

Roberto Furduc

REST-POHJAISEN OHJELMISTORAJAPINNAN
TOTEUTTAMINEN PILVIPALVELULLE

Tietotekniikan koulutusohjelma
2018

Furduc, Roberto
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Joulukuu 2018
Sivumäärä: 41
Liitteitä: 0

Asiasanat: ohjelmistokehitys, ohjelmistoarkkitehtuuri, ohjelmistosuunnitelu, REST, API

Insinööriyön tarkoituksena oli kertoa toimintatutkimusmallin mukaisesti, miten Tavu Cloud -sovellukselle suunniteltiin ja toteutettiin olemassa olevaan pilvipalveluun rajapinta Multim Oy:n ja sen asiakkaiden käyttöön.

Työssä tarkasteltiin läheisesti REST-arkkitehtuurimallia ja sen rajoitteita. REST-arkkitehtuurimallin lisäksi tarkasteltiin sitä ympäröiviä teknologioita, kuten rajapintoja, HTTP-protokollaa ja HATEOAS-mallia. Lopuksi tarkasteltiin vielä Tavu Cloud -sovelluksen ja Tavu API:n kehitykseen käytettyä Ruby on Rails -ohjelmistokehystä, sekä taustalla pyörivää OpenStack-järjestelmää.

Työssä tarkasteltiin rajapinnan suunnittelua olemassa olevan sovelluksen nykytilanteen perusteella. Lisäksi tarkasteltiin mitä ongelmakohtia nykypalvelun beetatestauksen aikana oli löytynyt ja niiden ratkaisumahdollisuuksia. Seuraavaksi työssä kerrottiin, miten suunniteltu rajapinta toteutettiin Ruby on Rails -ohjelmistokehityksen avulla.

Tuloksena saatiin varsin pätevä beetaversio rajapinnasta, joka täyttää toimeksiannon vaatimukset sekä sallii laajennukset tulevaisuutta varten. Puutteitakin rajapinnalle tuli, suurimpana mainittakoon, että rajapinta ei täysin noudata REST-arkkitehtuurimallin kaikki rajoitteita. Puutteista huolimatta, rajapinta paransi merkittävästi nykyisen sovelluksen suorituskykyä, ja siten myös nykyisen sovelluksen käyttäjäkokemusta ja avasi samalla yritykselle ovet rajapinnan jatkokehitykselle tulevaisuudessa.

IMPLEMENTING A REST-BASED API FOR A CLOUD SERVICE

Furduc, Roberto

Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences

Degree Programme in Information Technology

December 2018

Number of pages: 41

Appendices: 0

Keywords: software engineering, software design, software architecture, REST, API

The purpose of the thesis was, using the action research method, to tell how an API was designed and implemented for the existing Tavu Cloud application for use for Multim Ltd and its customers.

The work closely examined the REST architectural model and its constraints. In addition to the REST architectural model, surrounding technologies, such as APIs, HTTP and the HATEOAS model, were examined. Finally, the Ruby on Rails software framework used for the development of the Tavu Cloud application and the Tavu API, as well as the underlying OpenStack system, were reviewed.

The thesis examined the design of the API based on the current service's present situation. In addition, problematic issues that were identified from the existing application's beta testing and their possible solutions were examined. Next, the thesis examines how the planned API was implemented using the Ruby on Rails software framework and what problems came up during implementation and how they were solved.

As a result, a very competent beta version of the API that fulfilled the requirements of the requirement specification and allowed for future extensions, was obtained. The API had some shortcomings, with the most notable being the API not fully complying with the restrictions of the REST architectural model. Despite the shortcomings, the API significantly improved the query performance, and thus the user experience of the existing Tavu Cloud application while also opening doors for future development opportunities for the company with its extensibility.

SISÄLLYS

| | | |
|-------|--------------------------------------|----|
| 1 | JOHDANTO..... | 6 |
| 1.1 | Työn tavoitteet | 7 |
| 1.2 | Työn rakenne | 7 |
| 2 | POHJUSTUS..... | 8 |
| 2.1 | Rajapinnat | 8 |
| 2.2 | HTTP-protokolla..... | 9 |
| 2.3 | Metodit | 10 |
| 2.3.1 | Post-metodi..... | 10 |
| 2.3.2 | Get-metodi..... | 10 |
| 2.3.3 | Put-metodi | 11 |
| 2.3.4 | Delete-metodi | 11 |
| 2.4 | REST-arkkitehtuurimalli..... | 11 |
| 2.4.1 | Asiakas-palvelin-arkkitehtuuri | 12 |
| 2.4.2 | Tilattomuus..... | 12 |
| 2.4.3 | Välimuistitettava..... | 13 |
| 2.4.4 | Yhdenmukainen rajapinta..... | 13 |
| 2.4.5 | Kerroksittainen järjestelmä..... | 14 |
| 2.4.6 | Koodia pyynnöstä..... | 14 |
| 2.4.7 | Yhteenvedo..... | 14 |
| 2.5 | HATEOAS | 15 |
| 2.6 | Ruby on Rails..... | 16 |
| 2.7 | OpenStack | 17 |
| 2.7.1 | Laskenta (Nova) | 18 |
| 2.7.2 | Verkotus (Neutron)..... | 18 |
| 2.7.3 | Tallennus (Cinder)..... | 18 |
| 2.7.4 | Identiteetti (Keystone) | 18 |
| 2.7.5 | Kojelauta (Horizon)..... | 19 |
| 2.7.6 | Rajapinta..... | 19 |
| 3 | SUUNNITTELU | 20 |
| 3.1 | Lähtökohdat | 20 |
| 3.2 | Vaatimukset | 20 |
| 3.3 | Ruby on Rails kehitysalustana | 21 |
| 3.4 | Tavu Cloud -sovellus | 22 |
| 3.4.1 | Pyynnön elinkaari..... | 23 |

| | | |
|----------|---|----|
| 3.4.2 | Virtuaalipalvelimen hallintasivu | 25 |
| 3.5 | Ongelman analysointia..... | 27 |
| 3.5.1 | Mahdollisia ratkaisuja | 28 |
| 4 | TOTEUTUS | 30 |
| 4.1 | Virheiden käsittely | 30 |
| 4.2 | Reititys | 30 |
| 4.3 | Todentaminen | 31 |
| 4.3.1 | Tiimit | 32 |
| 4.4 | Ohjaimet..... | 33 |
| 4.4.1 | Response-apumoduuli | 33 |
| 4.4.2 | Resurssien haku | 34 |
| 4.4.3 | Resurssien luonti, päivitys ja poisto | 35 |
| 4.5 | Mallit..... | 36 |
| 4.5.1 | OsTools..... | 36 |
| 4.5.2 | Attribuutit ja as_json | 37 |
| 4.5.3 | SQL-kyselyt..... | 38 |
| 5 | TESTAUS | 40 |
| 6 | YHTEENVETO | 41 |
| LIITTEET | | |

