OULUN AMMATTIKORKEAKOULU

Tuomas Sipola

WEB-KEHITTÄJÄN PÄIVÄKIRJA

WEB-KEHITTÄJÄN PÄIVÄKIRJA

Tuomas Sipola
Opinnäytetyö
Syksy 2019
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Tuomas Sipola
Opinnäytetyön nimi: Web-kehittäjän päiväkirja
Työn ohjaaja: Jukka Jauhiainen
Työn valmistumislukukausi ja -vuosi: Syksy 2019 Sivumäärä: 50

Opinnäytetyö on päiväkirjamuotoinen tutkielma, jonka tarkoituksena on kertoa työstä web-kehittäjänä eri asiakasprojekteissa. Opinnäytetyö koostuu lähtötilanteen esittelystä, projektien kuvauksesta, päiväkirjasta ja pohdinnasta.

Lähtötilanteen kuvaus kertoo kirjoittajan alkutilanteesta asiakasprojekteissa sekä kuvataan työyhteisöä ja vuorovaikutusta työpaikalla. Projektien kuvauksessa käydään läpi alkutilanne ja mitä on tarkoitus kehittää seurantajakson aikana. Päiväkirja muodostuu seurantajaksosta, joka on jaettu viikko-osuuksiin. Jokaisen viikko-osuuden kohdalla kerrotaan päiväkohtaisesti työstä web-kehittäjänä. Viikko-osuuden lopussa esitetään analyysi, jossa kerrataan viikon aikana esille tulleita ongelmia ja niiden ratkaisuja. Viimeisenä päiväkirjassa on pohdintaosuus, jossa käydään läpi kirjoittajan kokemuksia päiväkirjamuotoisen opinnäytetyön tekemisestä sekä pohditaan seurantajakson aikana esille tulleita ajatuksia web-kehittäjän työstä.

Toimeksiantajana toimii yritys, jonka toimipisteellä työskentely pääasiassa tapahtuu. Päiväkirjan seurantajakso on 3.9.–25.10.2018.

Seurantajakson aikana esille tuli web-kehittäjän työn monipuolisuus ja se, kuinka tärkeä rooli kokemuksella on työssä. Kokemuksen kasvaessa työ web-kehittäjänä helpottuu, koska pystyy osittain soveltamaan aikaisempia ratkaisuja uusissa projekteissa. Lisäksi tärkeä havainto oli, että työ kehittäjänä tulee aina vaatimaan kohtuullisen paljon uusien asioiden opiskelua ja tiedon etsimistä. Myös oma alan harrastuneisuus on tärkeää omien taitojen kehittymisen kannalta.

Asiasanat: web-kehittäjä, päiväkirja, ohjelmistokehitys, Ruby on Rails

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software development

Author: Tuomas Sipola
Title of thesis: Diary of Web developer
Supervisor: Jukka Jauhiainen
Term and year when the thesis was submitted: Fall 2019 Number of pages: 50

The thesis is a diary-based study, which purpose is to describe working as web developer in various client projects. The thesis can be divided in to three main chapters.

The first chapter is about project descriptions, developer's working environment and interactions in a workplace. Projects descriptions tells mainly about what has been done in a previous project and what will be developed in the future projects. Second chapter consists of diary-part, which tells about working as a web-developer in daily basis. At the end of every week, there is analysis-part, which purpose is to summarize the daily problems and solutions. Third chapter consists of developer's own thoughts about making the thesis and working as a web developer.

The employer is a company which is located in Kempele and where the working mainly happens. The follow up period for this theses is 3.9.–25.10.2018.

Working as a web-developer a few important findings emerged. How eventually experience has an important role to play in development process and how much different technical skills are actually needed. Eventually web-developer's own personal projects are very important when building up programming skills.

Keywords: web developer, diary, software development, Ruby on Rails

SISÄLLYS

KÄYTETYT KÄSITTEET JA LYHENTEET

Apache ActiveMQ	Avoimeen lähdekoodin perustuva viestinvälityssovellus.
C++	Keskitason ohjelmointikieli, jota käytetään mm. resurssirajoitteisessa ympäristössä.
Docker	Ohjelma, joka suorittaa käyttöjärjestelmätason virtualisointia ja ajaa ohjelmia omissa sovelluskonteissaan.
DriveLooper	Sovellus ja web-palvelu, joka mahdollistaa automaattisen ja älykkään ajotiedon hallinnan.
eAutokoulu	Web-palvelu autokouluille ajo-opetuksen järjestämiseen.
Elasticsearch	Avoimeen lähdekoodiin perustuva hakumoottori.
GDAL	Sovelluskirjasto geospatiaalisen tiedon käsittelyyn.
HTTP	Lyhenne sanoista Hypertext Transfer Protocol. Selaimet ja WWW-palvelut käyttävät HTTP-protokollaa tiedonsiirtoon.
HTTPS	Laajennus HTTP-protokollaan, jonka tarkoitus on salata internetselaimen ja palvelimen välinen liikenne.
Integer	Ohjelmointikielissä käytetty muuttujatyyppi. Tarkoittaa kokonaislukua.
JavaScript	Korkean tason ohjelmointikieltä, jota käytetään pääasiassa webympäristössä.
JSON	Lyhenne sanoista JavaScript Object Notation. Avoimen standardin tiedostomuoto, jota käytetään tiedonvälitykseen.

JWT Token	Lyhenne sanoista JSON Web Token. Avoimen standardin menetelmä käyttöoikeustietueiden hallinnoimiseen eri ohjelmistojen välillä.
Kontena	Alusta, jonka avulla sovelluskehittäjät voivat helposti rakentaa ja hallita mikropalveluita.
Käsittelijä (Controller)	Rails-sovelluksen komponentti, joka vastaa verkkoselaimen HTTP-pyyntöjen käsittelystä.
LHR	Meta System server Java- komponentti, joka vastaa telematiikkalaitteilta tulevan datan käsittelystä Apache ActiveMQ -jonoon.
Mikropalvelu	Tarkoittaa itsenäisesti kehitettäviä ja hallittavia sovelluskomponentteja. Mikropalvelun tehtävänä on hoitaa yhden tiukasti rajatun kokonaisuuden sovelluksesta.
MVC-arkkitehtuuri	Ohjelmistoarkkitehtuurityyli, jossa ohjelma jaetaan kolmeen osaan: malliin (Model), näkymään (View) ja käsittelijään (Controller).
Makefile	Tiedosto, joka sisältää sarjan ohjeita, jonka avulla voidaan kääntää ja linkittää lähdekoodi suoritettavaksi ohjelmaksi.
Malli (Model)	Rails-sovelluksen komponentti, joka kuvaa sovelluksen tiedon tallentamista, ylläpitoa ja käsittelyä.
Migraatio-tiedosto	Tiedosto, jonka avulla voidaan luoda ja päivittää relaatiotietokannassa olevia tauluja Rails-sovelluksessa.
Näkymä (View)	Rails-sovelluksen komponentti, joka määrittää ulkoasun ja tietojen näyttämisen selaimessa.

OBD Data	Tarkoittaa ajoneuvoissa olevan telematiikkalaitteen lähettämää sensori- tai vikakoodi-datapakettia.
Polygoni	Tarkoittaa yksittäistä kartalla näkyvää viivojen rajamaa aluetta.
PostgreSQL	Avoimen lähdekoodiin perustava relaatiotietokannan hallintajärjestelmä.
Postman	Sovellus web-rajapintojen testaamiseen.
QGIS	Avoimeen lähdekoodiin perustuva ohjelma, jolla voidaan helposti näyttää ja editiota geospatiaalista dataa.
REST	Lyhenne sanoista Representational State Transfer. HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen.
Ruby on Rails	Ruby-ohjelmointikieleen perustuva ohjelmistokehys, joka pohjautuu MVC-arkkitehtuuriin.
RubyMine	**I**ntegroitukehitysympäristö Ruby- ja Ruby on Rails-sovelluksien kehittämiseen.
Shapefile	Avoin vektoripohjainen formaatti, jota käytetään geospatiaalisen tiedon tallentamiseen paikkatietojärjestelmissä.
UUID	Lyhenne sanoista Universally Unique Identifier, joka tarkoittaa 128-bittistä numeroa, jota käytetään tietojen tunnistamiseen tietokonejärjestelmissä.
UpCloud	Yritys, joka tarjoaa pilvipalveluratkaisuja.
Xcode	Integroitu kehitysympäristö macOS-käyttöjärjestelmälle.

1 JOHDANTO

Tämä opinnäytetyö kertoo päiväkirjan muodossa web-kehittäjänä työskentelemisestä 40 päivän ajalta. Jokaisen viikon lopussa esitetään yhteenveto tehdyistä tehtävistä ja niiden ratkaisuista. Viimeisenä on pohdintaosuus, jossa kirjoittaja käy läpi kokemuksiaan työstään web-kehittäjänä. Päiväkirja on kirjoitettu aikavälillä 3.9.–25.10.2018.

Toimeksiantaja toimi Looper-IT (Looper-IT 2019), joka on pieni ohjelmistoyritys Kempeleestä. Looper-IT:n erikoisalaa on älykkäiden ajonhallintasovellusten kehittäminen mobiililaitteisiin ja web-ympäristöön. Looper-IT:n päätuote on DriveLooper-sovellus älykkääseen ajonhallintaan (Drivelooper 2018).

Tein työtä web-kehittäjänä usean eri projektin parissa. Projektit käsittävät mm. uuden autoilupalvelun kehittämisen asiakkaalle sekä eAutokoulu-palvelun jatkokehittämisen (eAutokoulu 2019).

Työtehtävissäni tarvitsin osaamista useista eri osa-alueista, kuten JavaScript, C++- ja Ruby-ohjelmointikielistä. Tämän lisäksi tarvitaan tuntemusta web-sovellusten arkkitehtuureista sekä relaatiotietokannoista. Työympäristöni koostui eri sidosryhmistä, joita ovat asiakkaat sekä yhteistyökumppanit.

2 LÄHTÖTILANTEEN KUVAUS

2.1 Oman nykyisen työn analyysi

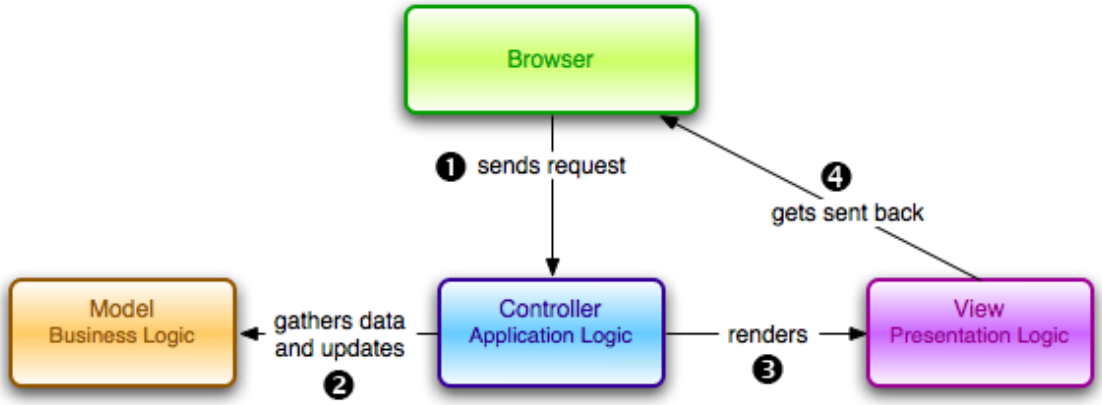
Työtehtäviini kuuluvat vanhojen projektien jatkokehittäminen, niiden ylläpitäminen sekä uusien projektien toteuttaminen. Teen työtäni web-kehittäjä-nimikkeellä.

Vastaan eAutokoulu-palvelun jatkokehittämisestä sekä ongelmatilanteiden selvittämisestä. eAutokoulu-palvelu on otettu käyttöön elokuussa 2018 ja sen on tarkoitettu autokouluille asiakashallintaan ja ajo-opetuksen järjestämiseen. Tehtäviini kuuluvat uusien ominaisuuksien kehittäminen asiakkaan vaatimusten mukaisesti, palvelun ylläpitäminen ja ongelmatilanteiden selvittäminen.

eAutokoulu-palvelun uusien ominaisuuksien kehittäminen tapahtuu yhteistyössä asiakkaan kanssa, jossa aluksi määritellään tarvittavat vaatimukset lisättävälle ominaisuudelle. Tämän jälkeen vastaan ominaisuuden kehittämisestä vaatimusten pohjalta käyttäen Ruby on Rails -ohjelmistokehystä (Ruby on Rails 2019). Ongelmatilanteet tarkoittavat yleensä virheiden korjaamista koodista tai pieniä muutoksia tietokannataan.

Uuden asiakasprojektin myötä olen päässyt rakentamaan REST-rajapintoja mikropalveluina, joiden kehittämisestä minulla ei ole aikaisempaa kokemusta (REST 2019).

Työtehtävissäni tarvitsen monipuolista osaamista ja tuntemusta ohjelmointikielistä sekä web-arkkitehtuureista. Varsinkin MVC-arkkitehtuurin tunteminen on oleellinen osa työtäni web-kehittäjänä (MVC 2019). Myös web-kehityksen yleisempien kielien, kuten JavaScript- ja CSS-perusteiden hallinta on oleellista. Näiden lisäksi tarvitaan osaamista relaatiotietokannoista ja niiden toimintaperiaatteista. Kuvassa 1 on esitetty MVC-arkkitehtuuri.



KUVA 1. MVC-arkkitehtuuri (Goodrich 2017).

Tärkein ominaisuus, jota tarvitsen sovelluskehittäjänä työskentelemisessä, on mielestäni halu etsiä uutta tietoja ja kyky soveltaa sitä. Tätä taitoa tarvitaan päivittäin, kun etsitään ratkaisuja ongelmatilanteisiin tai ohjelmointiesimerkkejä siitä, miten uusi ohjelmointikirjasto voidaan ottaa käyttöön projektissa.

Mielestäni osaamiseni taso on kohtuullinen suhteessa työni vaatimuksiin. Välillä kuitenkin huomaan, että tarvitsisin enemmän kokemusta ja osaamista tiettyjen työtehtävien suorittamiseen. Kokemuksen puuttuminen näkyy selkeimmin silloin, kun kysymyksessä on laaja tehtävä ja aikataulu on tiukka. Joudun käyttämään paljon työajastani tarvittavan informaation etsimiseen. Kokemuksen myötä tämä on kuitenkin helpottanut, koska aikaisempia ratkaisuja voidaan soveltaa uudestaan monessakin tilanteessa. Myös virheiden osalta kokemuksella on suuri merkitys. Kokemuksen myötä teet vähemmän virheitä ja koodista tulee selkeämpää. Virheitä ei tule kuitenkaan pelätä, koska ne tarjoavat hyvän mahdollisuuden oppia ja kehittää omia taitojaan.

