Rest-rajapinnan hyödyntäminen sovellusprojektissa

Tom Ekholm

2021 Laurea

Laurea-ammattikorkeakoulu

Rest-rajapinnan hyödyntäminen sovellusprojektissa

Tom Ekholm

Tradenomi

Opinnäytetyö

Kesäkuu 2021

Laurea-ammattikorkeakoulu **Tiivistelmä**

Liiketalouden ammattikorkeakoulututkinto, tietojenkäsittely

Tradenomi (AMK)

Ekholm Tom

Rest-rajapinnan hyödyntäminen sovellusprojektissa

Vuosi 2021 Sivumäärä 32

Tämän kehittämistyön tavoitteena oli tutustua avoimiin rajapintoihin, niiden toimintaan ja toteuttaa sovellusprojekti hyödyntäen valittua rajapintaa. Työlle ei ollut toimeksiantajaa vaan aihe syntyi tekijän omasta mielenkiinnosta aihetta kohtaan.

Teoriaosuudessa tutustutaan avoimeen dataan, avoimiin rajapintoihin ja REST-arkkitehtuuriin. Tarkempaan tarkasteluun otetaan Digitrafficin avoin REST-rajapinta, joka tarjoaa tietoja Suomen rataverkosta ja sen liikenteestä. Toteuttamisosuudessa alkuun tutustutaan rajapinnan dokumentaatioon, minkä jälkeen suunnitellaan ja rakennetaan sovellus, jonka tarkoituksena on toimia mobiilina juna-aikatauluna. Lopuksi tarkastellaan projektin lopputuloksia, kulkua ja jatkokehitystarpeita.

Asiasanat: api, rest, web-kehitys, avoin data

Laurea University of Applied Sciences **Abstract**

Degree Programme in Business Information Technology

Bachelor's Thesis

Tom Ekholm

Leveraging a REST API in a Software Development Project

Year 2021 Pages 32

The objective of this Bachelor's thesis was to learn about public API's (Application Programming Interface), how they work and to realise a software project with the gained knowledge. The thesis had no commissioner and its topic was chosen because it seemed interesting.

The theoretical background starts by briefly reviewing open data, public API's and REST architecture. Once the theoretical background has been established the project portion of the thesis is introduced. Digitraffic's API is introduced and chosen as the data source for the project and it's examined further in order to utilize it. After getting acquinted with the API documentation, a rough design for a web application is devised and then carried out. Finally after completing the project, the final product, the development process and ideas for further development are reviewed.

Keywords: web development, rest, api, open data

Sisällys

1 Johdanto

Tässä opinnäytetyössä käsitellään REST-tyyppistä Digitrafficin ohjelmointirajapintaa ja sen tietojen hyödyntämistä työssä kehitettävän web-sovelluksen tietolähteenä. Ennen toteutusta tutustutaan rajapinnan dokumentaatioon ja erilaisiin projektissa käytettyihin web-kehityksen tekniikoihin. Tämän jälkeen kuvataan kehitetyn sovelluksen toimintaa ja käydään läpi projektissa vastaan tulleita ongelmia ja miten niitä ratkaistiin.

Web-rajapinnat ovat kiinnostava aihe, sillä ne tuntuvat olleen jo tovin kovassa suosiossa ja ovat iso osa web-ohjelmointia. Lisäksi verkko on täynnä erilaisia avoimia rajapintoja, joita opitun perusteella voin tulevaisuudessa hyödyntää.

2 Työn lähtökohdat

Opinnäytetyön aihe valikoitui omasta mielenkiinnosta ohjelmointia ja ohjelmointirajapintoja kohtaan. Työllä ei ollut toimeksiantajaa vaan se toteutettiin oman osaamisen kartuttamiseksi.

Työn tavoitteena oli päästä tarkemmin tutustumaan web-ohjelmointiin, sen erilaisiin tekniikoihin ja ohjelmointirajapintojen toimintaan. Avoimien rajapintojen saatavuus ja tarjonta mahdollistaa kenen vain lähteä kehittämään sovelluksia, jotka käyttävät oikean elämän dataa. Ilman avoimia rajapintoja tietolähteen löytäminen yksityishenkilönä voi olla vaikeaa ja useimmissa tapauksissa mahdotonta. Oikeastaan ainoa tapa on luoda oma tietokanta olemassa olevien tietojen perusteella tai tuottaa dataa itse, oli se sitten sensoridataa, koneellisesti tuotettua (esim. web-scraping) tai muuta.

2.1 Kehittämiskohteen kuvaus ja kehittämistavoitteet

Työssä tutustuttiin ensin avoimeen dataan ja avoimiin rajapintoihin. Tämän jälkeen perehdyttiin Digitrafficin avoimeen rajapintaan ja rakennettiin sovellus, joka hyödyntää rajapinnasta saatavia tietoja. Sovelluksen noutaa rajapinnasta junien aikataulutietoja ja tulostaa niitä käyttäjälle tarpeen mukaan. Vastaavanlaisia, huomattavasti laajempia ja parempia sovelluksia on toki olemassa mm. HSL:n kehittäminä ja ylläpitäminä, mutta suurimpana tavoitteena on oman osaamisen edistäminen.

2.2 Aihealueen rajaus

Opinnäytetyö on rajattu Digitrafficin rautatieliikenteen avoimen rajapinnan palauttamiin tietoihin ja niiden käsittelyyn. Lisäksi osana työtä suunnitellaan ja rakennetaan sovellus rajapinnan dokumentaatiota ja tietoja hyödyntäen.

2.3 Keskeisiä käsitteitä

API (Application Programming Interface) = Ohjelmointirajapinta.

HTTP (Hypertext Transfer Protocol) = Protokolla resurssien noutamiseen.

GET-metodi = HTTP metodi, joka tekee pyynnön palvelimelle täsmennetystä resurssista. Vain datan noutamiseen.

POST-metodi = HTTP-metodi, joka lähettää tietoa palvelimelle täsmennettyyn resurssiin, usein muuttaa tilaa palvelimella.

RESt (Representational State Transfer) = HTTP-protokollaan pohjautuva ohjelmointirajapintojen arkkitehtuurimalli.

Endpoint = Ohjelmointirajapinnan polku tai osoite.

ISO-8601 = Kansainvälinen standardi päivämäärän ja ajan esittämiseen.

HTML (Hypertext Markup Language) = Hypertekstin merkintäkieli.

URL (Uniform Resource Locator) = Yhdenmukainen resurssin paikannin.

3 Avoin data

Avoin data on tietoa, jonka yritys, organisaatio, viranomaiset tai mikä vain taho on tehnyt julkiseksi kenen tahansa vapaasti käytettäväksi. Tiedonjakamisen toteutus on tarjoajan vapaasti päätettävissä, kunhan sen käyttöön ei liity minkäänlaisia rajoitteita tai ehtoja. Tiedon tulee olla vapaasti tarjolla ja lisensoitu niin ettei sen hyödyntämistä tai jakamista voida rajoittaa tai kieltää. (Avoindata 2021)

Yleisiä julkisen tiedon tarjoajia ovat usein viranomaiset, sillä julkinen hallinto tuottaa ja varastoi paljon arvokasta dataa. Sen tarjoaminen vapaasti on yleishyödyllistä, sillä sen hyödyntämiselle ei välttämättä löydy resursseja, mutta laittamalla se vapaaseen jakoon kuka vain voi hyödyntää jaettua tietoa ja luoda siitä lisäarvoa. Datan jakaminen vaatii yleensä toki teknistä toteutusta ja siten taloudellisen satsauksen, mutta sen hyödyt ovat monitasoiset

alkaen julkishallinnon parantuneesta tehokkuudesta ja avoimen data hyödyntäjien taloudellisesta kasvusta aina laajempaan yhteiskunnalliseen hyvinvointiin. (Avoindata 2021; Dataeuropa 2021)

Avoimen datan jakoon käytetään nykypäivänä yleensä avoimia rajapintoja, tämän työn kannalta keskitytään internetissä käytettävään avoimeen REST-pohjaiseen Web Service-ohjelmointirajapintaan, mutta siitä enemmän myöhemmin.

Ohjelmointirajapinnat tai API:t (Application Programming Interface) ovat keino jakaa tietoa erilaisten tietojärjestelmien välillä. Niiden avulla voidaan jakaa pelkästään tietoa (dataa) tai sitten tarjota erilaisia toiminnallisuuksia. Jotta rajapintaa voitaisiin kutsua avoimeksi, tulee kolmen ehdon täyttyä: rajapinnan tulee olla avoimesti dokumentoitu, käyttöönotettava ja testattava.

Avoimesti dokumentoidulla tarkoitetaan, että rajapinnan dokumentaatio on vapaasti saatavilla ja käytettävissä ja, että sen tiedot ovat tarpeeksi yksityiskohtaiset, että ulkoinen taho eli hyödyntäjä voi ottaa sen käyttöön. Eli ohjeista pitää löytyä kaikki tarvittava rajapinnan käyttöön.

