolla kuvasta havaitut koordinaatit pystyttiin muuntamaan karttapohjalle kuljetuksi reitiksi ja lämpökartaksi.

Ensimmäinen askel kehityksessä oli tutustuminen valmiina olevaan taustaohjelmaan ja liikenteen seurantasovellukseen, joiden tietämys auttoi kehittäessä verkkosovellusta. Sen jälkeen alkoi itse sovelluksen kehittäminen ja samalla JavaScriptin opetteleminen. Sopivan karttakirjaston löydyttyä reittien ja lämpökarttapisteiden piirtoa pystyi alkaa kokeilemaan testidatalla, jotta logiikka reittien piirtämiselle hahmottui paremmin. Tämän jälkeen täytyi selvittää kuinka kuvakoordinaateista voitaisiin johtaa karttapohjaan vastaavat koordinaatit ja tehdä vaaditut muutokset liikenteen seurantasovellukseen sekä taustaohjelmaan. Viimeinen vaihe oli sovelluksen kytkeminen taustaohjelman REST-rajapintaan hyödyntämään havannoitua reittitietoa.

Työ oli sopivan haastava ja se kartutti ammattitaitoani sekä vahvisti ongelmanratkaisutaitojani. Taustaa verkko-ohjelmoinnista oli suhteellisen vähän, ja työssä pääsin tutustumaan verkko-ohjelmoinnissa käytettyihin teknologiapaketteihin sekä Kotlin-ohjelmointikieleen, jota en koskaan ennen ollut käyttänyt. Docker tuli myös tutuksi palvelimen sekä liikenteen seurantasovelluksen ollessa dockerisoitu, eli niiden kehitysympäristö riippuvuuksineen oli tehty Docker-konttiin. Projekti vahvisti myös ymmärrystäni REST -rajapinta-arkkitehtuurista. Haastavin ongelma kehitysvaiheessa oli kuvakoordinaattien ja karttakoordinaattien välinen kalibraatio. Kalibraatiovaiheessa huomasin, että pienetkin virheet pisteiden välillä moninkertaistuivat karttapohjalle piirretyssä reitissä.

On hieno huomata, että valmiita ohjelmakirjastoja löytyy laidasta laitaan – matemaattisesti ja geometrisesti haastaville laskutoimituksille ja muunnoksille on valmiiksi olevia kirjastoja, jotka mahdollistavat esimerkiksi opinnäytetyössä kehitetyn sovelluksen toiminnan.

# LÄHTEET


1. Tampereella testattiin Ratikan täyttöasteen mittaamista kameratekniikalla. Smart Tampere. 2018. Saatavissa: https://smarttampere.fi/tampereella-testattiin-ratikan-tayttoasteen-mittaamista-kameratekniikalla/. Hakupäivä 26.3.2020.

2. Back-end Architecture. Codecademy. Saatavissa:https://www.codecademy.com/articles/back-end-architecture. Hakupäivä 7.2.2020.

3. What is backend development? Guru99. Saatavissa: https://www.guru99.com/what-is-backend-developer.html. Hakupäivä 24.2.2020.

4. Server. 2019. Computer Hope. Saatavissa: https://www.computerhope.com/jargon/s/server.htm. Hakupäivä 6.2.2020.

5. Improve Server Response Time. 2018. Google. Saatavissa: https://developers.google.com/speed/docs/insights/Server. Hakupäivä 10.3.2020.

6. Database. 2009. Saatavissa: https://techterms.com/definition/database. Hakupäivä 17.2.2020.

7. Server-side programming. 2019. MDN contributors. Saatavissa: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction. Hakupäivä 27.2.2020.

8. REST API: What is it, and what are its advantages in project development? 2016. BBVAOPEN4U. Saatavissa: https://bbvaopen4u.com/en/actualidad/rest-api-what-it-and-what-are-its-advantages-project-development. Hakupäivä 3.3.2020.

9. HTTP. 2019. MDN contributors. Saatavissa: https://developer.mozilla.org/fi/docs/Web/HTTP. Hakupäivä 16.2.2020.

10. HTTP response status codes. 2019. MDN contributors. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status. Hakupäivä 1.3.2020.

11. Introducing JSON. Saatavissa: https://www.json.org/json-en.html. Hakupäivä 2.2.2020.

12. Developer survey results. 2019. Stack Overflow. Saatavissa: https://insights.stackoverflow.com/survey/2019#technology-_-programming-scripting-and-markup-languages. Hakupäivä 12.2.2020.

13. A Practical Guide to Choosing between Docker Containers and VMs. 2020. Weaveworks. Saatavissa: https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms. Hakupäivä 7.3.2020.

14. Fetch. 2020. JavaScript.info. Saatavissa: https://javascript.info/fetch. Hakupäivä 16.2.2020.

15. OpenCv official API documentation 4.2.0. 2019. Saatavissa: https://docs.opencv.org/4.2.0/d9/d0c/group__calib3d.html. Hakupäivä 17.2.2020.

16. The Homography transformation. 2013. CorrMap. Saatavissa: http://www.corrmap.com/features/homography_transformation.php. Hakupäivä 1.2.2020.

17. Google Cloud Platform. 2020. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Google_Cloud_Platform. Hakupäivä 9.3.2020.