2.2 Sidosryhmät työpaikalla

Sidosryhmät työssäni koostuvat pääasiassa työpaikkani kehitystiimistä, yhteistyökumppaneista ja asiakkaista, joille olemme toteuttamassa projekteja. Käytän suurimman osan ajastani oman kehitystiimin parissa, mutta joudun myös välillä osallistumaan erilaisiin kehityspalavereihin asiakkaiden kanssa. Näissä kehityspalaverissa käydään yleisesti läpi asiakkaan vaatimuksia siitä, miten heidän tuotteitaan kehitetään eteenpäin. Lisäksi pidämme välillä kehityspalavereita yhteystyökumppaneiden kanssa.

2.3 Vuorovaikutustaidot työpaikalla

Työni web-kehittäjänä on pääasiassa melko itsenäistä, mutta tarvitsen hyviä vuorovaikutustaitoja oman kehitystiimin sekä asiakkaiden kanssa kommunikoimiseen. Hyvät vuorovaikutustaidot ovat tärkeitä, koska niiden avulla voidaan ennaltaehkäistä ongelmia projektin aikana. Kehitystiimin sisällä on tärkeää tietää, mitä tehdään ja milloin, jotta omat ratkaisut eivät vaikuta haitallisesti muiden kehittäjien työhön. Asiakkaan kanssa keskusteltaessa on tärkeää saada ymmärrys siitä, mitkä ovat heidän vaatimuksensa ja miten pystymme yhdessä miettimään mahdollisimmat hyvät ratkaisut. Web-kehittäjän on tärkeää pystyä kertomaan asiakkaalle, mikäli heidän esittämänsä ratkaisu on huono tai parempia vaihtoehtoja on saatavilla. Tällä tavoin pystytään asiakkaalle rakentamaan hyvä lopputuote.

3 PROJEKTIEN ALKUTILANNE

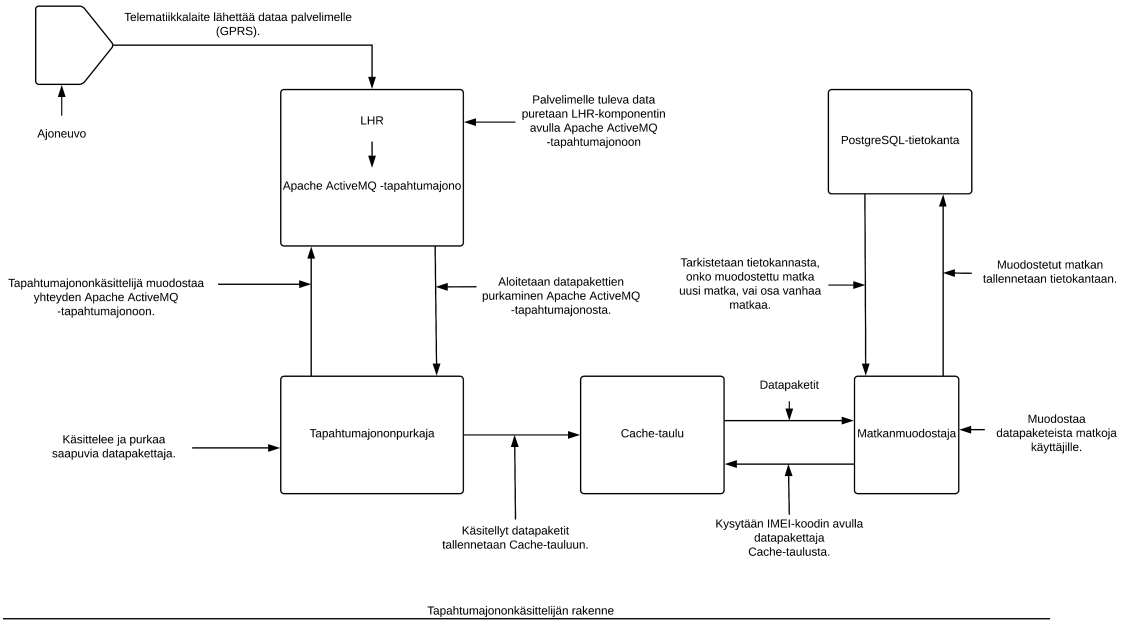
Luvussa esitellään aikaisemman yritysprojektin lopputulos ja sen kehittäminen eteenpäin. Samalla myös käydään läpi, mitä REST-rajapintoja rakennettiin ja kehitettiin uutta asiakasprojektia varten.

3.1 Tapahtumajononkäsittelijä

Looper-IT on ollut aikaisemmin mukana toteuttamassa ajotapapilottia. Pilotin tarkoituksena oli kerätä informaatiota kuljettajien ajotavoista. Kerättyä informaatiota pystyi tarkastelemaan web-sovelluksen avulla kuljettajakohtaisesti. Käyttäjä pystyi tarkastelemaan ajettua matkaa kartalla ja näkemään hänelle annetut pisteet mm. ajotyylistä.

Datan kerääminen tapahtui autoon kiinnitettävän telematiikkalaitteen avulla. Telematiikkalaite lähetti kerätyn datan palvelimelle, jossa se käsiteltiin ja tallennettiin Apache ActiveMQ -komponentin tapahtumajonoon odottamaan jatkokäsittelyä (Apache ActiveMQ 2019). ActiveMQ-tapahtumajonosta data purettiin Ruby on Rails -ohjelmistokehyksellä rakennetun sovelluksen avulla tietokantaan ja tallennettua dataa käsiteltiin myöhemmin muodostaen ajotapa-analyysia käyttäjille.

Yritysprojektissa rakennettiin vastaavanlainen kokonaisuus käyttäen C++-kieltä ja samalla karkeasti arvioitiin komponenttien suorituskyky eroa. C++-komponentti käytti osittain samoja ratkaisuja, kuin aikaisempi Rails-sovellus. Uudessa C++-ratkaisussa käytettiin myös Apache ActiveMQ -tapahtumajonoa tallentamaan telematiikkalaitteilta tuleva data PostgreSQL-relaatiotietokantaan (PostgreSQL 2019). C++-kielellä toteutetun tapahtumajonokäsittelijän rakenne on esitetty kuvassa 2.



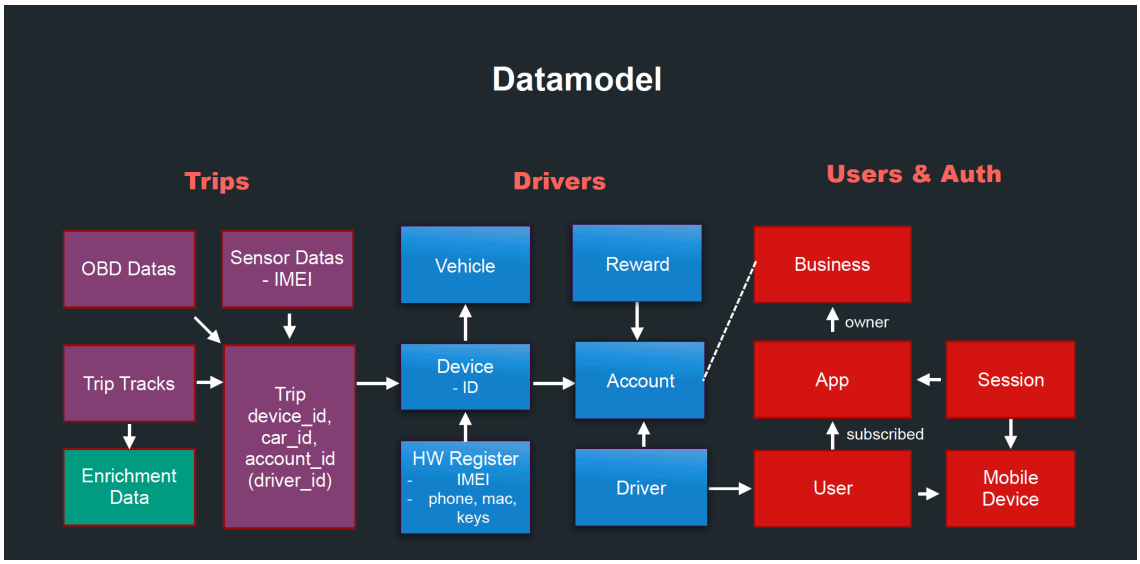
KUVA 2. Tapahtumajononkäsittelijän rakennekaavio

3.2 Tapahtumajononkäsittelijän muutokset

Tapahtumajononkäsittelijään tehtiin muutoksia asiakkaalle rakennettavaa uutta palvelua varten. Projektissa tietokantarakenne muuttui, joten nämä muutokset täytyi ottaa huomioon tapahtumajononkäsittelijässä. Selkein muutos oli, että käyttäjän tai ajoneuvon informaation käsittely täytyy tehdä REST-rajapinnan kautta. Aikaisemmin sama informaatio on hankittu suoraan relaatiotietokannasta. Uusi projekti rakennettiin useista mikropalveluista, jolloin myös luonnollisesti tapahtumajononkäsittelijä täytyy rakenteensa ulkopuolinen data hankkia mikropalveluista REST-rajapintojen välityksellä.

3.3 Trip-API- ja Driver-API-rajapinnat

Asiakkaalle kehitettävän autoilualustan myötä rakennettiin uudet REST-rajapinnat. Nämä rajapinnat ovat nimeltään Trip-API- ja Driver-API-rajapinnat. Kummatkin rajapinnat toteutettiin erillisinä mikropalveluina käyttäen Rails-ohjelmointikehystä. Kuvassa 3 on esitetty REST-rajapintojen tietomallien rakenne.



KUVA 3. Trip- ja Driver-tietomallien rakenne.

Trip-API-rajapinta vastaa kokonaisuudessaan pelkästään matkaan liittyvien tietojen hallinnasta. Rajapinnan avulla voidaan tallentaa paikkatietopisteet ja matkan informaatio tietokantaan. Lisäksi rajapinnasta voidaan kysyä matkoihin liittyvää informaatiota HTTP-kyselyiden avulla.

Driver-API-rajapinta vastaa käyttäjätietojen, ajoneuvotietojen ja telematiikkalaitteiden hallinnasta. Rajapinnan avulla luodaan käyttäjät, telematiikkalaitteet ja ajoneuvot sekä tarvittaessa pyydetään näihin liittyvää informaatiota matkatietojen rikastamiseen.

3.4 eAutokoulu

Looper-IT on aikaisemmin toteuttanut EDlooper-palvelun, joka oli tarkoitettu ajo-opetuksen järjestämiseen (Älykästä ajonseurantaan ja ajo-opetusta 2016). Palvelu oli tuotevariantti Drivelooper-palvelusta. EDlooperia käytettiin uuden ajokorttilain valmistautumiseen ja oli osana trafin kokeilulupaa niin sanottua 10+10-mallia. 10+10-mallissa oppilaalla oli lupa suorittaa kuljettajantutkinto suoritettuaan 10 tuntia ajo- ja teoriaopetusta sekä itseopiskelua. Kokeilussa tutkittiin, miten 10+10-malli vaikuttaa kuljettajantutkinnossa menestymiseen ja nuorten liikenneturvallisuuteen. Kokeilu suoritettuun 2017–2018. Syksyllä 2018 EDlooper uudelleen brändättiin eAutokouluksi.

Seurantajakson aikana eAutokouluun kehitettiin asiakkaan vaatimusten mukaisesti uusia ominaisuuksia ja korjattiin havaittuja virheitä. Uusien ominaisuuksien lisääminen aloitettiin opettajan näkymästä, jossa opetuskalenteriin lisättiin enemmän informaatiota valitusta kalenteritapahtumasta.

Myöhemmin korjattiin palvelun käyttämiseen liittyviä ohjelmointivirheitä sekä muokattiin toiminnallisuutta paremmaksi.

KUVA 4. EDlooper-palvelu (Älykästä ajonseurantaan ja ajo-opetusta 2016).

4 PÄIVÄKIRJARAPORTOINTI

4.1 Viikko 1

Maanantai 3.9.2018

Tänään aloitin tapahtumajononkäsittelijän muokkaamisen autoilualustaa varten. Komponentti on rakennettu ensimmäisen yritysprojektin aikana Looper-IT:lle ja sen tehtävänä on ollut muodostaa matkoja ajoneuvojen lähettämän datan perusteella. Kerätyt matkat on tallennettu PostgreSQL-tietokantaan. Tapahtumajononkäsittelijä on pyörinyt Looper-IT:n Amazon AWS -palvelimella (Amazon AWS 2019). Tarkoitus on alussa tehdä tapahtumajononkäsittelijästä uusi mikropalvelu ja siirtää se UpCloud-pilvipalveluun osaksi uutta projektikokonaisuutta.

Uudessa projektissa tietokantarakenne muuttuu, joten tapahtumajononkäsittelijän toiminnallisuutta pitää hieman muuttaa vastaamaan uutta rakennetta. Tämä tarkoittaa muutamien funktioden muuttamista ja kokonaan uuden Driver-API-luokan lisäämistä tapahtumajononkäsittelijään. Uuden Driver-API-luokan tekemisessä tutustuin cpp-httplib-kirjastoon (yhirose/cpp-httplib 2018).

Driver-API-luokan avulla kysytään matkoja luodessa Driver-API-rajapinnalta tietoja ajoneuvosta, käyttäjätilistä sekä kuljettajasta. Kysely tapahtuu käyttäen yksilöllistä IMEI-koodia. Ajoneuvon ja käyttäjän tietoja tarvitaan, jotta tallennettu matka voidaan osoittaa tietylle käyttäjälle.

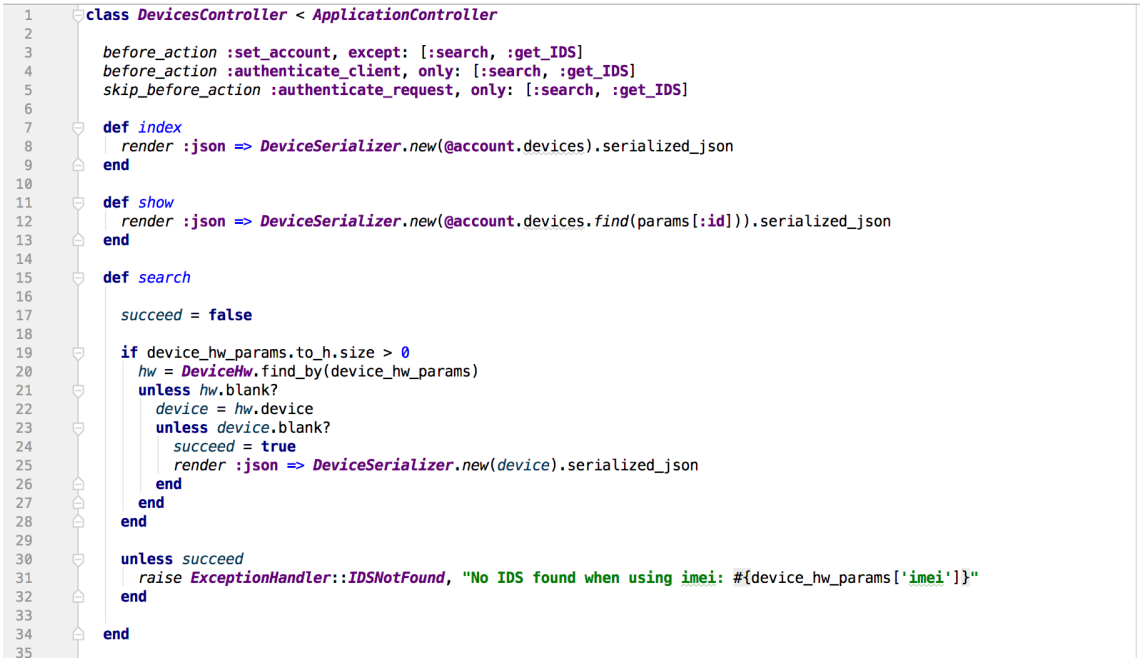
Tänään sain suurimman osan edellä mainituista muutoksista valmiiksi, mutta menee varmasti vielä osa huomisesta, jotta saan kokonaisuuden toimimaan oikein. Kuvassa 5 on esitetty Driver-API-luokan getDriver-funktio.

```
// ------------------------------------------------------------------------
// DriverApi getDriver function
// Used for getting driver_id, account_id and vehicle_id from DriverApi service
// ------------------------------------------------------------------------
//
string DriverApi::getDriver(const string& a_imei)
{
    string result = "no_data";
    string url = "";
    string query = "?imei=" + a_imei;
    url = m_device_path + query;
    m_client = new SSLClient(m_driver_api_base_url.c_str(), 443,60);

    if (m_client != NULL) {
        try {
        auto res = m_client->Get(url.c_str(),m_headers);
        if (res && res->status == 200) {
            result =  res->body;
        } else {
            if (res && res->status != 200 ) {
             string error = res->body;
             m_logger->debug("Class DriverApi getDriver function: " + error);
            }
        }
    } catch (exception &e) {
        string error = e.what();
        m_logger->error("Class TripDB connectDB Function: " + error);
        }
    }
    return result;
}
```

KUVA 5. Driver-API-luokan getDriver-funktio.