Käyttöönotettavalla tarkoitetaan, että käyttöönotto pitäisi onnistua ilman toimittajan tai ylläpitäjän apua, päivämäärästä ja kellonajasta riippumatta. Rajapinta voi vaatia esimerkiksi rekisteröintiä, mutta prosessin tulisi olla ilmainen ja edellä mainitusti, kalenterista riippumaton (automatisoitu).

Testattavalla tarkoitetaan, että rajapinnan tulee olla testattava ja tähän tulee olla tarjolla jonkinlaista testiaineistoa.

Kun rajapinta täyttää nämä kolme ehtoa, kenen vain pitäisi pystyä ottamaan se käyttöön ja sitä voidaan kutsua avoimeksi. (Avoinrajapinta 2021)

3.1 Esimerkki avointa rajapintaa hyödyntävästä palvelusta

HSL:n reittiopas (reittiopas.hsl.fi) hyödyntää Digitransit-palvelualustaa, joka on HSL:n omistama ja kehittämä palvelukokonaisuus joukkoliikenteen reititykseen. Se kokoaa HSL:n omia tietolähteitä sekä ulkoisia rajapintoja yhteen (mm. työssä tutkittava Digitrafficin rautatieliikenteen rajapinta) kokonaisuuteen. Käyttäjä syöttää palveluun lähtö- ja päätepisteen ja sovellus kertoo miten sinne päästään, kuinka pitkään matka kestää, mikä lippu matkaan tarvitaan ja paljon muuta. Palvelusta on myös mahdollisuus saada kävely- ja pyöräilyreittejä. (Digitransit 2021)

3.2 Avoimien rajapintojen käyttötapauksia

Twitter, Spotify ja monet muut it-alan jättiläiset tarjoavat tietojaan avoimien rajapintojen kautta, jolloin kehittäjät voivat rakentaa sovelluksia, jotka laajentavat alkuperäisten sovelluksien toimintojen skaalaa. Rajapintojen kautta voidaan rakentaa esimerkiksi vaihtoehtoinen käyttöliittymä tai hyödyntää sen tarjoamia tietoja ja toimintoja (osa tukee myös tiedon lähettämistä, esim. tweetin lähettäminen) erilaisiin käyttötarkoituksiin kuten vaikkapa analytiikkaan. Yleensä jos rajapinta tarjoaa mahdollisuuden muokkauksiin tai tiedon lähetykseen, se vaatii jonkinlaista autentikointia ja joskus auktorisointia ennen kuin käyttäjä pääsee rajapintaan käsiksi. Autentikointi viittaa siihen, että tarkastetaan käyttäjän identiteetti, joka voidaan toteuttaa esim. API-avaimella, jonka avulla käyttäjän kutsut rajapintaan sidotaan tiettyyn käyttäjään tai autentikointiavaimella, jolloin ongelmatilanteissa saadaan helposti selvyys ketä niitä aiheuttaa. Auktorisointi (authorization) viittaa siihen mitä tietoa ja toimintoja autentikoitunut käyttäjä pääsee tarkastelemaan ja käyttämään. (Twitter 2021; Johnson 2019)

3.3 Digitraffic

Digitraffic on Fintrafficin (TMF) tarjoama palvelu, jonka kautta on saatavissa ajantasaista liikennetietoa suomen tieverkolta, rautatie- ja meriliikenteestä. Tiedot ovat avointa dataa, jota jaetaan avointen rajapintojen kautta. (Digitraffic 2021)

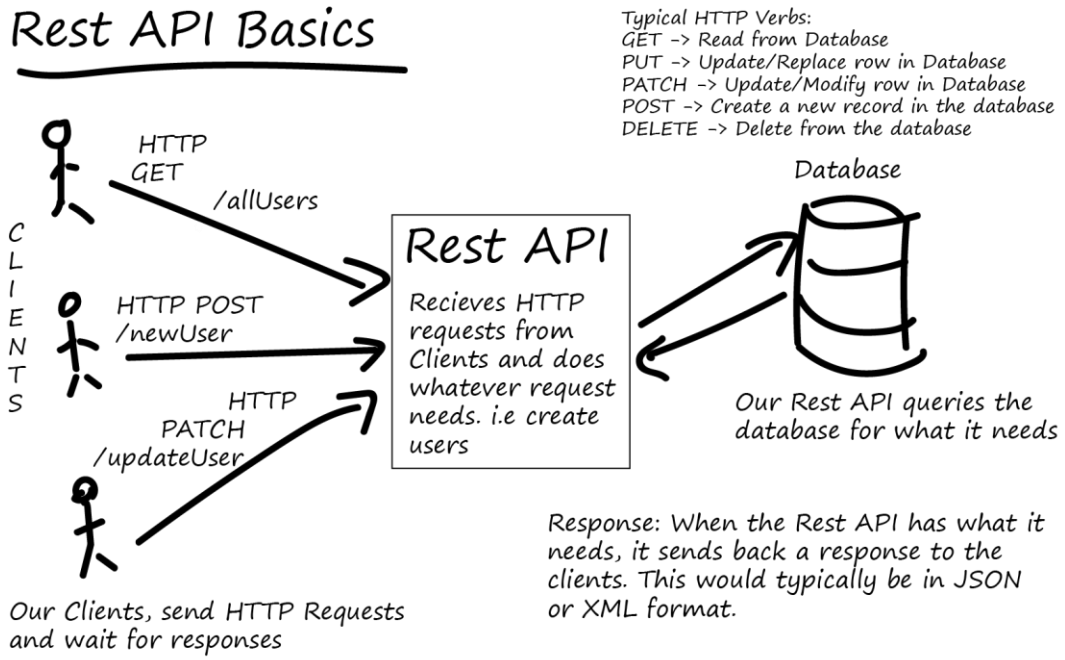
Tässä opinnäytetyössä käsitellään Digitrafficin rautatieliikenteen REST ohjelmointirajapintaa. Sen tietolähteenä toimii Fintrafficin ratakapasiteetin ja liikenteenohjauksen Liike-perheen sovellukset sekä matkustajainformaatiojärjestelmä MIKU. (Digiraffic 2021)

Rajapinta ei vaadi rekisteröitymistä, on maksuton, sille löytyy kattava dokumentaatio, joka on vapaasti saatavilla ja sen tietojen käyttölisenssi on Creative Commons Attribution 4.0. Lisenssi antaa oikeuden kopioida, muokata ja jakaa aineistoja eteenpäin joko alkuperäisessä tai muutetussa muodossa. Aineistoja voi myös yhdistää muihin aineistoihin ja käyttää sekä kaupallisiin että epäkaupallisiin tarkoituksiin. Ehtona on lähdetietojen ja lisenssin sisällyttäminen. (Digitraffic 2021)

4 REST

REST on suosittu arkkitehtuurimalli web-rajapintojen toteutukseen. Sen suosio perustuu yksinkertaisuuteen: sen ominaisuudet ovat käyttäjäystävällisiä ja käyttäjien on helppo seurata sen logiikkaa. REST:in lyhentämätön nimi ”Representational State Transfer” kertoo jo paljon sen toiminnasta, sillä rajapintaa kutsuessa palvelin siirtää (transfer) asiakkaalle

representaation (representation) pyydetyn resurssin tilasta (state). Alla kuvattuna palvelin-asiakas-mallin mukainen HTTP-pyyntö. (Avraham 2017)



Kuvio 1: Pyynnöt Rest-rajapintaan (Forbes 2017)

Arkkitehtuurimallin esitteli ensimmäistä kertaa Roy Fielding vuonna 2000 julkaistussa väitöskirjassaan. Siinä Fielding määrittelee REST:ille 6 vaatimusta, jotka tulisi täyttää, jotta rajapintaa voidaan kutsua aidosti REST:iksi.