LYHENTEET

| | |
|------|--|
| CRUD | Create, read, update ja delete. Verbit, joita usein käytetään HTTP:n metodeista. Verbien avulla kuvataan HTTP: |
| cURL | Komentoriviohjelma, jota voidaan käyttää datan siirtoon. Tukee monia eri protokollia. |
| DB | Tietokanta. Sähköisesti tallennettu organisoitu kokoelma tietoa. |
| JSON | Standardoitu tiedonsiirtoformaatti. |
| SQL | Structured Query Language. Standardoitu kieli, jonka avulla voidaan hakea tietoa relaatiotietokannoista. |

TERMIT

| | |
|-----------------|---|
| Avain-arvo-pari | Tietorakenne, jossa avain yhdistetään tiettyyn arvoon. |
| Debugaus | Virheenjäljitys. Ohjelmistotuotannon osa, jossa paikallistetaan ja korjataan ohjelmakoodista löytyviä virheitä. |
| Href | HyperText Reference. Attribuutti, jonka avulla voidaan linkittää eri verkkosivulle, tai saman verkkosivun eri osioon. |
| Metodi | Funktio, aliohjelma. Ohjelmakoodin itsenäinen osa, joka suorittaa tietyn toiminnon ja jota voidaan kutsua ohjelmakoodin eri osista. |
| Päätepiste | Kommunikaatiokanava. Rajapinnoista puhuttaessa, päätepiste on sijainti, johon asiakas voi lähettää pyynnön. |
| Refaktorointi | Ohjelmoinnin käsite, jolla tarkoitetaan ohjelman lähdekoodin uudelleenkirjoittamista. |
| Todennus | Todennuksella tarkoitetaan käyttäjän tunnistusta, eli identiteetin varmistusta. |

1 JOHDANTO

1.1 Työn tavoitteet

Tehtävänä oli suunnitella ja toteuttaa Multim Oy:lle web-rajapinta yrityksen uuteen Tavu Cloud -palveluun. Palvelu mahdollistaa virtuaalisten resurssien, kuten virtuaali-palvelimien, virtuaalisten tallennuslaitteiden ynnä muiden virtuaalisten resurssien luonnin ja hallinnoinnin. Palvelusta kerrotaan lisää luvussa 3.

Työn tavoitteena oli pohjustaa tärkeitä toteutuksessa käytettyjä teknologioita, kuten rajapintoja ja Ruby on Rails -ohjelmistokehystä, ja malleja, kuten REST ja HATEOAS. Lisäksi työn tarkoitus oli näyttää, miten toimintatutkimusmallia voidaan käytännössä käyttää hyödyksi sovelluksien kehityksessä. Työn tarkoituksena oli myös pohtia ja tarkastella miten toteutus onnistui ja pohtia jatkokehitysmahdollisuuksia.

1.2 Työn rakenne

Työ aloitetaan pohjustamalla lukijalle usein käytettyjä lyhenteitä ja käsitteitä. Web-rajapintoja ja teknologioita, kuten REST-arkkitehtuurimallia ja Ruby on Rails -ohjelmistokehystä, pohjustetaan tarkemmin, sillä ne ovat keskeisiä teknologioita toteutuksen kannalta. Seuraavaksi työssä tarkastellaan, miten toteutuksen suunnittelu eteni. Mitkä olivat lähtökohdat toteutukselle, mitä ongelmia toteutuksella pyrittiin ratkaista ja mitä ongelmia oltiin saatu selville yrityksen nykyisen sovelluksen, joka toimi pohjana rajapinnalle, beetestauksen aikana.

Neljännessä luvussa tarkastellaan toteutuksen yksityiskohtia ja kerrotaan toteutuksen toiminnasta, sikäli kun yrityksen kannalta kyetään. Viidennessä kappaleessa tarkastellaan lyhyesti testausmenetelmiä ja testaukseen käytettyjä työkaluja, sekä testauksen tuloksia. Lopuksi tarkastellaan, miten toteutukselle asetetut päämäärät täyttyivät, jäikö jotain puuttumaan ja mitä mahdollisia parannuksia voidaan tulevaisuudessa tehdä.