Tiistai 4.9.2018

Aloitin päivän tekemällä Driver-API-rajapintaan search-funktion, joka palauttaa kuljettajan ja ajoneuvon tiedot HTTP-vastauksessa (kuva 6). Tämän jälkeen aloitin testaamaan tapahtumajononkäsittelijän muutoksien toimivuutta paikallisesti käyttäen Driver-API-rajapintaa. Testaaminen tapahtui ajamalla paikallisesti Driver-API-rajapinta-sovellusta käyttäen RubyMine IDE:tä (RubyMine 2019) sekä tapahtumajononkäsittelijää Xcode IDE:n kautta (Xcode 2019). Tarkoitus oli nähdä, että mikäli tapahtumajononkäsittelijän kautta tulee rajapintakutsu, niin imei-koodia vastaavat tilin, kuljettajan ja ajoneuvon tiedot lähtevät HTTP-vastauksessa takaisin. Näitä tietoja tarvitaan myöhemmin matkan muodostuksessa.



```ruby
class DevicesController < ApplicationController

  before_action :set_account, except: [:search, :get_IDS]
  before_action :authenticate_client, only: [:search, :get_IDS]
  skip_before_action :authenticate_request, only: [:search, :get_IDS]

  def index
    render :json => DeviceSerializer.new(@account.devices).serialized_json
  end

  def show
    render :json => DeviceSerializer.new(@account.devices.find(params[:id])).serialized_json
  end

  def search

    succeed = false

    if device_hw_params.to_h.size > 0
      hw = DeviceHw.find_by(device_hw_params)
      unless hw.blank?
        device = hw.device
        unless device.blank?
          succeed = true
          render :json => DeviceSerializer.new(device).serialized_json
        end
      end
    end

    unless succeed
      raise ExceptionHandler::IDSNotFound, "No IDS found when using imei: #{device_hw_params['imei']}"
    end

  end

```

KUVA 6. Driver-API-rajapinnan search-funktio.

Ongelmia kuitenkin esiintyi, koska en saanut HTTP-vastauksessa mitään takaisin. Rajapinta palautti aina null-arvon, vaikka tiesin, että kyseiselle IMEI-koodille löytyvät tiedot tietokannasta.

Osan päivästä myös etsin tietoa siitä, miten C++ projektissa voidaan lukea ympäristömuuttujia. Rajapintakutsuja ja tietokantayhteyksiä varten on tarkoitus lukea tarvittavat osoitteet, tunnukset sekä salasanat docker-sovelluskontin sovelluksen määrittelytiedostosta.

Keskiviikko 5.9.2018

Tänään viimeistelin tarvittavat muutokset tapahtumajononkäsittelijään, jotta komponenttia voidaan hyödyntää autoilualustassa. Sain ympäristömuuttujien lukemisen toimimaan, koska C++:n oma standardikirjasto tarjoaa siihen hyvät välineet getenv-funktion kautta (std::getenv 2014).

Lisäksi korjasin search-funktion Driver-API-rajapinnasta, joka palautti vääränlaisia arvoja HTTP-vastauksessa. Vika löytyi lopulta tietokannan device-hw -ja device-tietomallien yhteydestä, joka oli määritelty väärin. Korjasin tietomallien yhteyden ja lisäsin uuden funktion device-tietomalliin, jotta laitetta etsittäessä saadaan HTTP-vastaukseen mukaan laitteeseen liitetty driver-id. Driver-id tarvitaan, jotta luotu matka voidaan tallentaa tietokantaan oikealle käyttäjälle. Korjasin ja lisäsin tarvittavia ympäristömuuttujia Driver-API-luokkaan ja tapahtumajononkäsittelijään.

Torstai 6.9.2018

Tämän päivän olin töissä Nativesoftin toimistolla (Nativesoft 2019). Navetivesoft on Looper-IT:n yhteistyökumppani uuden autoilualustan kehittämisessä. Tarkoitus oli jatkaa tapahtumajononkäsittelijän parissa, jotta se saadaan toimimaan UpCload-pilvipalvelussa. Aluksi oli hieman ongelmia, koska tapahtumajononkäsittelijä ei saanut yhteyttä ActiveMQ-tapahtumajonoon. Lopulta syyksi paljastui alaviivan käyttö ympäristömuuttujassa. Poistamalla alaviivat ympäristömuuttujista tapahtumajononkäsittelijä pystyi lukemaan ympäristömuuttujat oikein ja muodostamaan yhteyden ActiveMQ-tapahtumajonoon.

Seuraavana oli vuorossa tarvittavan cache-taulun tekeminen tietokantaan. Termillä ”cache” tarkoitetaan väliaikaisesti tallennettavaa informaatiota. Tässä tapauksessa ActiveMQ-tapahtumajonosta purettuja tapahtumia, joista myöhemmin muodostetaan matkoja. Vanhassa käsittelijässä cache-tietokantataulun luominen ei ollut ongelma, koska tarvittava taulu voitiin tehdä manuaalisesti AWS-palvelimelle. Uudessa autoilualustassa projektin kaikista komponenteista on tarkoitus tehdä omat erilliset mikropalvelut. Tämä tarkoittaa sitä, että tarvittavien tietokantataulujen rakentaminen pitää tapahtua automaattisesti, kun mikropalveluja ollaan ajamassa suoritukseen. Loppupäivä sujui cache-taulun määrittämisen parissa.

Perjantai 7.9.2018

Tänään viimeistelin cache-taulun luomisen tapahtumajononkäsittelijään. Tämän jälkeen tein palvelimelle tietokannan tallennettavia matkoja varten. Tämä tapahtui ajamalla RubyMine IDE:ssä migraatiotiedosto, joka luo siinä määritellyt taulut tietokantaan automaattisesti (Active Record Migrations 2018).

Tämän jälkeen oli vuorossa Postman-ohjelman avulla tilin, kuljettajan, laitteen ja ajoneuvon luominen tietokantaan (Postman 2019). Postman on työkalu REST-rajapintojen testaamiseen (Sending the first request 2018). Ohjelma toimii siten, että siinä luodaan tarvittava ympäristö testaamiseen. Luodussa ympäristössä määritellään mihin osoitteeseen lähetetään rajapintakutsuja ja minkälaisia parametreja halutaan käyttää.

Onnistuin luomaan tilin, kuljettajan, laitteen ja ajoneuvon tietokantaan. Aikaisemmin olin testannut vastaavien luomista paikalliseen tietokantaan omalla koneella. Nyt oli vuorossa näiden luominen varsinaiselle tietokantapalvelimelle.

Edelleen esiintyy ongelmia Driver-API-luokan kanssa tapahtumajononkäsittelijässä. Paikallisesti saan yhteyden muodostettua Driver-API-rajapintaan, mutta yhteyden muodostaminen pilvipalvelussa olevalle rajapinnalle ei onnistu.

Lauantai 8.9.2018

Tänään sain tapahtumajononkäsittelijän Driver-API-luokan toimimaan ja muodostamaan yhteyden Driver-API-rajapintaan. Tarvittavat muutokset olivat portin määrityksen muuttaminen sekä ympäristömuuttujassa osoitteen muuttaminen jättämällä pois HTTP-protokollan. Nyt yhteys saatiin muodostumaan rajapintaan ja sain hankittua IMEI-koodin avulla tiedot kuljettajasta ja ajoneuvosta.

Käänsin sovelluksen, latasin sovelluksesta rakennetun imagen serverille ja laitoin mikropalvelun suoritukseen. Matkankäsittelijä teki yhden matkan tietokantaan, mutta hyytyi sitten siihen. Aloin selvittämään, miksi pelkästään yksi matka tehtiin. Tiesin kuitenkin, että cache-taulussa on dataa useammalle eri matkalle. Loppupäivä sujui kyseisen ongelman selvittämisessä.

Viikon 1 analyysi

Viikko kului tapahtumajononkäsittelijän muutostöiden parissa ja uutta Driver-API-rajapintaa rakentaessa. Viikon lopputulos oli onnistunut ja mielestäni aloitin hyvin muutoksien tekemisen autoilualustaa varten, vaikka ongelmia esiintyi välillä. Suurimmat ongelmat liittyivät kuljettajan tietojen HTTP-kyselyyn Driver-API-rajapinnasta.

Viikon lopputulos oli seuraavanlainen:

- Tein tapahtumajononkäsittelijään uuden Driver-API-luokan. Luokan tarkoitus on tarjota funktiot, joiden avulla saadaan informaatiota kuljettajasta, ajoneuvosta ja tilistä. Tämä informaatio kysytään Driver-API-rajapinnalta.

- Tein uuden search-funktion Driver-API-rajapintaan. Search-funktio vastaa rajapinnalle tulevista HTTP-pyynnöistä, jotka haluavat saada informaatiota kuljettajista ja ajoneuvoista käyttäen IMEI-koodia.

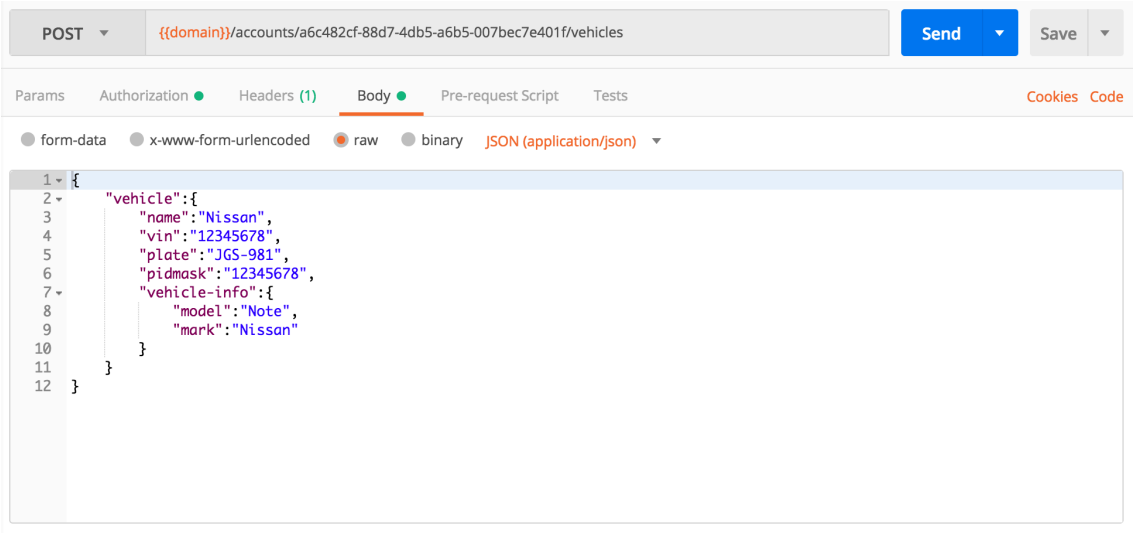
4.2 Viikko 2

Maanantai 10.9.2018

Tänään jatkoin tapahtumajononkäsittelijän matkan tallentamiseen liittyvien ongelmien selvittämistä. Päätin lisätä koodiin enemmän lokitusta ja toivoin, että siitä olisi apua ongelman selvittämisessä. Lokituksella tarkoitetaan koodipätkien lisäämistä suoritettavaan koodin, joiden tulokset kirjoitetaan erilliseen lokitiedostoon. Lokitiedostoa tarkastelemalla nähdään, mitä ohjelman suorituksen aikana on tapahtunut. Lokituksen lisääminen auttoi, koska sain selville, että ongelma liittyi olemassa olevan matkan päivittämiseen tietokannassa. Tietoja käsitellään indeksin avulla taulukosta ja lopulta virhe löytyi siitä, miten indeksiä käytetään.

Trip-tietokantataulun rakenne oli hieman muuttunut uudessa projektissa, jolloin en enää voinut käyttää vanhaa indeksointia. Tästä syystä tapahtumajononkäsittelijä ei tehnyt uusia matkoja. Korjasin indeksoinnin vastaamaan uutta rakennetta ja käsittelijä onnistui tekemään matkan cache-taulussa olevasta datasta.

Loppupäivän testasin Driver-API-rajapinnan toimivuutta Postman-ohjelmalla. Tein HTTP-pyyntöjen avulla uuden tilin, kuljettajan, ajoneuvon ja laitteen tietokantaan. Kuvassa 7 on esitetty ajoneuvon luominen.



KUVA 7. Ajoneuvon luonti Postman-ohjelmalla.

Tiistai 11.9.2018

Sairastelin tämän päivän kotona.

Keskiviikko 12.9.2018

Tänään testasin Trip-API-rajapintaa Postman-ohjelman avulla. Tein eri HTTP-kyselyitä rajapinnalle ja tarkistin, että oikeanlaista dataa saatiin HTTP-vastauksessa takaisin. Rajapinnan testaamisen jälkeen muutin tapahtumajononkäsittelijän Driver-API-luokkaa sitten, että yhteys Driver-API-rajapintaan saadaan muodostettua käyttäen HTTPS-protokollaa. Lopulta tein tarvittavat muutokset matkankäsittelijän makefile-tiedostoon, jotta ohjelman kääntäminen suoritettavaksi tiedostoksi onnistuu.

Tein myös pieniä muutoksia muihin osiin käsittelijän koodissa. Kyse oli lähinnä säädöistä, että kuinka kauan matkankäsittelijä-thread nukkuu, ennen kuin se alkaa muodostamaan matkoja.