1. Client-Server. Asiakas-Palvelin-malli: Erottamalla käyttöliittymän ongelmat tiedon säilönnän ongelmista, parannetaan käyttöliittymän siirrettävyyttä useille eri alustoille ja parannetaan skaalautuvuutta yksinkertaistamalla palvelimen komponentteja. Tärkein hyöty on erottelu, joka mahdollistaa komponenttien kehityksen itsenäisesti. Asiakas ja palvelin toimivat itsenäisesti ja ainoa vuorovaikutus niiden välillä on pyyntöjen muodossa, jonka asiakas aina aloittaa ja johon palvelin vain vastaa. (Fielding 2000, luku 5)
2. Stateless. Tilattomuus: Kommunikaatio asiakkaan ja palvelimen välillä pitää olla tilatonta. Jokainen pyyntö asiakkaalta palvelimelle täytyy sisältää tarpeellisen tiedon pyynnön ymmärtämiseen, eikä pyyntö voi hyödyntää palvelimelle tallentunutta kontekstia, vaikka sama asiakas olisi lähettänyt pyyntöä putkeen. (Fielding 2000)
3. Välimuisti. Välimuisti-rajoite vaatii, että tietoon, jota palautetaan, on määritelty selkeästi saako se jäädä välimuistiin vai ei . Jos tieto on määritelty jäämään

välimuistiin, asiakas saa luvan käyttää palautunutta dataa uudelleen vastaavissa pyynnöissä. (Fielding 2000)

4. Uniform Interface. Yhdenmukainen rajapinta. Tällä rajoitteella on 4 osaa: palvelimelle lähetetyn pyynnön pitää sisältää resurssin tunniste. Palvelimen vastauksen pitää sisältää tarpeeksi tietoa, jotta asiakas voi muokata resurssia. Jokainen pyyntö rajapintaan sisältää kaiken palvelimen vaatiman informaation, jotta se voi suorittaa pyynnön ja jokaisen palvelimen palauttaman vastauksen tulee sisältää kaikki tieto mitä asiakas tarvitsee ymmärtääkseen vastauksen. Hypermedian käyttö sovelluksen tilan moottorina. Hypermedialla tarkoitetaan linkkejä muihin medioihin (esim. kuvat, videot, teksti) eli rajoite vaatii, että vastauksien tulisi sisältää linkkejä. (Fielding 2000; Avraham 2017)
5. Layered System. Kerrostettu järjestelmä: Mahdollistaa sen, että arkkitehtuuri rakentuu hierarkkisista tasoista rajoittamalla komponenttien toimintaa niin, etteivät ne voi nähdä pidemmälle kuin tasolle, jonka kanssa ovat tekemisissä. (Fielding 2000)
6. Code-on-Demand. Ladattava koodi: tämä rajoite on vapaaehtoinen, rajapinta voi olla REST:in mukainen, vaikka se ei tarjoisikaan ladattavaa koodia. Asiakas voi pyytää koodia palvelimelta ja vastaus palvelimelta sisältää koodia, jota asiakas voi ajaa. (Fielding 2000; Avraham 2017)

5 Ohjelmointityökalut

Tässä luvussa käydään läpi työssä käytettyjä ohjelmointityökaluja. Opinnäytetyössä toteutetun projektin tekstieditorina käytettiin Visual Studio Codea. Ohjelmointikielenä käytettiin JavaScriptiä, palvelimena Nodea Express ja EJS moduulien avulla.

5.1 Visual Studio Code

Visual Studio Code on Microsoftin kehittämä tekstieditori, mikä soveltuu hyvin sovelluskehitykseen, sillä se tarjoaa tuen useille ohjelmointikielille. Se on ilmainen, kevyt asentaa ja ajaa eikä asennuksen mukana tule mitään ylimääräistä verrattuna moneen muuhun täysveriseen ohjelmointiympäristöön verrattuna. Se oli myös entuudestaan tuttu työkalu. (Visualstudio 2021)

5.2 Postman

Postman on mainio työkalu REST API:en ja niistä saatavien vastauksien tutkimiseen. Se soveltuu ehkä paremmin oman rajapinnan testaamiseen ilman tarvetta kirjoittaa ylimääräistä koodia rajapinnan toiminnan todentamiseen sekä muiden HTTP-metodien kuin GET:in käyttöön. Se on kuitenkin hyödyllinen työkalu oppia käyttämään, minkä takia sitä käytettiin projektin aikana. (Farmer 2021)

5.3 Node.js ja Express

Node.js on avoimen lähdekoodin järjestelmäriippumaton JavaScript-suoritusympäristö, joka on rakennettu Googlen Chrome-selaimeen kehitetyn V8 JavaScript-moottorin pohjalta. Ennen sen julkaisua, JavaScript-koodia ei voitu suorittaa kuin selaimen (front-end) puolella ja web-sovelluksissa palvelimen kanssa kommunikointiin tuli käyttää jotain toista ohjelmointikieltä. Node kuitenkin mahdollistaa JavaScript-koodin suorittamisen selaimen ulkopuolella ja siten myös sovelluksen palvelinpuolella (back-end), jolloin sovellus voidaan toteuttaa kokonaisuudessaan yhdellä ohjelmointikielellä. (Henderson 2019)

Noden toiminta perustuu moduuleihin, joita voidaan pitää pieninä ohjelmistokirjastoina tai apufunktioina. Siinä on asennettaessa omat sisäänrakennetut moduulinsa, mutta tarvittaessa käyttäjä voi luoda omia tai ladata muiden rakentamia moduuleja helposti Npm:n avulla (Node Package Manager). Npm on eräänlainen avoin verkkokirjasto Node-moduuleille, jonka avulla niitä voidaan julkaista ja jakaa helposti verkon yli. Npm on integroitu Nodeen, joten asentaminen käy helposti syöttämällä komento ”npm install <paketin nimi>” (npm install express) Noden komentoriville. Npm-kirjasto ja moduulien dokumentointi löytyy selaimella osoitteesta npmjs.com. Asentamisen jälkeen käyttäjän tulee vain ottaa moduuli omaan projektiinsa ja se on tämän jälkeen vapaasti käytettävissä moduulin dokumentaation kuvaamalla tavalla.

Node pystyy toimimaan palvelimena ydinmoduuliensa kanssa, mutta Express vähentää vaadittua koodimäärää huomattavasti. Express on minimalistinen, mutta joustava ja tehokas webkehityksen viitekehitys Nodelle. Se on minimalistinen, koska siinä ei tule mukana mitään turhaa ylimääräistä ja tuetut toiminnot ovat oletuksena pois päältä. Käyttäjällä on mahdollisuus valita niitä tarpeidensa mukaan, mutta häntä ei hukuteta toiminnallisuuksiin. Expressin joustavuus tulee middlewarejen käytöstä ja Noden moduulirakenteesta. Ne ovat helposti käyttöönotettavia JavaScript komponentteja mikä tekee Express-sovelluksista modulaarisia, joustavia ja jatkettavia. Tehokkuus tulee siitä, että se antaa täyden pääsyn Noden ydinrajapintoihin. Kaikki mitä voidaan tehdä Nodella, onnistuu myös Expressillä. (Yaapa 2013)

5.4 EJS-sivupohjat

Express.js tukee erilaisia sivupohjamoottoreita, joiden avulla html-sivupohjaan voidaan sivupohjamoottorin dokumentaation mukaisella syntaksilla määritellä muuttujia tai koodia, joiden avulla staattisesta sivusta tulee dynaaminen. Projektissa käytettiin EJS:ää, jota käytettäessä tiedostojen tiedostopäätteeksi tulee antaa ”.ejs”. Sivupohjamoottori alustetaan käyttöön, jotta Express tietää mitä sivupohjamoottoria käytetään ja mistä projektin kansiosta sivupohjat tulee hakea. Toimintaperiaatteena sivupohjamoottorissa on, että sivupohjaan merkitään muuttujia, joita ei tulosteta käyttäjän avatessa sivua. Vasta kun käyttäjä tekee

muutoksia sivulla, esimerkiksi syöttää hakukenttään tekstiä ja painaa hakunappia, tieto siirtyy POST-metodin avulla palvelimelle, missä lähetetty tieto prosessoidaan ja käyttäjälle palautetaan jotain tietoa takaisin sivupohjaan määriteltyihin muuttujiin. EJS mahdollistaa myös JavaScriptin suorittamisen sivupohjassa ilman script-tageja. EJS:n lisäksi suosittuja sivupohjamoottoreita ovat mm. Pug (ent. nimeltään Jade), Handlebars ja Mustache. Express käyttää oletussivupohjamoottorina Pugia, jolloin EJS pitää erikseen määritellä käyttöön. Alla pätkä koodia, jonka perusteella sovellus saa tiedon, että käytetään EJS:ää. (Expressjs 2021)

```
app.set("view engine", "ejs")
```

Kuvio 2: Otetaan käyttöön sivupohjamoottori ja määritellään EJS käyttöön. Muuttuja App on Express-moduulin ilmentymä

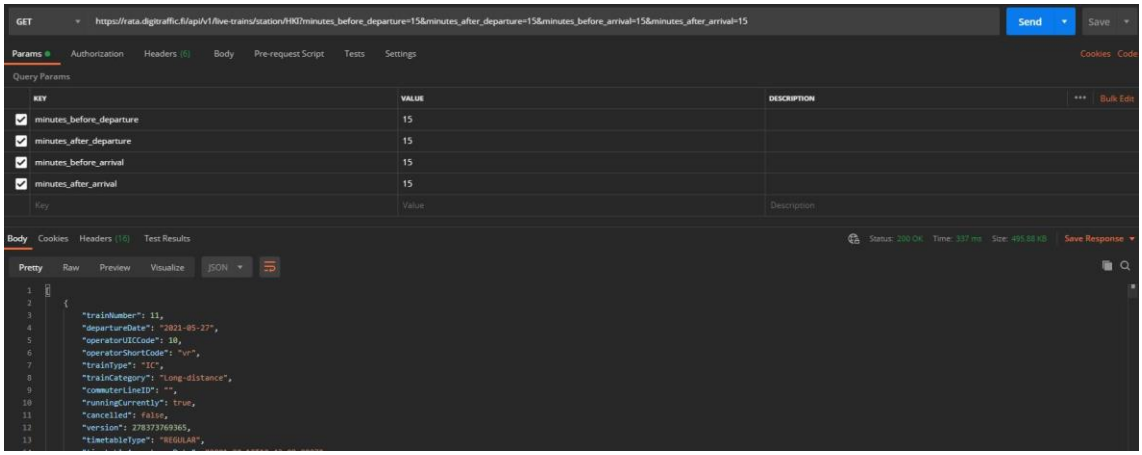
EJS asennetaan muiden Node moduulien lailla komennolla *npm install ejs*.