2 POHJUSTUS

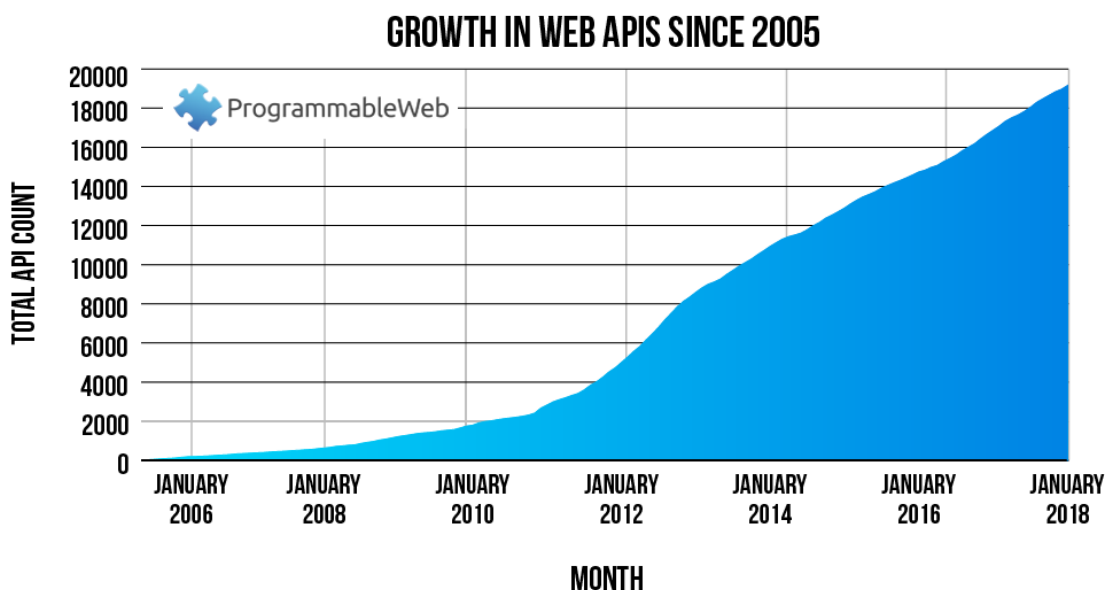
2.1 Rajapinnat

API, eli toisinsanoen ohjelmisto- tai ohjelmointirajapinta, on akronyymi joka tulee sanoista **A**pplication **P**rogramming **I**nterface. Rajapinta koostuu kokoelmasta funktioita, joiden avulla rajapintaa käyttävä ohjelmoija voi rakentaa ohjelman. Jokainen rajapinnan funktio suorittaa jonkin tietyn operaation tai kasan operaatioita.

Samalla tavoin kuin käyttöliittymänkin on tarkoitus helpottaa käyttäjän elämää, rajapinnan tarkoituksena on usein toimia tietynlaisena abstraktiotasona sen suorittamien operaatioiden ja sen funktioiden välillä, joka puolestaan usein helpottaa rajapinnan käyttäjän elämää. Esimerkiksi rajapinnassa voisi olla funktio, joka lisää tiedostoon käyttäjän antaman tekstin, ja palauttaa onnistuessaan arvon ”OK”. Rajapinnan funktiota kutsumalla käyttäjän ei ole välttämätöntä tuntea tai tietää taustalla tapahtuvien rajapinnan suorittamista operaatioista mitään, koska rajapinta hoitaa tämän käyttäjän puolesta (DrDobbs, 2004). Lisäksi rajapinnan tarkoituksena on eriyttää ohjelmalogiikka muista ohjelman komponenteista, esimerkiksi käyttöliittymästä.

Rajapinnat voivat olla julkiseen tai sisäiseen käyttöön tarkoitettuja. Riippumatta rajapinnan julkisuudesta, sen ohella on yleensä dokumentaatio, jonka tarkoitus on ohjastaa rajapinnan käyttäjää rajapinnan käyttöön.

Rajapintoja voidaan kutsua koodilla monilla eri tavoin, mutta varsinkin viime aikoina ovat yleistyneet ns. web-rajapinnat. Web-rajapinnalla tarkoitetaan sellaista ohjelmistorajapintaa, jonka funktiot ovat kutsuttavissa niin sanotusti etäkutsuina, esimerkiksi HTTP-protokollaa hyväksikäyttäen. ProgrammableWeb:n julkaiseman artikkelin mukaan, heidän API-luettelonsa ylitti 19000 API:a 2018 tammikuussa. Vaikkakin ProgrammableWeb:n tilastot saattavat olla hieman vääristyneitä, sillä se on käsin ylläpidetty, itseilmoitettu ja pitää kirjaa vain julkisista rajapinnoista. Dataa on kuitenkin kerätty niin pitkältä ajalta, että siitä näkee selvän trendin ylöspäin. Kuvassa 1 nähdään ProgrammableWebin API-luettelon kasvu vuodesta 2005 lähtien. (ProgrammableWeb, 2018)



Kuva 1. Web-rajapintojen nousu (ProgrammableWeb, 2018).

2.2 HTTP-protokolla

HTTP tulee sanoista **H**ypertext **T**ransfer **P**rotocol. Se on ohjelmistoprotokolla, josta on tullut standardi kommunikointimetodi WWW-maailmassa. HTTP:tä käytetään viestien välittämiseen asiakkaan ja palvelimen välillä. Esimerkiksi verkkoselain voi pyytää palvelimelta haluamansa verkkosivun, ja palvelin vastaa pyyntöön asiakkaan haluamilla tiedoilla, tässä tapauksessa verkkosivulla ja sen sisältämillä tiedoilla. Pyynnöt ja vastaukset voivat lisäksi sisältää muita tietoja, kuten otsikoita, tilakoodeja ja niin edelleen. Näitä tietoja käytetään yleensä pyyntöjen ja vastauksien tarkentamiseen. (RFC2616, 1999)

Pyyntöjä tehdään URI:ien (**U**niform **R**esource **I**dentifier) avulla. Tarkemmin, yleisimmin käytetty URI:n alalajike on URL (**U**niform **R**esource **L**ocator), joita myös arkikielessä verkko-osoitteiksi kutsutaan. URL osoittaa tietyn resurssin tai resurssien sijainnin tietokoneverkossa. (RFC2616, 1999)

2.3 Metodit

HTTP:lle on määritelty standardimetodeja, joiden on tarkoitus ilmaista palvelimelle minkä operaation käyttäjä haluaa suorittaa resurssille. RFC2616 määrittää HTTP 1.1 protokollalle kahdeksan metodia: options, get, head, post, put, delete, trace ja connect. (RFC2616, 1999)

Näitä metodeja saatetaan myös kutsua ”verbeiksi”, vaikka jotkin metodeista ovatkin substantiiveja. REST:n kannalta, merkittävimmät metodit ovat: post, get, put ja delete. Tätä neljän metodin ryhmää nimitetään myös akronyymilla CRUD, joka tulee sanoista create, read, update ja delete. CRUD kertoo yhdellä sanalla minkälaisen toiminnon kukin metodi suorittaa resurssille.

Olennaista metodeille on myös metodin idempotenttisuus. Idempotenttisuudella tarkoitetaan metodin uudelleensuorituksen turvallisuutta. Turvallisen metodin uudelleensuorittamisen ei tule muuttaa lopputulosta. Esimerkiksi jos käyttäjä hakee listauksen olemassa olevista resursseistaan, ja suorittaa saman listauspyynnön uudelleen ilman muita välitoimintoja, vastauksen tulee olla identtinen. (RFC2616, 1999)

2.3.1 Post-metodi

Post-metodin avulla luodaan uusi resurssi resurssikokoelmaan. Post-metodi ei ole idempotentti, sillä saman pyynnön suorittaminen kahteen kertaan loisi kaksi erillistä resurssia, molemmat omilla tunnisteillaan (ID). Post pyynnot osoitetaan resurssikokoelmalle, <http://www.exampleapi.com/users>. (RFC2616, 1999)