Torstai 13.9.2018

Olin vapaalla.

Perjantai 14.9.2018

Olin vapaalla.

Viikon 2 analyysi

Kulunut viikko oli uuden autoilualustan osalta lyhyt, koska sairastelin ja olin kaksi päivää vapaalla. Viikon lopputulos oli kuitenkin onnistunut, koska sain alustavasti tehtyä tarvittavat muutokset tapahtumajononkäsittelijään sekä selvitettyä ongelmat, joita edellisellä viikolla esiintyi.

Muutoksien myötä tapahtumajononkäsittelijä on muuttunut seuraavasti:

- Aikaisemmin ajoneuvoon, kuljettajaan ja käyttäjän tiliin liittyvä informaatio on haettu suoraan tietokannasta käsittelijän sisällä. Tämä on toiminut käytännössä siten, että on otettu tietokantakirjasto käyttöön ja kirjaston kautta tehty suoraan tietokantaan kyselyitä. Nyt tätä varten on rakennettu oma luokka, jonka avulla kysytään Driver-API-rajapinnasta vastaavat tiedot HTTP-kyselynä.
- Aikaisemmin edellä mainitut tiedot olivat tyypiltään kokonaislukuja (Integer), mutta uudessa tietomallissa ne on muutettu tyypiksi (UUID). Tätä varten on pitänyt tehdä pieniä muutoksia käsittelijässä.
- Lisäksi aiemmin tarvittavat kirjastot on itse manuaalisesti käännetty ja asennettu Amazon AWS -palvelimelle, kuten myös tapahtumajononkäsittelijä. Uudessa autoilualustassa kokonaisuus on siirretty mikropalveluksi, jolloin edellä mainitut toimenpiteet pitää automatisoida.

4.3 Viikko 3

Maanantai 17.9.2018

Päivä alkoi miettimällä, miten Digiroad-tietokanta voitaisiin ottaa käyttöön uudessa autoilualustassa (Digiroad 2019). Digiroad-tietokannan avulla olisi mahdollista selvittää, millä tietyypillä käyttäjä on milloinkin ajanut. Tämä tapahtuu siten, että jokaiselle paikkatietopisteelle, joista kuljettu matka koostuu, täytyy löytää sitä vastaava tietyyppi Digiroad-tietokannasta. Lopulta käyttäjälle voidaan

esittää jokaisen matkan kohdalla, kuinka paljon matkasta on ajettu eri tietyypeillä. Tietyyppejä voivat olla mm. moottoritie, maantie tai taajama.

Tiistai 18.9.2018

Jatkoin edelleen pohtimista, miten Digiroad-tietokanta voitaisiin ottaa käyttöön projektissa. Digiroad-materiaali tulee shapefile-formaatissa, joka on suosittu avoin formaatti geospatiaalisen-tiedon tallentamiseen (Shapefile 2018). Mikäli Digiroad-materiaalia halutaan hyödyntää esimerkiksi web-kehityksessä, pitää se ensin purkaa haluttuun tietokantaan. Projektissa on tarkoitus lukea tarvittava data suoraan Elasticsearch-hakumoottoriin (Elasticsearch introduction 2018).

Kokeilin aluksi ratkaista tehtävän käyttäen apuna Rgeo -ja Rgeo-shapefile-kirjastoja Rails-ympäristössä (Andres 2014). Tämä ratkaisu osoittautui lopulta huonoksi, joten aloin etsimään uusia ratkaisuja Digiroad-tietokannan lukemiseksi Elasticsearch-hakumoottoriin.

Keskiviikko 19.9.2018

Hylättyäni Rgeo-kirjastot jatkoin etsimistä, millä tavalla pystyisin lukemaan Digiroad-materiaalin Elasticsearch-hakumoottoriin. Löysin lopulta GDAL-kirjaston, jota päätin kokeilla (Geographically Encoded Objects for ElasticSearch 2018). GDAL-kirjasto on työkalu, jonka avulla geospatiaalista dataa sisältäviä tiedostoja voidaan lukea tietokantaan käyttäen komentorivikomentoja. Loppupäivä sujui tutustuen GDAL-kirjastoon ja miten se toimii.

Torstai 20.9.2018

Tutustuttuani GDAL-kirjastoon päätin tehdä bash-skriptin, joka käynnistetään komentoriviltä. Skripti käy läpi määritellyn kansio rakenteen ja lukee tarvittavat tiedostot Elasticsearch-hakumoottoriin. Päivä sujui bash-skriptin tekemisessä ja testaamisessa (kuva 8).

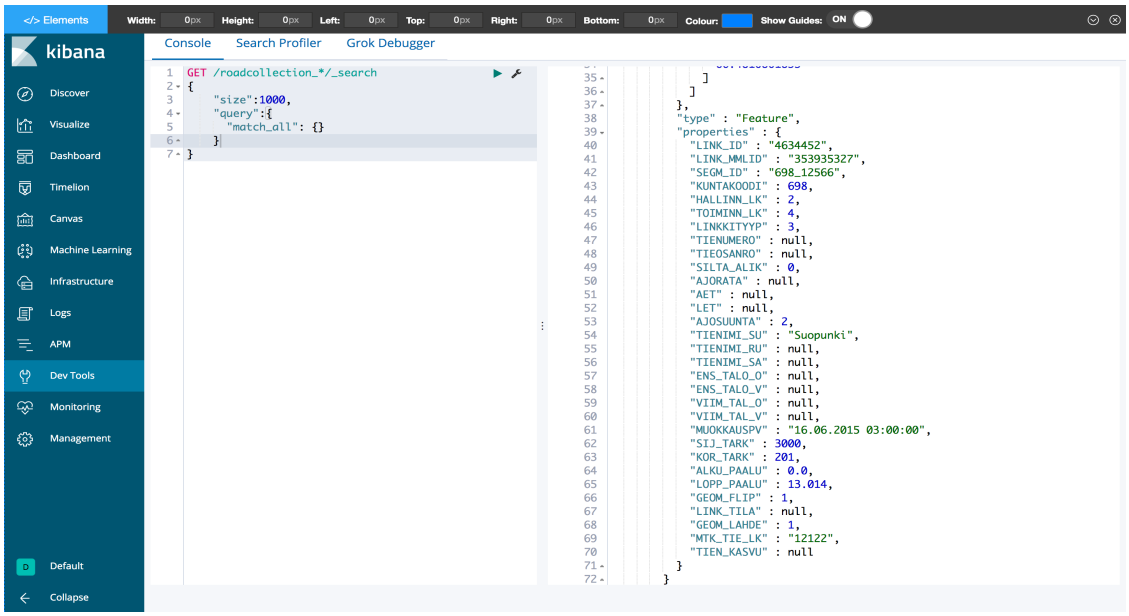
```
#!/bin/bash

#Insert this script to the root folder where the DigiRoad files are located.
#Excuting this script imports all the road files (DR_LINKKI) to the elasticsearch
#If previous road indexes exists, they will be delete from the elasticsearch
#It is possible in this stage to transform shapefile srid format
DIR=$(pwd)
counter=0

function start_file_importing() {
 echo Starting file importer..
 import_conurbation_files
 import_weather_station_data
 import_winter_speedlimit_files
 import_speedlimit_files
 import_road_files
 echo Exiting file importer.
}

function delete_road_indexes() {
 echo Deleting previous road indexes...
 curl -X DELETE ${ELASTICSEARCH_URL}/_all
 echo Finished deleting previous road indexes.
}

function import_road_files() {
 echo Starting importing road files...
 files=$(find . -type f -name "*DR_LINKKI_K.shp")
 #ogr2ogr -progress --config ES_META $DIR/map.txt -f "ElasticSearch" ${ELASTICSEARCH_URL} $file[0]
for file in $files; do
 echo "Processing file: $file"
 #ogr2ogr -progress --config ES_META $DIR/map.txt -f "ElasticSearch" ${ELASTICSEARCH_URL} $file
 ogr2ogr -progress -lco BULK_SIZE=5000000 -f "ElasticSearch" ${ELASTICSEARCH_URL} $file -nln "RoadCollection_$counter"
 let "counter++"
 echo Finished processing file.
 done
 echo Finished importing road files.
}
```

KUVA 8. Skripti shapefile-tiedostojen lukemiseen Elasticsearch-hakumoottoriin.

Perjantai 21.9.2018

Sain bash-skriptin valmiiksi ja onnistuin lukemaan Digiroad-tietokannan Elasticsearch-hakumoottoriin. Tietojen lukemisen jälkeen vahvistin Kibanan (Kibana Guide 2018) avulla, että Digiroad-tietokanta on onnistuneesti luettu.

Kibana on avoimen lähdekoodin laajennus, jolla voidaan helposti visualisoida dataa Elasticsearch-hakumoottorista käyttäen selainta. Tarkistin Kibanalla, että pystyin tekemään yksinkertaisen kyselyn Elasticsearch-hakumoottoriin ja samaan oikean vastauksen takaisin (kuva 9).



KUVA 9. Hakutyökalu Kibanassa. Haetaan Elasticsearch-hakumoottorista ensimmäiset 1000 hakutulosta.

Viikon 3 analyysi

Viikko kului pääasiassa pohtiessa, miten onnistun lukemaan Digiroad-tietokannan Elasticsearch-hakumoottoriin. Suurin ongelma liittyi sopivien työkalujen löytämiseen, jolla materiaalin lukeminen olisi mahdollista. Lopulta löysin GDAL-kirjaston ja rakensin bash-skriptin, jolla onnistuin lukemaan Digiroad-tietokannan Elasticsearch-hakumoottoriin. Viikon lopputulos oli onnistunut, koska autoilualustan kannalta tärkeä Digiroad-tietokanta saatiin otettua käyttöön.

Yhteenvetona viikon lopputulos oli seuraavanlainen:

- Hankin tietoa, miten Digiroad-tietokanta voidaan tallentaa Elasticsearch-hakumoottoriin.
- Kokeilin aluksi Rgeo -ja Rgeo-shapefile-kirjastoja Rails-ympäristössä, mutta nämä ratkaisut eivät toimineet.
- Löysin GDAL-kirjaston, jonka avulla rakensin bash-skriptin suorittamaan Digiroad-tietokannan tallentamisen. Skripti käy läpi määritellyn kansiorakenteen ja tallentaa shapefile-tiedostoissa olevan datan Elasticsearch-hakumoottoriin.
- Kibanan avulla tarkistin, että Digiroad-tietokannan tallentaminen Elasticsearch-hakumoottoriin on onnistunut.

4.4 Viikko 4

Maanantai 24.9.2018

Tänään jatkoin Elasticsearch-hakumoottoriin tutustumista. Tarkoitus on, että tapahtumajononkäsittelijä tallentaa tietyn matkan ja matkan pisteet PosgtresSQL-tietokantaan. Tämän jälkeen Trip-API-rajapinnassa ajossa oleva taustatehtävä indeksoi tallennetut pisteet Elasticsearch-hakumoottoriin.

Indeksoinnin yhteydessä pisteelle haetaan lisää informaatiota, kuten mihin tietyyppiin kyseinen paikkatietopiste kuuluu. Indeksoinnin jälkeen matkan id-arvon avulla on mahdollista hakea Elasticsearch-hakumoottorista matkalle kuuluvat paikkatietopisteet.

Pieniä muutoksia pitää tehdä tapahtumajononkäsittelijään, koska jokaisella tallennettavalla paikkatietopisteellä pitää olla uusi boolean-tyyppinen kenttä tietokantataulussa. Boolean-kenttä tarvitaan, jotta Trip-API-rajapinnassa toimiva taustatehtävä tietää, mitä pisteitä voidaan indeksoida Elasticsearch-hakumoottoriin. Lisäsin tarvittavan boolean-kentän tietokantatauluun.

Tiistai 25.9.2018

Tämän päivän teema oli edelleen matkan paikkatietopisteiden indeksoiminen Elasticsearch-hakumoottoriin. Aloin selvittämään, miten Rails-sovelluksessa saadaan taustatehtävä suoritukseen. Indeksointitaustatehtävän olisi tarkoitus olla jatkuvassa suorituksessa taustalla.
Lähdin ratkaisemaan tehtävää siten, että tein Trip-API-rajapintaan yhden Activejob-luokan, joka käynnistetään sovelluksen alustusvaiheessa komennolla ”Luokannimi.queue” (Active Job Basics 2018). Tämän jälkeen Rails-sovellus ajaa suoritettavan luokan perform-funktion heti, kun sen suorittaminen on mahdollista. Samassa funktiossa on tarkoitus kutsua luokan perform-funktiota uudelleen, jolloin käytännössä kysymyksessä olisi rekursio.

Pienen tutkistelun jälkeen tulin siihen tulokseen, että tämä ei ole hyvä ratkaisu ja, koska Trip-API-rajapintaan oli jo valmiiksi määritelty Resque- ja Redis-kirjastot, päätin ottaa kyseiset kirjastot käyttöön. Resque on Redis-tietokantaan pohjautuva Ruby-kirjasto, jonka avulla voidaan helposti luoda taustatehtäviä ja asettaa ne suoritukseen (resque/resque 2018).

Sain otettua kirjastot käyttöön ja pystyin tarkastelemaan osoitteessa localhost:3000/resque, että suoritettava taustatehtävä ilmestyi jonoon. Ongelmana oli kuitenkin vielä, että taustatehtävä suoritettiin vain kerran. Tarkoitus on suorittaa indeksointia taustalla jatkuvasti, jotta kaikki uudet paikkatietopisteet saadaan indeksoitua Elasticsearch-hakumoottoriin. Hetken etsimisen jälkeen löysin Resque-scheduler-kirjaston, joka on laajennus Resque-kirjastoon ja näyttää ratkaisevan tämän ongelman (Shah 2018).

Keskiviikko 26.9.2018

Aloitin päivän tutustumalla Resque-scheduler-kirjastoon sekä miettimään, miten sen voisi ottaa käyttöön projektissa. Aluksi oli hieman ongelmia, mutta sain lopulta kirjaston onnistuneesti käyttöön. Rakensin aluksi yhden taustatehtävän, jonka tarkoituksena on ottaa tietty määrä paikkatietopisteistä PostgreSQL-tietokannasta ja indeksoida pisteet Elasticsearch-hakumoottoriin (kuva 10). Indeksoinnin jälkeen merkitään paikkatietopisteille tietokannassa arvoksi ”true” indikoimaan, että kyseiset pisteet on käsitelty.

```
require 'track_update_task'
require 'logger'
require 'resque'

class TripTrackIndexingMainTask
  @queue = :indexing_task
  def self.perform
    last_record = Redis.current.get("last_record") || Time.now.strftime('%Y-%m-%d')
    results = TripTrack.select(:id, :unix_time).where(:elasticsearch_indexed => false).where('created_at >= ?', last_record).limit(2000).order('created_at asc')
    unless results.blank?
ask.rb  results.in_batches(of: 200) do |tracks|
        Resque.enqueue(TrackUpdateTask, tracks.map {|t| t.id})
      end
      Redis.current.set( key "last_record", results.last.created_at.to_s)
    end
  end
end
```

KUVA 10. Paikkatietopisteiden indeksointitaustatehtävä Trip-API-rajapinnassa.