5.5 Nodemon

Nodemon on Node-moduuli, joka projektikansioon asennuksen ja konfiguroinnin jälkeen osaa tarkkailla muutoksia koodissa ja käynnistää sovelluksen uudelleen, kun muutoksia tallennetaan. Ilman Nodemonia tämä prosessi pitää tehdä manuaalisesti, mikä on toiminnallisuuksia testattaessa pieni vaiva. Npm-asennuksen yhteydessä komentoon annetaan ”-d” parametri, mikä asentaa moduulin kehittäjäriippuvuutena. Erona normaaliin riippuvuuteen on se, että jos projekti laitetaan jakoon tai siirretään toiseen tiedostosijaintiin, sen toimintaan tarvitut moduulit löytyvät package.json-tiedostosta ja ne saadaan asennettua helposti komennolla ”npm install”. Nodemon ei kuitenkaan ole sovelluksen toimintaan tarpeellinen, joten se halutaan eriyttää muista moduuleista. (Nodemon 2021)

5.6 Pyynnöt rajapintaan ja Axios

Jotta rajapinnasta voidaan saada tietoa ulos, on siihen tehtävä GET-pyyntö. Yksinkertaisuudessaan GET-metodia käytetään tietojen hakuun tietystä resurssista palvelimella, eli mitä URL:in takaa löytyy, kun selaimella avataan jokin web-sivu: selain lähettää GET-pyynnön palvelimelle, jossa web-sivu pyörii, jonka jälkeen palvelin lähettää tarvittavat tiedot (pyydetyn resurssin representaation) takaisin ja näyttää ne selaimessa. Digitrafficin rajapinnan tapauksessa GET-pyyntö palauttaa JSON-muotoista dataa.



Kuvio 3: Pyyntö rajapintaan Postmanin avulla ja rajapinnan vastaus

Node tukee HTTP-pyyntöjen lähettämistä ja vastaanottoa omalla standardilla HTTP-moduulillaan, mutta käyttämällä pidemmälle jalostettua moduulia, säästetään aikaa ja koodirivejä. Tässä projektissa käytettiin Axios.js:ää pyyntöjen tekoon Digitrafficin rajapintaan. JavaScript itsessään tukee Fetch API:a, jonka avulla voitaisiin ihan yhtä hyvin suorittaa pyyntöjä rajapintaan. Suurimpana erona näitä molempia kokeiltaessa oli, että Fetch palautti haetut tiedot tekstimuodossa, jolloin vastaus piti vielä parsia JSON-muotoon sen käsittelyn helpottamiseksi, kun taas Axios teki tämän automaattisesti. Lisäksi Axios on Promise-pohjainen, mikä auttaa asynkronisen koodin ajamisessa. Muita suosittuja vastaavanlaisia moduuleja ovat Request, SuperAgent, Got ja Node-fetch. (Axios 2021)

6 Projektin suunnittelu

Tässä luvussa käydään läpi sovelluksessa hyödynnettävän rajapinnan dokumentaatiota. Projektin tietolähteeksi valikoitui Digitrafficin rautatieliikenteen rajapinta, koska sen tarjoama data on ajankohtaista ja sen hyödyntämiselle on junamatkustajana perusteltu käyttötarve.

6.1 Digitraffic dokumentaatio

Digitraffic tarjoaa kahden tyyppisiä REST-rajapintoja rautatieliikenteelle: GraphQL-rajapinta ja staattiset rajapinnat. GraphQL tarjoaa käyttäjälle mahdollisuuden filtteröidä, järjestää ja päättää mitä tietoa palvelu palauttaa vastauksena. Staattiset rajapinnat palauttavat tiedot aina samassa formaatissa: vapaasti filtteröinti, järjestäminen tai määrittely ei ole mahdollista. (Digitraffic 2021)

GraphQL-rajapinta toimii loppukäyttäjän näkökulmasta hieman samalla tapaa kuin normaali tietokanta, johon tehtävät tietokantapyynnöt voidaan rajata tarkasti.

Digitrafficin staattisten rajapintojen tietojen rajaamiseen taas ei löydy muita työkaluja kuin palvelun tarjoajan antamat endpointit tai päätepisteet, joiden kautta tehdessä hakuja filtteröinti, järjestäminen ja määrittely tulee tapahtua käyttäjän omassa päässä sen jälkeen, kun tieto on haettu palvelusta.

Digitrafficin rautatieliikenteen endpointit ovat jaettu kahdeksaan osaan tai polkuun, joista jokaisella polulla on omanlaisensa dokumentaatio minkä perusteella tietoja voidaan rajapinnasta hakea.

- Junien tiedot (/trains)
- Aktiivisten junien seuranta (/live-trains))
- Junan GPS-sijainnit (/train-locations)
- Kulkutietoviestit (/train-tracking)
- Kokoopanotiedot (/compositions)
- Kulkutievaraukset (/routesets)
- Ratatyötiedot (/trackwork-notifications) ja liikenteen rajoitetiedot (/trafficrestriction-notifications)
- Metatiedot (/metadata)

Kuvio 4: Staattisen rajapinnan endpointtien jaottelu rajapinnan dokumentaatiossa

Opinnäytetyössä tehtävän projektin tarkoituksena oli keskittyä aktiivisiin juniin, jotka ovat juuri saapumassa etsitylle asemalle. Tämä tarkoittaa, että suurimmalta osin keskityttiin Aktiivisten junien seurantaan (/live-trains).

Aktiivisten junien seurannasta löytyy neljä eri perustetta tai tapaa rajata rajapinnan vastausta: liikennepaikalle saapuvat ja lähtevät junat (lukumäärärajoitus), liikennepaikalle saapuvat ja lähtevät junat (aikavälirajoitus), reittiperusteinen haku ja kohta lähtevien tai saapuvien junien seuranta. Riippuen käyttötarkoituksesta mikä vain vaihtoehto voisi toimia, mutta tämän opinnäytetyön projektissa haluttiin keskittyä Liikennepaikan saapuviin ja lähteviin juniin aikavälirajoituksella.

Aikavälirajoitettu pyyntö rajapintaan palauttaa asemalla pysähtyvistä junista viimeksi lähteneet tai saapuneet, tai seuraavaksi lähtevät tai saapuvat. Parametreillä voidaan rajoittaa lähtevien ja saapuvien junien aikahaarukkaa. (Digitraffic 2021)

Dokumentaation esimerkki Helsingin Rautatieaseman pysäkkiin kohdistettu hausta:

https://rata.digitraffic.fi/api/v1/live-trains/station/HKI?minutes_before_departure=15&minutes_after_departure=15&minutes_before_arrival=15&minutes_after_arrival=15

Käyttäjän määrittelemät ehdot alkavat kohdasta ”HKI”, joka on lyhennekoodi Helsingin rautatieasemalle, jonka perusteella haku rajapintaan saadaan kohdistettua tiettyyn asemaan

tai pysäkkiin. Tämän jälkeen tulevat aikavälimääritykset sekä muut hakuparametrit, joille selitteet löytyvät rajapinnan dokumentaatiosta.

Hakuehdot

	Nimi	Formaatti	Oletusarvo	Esimerkki	Selitys
⊕	station	string		"HKI"	Aseman lyhenne. Esimerkiksi HKL, TPE, PSL. Lista lyhenteistä löytyy täältä.
⊕	minutes_before_departure	positive integer, 0-1440		20	Kuinka monta minuuttia juna näytetään ennen sen lähtöä.
⊕	minutes_after_departure	positive integer, 0-1440		20	Kuinka monta minuuttia juna näytetään sen lähdön jälkeen.
⊕	minutes_before_arrival	positive integer, 0-1440		20	Kuinka monta minuuttia juna näytetään ennen sen saapumista.
⊕	minutes_after_arrival	positive integer, 0-1440		20	Kuinka monta minuuttia juna näytetään sen saapumisen jälkeen.
?	include_nonstopping	true/false	false	true	Palautetaanko aseman ohi pysähtymättä ajavat junat.
?	train_categories	string		Commuter,Long-distance	Junalaji-rajaus pilkulla eroteltuna. Lista junalajeista löytyy täältä
?	version	positive integer		159123295871	Versiorajoitus. Palauttaa kaikki junat, jotka ovat muuttuneet sitten version-version. Jos versionumeroa ei anneta, palautetaan uusimmat tiedot.