2.3.2 Get-metodi

Get-metodin avulla noudetaan tietoa olemassa olevasta resurssista tai resurssikokoelmasta. Get-pyyntöjen ei tule muuttaa resurssien tilaa, toisin sanoen get-pyyntö tulee olla idempotentti. Get pyynnot osoitetaan joko yksittäiselle resurssille, <http://www.exampleapi.com/users/1>, tai resurssikokoelmalle <http://www.exampleapi.com/users>. (RFC2616, 1999)

2.3.3 Put-metodi

Put-metodin avulla päivitetään olemassa oleva resurssi. Eri HTTP:n määrittelyissä on myös lisäksi erillinen patch-metodi, jonka on tarkoitus päivittää resurssi vain **osittain**, kun taas put-metodin on tarkoitus päivittää resurssin kaikki tiedot. Käytännössä kuitenkin näitä metodeja käytetään usein samaan tarkoitukseen. Put-pyynnöt osoitetaan yksittäiselle resurssille, <http://www.exampleapi.com/users/1>. (RFC2616, 1999)

2.3.4 Delete-metodi

Delete-metodi poistaa olemassa olevan resurssin. Delete-pyynnöt ovat idempoteetteja ja osoitetaan aina tiettyyn resurssiin, <http://www.exampleapi.com/users/1>. (RFC2616, 1999)

2.4 REST-arkkitehtuurimalli

REST eli **RE**presentational **S**tate **T**ransfer on Roy Fieldingin kehittämä arkkitehtuurimalli hajautettujen hypermediajärjestelmien suunnitteluun. Alunperin REST kehitettiin pääasiallisesti tavaksi kuvata verkkokonsepteja HTTP-protokollamäärittelyä työstettäessä. (Fielding, 2000)

Alunperin REST:stä käytettiin nimitystä ”HTTP object model”, mutta nimi johti usein väärinkäsityksiin, joten se muutettiin paremmin arkkitehtuurimallia kuvaavaksi. Fieldingin mukaan ”nimen pitäisi antaa lukijan mieleen kuva siitä, miten hyvin toteutettu verkkosovellus toimii: verkko nettisivuja (virtuaalinen tilakone), jossa käyttäjä selaa sovelluksen läpi linkkien avulla (tilamuutokset), ja tuloksena on seuraavan sivun (sovelluksen seuraavan tilan kuvaus) siirtäminen ja renderöinti käyttäjän käyttöön” (Fielding, 2000).

RESTin on tarkoitus toimia mallina verkkojärjestelmille keskeisiksi todetuille käytös- ja suorituskykyvaatimuksille niin, että sitä sovellettaessa lopputulos on optimaalinen. Näiden keskeisiksi todettujen vaatimusten perusteella Fielding esitti kuusi rajoitetta, joita sovelluksen tulisi seurata, jotta sen katsotaan täyttävän REST:n määrittelyn:

client-server (asiakas-palvelin), stateless (tilaton), cacheable (välimuistitettava), uniform interface (yhdenmukainen rajapinta), layered system (kerroksittainen järjestelmä) ja code on demand (koodia pyynnöstä). (Fielding, 2000)

2.4.1 Asiakas-palvelin-arkkitehtuuri

Asiakas-palvelin-arkkitehtuurin ajatuksena on, että sekä asiakkaalla että palvelimella, on omat huolenaiheensa. Asiakkaan ja palvelimen huolenaiheiden ei tulisi koskaan limittyä. Rajoite erottaa asiakas- ja palvelinsovellukset toisistaan ja mahdollistaa niiden toisistaan riippumattoman kehityksen, sekä edesauttaa esimerkiksi asiakassovelluksen siirrettävyyttä. Esimerkiksi taustalla toimiva palvelinjärjestelmä voidaan uudelleentoteuttaa täysin eri tekniikalla, joka parantaisi palvelimen nopeutta, mutta asiakas ei huomaisi muuta kuin sovelluksen nopeamman toiminnan. (Fielding, 2000)

2.4.2 Tilattomuus

Asiakkaan ja palvelimen välisen kommunikaation tulee olla tilatonta. Toisin sanoen palvelin ei saa tallentaa istunto- tai tilatietoja asiakkaasta tai asiakkaan aikaisemmin tekemistä pyynnöistä. Tästä seuraa, että jokaisen pyynnön on sisällettävä kaikki tarvittava tieto pyynnön loppuunviemiseksi. Istuntotietojen ja sovelluksen tilan ylläpito jää siis asiakassovelluksen vastuuksi. (Fielding, 2000)

Tilattomuus parantaa sovelluksen näkyvyyttä, luotettavuutta ja skaalautuvuutta. Skaalautuvuus paranee, kun palvelimen ei tarvitse tallentaa sovelluksen tilaa pyyntöjen välillä, jolloin niiden tallentamiseen vaadittu tallennustila ja muut resurssit vapautuvat. Näkyvyydenkin kannalta asiat paranevat, sillä esimerkiksi virhetilanteiden tai muunlaisien analyysien tekemiseksi jokaista pyyntöä voidaan tarkastella yksitellen. (Fielding, 2000)

Rajoitteella on ongelmansatkin. Verkon suorituskyky saattaa kärsiä, koska jokaisen pyynnön tulee sisältää kaikki sen loppuunviemiseen tarvittava tieto, sen sijaan, että osa tiedoista tallennettaisiin palvelimen puolelle tulevia pyyntöjä varten, joskin seuraava rajoite (välimuistitettavuus) pyrkii parantamaan verkon suorituskykyä. Lisäksi, koska

tila on asiakassovelluksen vastuulla, niin palvelimen kyky vaikuttaa sovelluksen johdonmukaiseen toimintaan eri asiakassovelluksien tai asiakassovellusversioiden välillä, pienenee. (Fielding, 2000)

2.4.3 Välimuistitettava

Kuten aiemmin mainittu, tämä rajoite pyrkii parantamaan verkon suorituskykyä ja vaatii, että palvelimen on, joko suorasti tai epäsuorasti, merkattava onko lähetetty vastaus välimuistitettava vai ei. Käytännössä tämä tarkoittaa, että jokaisen palvelimelta saadun vastauksen mukana on oltava tieto, saako asiakassovellus käyttää tallentaa vastauksen ja käyttää samaa vastausta tulevaisuudessa identtisille pyynnöille. Verkon suorituskyvyn parantamisen lisäksi, rajoite parantaa skaalautuvuutta ja käyttäjän näkökulmasta huomattavaa viivettä, sillä jotkin pyynnot pystytään jättämään täysin käsittelemättä välimuistituksen ansiosta. Välimuistituksen huono puoli on, että joskus asiakassovelluksella saattaa olla vanhentunutta tietoa, vähentäen sovelluksen luotettavuutta. (Fielding, 2000)