Tarkastelemalla selaimessa osoitteessa localhost:3000/resque/overview näin, että indeksointitaustatehtävä tulee suoritukseen toistuvasti määritellyn 30 sekunnin välein. Lisäksi katsoin tietokannasta, että paikkatietopisteiden boolean-kenttä ”elasticsearch_indexed” saa arvon ”true”, eli pisteet oli käsitelty taustatehtävän toimesta. Huomasin kuitenkin pian, että vaikka relaatiotietokannasta tarkasteltuna pisteiden indeksointi onnistuu, niin varsinaiseen Elasticsearch-hakumoottoriin ei tule tuloksia. Aloin selvittämään ongelmaa ja samalla miettimään, miten indeksointia voitaisiin hieman nopeuttaa.

Torstai 27.9.2019

Tämä päivä jatkui edelleen indeksointitaustatehtävän parissa. Indeksointi oli edelleen puutteellinen. Elasticsearch-hakumoottoriin ei saatu luettua dataa, vaikka varsinainen taustatehtävä onnistuneesti oli suorituksessa Resque-jonossa.

Lähdin aluksi muuttamaan taustatehtävän rakennetta. Yhden taustatehtävän sijaan päätin tehdä varsinaisen päätehtävän, joka pilkkoo indeksoitavan datan osiin. Tämän jälkeen käynnistetään sivutehtäviä, jotka suorittavat pilkottujen osien indeksoimisen Elasticsearch-hakumoottoriin. Tämä oli suhteellisen helppo tehtävä. Tein päätehtävän, joka hakee rajoituksen mukaan tietokannasta indeksoitavien pisteiden id-arvot. Nämä id-arvot pilkotaan eriin, jotka sitten asetetaan sivutehtäväjonoon odottamaan käsittelyä. Sivutehtäväjonoa puretaan usealla jononpurkajalla, jolloin saadaan rinnakkaisesti indeksoitua nopeasti paikkatietopisteitä.

Yhden indeksointitehtävän jakaminen osiin onnistui hyvin, mutta datan indeksointi Elasticsearch-hakumoottoriin ei vieläkään toiminut. Lopulta syy löytyi siitä, että Elasticsearch-hakumoottorin callback-funktiot eivät toimineet odotetulla tavalla paikkatietopisteiden tietomallia päivittäessä. Muutin indeksoinnin taustatehtävää poistamalla hakumoottorin callback-funktiot käytöstä ja suoritin indeksoinnin suoraan taustatehtävän koodissa. Lopulta päivitin jokaisella kierroksella PostgreSQL-tietokantaan indeksoinnin tulokset. Näiden korjauksien myötä sain indeksoinnin toimimaan.

Perjantai 28.9.2018

Tämä päivä kului tutustuessa Elasticsearch-hakumoottoriin. Hankin tieto siitä, miten suoritetaan erilaisia hakuja hakumoottorissa (Ojo 2016). Päivän päätteeksi testasin hakuja paikallisesti käyttäen Kibanaa ja myös osittain tuotannon puolella.

Viikon 4 analyysi

Kulunut viikko sisälsi paljon uusia asioita minulle. Elasticsearch-hakumoottori ei ollut entuudestaan tuttu, joten pelkästään hakumoottoriin tutustuminen on vienyt melkoisesti aikaa. Tämän lisäksi en ollut koskaan aikaisemmin tehnyt Rails-sovellukseen taustatehtäviä. Varsinaisen tekemisen ohella aikaa on kulunut paljon tarvittavan osaamisen hankkimiseen, miten tarvittavat kirjastot kuten Resque ja Resque-scheduler saadaan käyttöön ja mitä niillä voi tehdä. Viikon lopputulos oli kuitenkin

onnistunut ja tämän myötä pystyin jatkamaan seuraavaan tehtävään, eli miten suoritetaan indeksoitujen pisteiden avulla matkojen analyysia käyttäjille.

Yhteenvetona viikon lopputulos oli seuraavanlainen:

- Lisäsin paikkatietopisteille tietokantaan boolean-kentän ”elasticsearch_indexed”.
- Otin onnistuneesti käyttöön Resque- ja Resque-scheduler-kirjastot.
- Suoritettava indeksointi tehtävä jaettiin onnistuneesti osiin, eli varsinaiseen päätehtävään, joka pilkkoo suoritettavan datan osiin sekä sivutehtäviin. Sivutehtävät suorittavat varsinaisen indeksoiminen. Tämä tapahtui hyödyntäen Resque- ja Resque-scheduler-kirjastoja.
- Muutin paikkatietopisteiden indeksointia siten, että Elasticsearch-hakumoottorin callback-funktioiden sijaan varsinainen indeksointi suoritetaan suoraan sivutehtävässä. Tämän muutoksen myötä indeksoiminen onnistui hakumoottoriin.

4.5 Viikko 5

Maanantai 1.10.2018

Viikko alkoi tieanalyysin suunnittelulla. Tarkoitus on hyödyntää Elasticsearch-hakumoottorin tuloksia, kun laaditaan analyysia matkakohtaisesti käyttäjälle. Käyttäjälle esitetään jokaisen matkan kohdalla, kuinka monta kilometriä on ajettu eri tietyypeillä. Eri tietyyppejä ovat esimerkiksi moottori- ja moottoriliikennetie.

Aloitin analyysin tekemisen siten, että aluksi tehdään yksi erillinen rajapintakutsu, jonka kautta matkatunnuksen avulla voidaan hakea matkan kaikki paikkotietopisteet Elasticsearch-hakumoottorista. Tämän jälkeen haetut pisteet käsitellään ja lähetetään HTTP-vastauksena takaisin. Itse analyysi on tarkoitus toteuttaa erilliseen luokkaan, jolloin koodi pysyy vähän siistimpänä. Analyysi on itsessään kohtuullisen yksinkertainen. Tarkoitus on saada matkan paikkatietopisteet aikajärjestyksessä. Aikajärjestystä hyödyntäen otetaan kaksi pistettä ja lasketaan näiden välinen etäisyys ja katsotaan, mihin tietyyppiin pisteet kuuluvat. Laskun tulos otetaan tietyypin mukaan talteen jokaisen pisteparin kohdalla. Tietyypin vaihtuessa kahden pisteen välimatka tallennetaan edellisen pisteen tietyypin

mukaan. Tiettyjä ongelmia esiintyy, koska esimerkiksi moottoritien alituksessa voidaan saada pisteitä siitä, että olemme ajamassa moottoritiellä, vaikka todellisuudessa kysymyksessä on tien alitus.

Tiistai 2.10.2018

Tänään viimeistelin tieanalyysin ensimmäisen version. Tietyt ongelmat, kuten moottoritien alitus on edelleen olemassa, mutta ne pitää ratkaista myöhemmin. Siirsin koodit tieanalyysin osalta tuotantoon ja aloitin testaamaan Postman-ohjelmalla analyysiin toimivuutta. Sain pelkästään virheilmoituksen HTTP-vastauksessa, että hakemaani indeksiä ei ole olemassa. Tiesin kuitenkin, että indeksi on luotu ja paikkatietopisteet on indeksoitu Elasticsearch-hakumoottoriin.

Aloitin selvittämään ongelmaa ja huomasin, että yritin hakea paikkatietopisteitä väärältä Elasticsearch-hakumoottori-instanssilta. Tieanalyysia varten Digiroad-tietokanta ja matkojen paikkatietopisteet on indeksoitu eri Elastisearch-hakumoottoreihin. Korjaamalla haun osalta, mistä käsitellyt paikkatietopisteet haetaan, sain tämän ongelman korjattua. Tein ratkaisun, että teen kaksi erillistä luokkaa, jotka hoitavat yhteyden Elasticsearch-hakumoottoreihin. Muutoksen myötä rajapinta toimi oikein, kun testasin sitä uudestaan Postman-ohjelmalla. Sain vastauksena kysytyn matkan tieanalyysin, eli montako metriä eri tietyypeillä on ajettu ja kuinka kauan sekunteina.

Tämän jälkeen aloin tutkimaan, miten analyysia voitaisiin käyttää matkakyselyn yhteydessä. Tarkoitus on, että kun käyttäjä pyytää matkoja Trip-API-rajapinnasta, niin matkatietojen yhteydessä palautetaan jokaiselle matkalle tieanalyysi.

Keskiviikko 3. 10.2018

Tämän päivä kului pitkälti tieanalyysin parissa ja siinä, miten analyysi voidaan palauttaa matkakyselyn yhteydessä takaisin. Tällä hetkellä analyysi toimii siten, että kutsutaan RoadAnalyzer-luokan funktiota, jolle annetaan parametrina yksilöllinen matka-id. Matka-id:n avulla RoadAnalyzer-luokka hakee kaikki matkan paikkatietopisteet Elasticsearch-hakumoottorista ja suorittaa analyysin pisteille. Seuraava tehtävä on selvittää, miten analyysin tulos voitaisiin ottaa talteen matkan tietomalliin.

Lähdin kokeilemaan aluksi, että voisiko matkan tietomallin sisällä suorittaa jokaiselle pisteelle analyysin, kun tietoa haetaan. En saanut tätä kuitenkaan toimimaan, joten päädyin hieman erilaiseen ratkaisuun. Tein matkan tietomalliin yhden muuttujan, johon voidaan tallentaa saatu tieanalyysi. Rails-sovelluksen käsittelijän puolella tämä toimii siten, että haetaan käyttäjän matkat tietokannasta. Tämän jälkeen jokainen matka käydään silmukassa läpi ja suoritetaan tieanalyysi. Saatu tulos tallennetaan matkan tietomallin analyysille tarkoitettuun muuttujaan. Nyt matka muiden tietojen ohella sisältää taulukon, jonka sisällä on JSON-objekti matkan analyysistä (JSON 2019).

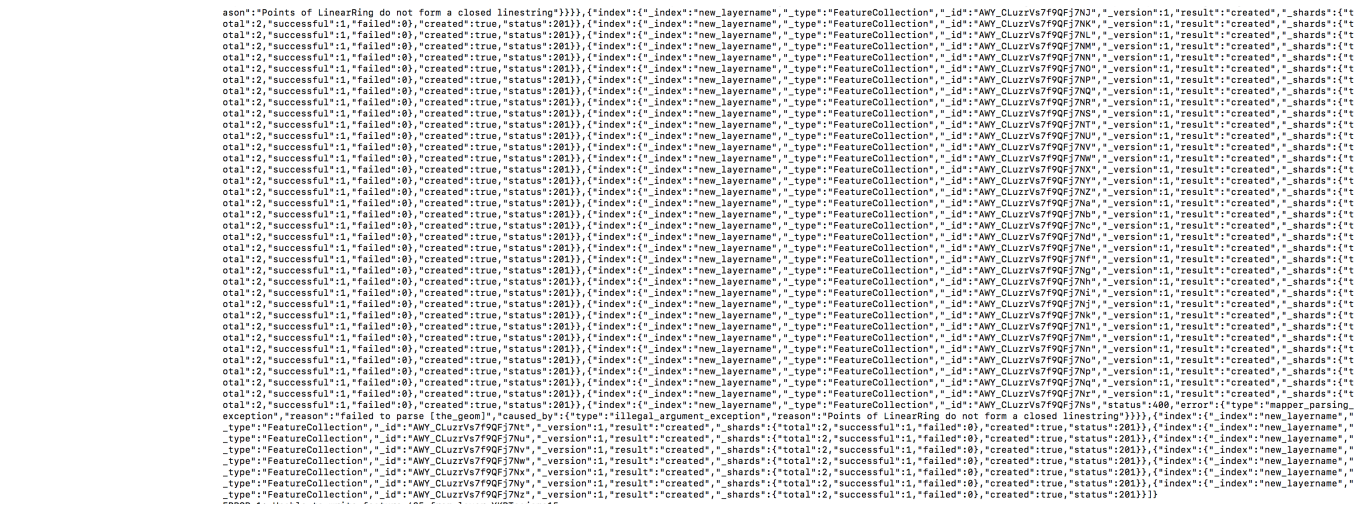
Seuraavana tehtävä olisi alkaa selvittämään, miten mahdollisesti taajamatietoja saataisiin mukaan tieanalyysiin. Kysymys on siitä, miten suoritan hakuja Elasticsearch-hakumoottorilla liittyen taajamatietoihin. Alustavasti näyttää siltä, että en voi ainakaan samankaltaista hakua käyttää kuin tietyyppien kohdalla.

Torstai 4.10.2018

Päivän aloitin selvittämällä, miten olisi mahdollista lukea taajamatiedot Elasticsearch-hakumoottoriin. Tarkoitus olisi, että käyttäjälle voitaisiin esimerkiksi mobiilisovelluksen kautta esittää tieanalyysi. Analysoitujen matkojen tiedoissa näkyisi kuinka monta kilometriä on ajettu erilaisilla tietyypeillä ja kohdistuvatko ajot taajamaan.

Olin aikaisemmin tehnyt muutoksia Trip-API-rajapinnassa matkan paikkatietopisteiden indeksointiin, joten tältä osin taajamatietojen lukeminen pitäisi onnistua. Ongelmana oli kuitenkin, että en saanut haettua dataa Elasticsearch-hakumoottorista koskien taajamatietoja. Lopulta selvisi, että osa taajamatiedoista oli lukematta Elasticsearch-hakumoottoriin. Tämä selvisi, kun yritin hakea dataa paikkatietopisteen sijasta syöttämällä paikkakunnan. Kokeilin hakusanalla ”Oulu” etsiä taajamatietoja Elasticsearch-hakumoottorista, mutta taajamatietoja ei löytynyt.

Poistin taajamatietojen indeksin Elasticsearch-hakumoottorista ja yritin lukea taajamatiedot uudestaan hakumoottoriin. Tällä kertaa en yrittänyt lukea dataa bulkkina, koska ajattelin, että mikäli siinä olisi tapahtunut jokin virhe. Datan lukeminen epäonnistui puolivälissä taajamatietojen lukemista. Yritin muutaman kerran uudestaan, mutta sain aina samanlaisen virheilmoituksen (kuva 11).



KUVA 11. Virheilmoitus taajamatietojen lukemisesta Elasticsearch-hakumoottoriin.

Seuraava tehtävä oli alkaa selvittämään, että mitä kyseinen virheilmoitus tarkoittaa ja miten se mahdollisesti korjataan.

Perjantai 5.10.2018

Tämä päivä jatkui selvittämistyöllä, eli miksi taajamatietojen lukeminen epäonnistui, vaikka aikaisemmin Digiroad-tietokanta oli onnistuneesti luettu Elasticsearch-hakumoottoriin. Kummassakin tapauksessa kysymyksessä on kuitenkin shapefile-tiedosto. Muutaman tunnin selvityksen jälkeen löysin mahdollisen syyn. Näyttäisi siltä, että taajamatietojen shapefile-tiedostot eivät ole täysin ISO 19017:2003 -standardin mukaisia. Elasticsearch-hakumoottori on puolestaan erittäin tarkka geospatiaalisen datan suhteen, joten on mahdollista, että tämän takia taajamatiedostojen lukeminen hakumoottoriin epäonnistuu. Seuraavana on tarkoitus alkaa selvittämään, että löytyykö työkaluja, joilla shapefile-tiedoston polygoneja voitaisiin yrittää korjata.