⊕ Pakollinen ? Vapaaehtoinen

Paluuarvo

Kuvio 5: Aikavälirajoitetun pyynnön hakuehdot dokumentaatiossa

Esimerkki palauttaa siis kaikki junat, 15 minuuttia ennen junan lähtöä asemalta, 15 minuuttia lähdön jälkeen, 15 minuuttia ennen junan saapumista ja 15 minuuttia saapumisen jälkeen.

Rajapinta palauttaa tiedot JSON-muotoisena (JavaScript Object Notation). JSON on kevyt tietoformaatti tiedon säilömiseen ja kuljetukseen. Sitä käytetään usein datan lähettämiseen palvelimelta web-verkkosivulle. Tieto kulkee JSON:issa avain-arvo-pareina ja se jaetaan uusiin pareihin pilkulla. Aaltosulkeet merkitsevät objekteja ja hakasulkeet matriiseja, joten syntaksiltaan se on identtinen JavaScript objektien luonnin kanssa, mutta syntaksista ja nimestä huolimatta sitä voidaan käyttää myös muidenkin ohjelmointikielien kanssa. Alla kuvankaappaus rajapinnan JSON-tyyppisestä vastauksesta ilman minkäänlaista formatointia. Objektin sisältämän tiedon takia se on melko helposti luettavissa, mutta todellisuudessa

objektien tai taulukoiden sisältämät tiedot voivat olla kymmeniä tai satoja rivejä pitkiä. (Mozilla. 2021)

Kuvio 6: Rajapinnan palauttamaa formatoimatonta JSON-dataa

Rajapinnan palauttama tieto on kaikki yhdessä köntässä Google Chrome selaimella avattuna, joten sen tulkitsemisen helpottamiseksi voidaan hyödyntää työkalua kuten JSON-formatter, joka parsii tiedon helpommin ihmisen luettavaan muotoon. Jotkin selaimet, kuten Mozilla Firefox tekevät tämän automaattisesti. Jotta voidaan tutkia rajapinnan palauttamia tietoja selkeämmin, on hyvä tutustua vain yhteen vastauksen ilmentymään tai objektiin. Tämä helpottaa huomattavasti datan tulkintaa, jotta sitä osataan toteutusvaiheessa käsitellä siihen muotoon, kun sitä halutaan näyttää loppukäyttäjälle. Alla kuvattuna rajapinnasta saatu vastaus, jonka sisältämiä objekteja on laitettu piiloon ja neljännes jätetty näkyviin tulkittavaksi.

Kuvio 7: Rajapinnan palauttama JSON-data siistittynä luettavampaan muotoon

JSON-formatter-työkalua käytettäessä JSON-objekteja voidaan piilottaa nuolen taakse puurakenteessa, jolloin datan rakennetta on helpompi pureskella. Yllä olevasta kuvasta selviää, että JSON-data on matriisissa ja se sisältää objekteja, joista jokainen objekti on yhden aseman ilmentymä. Lisäksi objekti sisältää tietoa asemasta avain-arvo-pareissa.

6.2 Suunnitelma

Opinnäytetyössä toteutettavan projektin tarkoituksena oli luoda yksinkertainen web-sovellus, johon käyttäjä voi syöttää pysäkin, jonka jälkeen sovellus palauttaa syötetylle pysäkille ennalta määritetyn ajan sisällä saapuvat ja lähtevät junat. Tätä toiminnallisuutta varten sivulle ei tarvita muuta kuin yksi syötekenttä.

Ideana oli, että kun käyttäjä syöttää aseman nimen, sovellus toteuttaa GET-pyynnön Aktiivisten junien polkuun, kohdennettuna haettuun asemaan. Päällimmäisenä ongelmana on kuitenkin, että kyseinen polku ottaa vastaan GET-pyyntöihin vain aseman lyhennekoodin, esimerkin tapauksessa ”HKI”, joka on Helsingin rautatieaseman lyhennekoodi. Onneksi Digitraffic tarjoaa kuitenkin metatietojen polun (/metadata), josta löytyy kaikkien liikennepaikkojen tiedot mukaan lukien niiden lyhennekoodit ja nimet.

Täten, sovelluksen toimintalogiikkaa oli muokattava niin, että kun käyttäjä syöttää hakukenttään tekstiä sovellus tekee pyynnön ensin pysäkkien Metadata-polkuun, jonka jälkeen se palauttaa hakuparametrejä vastaavan pysäkin lyhennekoodin, jonka perusteella sovellus tekee uuden pyynnön aktiivisten junien polkuun. Tämä tulee toteuttaa asynkronisesti, koska muuten sovellus tekee pyynnön Aktiivisten junien polkuun ennen kuin se on saanut vastauksen metadata-polusta siihen mille asemalle haku tulisi kohdistaa. Tämä onnistuu JavaScriptin await-operaattorilla, joka pistää kyseisen funktion tauolle, kunnes asynkronisen funktion lupaus (promise) on saatu selvitettyä (voi onnistua tai epäonnistua). Tässä tapauksessa lupaus on funktio, joka hakee hakuparametrejä vastaavan aseman lyhennekoodin. Jos hakuparametreillä ei löydy tuloksia metadatasta niin funktio palauttaa vikailmoituksen. Jos se taas onnistuu niin prosessi jatkaa etenemistä palautuneella lyhennekoodilla, jolloin sovellus tekee haun aktiivisten junien polkuun ja palautuneiden tietojen prosessoinnin jälkeen tulostaa asemalle saapuvat ja lähtevät junat.

6.3 Rajapinnan vastaukset

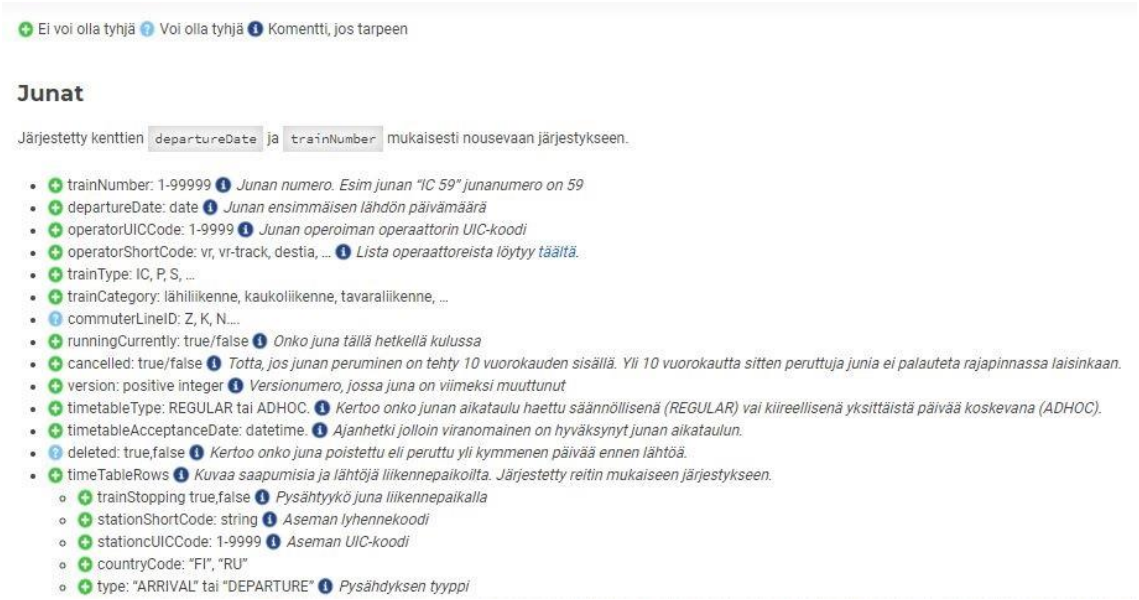
Kuten aiemmin mainittiin, Digitrafficin rajapinnan endpointit palauttavat tiedot aina samassa muodossa ja ainoat hakua rajaavat ehdot annetaan pyyntöä tehdessä osoiterivillä. Pyyntö Aktiivisten junien endpointiin palauttaa matriisin kaikista niistä junista, jotka ovat annetun aikamääreen sisällä kulkemassa aseman kautta. Tämä matriisi sisältää aina yhden junan ilmentymän objektina ja jokaisen junan objektin alta löytyy junan tietojen lisäksi oma aikataulumatriisi, joka sisältää pysähdys objekteja, jotka ovat pysähdyksen ilmentymiä kyseiselle junalle. Tämä aikataulumatriisi sisältää aina yhden junan jokaisen pysähdyksen, mukaan lukien ne mitkä eivät liity mitenkään rajapintaan lähetetyn pyynnön asemaan. Lisäksi lähes jokaisesta pysähdyksestä löytyy kaksi ilmentymää, asemalle saapuminen ja asemalta

lähtö. Pisararadan tapauksessa juna voi kulkea saman aseman kautta neljästi: menosuuntaan asemalle saapuminen, asemalta lähtö ja paluumatkalla asemalle saapuminen ja asemalta lähtö.

```
[
  {
    "trainNumber": 9,
    "departureDate": "2021-05-03",
    "operatorUICCode": 10,
    "operatorShortCode": "vr",
    "trainType": "IC",
    "trainCategory": "Long-distance",
    "commuterLineID": "",
    "runningCurrently": true,
    "cancelled": false,
    "version": 278043049477,
    "timetableType": "REGULAR",
    "timetableAcceptanceDate": "2021-02-19T10:43:08.000Z",
    "timeTableRows": [ … ] // 134 items
  },
```


Kuvio 8: Aaltosulkeissa yksi junan ilmentymä: IC-juna numero 9. TimeTableRows matriisin alta löytyy kyseisen junan jokaisen pysähdyksen saapuminen ja lähtö.