2.4.4 Yhdenmukainen rajapinta

Yhdenmukaisen rajapinnan takaamiseksi, REST:llä on erikseen yhdenmukaisuuteen määritellyt rajoitteet resurssien tunnistusta, resurssien manipulointia, itseäänkuvaavia viestejä ja HATEOAS:ää koskien. Yhdenmukainen rajapinta yksinkertaistaa järjestelmän arkkitehtuuria, lisäksi se parantaa kanssakäymisten näkyvyyttä. Yhdenmukainen rajapinta myös rohkaisee aiemmin mainittua asiakas- ja palvelinsovelluksien toisistaan riippumatonta kehitystä, koska toteutukset eivät ole kytketty niiden tarjoamiin palveluihin. Rajoitteen miinuspuolena on sovelluksen hyötysuhteen huonontuminen, sillä rajoituksen myötä tiedonsiirtoformaatin on oltava standardoitu, eikä sovelluskohtainen. (Fielding, 2000)

HATEOAS:stä kerrotaan tarkemmin luvussa 2.5. Muutkin rajoitteet tulevat vielä työssä uudelleen esille, mutta niistä ei sen tarkemmin puhuta.

2.4.5 Kerroksittainen järjestelmä

Kerroksittaisessa järjestelmässä järjestelmä jaetaan hierarkisiin kerroksiin, jossa komponentit ovat tietoisia ja kommunikoivat ainoastaan välittömien naapureidensa kanssa. Näin saadaan asetettua teoreettinen yläraja järjestelmän monimutkaisuudelle ja helpotetaan yksittäisten komponenttien korvausta. Lisäksi kerroksittainen malli mahdollistaa palvelujen kuormantasauksen useamman verkon tai käsittelijän läpi, parantaen järjestelmän skaalautuvuutta. (Fielding, 2000)

2.4.6 Koodia pyynnöstä

Code on demand, eli koodia pyynnöstä, tarkoittaa, että asiakassovellus voi ladata palvelimelta suoritettavaa koodia. Toisin sanoen siis palvelimen vastaus ei tule esimerkiksi JSON-muodossa, vaan suoritettavan koodin muodossa, ladattu ohjelmakoodi sitten suoritetaan asiakassovelluksen puolella. Näin sallitaan asiakassovelluksen toiminnallisuuksien laajentaminen ja yksinkertaistaminen, sillä uusia ominaisuuksia voidaan ladata suoraan palvelimelta, eikä niitä tarvitse asiakassovelluksen puolella toteuttaa. (Fielding, 2000)

Toisaalta, rajoite myös vähentää näkyvyyttä, joten se on määritelty vaihtoehtoiseksi rajoitteeksi (Fielding, 2000). Sovelluksen ei ole siis pakko seurata tätä rajoitetta jotta sitä voitaisiin kutsua ”RESTfulliksi”.

2.4.7 Yhteenveto

Nämä keskeiset rajoitteet on tärkeää pitää mielessä REST:n mukaista palvelua kehitäessä. Vaikka rajoitteet ovat teknisesti ottaen pakollisia, niin käytännössä kaikkia rajoitteita ei kuitenkaan aina noudateta. ”Oikeita” palveluita kehitetään usein ratkaisemaan ongelmia tai tarjoamaan ominaisuuksia, eikä niinkään seuraamaan jonkin teknisen määritelmän rajoitteita. Tämä usein johtaa siihen, että määritelmiä käytetään enemmänkin suuntaa antavina ohjeistuksina.

2.5 HATEOAS

HATEOAS (hypermedia as the engine of application state), eli hypermedia sovelluksen tilakoneena, on yksi neljästä REST:n yhdenmukaisen rajapinnan rajoitteen tarkenuksista. Sen tarkoituksena on mahdollistaa koneiden ymmärtää rajapintoja ja navigoida niiden lävitse ilman etukäteistietoja. Ideana on, että palvelinsovellus antaa vastauksissaan tarpeeksi tietoa, jotta koko sovelluksen läpi voidaan navigoida ilman etukäteistietoja.

HATEOAS:n perusideaa voisi verrata, vaikka ihmiseen, joka pystyy navigoimaan kokonaisen verkkosivuston läpi tietämällä vain verkkosivuston juuriosoitteen. Esimerkiksi henkilö voisi selata verkkokaupan juuriosoitteeseen, ja sovellus tarjoaa henkilölle hakulaatikkoon, listaa suosittimista tuotteista, nappulan tuotteiden ostoskoriin lisäämiselle ja niin edelleen. Henkilön suorittaessa operaatioita, sovellus antaa henkilölle aina asiayhteyteen (sovelluksen nykytilaan) liittyvät mahdolliset toiminnot, joita voitaisiin rinnastaa rajapinnan tarjoamiin hyperlinkkeihin, ja näin henkilö on saanut ostoretkensä suoritettua ainoastaan tietämällä verkkosivuston juuriosoitteen, loput mahdollisista osoitteista ja operaatioista henkilö on oppinut ostoretken aloituksen jälkeen.

Tarkastellaan esimerkkiä 1. Esimerkissä on asiakas-objekti, jonka ID arvo on 1. Lisäksi objekti sisältää ”links”-nimisen attribuutin, jonka sisältä löytyy taulukko link-objekteja. Jokainen link-objekti muodostuu ”href” ja ”rel” attribuuteista. Href (hyper-text reference) attribuutti, jonka arvona on URL, osoittaa kyseisen asiakkaan tietojen lokaatioon. Rel (relationship), attribuutin arvo kertoo minkälainen suhde kyseisellä linkillä on kyseiseen resurssiin. Esimerkin tapauksessa suhde on ”self” eli kyseinen linkki on linkki resurssiin itseensä.

```
1. {  
2.   "id": "1",  
3.   "links": [  
4.     {  
5.       "rel": "self",  
6.       "href": "http://localhost:8080/asiakkaat/1"  
7.     }  
8.   ]  
9. }
```

Esimerkki 1. Esimerkkirajapinnan vastaus, joka sisältää HATEOAS mukaisen link-objektin.