Löysin muutaman työkalun, kuten QGIS, joka perustuu avoimeen lähdekoodiin (Dunks – Brelsford – Pappas 2018). QGIS-työkalulla on mahdollista tarkastella, editoida ja analysoida geospatiaalista dataa. Sain QGIS-työkalun asennettua ja sain avattua taajamatiedoston onnistuneesti. Ajoin virheiden analysointityökalun läpi ja löysin paljon puutteita taajamatiedoston polygoneista. Yritin korjata löydetyt virheet ohjelman oletusasetuksilla ja lukea taajamatietoja uudestaan Elasticsearch-hakumoottoriin, mutta lukeminen epäonnistui edelleen. Loppupäivän yritin selvittää ongelmaa ja

tulin lopulta siihen päätökseen, että jätän ainakin tässä vaiheessa taajamatietojen käyttämisen pois.

Viikon 5 analyysi

Tämä viikko kului tieanalyysin rakentamisen parissa ja tutkin mahdollisuutta ottaa mukaan taajamatietoja tieanalyysissä. Taajamatietojen kohdalla tilanne osoittautui hieman vaikeaksi, joten taajatiedot päätettiin jättää pois toistaiseksi. Rakensin tieanalyysin ensimmäisen version valmiiksi ja se osoittautui kohtuullisen hyväksi ratkaisuksi. Huolimatta alku viikon ongelmista viikon lopputulos oli onnistunut, koska sain ensimmäisen version tieanalyysista valmiiksi.

Yhteenvetona viikon lopputulos oli seuraavanlainen:

- Ensimmäinen versio tieanalyysista valmiiksi.
- Analyysi toimii siten, että RoadAnalyzer-luokalle annetaan matkan id-arvo, jonka mukaan haetaan matkalle kuuluvat paikkatietopisteet aikajärjestyksessä Elasticsearch-hakumoottorista.
- Saadut pisteet analysoidaan siten, että otetaan kaksi pistettä, lasketaan pisteiden välinen etäisyys ja katsotaan mihin tietyyppiin pisteet kuuluvat. Saatu tulos summataan tietyypin matkalaskuriin.
- Lopulta saatu analyysi tallennetaan JSON-objektina kyseisen matkan muuttujaan.
- Yritin lukea taajamatiedot Elasticsearch-hakumoottoriin, mutta en saanut toimivaa ratkaisua valmiiksi.

4.6 Viikko 6

Maanantai 8.10.2018

Tämä päivä alkoi eAutokoulu-palvelun muutostöiden parissa. Tänään olisi tarkoitus muuttaa opettajan aloitussivun kalenterinäkymää siten, että valittuun kalenteritapahtumaan saadaan enemmän informaatiota näkyviin. Samalla lisätään mahdollisuus tehdä suoritusmerkintä valituille oppilaille suoraan kalenterimerkinnästä. Myös oppilaan tietoja olisi hyvä päästä tarkastelemaan suoraan kalenterimerkinnän kautta.

Kysymyksessä ei ollut kovin iso työ, koska kaikki tarvittavat komponentit olivat jo olemassa. Tein hieman muutoksia vanhaan koodiin, jossa kalenteritapahtuman HTML-komponentit rakennetaan. Kysymyksessä oli lähinnä uuden informaation purkaminen HTTP -vastauksesta sekä sen esittäminen tapahtumassa. Sain lopulta tarvittavat muutokset tehtyä onnistuneesti (kuva 12).

KUVA 12. Opettajan kalenterimerkintä.

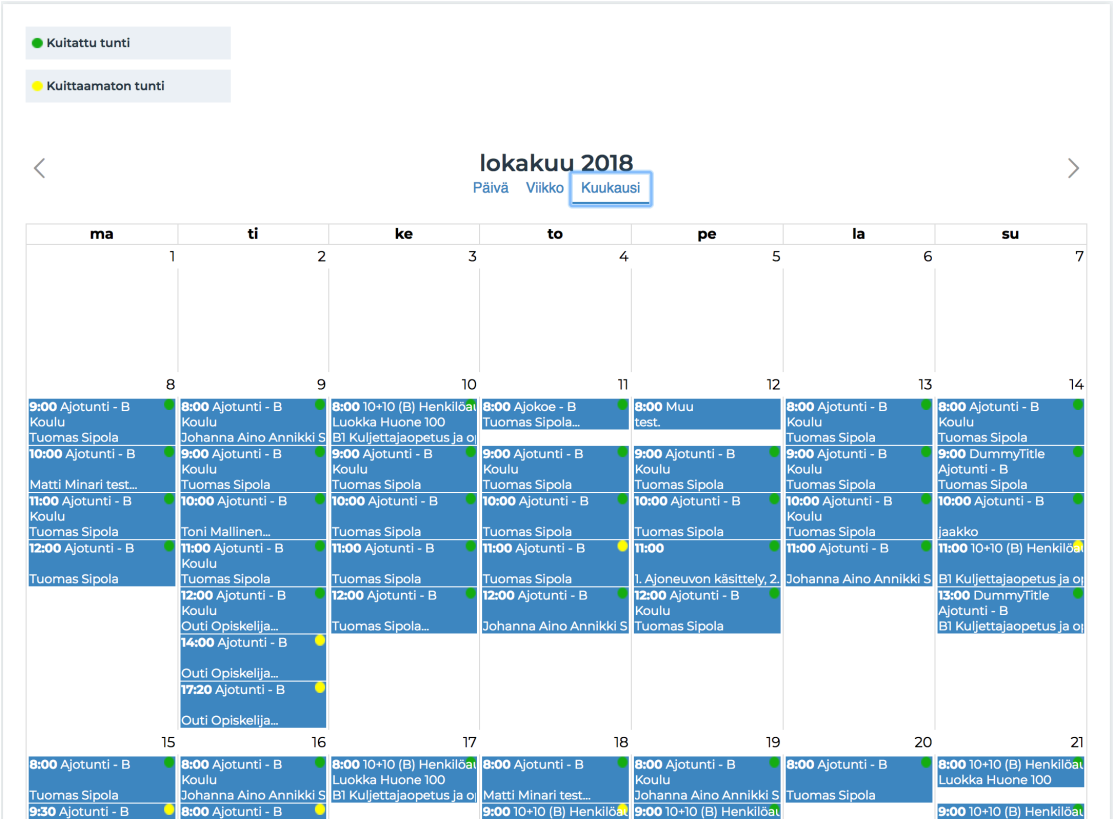
Tiistai 9.10.2018

Seuraavana oli vuorossa tehdä muutoksia opettajan kalenterinäkymään, jonka avulla opettajat voivat luoda kalenteritapahtumia. Tarkoitus on lisätä ja esittää enemmän informaatiota, jotta opettajat voivat nähdä selkeästi suoritetut ajo- ja teoriatunnit eri tapahtumien kohdalla.

Aloin miettimään, miten tehtävä mahdollisesti suoritetaan. Asiakkaan toiveen mukaisesti päädyin ratkaisuun, että kalenterissa tapahtumien kohdalla esitetään värillisenä ympyränä oikeassa yläkulmassa tapahtuman tilanne. Vihreä ympyrä tarkoittaa, että tapahtuma on suoritettu ja keltainen puolestaan tarkoittaa, että tapahtumaa ei ole vielä kuitattu. Koko päivä kului tämän tehtävän parissa.

Keskiviikko 10.10.2018

Tänään jatkoin vielä tekemällä eAutokoulu-palvelun muutostöitä. Jatkoin eilistä tehtävää, eli miten opettajalle voidaan esittää kalenterinäkymässä selkeästi eri tuntien tilanne. Sain lopulta lisättyä kuvan yksittäiseen tapahtumaan kertomaan suoritusmerkinnästä. (kuva 13).



KUVA 13. Opettajan kalenterinäkymä. Vihreä / Keltainen ympyrä kertoo suoritusmerkinnästä.

Torstai 11.10.2018

Tänään oli suunnittelupäivä, eli kävimme yhdessä oman kehitystiimin kanssa läpi, mitä vaatimusta alkaisin seuraavana tekemään autoilualustaan. Mietimme mitä muuta dataa voitaisiin hyödyntää paikkatietopisteiden kanssa ja päätimme, että tiesää olisi hyvä lisäys siihen. Tarkoitus on tehdä seuraavat muutokset paikkatietopisteiden indeksointiin, jotta tiesäätä voitaisiin hyödyntää:

- Luetaan tiesääasemat Elasticsearch-hakumoottoriin.
- Indeksoinnin yhteydessä tallennetaan mm. sääaseman nimi, koordinaatit, id sekä etäisyys paikkatietopisteelle.
- Saadun sääaseman id-arvon avulla haetaan HTTP-kyselyllä sääaseman säätiedot ulkopuoliselta REST-rajapinnalta. Tätä varten rakennetaan oma luokka, joka hoitaa reaaliaikaisen datan hakemisen ja käsittelemisen.
- Tarkoitus on kerätä mm. seuraavia säätietoja, kuten ilmanlämpötila, tienlämpötila, sateen olomuoto ja yleistä sääinformaatiota.

- Tehdään oma luokka saadun säätiedon analysointiin, jotta Elasticsearch-hakumoottoriin saadaan tallennettua käsiteltyä dataa.
- Tarkoitus on, että jokaiselle pisteelle ei erikseen haeta HTTP-kyselyn avulla säätietoja, joten otetaan avuksi cachetus. Cachetus tarkoittaa sitä, että mikäli tietyllä id arvolla ei löydy säätietoja cachesta, niin suoritetaan HTTP-kysely ja saatu data otetaan cacheen talteen. Dataa on tarkoitus säilyttää cachessa 10-minuuttia, jonka jälkeen se poistetaan. Cachetuksen avulla ei tarvitse suorittaa tuhansia HTTP-kyselyitä, mikä olisi väistämätöntä paikkatietopisteiden lukumäärän vuoksi.

Perjantai 12.10.2018

Tänään aloin tekemään muutoksia Trip-API-rajapinnan paikkatietopisteiden indeksointiin, jotta tiedot tiesäästä saadaan luettua Elasticsearch-hakumoottoriin. Ensimmäisenä tehtävänä oli saada paikallisesti luettua tiesääasemat Elasticsearch-hakumoottoriin. Tätä tietoa tarvitaan, kun haetaan paikkatietopisteen mukaan lähintä tiesääasemaa, jolta voidaan kysyä tarvittava sääinformaatio.

Säätietojen indeksointimuutosten tekeminen oli helppoa, koska minulla oli olemassa melkein valmista koodia tätä varten. Pystyin käyttämään samoja ratkaisuja, joita olin tietyyppien kyselyssä tehnyt. Suurin työ liittyi vastauksena tulevan säätietojen käsittelyyn, jotta sitä voidaan hyödyntää paikkatietopisteiden indeksoinnissa. Tätä varten aloitin tekemään HTTP-kyselyn yhteyteen funktiota, jonka tehtävä on vastauksen käsitteleminen sellaiseen muotoon, että se on helppo purkaa indeksoinnin yhteydessä omiin tietokenttiinsä. Loppu päivä sujui säätietojen käsittelyn rakentamisen parissa.

Viikon 6 analyysi

Kulunut viikko oli monipuolinen, koska viikon ensimmäiset päivät kuluivat eAutokoulu-palvelun muutostöiden parissa ja loppu viikko puolestaan työstäessä autoilualustaa. Viikon lopputulos oli onnistunut, koska sain tehtyä tarpeellisia muutoksia eAutokoulu-palveluun sekä aloitettua tiesäätietojen integroimisen matkojen paikkatietopisteisiin.

Yhteenvetona viikon lopputulos oli seuraavanlainen:

- Muutin onnistuneesti eAutokoulu-palvelun yksittäisen kalenteritapahtuman näkymää. Lisäsin näkymään enemmän tarpeellista informaatiota ja samalla mahdollisuuden oppilaan nimeä klikkaamaalla päästä oppilaan tietoja käsittelemään.
- Lisäsin uuden ominaisuuden eAutokoulu-palvelun opettajan kalenterinäkymään. Nyt opettajat näkevät, että mitkä ajotunnit on kuitattu ja mitkä ovat vielä kuittaamatta.
- Kävimme onnistuneesti kehitystiimin kanssa läpi, että mitä osa-aluetta alkaisin seuraavana tekemään autoilualustan parissa.

4.7 Viikko 7

Maanantai 15.10.2018

Tämä päivä alkoi sillä, että jatkoin viime perjantaina aloittamaani tehtävää, joka liittyi tiesäähän. Tarkoitus on, että matkan jokaisen paikkatietopisteen kohdalla on indeksoitu Elasticsearch-hakumoottoriin informaatiota lähimmän tiesääaseman lukemista.

Sain perjantaina aloittamani lisäykset valmiiksi ja seuraavana oli vuorossa Rails-cache ominaisuuden käyttöön ottaminen indeksoinnissa. Cachetusta tarvitaan, jotta jokaisen paikkatietopisteen kohdalla ei tarvitse erikseen käydä HTTP-kyselyllä hakemassa sääaseman tietoja, koska monelle pisteelle löytyy sama sääasema. Rails-cache toimii siten, että cachessa säilytetään 10 minuuttia sääasemien tuloksia, joita voidaan hyödyntää indeksoinnin yhteydessä. Mikäli halutulle sääasemalle ei löydy tuloksia cachesta, niin suoritetaan HTTP-kysely ulkoiseen REST-rajapintaan ja cachetetaan tulokset. Loppupäivä kului paikkatietopisteiden cachetuksen rakentamisessa.

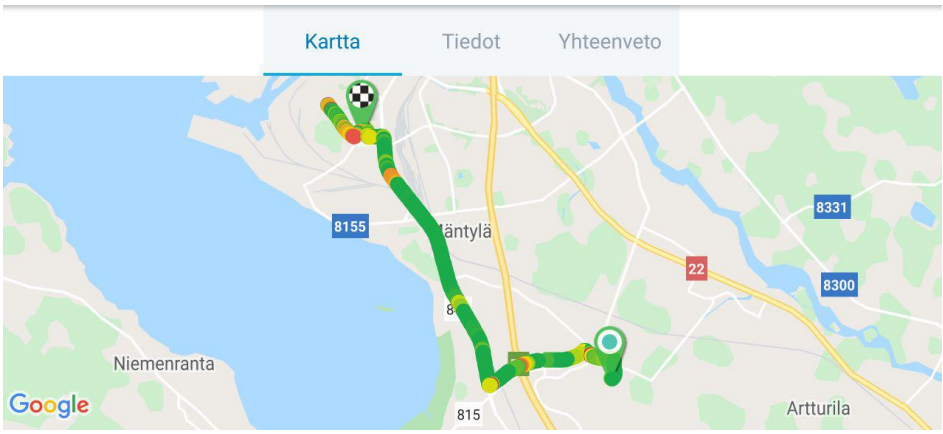
Tiistai 16.10.2018

Tänään korjasin koodia säätietojen lukemiseen liittyen. Otin eri komponenteiksi säätietojen HTTP-kyselyn ja säätietojen vastauksen purkamisen. Tein kummastakin omat luokkansa, joten koodista tuli selkeämpi tältä osin.