Sovelluksen toiminnan kannalta hauttiin siis prosessoida rajapinnasta saatava tietoa niin, että junan objektista otettaisiin talteen oleelliset siihen liittyvät tiedot, joita halutaan tulostaa loppukäyttäjälle ja suodatettaisiin TimeTableRows-matriisista junan pysähtymisistä käyttäjän hakeman aseman tietoja kuten asemalle saapuminen ja sen suunniteltu kellonaika. Jos pysähdykselle tai junalle on tarjolla reaaliaikaista seurantatietoa, niin nämäkin haluttiin poimia tulostettavaksi. Myöskään peruttuja junavuoroja ei haluttu huomioida.



Kuvio 9: Junan ilmentymän rakenne ja arvo-avain-parien selitteitä rajapinnan dokumentaatiossa

Rajapinnan vastauksen aikaleimat ovat ISO-8601 standardin mukaisia ja niiden aikavyöhykkeenä toimii Z eli ne ovat UTC-ajassa (Coordinated Universal Time).

Projektin tarkoituksena oli rakentaa yksinkertainen sovellus haetulle pysäkille saapuvista junista, joten minimivaatimukset loppukäyttäjälle tulostettavista tiedoista olivat:

- trainNumber. Junan numero.
- commuterLineID. Esim. K-juna.
- scheduledTime. Arvioitu saapumisaika pysäkille.
- commercialTrack. Pysähdyksen raide.
- Jos juna on myöhässä tai etuajassa ja siitä on tieto rajapinnan vastauksessa, tulostetaan tämä arvion lisäksi.

7 Sovelluksen kehitys

Kun rajapinnan vastauksiin saatiin jonkinlainen selvyys ja luotiin alustava suunnitelma, voitiin siirtyä itse sovelluksen kehittämiseen. Sovellus vaatii kaksi keskeistä komponenttia, palvelimen, jossa sovellus pyörii ja käyttöliittymän, minkä kautta sovellusta käytetään.

Projektin ensimmäinen askel oli luoda projektikansio. Tämän jälkeen ladattiin Visual Studio Code-tekstieditoriin lisäosa, jonka avulla Noden komentorivi saatiin integroitua editorin näkymään, jolloin tuotettua ohjelmakoodia voitiin ajaa helposti samassa paikassa, kun sitä

kirjoitettiin. Tämän jälkeen navigoitiin VS Coden komentokehoitteen kautta luotuun projektikansioon ja alustettiin projekti Noden npm init-komennolla, joka generoi package.json-tiedoston. Tiedoston avulla voidaan mm. pitää kirjaa sovelluksen käyttöön tarvittavista riippuvuuksista (asennetuista moduuleista). Tämän jälkeen asennettiin Express ja EJS moduulit, jotka muodostavat Noden ohella sovelluksen pohjan. Expressin avulla saatiin helposti pystytettyä paikallinen palvelin, jossa projektia pystyttiin ajamaan ja testaamaan kuin se olisi mikä tahansa verkkosivu.

7.1 Node ja Express

Sovelluksen palvelimen puolen tai back-endin toiminnallisuuden muodostavat Node, Express ja EJS. Express tekee palvelimen pystyttämisestä helppoa ja nopeaa. Vaikka sen käyttöönotto on helppoa, Express on myös pitkälle skaalautuva ja helposti laajennettava web-sovelluksien toteutustekniikka. (White 2021)

Alla yksinkertainen esimerkki Express palvelimen pystytykseen, se toimii URL-osoitteen http://localhost/3000/:n juuressa (voidaan myös kutsua poluksi) ja kun käyttäjä yhdistää (tekee GET pyynnön) tähän osoitteeseen palvelin palauttaa vastauksen, joka sisältää tekstin ”Hello World”. Tämän lisäksi Noden lokiin tulostuu viesti sovellusta käynnistettäessä siitä, että palvelin on ylhäällä ja toimii määritellyssä portissa. Myöhemmin projektissa on tarkoituksena, että palvelin palauttaa polkuun pyrkivälle käyttäjälle .EJS-tyyppisen HTML-dokumentin, joka toimii sovelluksen käyttöliittymänä ja ”front-endinä”.

```
1  const express = require('express')
2  const app = express()
3  const port = 3000
4
5  app.get('/', (req, res) => {
6    res.send('Hello World!')
7  })
8
9  app.listen(port, () => {
10   console.log(`Example app listening at http://localhost:${port}`)
11 })
```

Kuvio 10: Hello World-esimerkki Expressin avulla

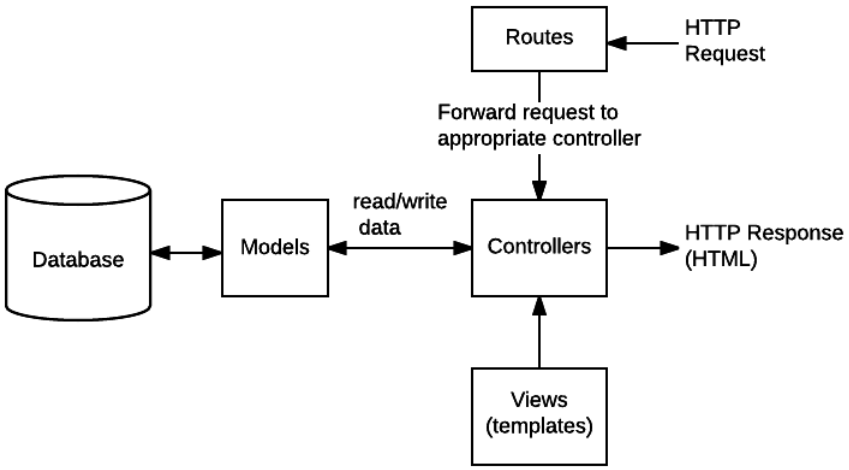
Kun palvelin saatiin pystyyn seuraava työvaihe, oli ottaa EJS-sivupohjamoottori käyttöön ja saada sovellus näyttämään muuta kuin pelkkä ”Hello world” tekstinpätkä. Expressin dokumentaation mukaan html-dokumentteja renderöidään res.send()-funktion sijaan res.render()-funktiolla. Ennen tätä kuitenkin tulee luoda kansio näkymille eli html-tiedostoille, joita ollaan näyttämässä käyttäjällä ja määritellä Expressille mistä tiedostot

löytyvät. Tämän jälkeen luodaan html-tiedosto, projektissa käytettiin aiemmin luotua pohjaa. Jotta EJS:ää voidaan käyttää, tulee html-tiedostoille antaa tiedostotyypiksi .ejs.

Tämän jälkeen res.send(”Hello World”) voidaan korvata res.render(html-tiedoston nimi)-funktiolla, jolloin selaimessa osoitteessa http://localhost:3000/ näkyy HTML-dokumentin representaatio.

Seuraava askel oli tehdä kutsuja Digitrafficin rajapintaan sovelluksen kautta. Tätä varten asennettiin projektiin Axios-moduuli, jonka avulla HTTP-pyyntöjen tekeminen olisi helpompaa. Sovelluksen kautta pyyntöjen tekeminen poikkeaa kuitenkin selaimen kautta tehtäviin pyyntöihin siinä mielessä, että rajapinta käyttää Gzip-paketointia, jonka suurin osa nykyselaimista osaa purkaa automaattisesti. Node ja Axios eivät tätä osaa oletuksena tehdä, joten pienen pähkäilyn jälkeen selvisi, että pyyntöjen tunnisteisiin piti lisätä määritys ”Accept-Encoding: gzip”, jonka jälkeen pyynnöt rajapintaan onnistuivat. Rajapinnan dokumentaatiossa asiasta mainittiin vain GraphQL-rajapinnan yhteydessä, mutta ilman tunnistetietoa lähetetyt pyynnöt rajapintaan palauttivat oikeaan suuntaan ohjaavan virheilmoituksen.

Kun pyynnöt rajapintaan saatiin onnistumaan, voitiin alkaa miettimään sovelluksen rakennetta. MVC-mallia (Model View Controller) hyvin löyhästi mukaillen pyrittiin eriyttämään sovelluksen eri toiminnallisuuksia, jotta kokonaisuutta on helpompi käsitellä ja laajentaa tulevaisuudessa. Alla oleva prosessikaavio kuvaa MVC-mallin nimikomponentit (model-view-controller) ja sen miten ne keskustelevat keskenään.