Esimerkin osoittamalla tavalla pystyttäisiin koneellisesti hakemaan asiakkaaseen liittyvä tieto seuraamalla href attribuutin sisältämää URL:ää ilman etukäteistietoa rajapinnasta. Rajapinnan voitaisiin sanoa olevan itsensä dokumentoiva.

2.6 Ruby on Rails

Rails on avoimen lähdekoodin ohjelmistokehys, joka on kirjoitettu Ruby-ohjelmointikielellä. Sen tarkoituksena on helpottaa ja nopeuttaa verkkosovellusten kehitystä teemmällä oletuksia verkkosovelluksissa usein toistuvista kaavoista. Railsin mielestä, verkkosovelluksia kehittäessä, on olemassa yksi paras tapa toteuttaa tiettyjä asioita, ja se on suunniteltu kannustamaan kehittäjiä käyttämään Railsin ennalta valitsemaa parasta tapaa. (Getting Started with Rails, 2018)

Rails perustuu kahteen pääfilosofiaan: Don't Repeat Yourself ja Convention Over Configuration. Nämä filosofiat ovat ohjanneet Rails-ohjelmistokehityksen kehitystä alusta asti. (Getting Started with Rails, 2018)

Tarkastellaan seuraavaksi lyhyesti mitä edellä mainitut filosofiat käytännössä tarkoittavat:

- Don't Repeat Yourself. DRY on ohjelmistokehityksen käsite, joka toteaa, että jokaisella tiedolla on oltava yksi, yksiselitteinen ja arvovaltainen edustus järjestelmässä. Eli toisin sanoen DRY kannustaa kehittäjiä välttämään koodin kahdentamista, täten tehden järjestelmästä ylläpidettävämmän, helpommin debugatattavan ja helpommin laajennettavamman. (Getting Started with Rails, 2018)
- Convention Over Configuration. Kuten aiemmassa kappaleessa mainittiin, Railsillä on mielipide, miten asiat tehdään parhaalla tavalla. Täten ohjelmistokehitykseen on ennalta asetettuja oletusasetuksia, jotka on kokemuksen perusteella katsottu parhaaksi. Näin nopeutetaan ja yhdenmukaistetaan verkkosovelluksen kehitystä. (Getting Started with Rails, 2018)

Asia, josta haluan vielä erikseen mainita, ja joka tulee vielä työssä uudestaan esille, on Railsin reititys. Railsin reititys seuraa tiukasti REST-arkkitehtuurimallin mukaista resursseihin pohjautuvaa reititystä, tarkoittaen, että Railsin automaattisesti generoimat URI:t noudattavat REST:ssä määriteltyä kaavaa. Oletusreitit resurssille saadaan kätevästi jopa yhdellä koodirivillä. Esimerkiksi kirjoittamalla `routes.rb`-tiedostoon `resources :photos` johtaisi kuvassa 2 nähtäviin reitteihin. (Rails Routing from the Outside In, 2018)

| HTTP Verb | Path | Controller#Action | Used for |
|-----------|------------------|-------------------|--|
| GET | /photos | photos#index | display a list of all photos |
| GET | /photos/new | photos#new | return an HTML form for creating a new photo |
| POST | /photos | photos#create | create a new photo |
| GET | /photos/:id | photos#show | display a specific photo |
| GET | /photos/:id/edit | photos#edit | return an HTML form for editing a photo |
| PATCH/PUT | /photos/:id | photos#update | update a specific photo |
| DELETE | /photos/:id | photos#destroy | delete a specific photo |

Kuva 2. Railsin määrittelemät oletusreitit. (Rails Routing from the Outside In, 2018)

2.7 OpenStack

OpenStack on ilmainen, avoimen lähdekoodin ohjelmistoalusta joka on tarkoitettu pilvilaskentaan. OpenStackin kehitys alkoi vuonna 2010 Rackspace Hostingin ja NASA:n yhteistyöhankkeena. Nykyään sitä tukee, kehittää tai käyttää yli 600 yritystä. Mukaan mahtuu isojakin nimiä, kuten HP, PayPal, Comcast, Best Buy ja niin edelleen. (OpenStack, 2018)

OpenStackiä voidaan ajatella IaaS:nä (Infrastructure as a Service), jolla tarkoitetaan, että asiakkaalle tarjotaan virtuaalisia resursseja, kuten palvelimia, tallennustilaa ja verkkolaitteita, verkon yli. Asiakkaat hallitsevat näitä virtuaalisia resursseja usein joko ohjelmallisesti rajapintojen, komentorivityökalujen ja/tai käyttöliittymien avulla.

OpenStack koostuu moduuleista, joille on annettu erilaisia koodinimiä, ja joista kukin hallitsee infrastruktuurin tiettyä osaa tai osia. Seuraavissa kappaleissa avataan lyhyesti OpenStackin päämoduulien tarkoitus. (OpenStack, 2018)

2.7.1 Laskenta (Nova)

Novan tehtävänä on hallita virtuaalipalvelimia. Sen avulla loppukäyttäjä voi luoda ja hallita virtuaalipalvelimiaan joko rajapinnan tai muiden työkalujen avulla. Toimiakseen Nova vaatii lisäksi seuraavat moduulit: Keystone, Glance ja Neutron.

2.7.2 Verkotus (Neutron)

Neutronin tarkoitus on hallita verkkoja ja IP osoitteita. Se mahdollistaa verkkoyhdistyneisyyden virtuaalisille liitännöille joita muut moduulit, kuten Nova, hallitsevat. Lisäksi Neutronin avulla loppukäyttäjä voi luoda omia verkkoja, sekä sallia tai estää verkkoliikenteen kulkua virtuaalipalvelimiin tai muihin liitännöihin.

2.7.3 Tallennus (Cinder)

Cinder mahdollistaa tallennuslaitteiden luonnin ja kiinnittämisen. Luotuja tallennuslaitteita voi kiinnittää Novalla luotuihin virtuaalipalvelimiin. Tallennuslaitteita voi kätevästi kiinnittää ja irrottaa lennosta. Tallennuslaitteista voi myös luoda tilannevedoksia, joiden perusteella voidaan palauttaa tai luoda uusia tallennuslaitteita.

2.7.4 Identiteetti (Keystone)

Keystonen avulla pidetään kirjaa kaikista OpenStackin käyttäjistä, sekä käyttäjien oikeuksista. Keystone myös hoitaa käyttäjien todentamisen.