Testasin paikallisesti, että indeksointi ja säätietojen cachetus toimii oikein. Tämän jälkeen oli vuorossa, miten luen sääasemat tuotantoon. Aluksi oli hieman ongelmia, mutta onnistuin lopulta ajamaan mikropalvelun suoritukseen. Tämän jälkeen pystyin tuotannon palvelimella ajamaan skriptin, joka luki tiedot tiesääasemista Elasticsearch-hakumoottoriin.

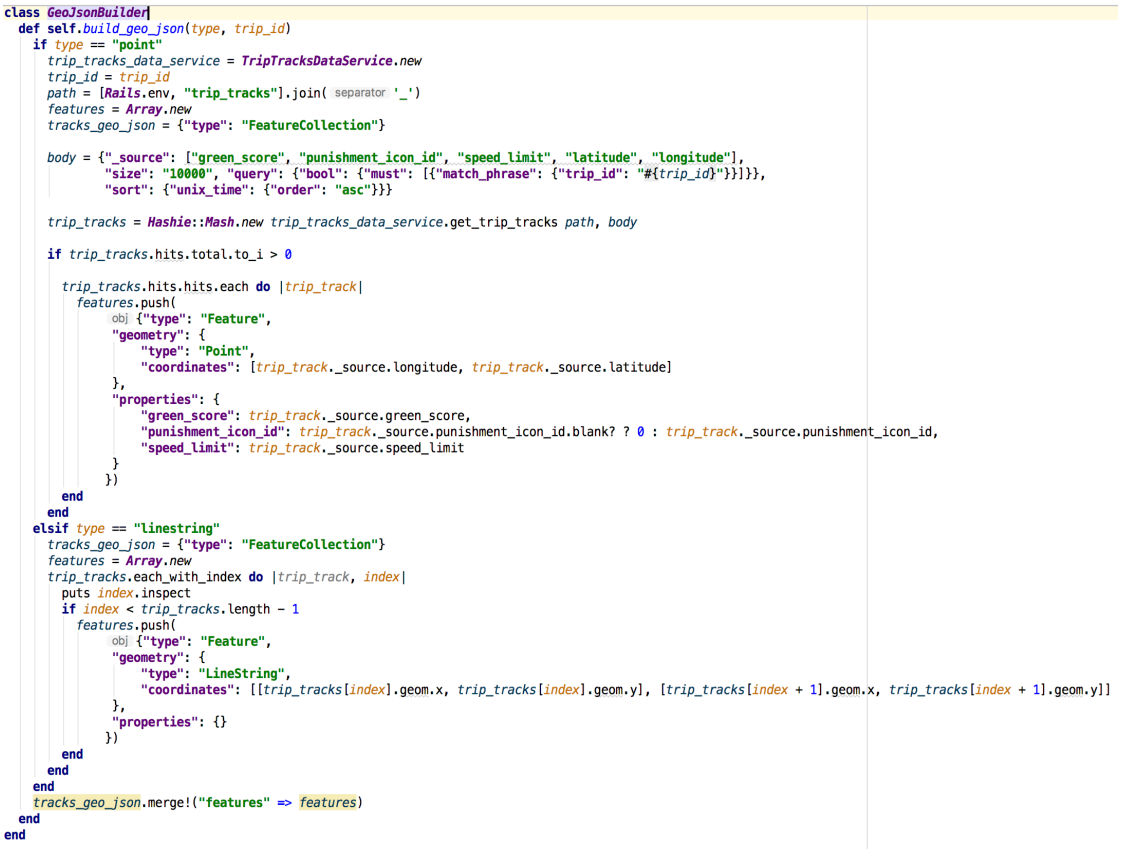
Keskiviikko 17.10.2018

Tämä päivä käynnistyi tutkimalla, miten olisi mahdollista palauttaa Trip-API-rajapinnan kautta paikkatietopisteet GeoJSON-formaatissa (GeoJSON 2018). Tarkoitus on, että matkan reitti voitaisiin helposti näyttää client-puolella syöttämällä karttaobjektille GeoJSON-formaatissa oleva HTTP-vastaus (kuva 14). Tämän pitäisi olla nopeampaa, kuin aikaisempi menetelmä, jossa karttapalvelimelle syötetään paikkatietopisteet, joiden avulla muodostetaan kartan päälle png-kuva reitistä.



KUVA 14. Indeksoinut paikkatietopisteet mobiili-clientin karttanäkymässä.

Päätin tehdä yksinkertaisen luokan, jonka avulla GeoJSON-vastaus saadaan. Kysymyksessä oli lopulta kohtuullisen helppo tehtävä, koska GeoJSON on rakenteeltaan yksinkertainen. GeoJSON-vastaus rakennetaan hakemalla matkan kaikki paikkatietopisteet Elasticsearch-hakumoottorista. Tämän jälkeen pisteet käydään läpi yksitellen ja pisteiden koordinaatit otetaan talteen yksittäiseen point-tyyppiseen objektiin ja objektit kerätään taulukkoon talteen. Kuvassa 15 on esitetty GeoJson-Builder-funktion rakenne.



```
class GeoJsonBuilder
  def self.build_geo_json(type, trip_id)
    if type == "point"
      trip_tracks_data_service = TripTracksDataService.new
      trip_id = trip_id
      path = [Rails.env, "trip_tracks"].join( separator '_')
      features = Array.new
      tracks_geo_json = {"type": "FeatureCollection"}

      body = {"_source": ["green_score", "punishment_icon_id", "speed_limit", "latitude", "longitude"],
              "size": "10000", "query": {"bool": {"must": [{"match_phrase": {"trip_id": "#{trip_id}"}}]}},
              "sort": {"unix_time": {"order": "asc"}}}

      trip_tracks = Hashie::Mash.new trip_tracks_data_service.get_trip_tracks path, body

      if trip_tracks.hits.total.to_i > 0

        trip_tracks.hits.hits.each do |trip_track|
          features.push(
                obj {"type": "Feature",
                "geometry": {
                    "type": "Point",
                    "coordinates": [trip_track._source.longitude, trip_track._source.latitude]
                },
                "properties": {
                    "green_score": trip_track._source.green_score,
                    "punishment_icon_id": trip_track._source.punishment_icon_id.blank? ? 0 : trip_track._source.punishment_icon_id,
                    "speed_limit": trip_track._source.speed_limit
                }
               })
        end
      end
    elsif type == "linestring"
      tracks_geo_json = {"type": "FeatureCollection"}
      features = Array.new
      trip_tracks.each_with_index do |trip_track, index|
        puts index.inspect
        if index < trip_tracks.length - 1
          features.push(
                obj {"type": "Feature",
                "geometry": {
                    "type": "LineString",
                    "coordinates": [[trip_tracks[index].geom.x, trip_tracks[index].geom.y], [trip_tracks[index + 1].geom.x, trip_tracks[index + 1].geom.y]]
                },
                "properties": {}
               })
        end
      end
    end
    tracks_geo_json.merge!("features" => features)
  end
end
```

KUVA 15. GeoJsonBuilder-funktio.

Päivän toisena tehtävä oli indeksoida paikkatietopisteet uudestaan lisäten indeksoinnin yhteydessä säätiedot pisteille.

Torstai 18.10.2018

Tämän päivän tehtävänä oli korjata tapahtumajononkäsittelijää ja tehdä uudistuksia siihen. Korjasin Driver-API-luokan getDriver-funktiota lisäten siihen try/catch-haaran, jotta mahdollisen virheen sattuessa ohjelma ei kaadu ja virhe saadaan otettua kiinni lokitiedostoon. Tämä oli melko suoraviivainen ja yksinkertainen lisäys olemassa olevaa koodiin.

Tämän jälkeen aloitin tekemään muutosta tapahtumajononkäsittelijän osaan, joka vastaa varsinaisesta matkan rakentamisesta ja tallentamisesta tietokantaan. Nykyisessä toteutuksessa uuden matkan viimeisen paikkatietopisteen ja mahdollisesti siihen kuuluvan matkan ensimmäisen paikkatietopisteen välistä etäisyyttä ei lasketa ollenkaan. Käytännössä tämä tarkoittaa sitä, että lopullisesta matkasta jää mahdollisesti pois muutamia kymmeniä metrejä. Korjaus on lopulta helppo, koska jo nyt matkanmuodostaja tarkistaa matkan rakentamisen lopussa, että kuuluuko kyseinen matka olemassa olevaan matkaan vai onko se täysin uusi matka. Mikäli muodostettu matka on osa

vanhaa matkaa, niin tässä yhteydessä voidaan tietokannasta saatujen tietojen avulla laskea puuttuva etäisyys. Käytin olemassa olevaa koodia tältä osin, jonka avulla lasketaan paikkatietopisteiden välinen etäisyys. Sain lopulta ongelman onnistuneesti ratkaistua.

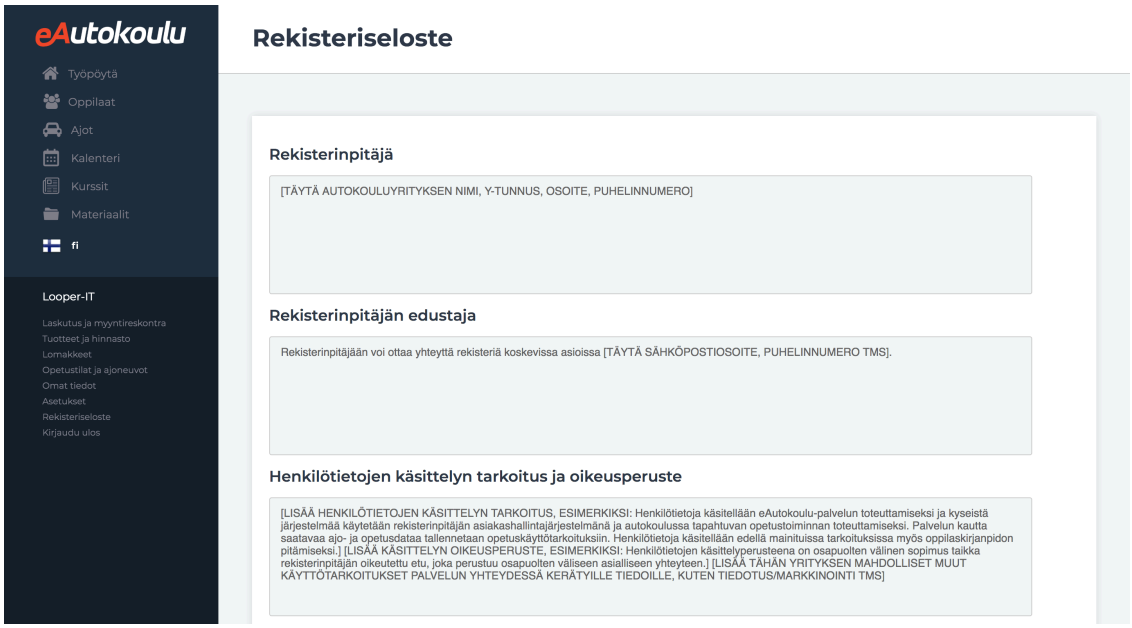
Perjantai 19.10.2018

Tämä päivä alkoi eAutokoulu-palvelun parissa. Tarkoitus on saada tarvittavia päivityksiä tehtyä palveluun. Ensimmäisenä tehtävänä oli tehdä muutoksia kalenteritapahtuman ajoneuvon valintaan. Nykyisellään kalenteritapahtumassa valitaan ajoneuvo pelkästään mallin ja merkin mukaan. Tarkoittaa sitä, että mikäli autokoululla on käytössä useampi saman merkkinen ja mallinen auto, niin kalenteritapahtumasta on mahdotonta nähdä, mikä ajoneuvo on valittuna. Tein pienen yksinkertaisen korjauksen laittamalla ajoneuvon valintaan mukaan rekisterikilven tiedot, joten jatkossa voidaan nähdä, mikä ajoneuvo on kysymyksessä tapahtumassa. Korjasin vielä oletusajoneuvon valinnan, joka ei aina toiminut oikein, eli koulun opettajan oma ajoneuvo ei ollut valittuna tapahtumaan automaattisesti. Tämä ongelma liittyi koulun ajoneuvojen hakuun ja siihen, että Rails-käsittelijän puolella hakua käsiteltiin väärin. Tämä oli kuitenkin helppo korjata ja sain tältä osin työt viimeisteltyä ja laitettua muutokset tuotantoon.

Lauantai 20.10.2018

Viikon viimeinen päivä jatkui eAutokoulu-palvelun tehtävien parissa. Tänään oli vuorossa rekisteriselosteen lisääminen palveluun, jonka avulla koulu voi laatia oman selosteen. Laadittu seloste näytetään koulun sisällä opettajille sekä myös kouluun tuleville oppilaille oppilaan omien sivujen kautta. Rekisteriselosteen tarkoituksena on antaa tietoja siitä, miten koulu käsittelee mm. oppilaiden henkilötietoja.

Tehtävänä rekisteriselosteen tekeminen oli melko yksinkertainen. Tein yksinkertaisen näkymän, jonka tehtävä on näyttää muokattava informaatio. Lopulta, kun muokkaukset halutaan tallentaa, niin se tapahtuu yksinkertaisesti painamalla tallenna-nappia. Mikäli koululla ei ole olemassa rekisteriselostetta, niin se luodaan ensimmäisellä kerralla, kun seloste sivulla käydään selaimella. Tämän jälkeen kysymyksessä on aina olemassa olevan selosteen muokkaaminen ja päivittäminen (kuva 16).



KUVA 16. Rekisteriseloste opettajan näkymässä.

Viikon 7 analyysi

Kulunut viikko oli monivaiheinen, koska tein töitä eAutokoulu-palvelun sekä autoilualustan parissa. Sain rakennettua Trip-API-rajapintaan funktion, joka palauttaa paikkatietopisteet GeoJSON-formaatissa ja muokkasin paikkatietopisteiden indeksointia Elasticsearch-hakumoottoriin. Lisäksi tein muutoksia eAutokoulu-palveluun, kuten rekisteriselosteen lisääminen ja korjasin kalenteritapahtuman ajoneuvon valinnan. Viikon lopputulos onnistunut, koska sain kaikki tarvittavat muutokset ja lisäykset onnistuneesti tehtyä.

Lopulta viikon yhteenveto on seuraavanlainen:

- Tein muutokset paikkatietopisteiden indeksointiin, jonka avulla pisteisiin luetaan lähimmän tiesäänaseman reaaliaikaiset säätiedot. Näitä tietoja ovat mm. tien- ja ilmanlämpötila.
- Lisäsin Trip-API-rajapintaan rajapinnan, jonka avulla mobiilisovellus voi matka-id:n avulla hakea matkan paikkatietopisteet GeoJSON-formaatissa.
- eAutokoulu-palvelun kalenteritapahtuman korjaaminen siten, että kalenteritapahtuman voi jatkossa paremmin liittää ajoneuvon mukaan.
- Lisäsin eAutokoulu-palvelun rekisteriselosteen, jota koulu voi vapaasti muokata. Lisäksi seloste on näkyvissä oppilaan omilla sivuilla sekä ilmoittautumisen yhteydessä etusivulla.