Kuvio 11: MVC-mallin rakennetta Expressillä (Mozilla 2021)

MVC-mallissa sovellus jaetaan nimensä mukaisesti malleihin (model), näkymiin (view) ja kontrollereihin (controller). Ideana on, että HTTP-pyyntö (käyttäjä avaa sivun) tulee

Expressin määritettyyn reittiin (route), jossa kontrolleri on ottamassa tätä vastaan. Kontrolleri keskustelee mallin kanssa, joka palauttaa halutut tiedot, sijoittaa ne näkymään ja palauttaa käyttäjälle renderöidyn näkymän.

```
app.use("/", router)
```

Kuvio 12: HTTP pyynnöt ohjautuvat reittiin (router)

```
router.get("/", controller.renderHomepage)

router.post("/", controller.getTimeTableRows)
```

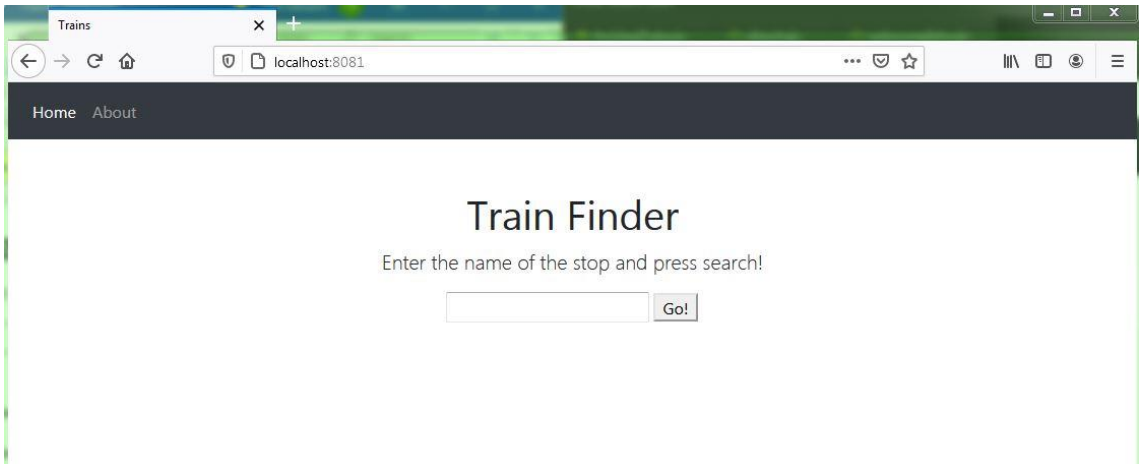
Kuvio 13: Reitti (router) ohjaa pyynnöt kontrollerille

```
res.render("index", { trains: trainsOrdered, stationcode: stationCode })
```

Kuvio 14: Kontrollerin funktio renderöi index.ejs tiedoston ja lähettää siihen tietoa

7.2 Html ja sivuston toimintaa

Aiemmin mainittuun html-dokumenttiin lisättiin lomake-elementti, johon lisättiin syötekenttä ja nappi, jota painamalla lomakkeen syötteen tiedot lähetetään palvelimelle. Lomakkeen metodiattribuutti määrittelee selaimelle missä muodossa data tulee lähettää palvelimelle, jotta sitä voidaan käsitellä. Lomakkeen dataa voidaan lähettää GET ja POST-metodeilla, joista GET lähettää lomakkeen tiedot sellaisenaan selaimen osoiterivin URL:issa kun taas POST:illa tiedot kulkevat pyynnön leipätekstissä tai bodyssä. Molemmilla metodeilla on käyttötarkoituksensa, mutta POST on vaihtoehdoista turvallisempi, koska sen lähettämä tieto ei jää mm. sivuhistoriaan talteen.



Kuvio 15: Sovelluksen käyttöliittymä

Kun sovelluksen hakukenttään syötetään tekstiä ja se lähetetään palvelimelle painamalla nappia, syötetty teksti muutetaan palvelimen päässä pieniin kirjaimiin, jotta sitä voidaan vertailla yhdenvertaisesti ja käyttäjän ei tarvitse kiinnittää huomiota oikeinkirjoitukseen. Jatkokehityksessä suunnitelmana on rakentaa toiminnallisuus ehdottamaan aseman nimiä kenttään kirjoitetun perusteella. Jos hakukenttä jätetään tyhjäksi, sovellus palauttaa vikailmoituksen ennen kuin se alkaa tekemään pyyntöjä Digitrafficin rajapintaan.

Jos hakukentästä palautuu tekstiä palvelimelle eli syöte ei ole tyhjä, tehdään haku rajapinnan metadata-endpointtiin ja vertaillaan syötettyä tekstiä kaikkiin rajapinnasta löytyviin asemiin, joiden kautta kulkee matkustajaliikennettä. Vertailussa ei käytetä yksi yhteen vertailua eli syötteen ei tarvitse olla muodoltaan täysin identtinen rajapinnan tietojen kanssa. Tämä

tapahtuu Match()-funktiolla, joka vertaa string-tyyppisiä muuttujia RegExp:iä vasten (Regular Expression) (Mozilla 2021).

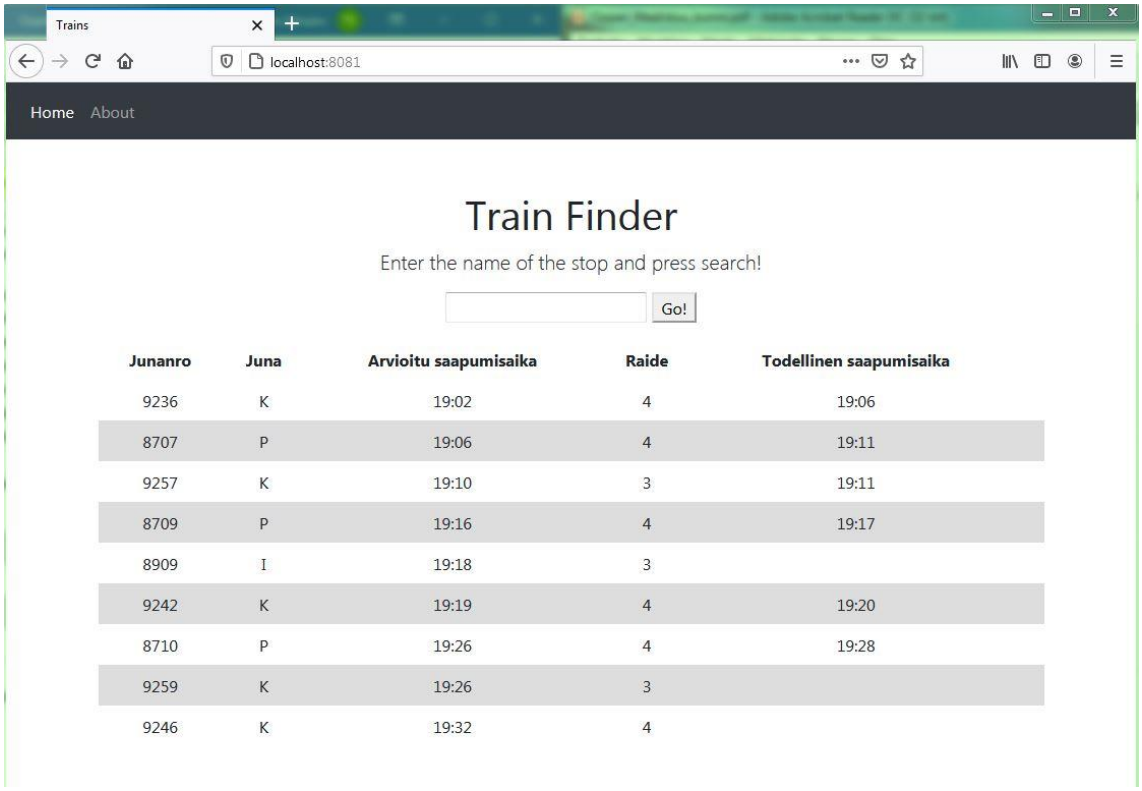
Jos käyttäjän antamalle syötteelle löytyy vastaava rajapinnan tiedoista, sovellus ottaa talteen aseman lyhennekoodin ja tekee perään uuden pyynnön rajapintaan, tällä kertaa aktiivisten junien-endpointiin. Tämä tapahtuu luomalla muuttuja, jonka arvoksi annetaan syötteen ja metadatan vertailun käsittelevä funktio. Muuttuja tämän jälkeen sijoitetaan rajapinnan pyynnön URL-osoitteeseen, jolloin pyyntö kohdistuu siihen asemaan, jonka perusteella käyttäjä antoi syötteen. Pyynnön URL-osoite ei kohdistaisi hakua mihinkään, jos metadatan hakevaa funktiota ei määriteltäisi odottamaan vastausta asynkronisella await-operaattorilla, sillä stationCode-muuttuja on tyhjä, kunnes funktio palauttaa vastauksen.

```
const stationCode = await getStationMetadata(station)
const URL = `https://rata.digitraffic.fi/api/v1/live-trains/station/${stationCode}?
```

Kuvio 16: Pyynnön URL:in rakentuminen sovelluksessa

Tässä vaiheessa sovellus saavutti sen toiminnallisuuden, mitä aiemmin selaimen kautta nähtiin. Erotuksena, että selaimella osoiteriviä voitiin muokata manuaalisesti ja aseman lyhennekoodi tuli tietää ulkoa tai käydä tarkastamassa metadata-endpointista. Rajapinnan ongelmana tai rajoitteena on jo aiemminkin mainittu tiedon määrä: ruuhka-aikaan lähiliikenteen asemilla voi kulkea useita kymmeniä junia 15 minuutinkin sisällä. Näistä jokainen juna sisältää ainakin kaksi ilmentymää jokaisesta pysähdyksestä mikä voi tarkoittaa jopa satoja pysähdyksiä per juna. Tästä syystä tässä vaiheessa JSON-dataa palautuu useita tuhansia rivejä, joten datamassa täytyy sovelluksen päässä käydä läpi ja noukkia sieltä käyttäjälle relevantit tiedot eli moneltako mikäkin juna on saapumassa asemalle ja mille raiteelle.