2.7.5 Kojelauta (Horizon)

Horizon tarjoaa sekä valvojille, että käyttäjille, web-pohjaisen käyttöliittymän virtuaalisten resurssien hallintaan ja monitorointiin.

2.7.6 Rajapinta

OpenStack tarjoaa REST-arkkitehtuurimalliin perustuvan web-rajapinnan, jonka avulla käyttäjät ja valvojat voivat hallita tai monitoroida virtuaalisia resursseja.

3 SUUNNITTELU

Tehtävänä oli tarjota asiakkaalle ohjelmallista rajapintaa, jonka avulla asiakas voi hallita virtuaalisia resurssejaan. OpenStack tarjoaa omaa rajapintaa, jota jo käytettiin sovelluksessa, mutta yritys tarvitsee paremman tavan monitoroida, laskuttaa ja rajoittaa rajapinnan käyttäjiä, ja OpenStackin tarjoamalla rajapinnalla tähän ei kyetty. Lisäksi yritys ei halunnut paljastaa OpenStackin rajapintaa loppuasiakkaalle.

Lopputuloksena oli siis, että päätettiin toteuttaa oma rajapinta, jotta saadaan parempi hallittavuus sekä laajennettavuus rajapinnalle. Suunnitelmassa oli virtuaalisten resursien hallinnan lisäksi mahdollistaa muihinkin yrityksen tarjoamiin palveluihin, esimerkiksi webhotelleihin ja domaineihin, liittyviä ominaisuuksia samaan rajapintaan tulevaisuudessa.

3.1 Lähtökohdat

Ymmärtääksemme paremmin suunnittelun ja toteutuksen aikana tehtyjä valintoja, on hyvä tietää millaisista asemista rajapintaa lähdettiin suunnittelemaan. Seuraavissa luvuissa tarkastellaan sovelluksen nykytilaa, ja tunnistetaan yksi isoimmista ongelmakohdista käyttäjäkokemuksen kannalta OpenStack-rajapinnan käytössä.

3.2 Vaatimukset

Vaikka rajapinnan toteutus hallintapaneelisovellukselle oli jo ennen rajapinnan julkaisua päätetty, niin rajapinnan määrittely oli vielä auki. Tämä johti siihen, että hallintapaneelisovelluksen rajatun beetestauksen aikana tehdyt tarkkailut ja saadut palautteet, auttoivat tarkentamaan toimeksiannon määrittelyä. Beetestauksen aikana esimerkiksi huomattiin, että tietyillä sivuilla tiedonhaku OpenStack-rajapinnasta kesti liian pitkään.

Rajapintaa lähdettiin suunnittelemaan niin, että seuraavat päämäärät täytyisivät mahdollisimman hyvin:

- Rajapinnan ja nykyisen hallintapaneelin toiminnallisuudet, virtuaalisten resurssien kohdalla, täytyy olla mahdollisimman lähellä toisiaan. Toisin sanoen kaikki mitä käyttäjä kykenee hallintapaneelin avulla tekemään, hänen pitäisi kyetä myös rajapinnan avulla tekemään.
- Tiedonhaku käyttäjän virtuaalisista resursseista toteutetaan tietokantakyselyillä tiedonhaun nopeuden parantamiseksi. Tämän vaatimuksen syntymisestä puhutaan lisää luvussa 3.5.1.
- Rajapinnan tulee olla laajennettavissa tulevaisuutta varten yrityksen muihin palveluihin.
- Rajapintaa pitää kyetä käyttämään myös yrityksen omaan ”sisäiseen” käyttöön, esimerkiksi erinäisiin admin toimintoihin.

3.3 Ruby on Rails kehitysalustana

Rajapinta tultaisiin toteuttamaan Ruby on Railsin avulla. Nykyinen hallintapaneelisovellus on myös toteutettu Ruby on Railsin avulla, joten tarvittaessa, koodinpätkiä voidaan kopioida suoraan projektista toiseen pienellä vaivalla. Tosin koodin kahdentamista haluttiin välttää mahdollisimman paljon, sillä kahdentaminen lisää aina työtä myöhemmin, koodia ylläpidettäessä tai päivitettäessä. Hallintapaneelin kehityksen aikana rajapintaa kehittävä tiimi oli jo tutustunut hyvin ohjelmointikehykseen, täten voitiin luottavaisin mielin olettaa, että työn jälki tulee olemaan hyvää ja kehitys nopeaa.

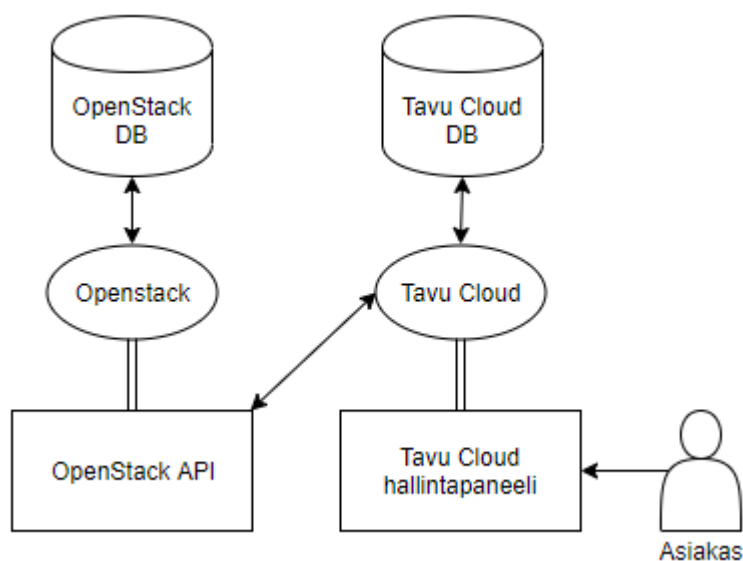
Railsistä löytyy myös niin kutsuttu API-only -moodi, joka projektia luodessa karsii sovelluksesta osan normaalissa asennuksessa mukana tulevista väliohjelmistoista. Nämä karsitut väliohjelmistot ovat usein esimerkiksi näkymään liittyviä asioita, joita ei rajapinnoissa tarvita. Ylimääräisten väliohjelmistojen karsiminen lisää sovelluksen suorituskykyä ja vähentää sovelluksen monimutkaisuutta, helpottaen sovelluksen ylläpitoa ja parantaen päivitettävyyttä.