4.8 Viikko 8

Maanantai 22.10.2018

Tämä viikko alkoi tapahtumajonokäsittelijän parissa. Ensimmäisenä tehtävänä oli selvittää, miksi käsittelijä kaatuu, kun yhteys Trip-tietokantaan katkeaa. Aloin selvittämään ongelmaa katsomalla matkankäsittelijän lokitiedostoa. Lokitiedosto ei sinällään paljastanut mitään erikoista informaatiota ohjelman kaatumisen syystä. Ainoastaan sain selville viimeisen lokiviestin, jonka mukaan yhteyttä Trip-tietokantaan sekä Driver-API-rajapintaan ei voida muodostaa, koska palvelimen nimeä ei voida muuntaa osoitteeksi. Viaksi lopulta osoittautui pieni ohjelmointivirhe koodissa, jonka korjasin.

Tiistai 23.10.2018

Aloitin tänään uusien funktioiden tekemisen Trip-API-rajapintaan. Funktioiden on tarkoitus palvella HTTP-pyyntöjä, joiden avulla saataisiin historiatietoja matkoista tai viimeisimmän matkan tiedot. Päivän päätteeksi sain funktiot valmiiksi.

Keskiviikko 24.10.2018

Tänään aloitin tekemään tapahtumajononkäsittelijään runkoa matkan pisteytystä varten. Tarkoitus on esittää käyttäjälle pisteitä ajotyylistä, joka lasketaan kiihtyvyysanturin tiedoista matkan muodostamisen yhteydessä. Käytin mallina vanhaa pisteytystä ja tein siitä osatoteutuksen, joka toimi kohtuullisen hyvin. Tämän lisäksi rakensin matkan käsittelyn yhteyteen etäisyyden laskemisen kahden paikkatietopisteen välille, joka tallennetaan pistekohtaisesti.

Tarkoitus on, että myöhemmin tätä tietoa voidaan käyttää hyväksi, kun Elasticsearch-hakumoottoriin tehdään kysely matkan paikkatietopisteistä ja halutaan saada tieanalyysi. Aikaisempi ratkaisu hakee pisteet ryhmiteltynä tietyypin mukaan ja laskee näiden pisteiden avulla etäisyydet käymällä pisteet läpi silmukassa. Uudessa ratkaisussa laskemista ei enää tarvita, koska tapahtumajononkäsittelijä on jo laskenut yksittäisten pisteiden välisen etäisyyden. Muutoksen myötä Elasticsearch-hakumoottoriin olisi mahdollista tehdä kysely, joka summaa haluttujen kenttien tulokset yhteen, jolloin etäisyyttä ei tarvitse enää erikseen laskea Rails-käsittelijän puolella. Päivän viimeisenä tehtävä luin nopeusrajoitukset Elasticsearch-hakumoottoriin.

Torstai 25.10.2018

Olin lomalla tämän päivän, mutta tein muutaman pienen muutoksen eAutokoulu-palveluun. Lisäsin puuttuvia näkymiä, joita tarvitaan, mikäli OBD datojen käsittelyssä tapahtuu virhe. Lisäksi tein uuden funktion Trip-API-rajapintaan, jota tarvitaan paikkatietopisteiden käsittelyssä.

Viikon 8 analyysi

Kokonaisuudessaan viikko oli monipuolinen. Viikon työtehtävät koskivat pääasiassa Trip-API-rajapinnan lisäyksiä, että myös matkankäsittelijää ja siihen liittyviä uusia ominaisuuksia, kuten ajotapapisteiden laskemista käyttäjille. Viikon lopputulos oli onnistunut, koska sain muutamia tärkeitä korjauksia ja lisäyksiä tehtyä autoilualustaan, kuten nopeusrajoituksien lukeminen Elasticsearch-hakumoottoriin sekä ohjelmointivirheen korjaaminen tapahtumajononkäsittelijässä.

Lopulta viikon yhteenveto oli seuraavanlainen:

- Korjasin pienen ohjelmointivirheen tapahtumajononkäsittelijän koodissa, jotta yhteys Trip-tietokantaan ei katkeaisi.
- Lisäsin uusia funktiota Trip-API-rajapintaan, joiden avulla voidaan kysyä viimeisimmän matkan tietoja rajapinnasta.
- Aloitin tekemään matkanpisteytystä matkankäsittelijään.
- Tallensin nopeusrajoitukset Elasticsearch-hakumoottoriin.

5 POHDINTA JA PÄÄTELMÄT

Päädyin lopulta tekemään opinnäytetyön päiväkirjan muodossa, koska ajattelin, että nykyiseen elämäntilanteeseen nähden se olisi järkevin ratkaisu. Päiväkirjamuotoinen opinnäytetyö sopii hyvin työssäkäyville mielestäni, koska sen yhtenä teemana on kuvata omaa työtä ja kehittymistä seurantajakson ajan, jolloin sen tekeminen tuntuu luontevalta. Ajattelin, että päiväkirjan tekeminen olisi helppoa, mutta huomasin lopulta päiväkirjan kirjoittamisen olevan haastavaa. Varsinkin välillä jouduin miettimään, miten kirjoitan asiat, jotta teksti ei olisi liian itseään toistavaa ja asiat tulisivat ymmärretyiksi. Toisaalta jouduin useasti pohtimaan ratkaisuja eri ongelmiin useita päiviä, jolloin myös päiväkohtainen raportointi oli pakostakin itseään toistavaa.

Seurantajakson aikana pääsin kehittämään uusia toiminnallisuuksia eAutokoulu-palveluun sekä kehittämään REST-rajapintoja uudessa asiakas projektissa. Lisäksi kehitin eteenpäin aiemmin yritysprojektin parissa rakentamaani matkankäsittelijää. Projektien lopputuloksiin olen tyytyväinen, eli pystyin vastaamaan asiakkaiden vaatimuksiin ja rakentamaan tarvittavat ratkaisut.

Seurantajakson aikana käytin useita eri ohjelmointikieliä eri ohjelmointiympäristöissä, kuten JavaScript, CSS, Ruby, C++ ja Ruby on Rails. Tutustuin paljon eri työkaluihin ja teknologioihin, joita pääsin seurantajakson aikana käyttämään projekteissa, joista tärkeimmät olivat mielestäni Postman-ohjelma sekä Rails-sovelluksen taustatehtävät. REST-rajapintojen kehittämisessä ja testaamisessa käytin Postman-ohjelmaa, koska sillä voidaan helposti testata rajapintojen toimivuutta suorittamalla HTTP-pyyntöjä. Paikkatietopisteiden indeksoiminen Elasticsearch-hakumoottoriin tapahtui käyttäen Rails-sovelluksen taustatehtäviä. Web-kehittäjän on tärkeä tietää, miten web-sovelluksessa voidaan suorittaa erilaisia taustatehtäviä, koska aina ei ole mahdollista palauttaa välittömästi HTTP-vastausta käyttäjälle. Eri työkalujen ja teknologioiden kautta pääsin näkemään fullstack-kehittäjän työnkuvan kokonaisuudessaan, joka tarkoittaa ohjelmointia rajapintojen serverеiltä aina käyttäjän selaimessa toimivaan kokonaisuuteen. Uskon, että tulevaisuudessa tulen käyttämään seurantajakson aikana tutuksi tulleita työkaluja uusissa projekteissa.

Seurantajakso oli sekä erittäin mielenkiintoinen sekä vaativa johtuen useista päällekkäisistä projekteista. Tämä aiheuttikin osaltaan ongelmia työskentelyssäni, koska jouduin pohtimaan ja hankkimaan tietoja useisiin täysin erilaisiin ongelmiin. Kokemattomuuteni web-kehittäjä asetti omat

haasteensa työssäni. Osaltaan se näkyi siinä, että en aina tiennyt mitä kaikkea web-kehitysprosessi pitää sisällään ja mitä siinä pitää ottaa huomioon. Myös kokemattomuuteni näkyi selkeästi siinä, että paljon aikaa kului tarvittavan osaamisen hankkimiseen. Yleensä tämä tarkoitti tiedon hankkimista käyttäen Google-hakukonetta ja hankitun tiedon soveltamista omassa koodissa. Sanonta ”työ tekijäänsä opettaa” pitää todellakin paikkaansa ohjelmoinnin parissa. Kokemuksen kasvaessa voit soveltaa osittain aikaisempia ratkaisumalleja uusissa projekteissa, jolloin myös vähemmän aikaa kuluu tarvittavan tiedon hankkimiseen. Toisaalta työ web-kehittäjänä tulee aina olemaan uusien asioiden opiskelua ja soveltamista, mutta hankitun kokemuksen myötä se on aina helpompaa. Kokemuksen kasvaessa myös ohjelmointivirheiden määrä vähenee ja koodista tulee lyhyempää sekä selkeämpää. Virheiden tekemistä ei saa kuitenkaan pelätä, koska niitä tulee pakostakin ja ne tarjoavat lopulta hyvän mahdollisuuden oppia.

Mielestäni olisi ollut helpompaa tehdä töitä pääsääntöisesti yhden projektin parissa seurantajakson ajan, mutta monesti tämä ei luonnollisesti ole mahdollista. Toisaalta koen, että työskentely juuri usean eri projektin parissa on kehittänyt osaamista todella paljon. Olen käyttänyt useita eri ohjelmointikieliä ja miettinyt ratkaisuja erilaisiin ohjelmointiongelmiin, jolloin osaamiseni on kehittynyt monipuolisesti.

Seurantajakson aikana tietyt käsitykseni web-kehittäjän työstä ovat vahvistuneet. Tärkein asia on edelleen mielestäni halu kehittyä ja olla valmis uhraamaan aikaa tarvittavan tiedon hankkimiseen ja soveltamiseen. Tämä tarkoittaa erityisesti omaa vapaa-aikaa, jota joudutaan uhraamaan, mikäli haluaa kehittyä ohjelmoinnin parissa. Työ web-kehittäjänä ei tule olla pelkästään työtä, vaan sen tulee olla myös harrastus.

LÄHTEET

Active Job Basics 2018. rubyonrails.org. Saatavissa: https://guides.rubyonrails.org/active_job_basics.html. Hakupäivä 25.9.2018.

Active Record Migrations 2018. rubyonrails.org. Saatavissa: https://edgeguides.rubyonrails.org/active_record_migrations.html. Hakupäivä 7.9.2018.

Andres 2014. Shapefiles in Ruby on Rails. andresiglesias.net. Saatavissa: http://www.andresiglesias.net/tag/rgeo-shapefile/. Hakupäivä 18.9.2018.

Amazon AWS 2019. amazon.com. Saatavissa: https://aws.amazon.com/. Hakupäivä 6.11.219.

Apache ActiveMQ 2019. activemq.apache.org. Saatavissa: https://activemq.apache.org/. Hakupäivä 6.11.2019.

Digiroad 2019. vayla.fi. Saatavissa: https://vayla.fi/avoindata/digiroad#.XbkoO0UzbOQ. Hakupäivä 6.11.2019.

Dunks, Richard – Brelsford, Eric – Pappas, Beth 2018. Introduction to GIS Fundamentals. datapolitan.com. Saatavissa: http://training.datapolitan.com/qgis-training/Introduction_to_GIS_Fundamentals/#1. Hakupäivä 5.10.2018.

DriveLooper 2018. drivelooper. Saatavissa: https://www.drivelooper.com/. Hakupäivä 12.11.2019.

eAutokoulu 2019. eautokoulu.fi. Saatavissa: https://www.eautokoulu.fi. Hakupäivä 6.11.2019.

Elasticsearch introduction 2018. elastic.co. Saatavissa: https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html. Hakupäivä 18.9.2018.

Geographically Encoded Objects for ElasticSearch 2018. gdal.org. Saatavissa: https://gdal.org/drivers/vector/elasticsearch.html. Hakupäivä 19.9.2018.

GeoJSON 2018. geojson.org. Saatavissa: https://geojson.org/. Hakupäivä 5.10.2018.

Goodrich, Glen 2017. Understanding the Model-View-Controller (MVC) Architecture in Rails. site-point.com. Saatavissa: https://www.sitepoint.com/model-view-controller-mvc-architecture-rails/. Hakupäivä 6.11.2019.

JSON 2019. json.org. Saatavissa: https://json.org/. Hakupäivä 7.11.2019.

Kibana Guide 2018. elastic.co. Saatavissa: https://www.elastic.co/guide/en/kibana/current/in-dex.html. Hakupäivä 21.9.2018.

Looper-IT 2019. looperit.fi. Saatavissa: https://www.looperit.fi/. Hakupäivä 6.11.2019.

MVC 2019. Saatavissa: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93cont-roller. Hakupäivä 6.11.2019.

Nativesoft 2019. nativesoft.fi. Saatavissa: https://nativesoft.fi/. Hakupäivä 6.11.2019.

Ojo, Tim 2016. 23 Useful Elasticsearch Example Queries. dzone.com. Saatavissa: https://dzone.com/articles/23-useful-elasticsearch-example-queries. Hakupäivä 28.9.2018.

PostgreSQL 2019. postgresql.org. Saatavissa: https://www.postgresql.org/. Hakupäivä 6.11.2019.

Postman 2019. getpostma.com. Saatavissa: https://www.getpostman.com/. Hakupäivä 6.11.2019.

REST 2019. wikipedia.org. Saatavissa: https://en.wikipedia.org/wiki/Representational_state_trans-fer. Hakupäivä 6.11.2019.

resque/resque 2018. github.com. Saatavissa: https://github.com/resque/resque. Hakupäivä 25.9.2018.

Ruby on Rails 2019. rubyonrails.org. Saatavissa: https://rubyonrails.org. Hakupäivä 6.11.2019.

RubyMine 2019. jetbrains.com. Saatavissa: https://www.jetbrains.com/ruby/. Hakupäivä 6.11.2019.

Sending the first request 2018. getpostman.com. Saatavissa: https://learning.getpostman.com/docs/postman/launching_postman/sending_the_first_request. Hakupäivä 7.9.2018.

Shah, Naiya 2018. What is Job Scheduling?. bootreetechnologies.com. Saatavissa: https://www.botreetechnologies.com/blog/job-scheduling-with-resque-in-ruby-on-rails. Hakupäivä 25.9.2018.

Shapefile 2018. wikipedia.org. Saatavissa: https://fi.wikipedia.org/wiki/Shapefile. Hakupäivä 18.9.2018.

std::getenv 2014. cppreference.com. Saatavissa: https://en.cppreference.com/w/cpp/utility/program/getenv. Hakupäivä: 5.9.2018.

Xcode 2019. apple.com. Saatavissa: https://developer.apple.com/xcode/ide/. Hakupäivä 6.11.2019.

yhirose/cpp-httplib 2018. github.com. Saatavissa: https://github.com/yhirose/cpp-httplib. Hakupäivä 3.9.2018.

Älykästä ajonseurantaan ja ajo-opetusta 2016. Ajokortti, nro 2.S. 12–13.