Projektin suunnitteluvaiheessa määriteltiin tai korvamerkittiin tietoja, joita käyttäjälle haluttiin tulostaa. Tämä tapahtuu noukkimalla tuhansien rivien joukosta jokaisen junan perustiedot: yksilöivä junanumero, junan linjatunnus ja tieto siitä onko juna peruttu. Tämän jälkeen halutaan noukkia vain ne junan pysähdykset, joiden lyhennekoodit täsmäävät käyttäjän hakemaan asemaan (esim. stationShortCode: ”HKI”), jotka ovat saapuvia (type: ”ARRIVAL”) ja jotka ovat pysähtymässä asemalle (trainStopping: true) ja tulostaa näiden ilmentymien tiedoista tarvitut.



Kuvio 17: Haetulle asemalle saapuvia junia sovelluksessa

8 Yhteenveto ja pohdintaa

Opinnäytetyön tavoitteena oli tutustua web-rajapintoihin ja opitun perusteella kehittää niitä hyödyntävä sovellus. Koska REST on nykypäivänä erittäin suosittu tapa suunnitella rajapintoja, voin hyödyntää opittua myös tulevaisuudessa muissa projekteissa

Tehdyn projektin lopputulos täytti sille suunnitteluvaiheessa asetetut tavoitteet. Kehitin sovelluksen, jonka avulla käyttäjä pystyy selvittämään hakemalleen asemalle saapuvien junien ajankohta. Myös kirjoittajan oma ymmärrys ohjelmointirajapinnoista ja web-ohjelmoinnista kasvoi.

Sovellusta kehitettäessä kuitenkin ilmeni useita mahdollisia parannuksia ja korjauksia, joita olisi hyvä huomioida joko seuraavassa projektissa tai jatkokehityksessä. Tällä hetkellä sovellus ei hyödynnä kuin murusen tiedoista, joita rajapinnasta saadaan vastauksena.

Sovelluksen ehkä suurimpana ongelmakohtana voitaisiin pitää sitä, että rajapinta rajoittaa siihen tehtävien pyyntöjen määrää. Omassa käytössä sovelluksella tehtyjen pyyntöjen määrä pysyy hyvin rajoitteiden puitteissa, mutta jos sivustoa haluttaisiin viedä pidemmälle siihen

pisteeseen, että se julkaistaisiin muiden käyttöön, niin rajoitteet tulisivat vastaan jo muutamalla samanaikaisella käyttäjällä.

Tähän ratkaisu olisi tehdä pyyntö rajapintaan kaikista junista esimerkiksi kerran vuorokaudessa ja tallentaa tiedot tietokantaan. Tämä ratkaisu helpottaisi myös rajapinnan palauttaman datan käsittelyä. Tämän jälkeen kaikki pyynnöt sovellukseen kohdistuisivat tietokantaan, jolloin rajapinnan rajoitteet saataisiin kierrettyä. Ratkaisu on hyvin yksinkertainen paperilla, mutta vaatii sovelluksen kehittämistyön aloittamisen lähes puhtaalta pöydältä. Kuitenkin kiitos Expressin ja sovelluksen rakenteen, kehitystyötä voitaisiin jatkaa helposti tekemällä tälle oma reittinsä, jolloin sovelluksen toimivia toiminnallisuuksia ei tarvitsisi rikkoa, vaan sitä voitaisiin kehittää omassa rauhassa.

9 Oman oppimisen arviointi

Opinnäytetyön aihe oli kirjoittajan mielestä mielenkiintoinen, joten projektin toteuttamiseen riitti motivaatiota alusta loppuun, vaikka välillä ongelmien ratkaiseminen olikin työlästä ja paikoin hidasta. Opin paljon erityisesti REST-rajapintojen toiminnasta ja ymmärrän paremmin mitä työkaluja niiden hyödyntämiseen voidaan käyttää.

Projektissa haluttiin rakentaa sovellus, joka hakee asemalle saapuvien junien tiedot ja tämä saavutettiinkin, mutta työn edetessä ja taitojen karttuessa törmättiin moniin parannusideoihin ja parempiin tapoihin toteuttaa jokin toiminnallisuus, joita jo lueteltiinkin aiemmassa luvussa. Näitä ei kuitenkaan toteutettu projektin aikana aikataulujen vuoksi ja siksi, että sovellus teki sen mitä alkujaan haluttiinkin ja pyrittiin välttämään ns. scope creep eli tahdottiin ajaa projekti maaliin ja käsitellä esille nousseet ongelmat jatkokehityksessä.

Opinnäytetyön kirjoittaminen vaati huomattavasti enemmän aikaa kuin kehittäminen heikohkon motivaation ja työkiireiden takia.

Lähteet

Sähköiset

Avoindata 2021. Viitattu 1.5.2021. https://www.avoindata.fi/fi/opas/mita-on-avoin-data

Avoinrajapinta 2021. Viitattu 1.5.2021. http://avoinrajapinta.fi/

Axios 2021. Viitattu 26.5.2021 https://github.com/axios/axios

Dataeuropa 2021. Viitattu 1.5.2021 https://data.europa.eu/fi/trening/what-open-data

Digitraffic 2021. Junien tiedot. Viitattu 1.5.2021
https://www.digitraffic.fi/rautatieliikenne/#junien-tiedot-trains

Digitraffic 2021. Käyttöehdot. Viitattu 1.5.2021 https://www.digitraffic.fi/kayttoehdot/

Digitraffic 2021. Palvelun esittely. Viitattu 1.5.2021 https://www.digitraffic.fi/palvelun-esittely/

Digitraffic 2021. Rautatieliikenne. Viitattu 1.5.2021
https://www.digitraffic.fi/rautatieliikenne/

Digitransit 2021. Liity. Viitattu 25.5.2021 https://digitransit.fi/liity/

Expressjs 2021. Using template engines with Express. Viitattu 25.5.2021
https://expressjs.com/en/guide/using-template-engines.html

Elliot Forbes 2017. What is a rest api. Viitattu 8.6.2021 https://tutorialedge.net/software-eng/what-is-a-rest-api/

Kevin Farmer 2021. What is Postman and why should I use it? Viitattu 26.5.2021
https://www.digitalcrafts.com/blog/student-blog-what-postman-and-why-use-it

Lucas White 2020. What Are The Reasons To Learn Express.Js In 2021? Viitattu 1.5.2021
https://codersera.com/blog/learn-express-js/

Michael Henderson 2019. Viitattu 1.5.2021 https://medium.com/@michaelhenderson/what-is-nodejs-and-why-you-need-to-learn-it-f0760ba9a76a

Mozilla 2021. JSON. Viitattu 25.5.2021 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

Mozilla 2021. Routes. Viitattu 25.5.2021 https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes)

Mozilla 2021. String.prototype match(). Viitattu 1.5.2021 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/match

Nodemon 2021. Viitattu 1.5.2021 https://nodemon.io/

Roy Fielding 2000. Architectural Styles and the Design of Network-based Software Architectures. Kappale 5. Viitattu 1.5.2021 https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Shif Ben Avraham 2017. What is REST - A simple explanation for beginners. Viitattu 1.5.2021 https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-1-introduction-b4a072f8740f

Shif Ben Avraham 2017. What is REST - A simple explanation for beginners. Viitattu 1.5.2021 https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-2-rest-constraints-129a4b69a582

Tom Johnson 2019. Viitattu 25.5.2021 https://github.com/tomjoht/learnapidoc/blob/main/_docs/documenting_conceptual_content/docapis_more_about_authorization.md

Twitter 2021. Publish and manage Tweets, and analyze Tweet data. Viitattu 25.5.2021 https://developer.twitter.com/en/use-cases

Visualstudio 2021. Viitattu 25.5.2021 https://code.visualstudio.com/

Yaapa, H. 2013. Express Web Application Development, 24-26. Viitattu 2.5.2021. http://ebookcentral.proquest.com/lib/laurea/detail.action?docID=1220932

Kuviot