Lisäksi kuten aiemmin mainittiin, Railsin reititys perustuu resurssimaiseen reititykseen, säästäen aikaa rajapinnan suunnittelussa, kun kehittäjän ei tarvitse miettiä miten rajapinta mallinnetaan HTTP:n kannalta. Lisäksi Railsistä löytyy hyvät URL työkalut URL:ien generointiin, joiden avulla HATEOAS:n toteuttaminen onnistuisi kätevästi.

3.4 Tavu Cloud -sovellus

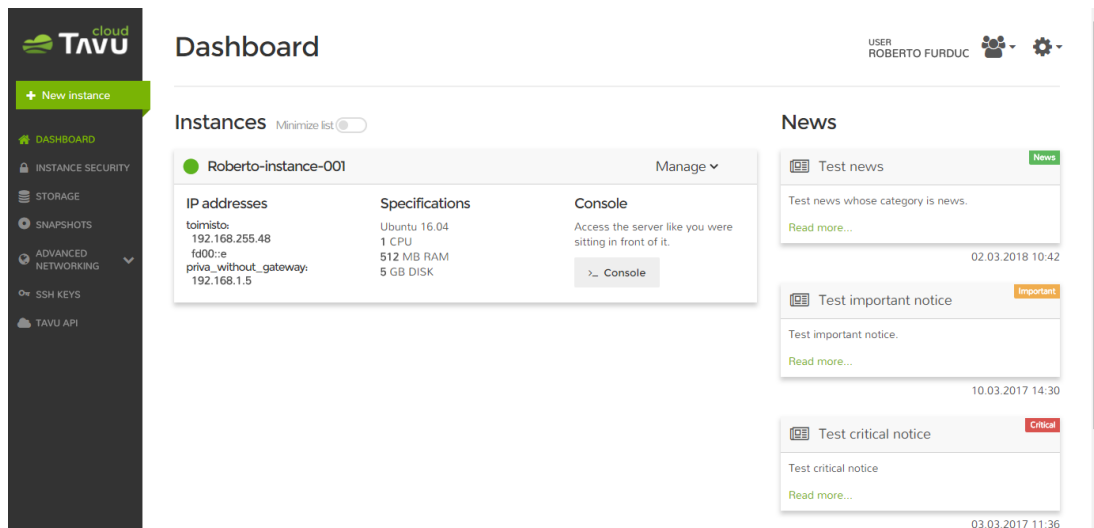
Tavu Cloud on Ruby on Rails -ohjelmistokehyksen avulla rakennettu web-sovellus. Sovelluksen avulla yritys kykenee tarjoamaan asiakkailleen helppokäyttöisempää käyttöliittymää (kuin OpenStackin tarjoama Horizon-käyttöliittymä) OpenStack-taustajärjestelmään hallintapaneelin muodossa. Omaa hallintapaneelia on myös helpompi tarpeen mukaan muokata ja laajentaa. Lisäksi sovellus auttaa yritystä pitämään kirjaa asiakkaiden käyttämistä virtuaalisista resursseista ja mahdollistaa asiakkaiden laskutuksen.

Kuvassa 3 esitellään pelkistetty kaavio taustajärjestelmien rakenteesta ja niiden välisestä keskustelusta. Kuvasta voidaan nähdä, miten OpenStackin rajapintaa hyväksi käyttäen Tavu Cloud -sovellus paljastaa asiakkaalle käyttöliittymän, jonka avulla asiakas voi olla vuorovaikutuksessa OpenStackin järjestelmien kanssa. Hallintapaneelia voidaan ajatella eräänlaisena rajapintana asiakkaan ja OpenStack-järjestelmien välillä.



Kuva 3. Pelkistetty kaavio järjestelmien ja asiakkaiden välisistä yhteyksistä.

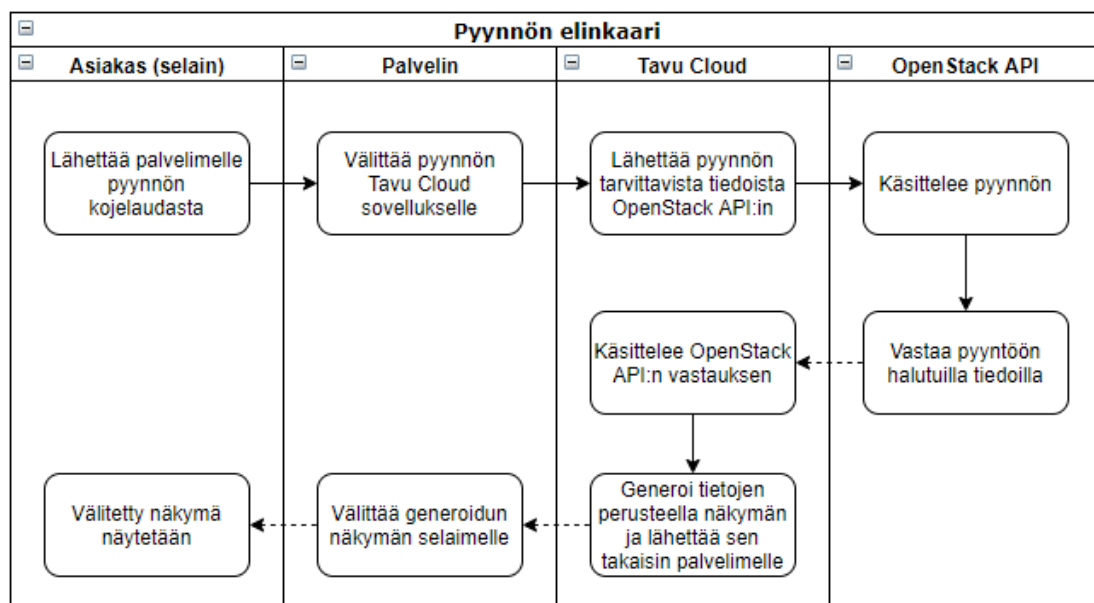
Kuvassa 4 nähdään asiakkaalle näkyvää hallintapaneelin käyttöliittymää. Käyttöliittymän vasemmalla puolella nähdään lista käyttäjän virtuaalipalvelimista ja oikealla puolella listaus tärkeistä päivityksistä ja uutisista.



Kuva 4. Kuvakaappaus hallintapaneelin kojelaudasta.

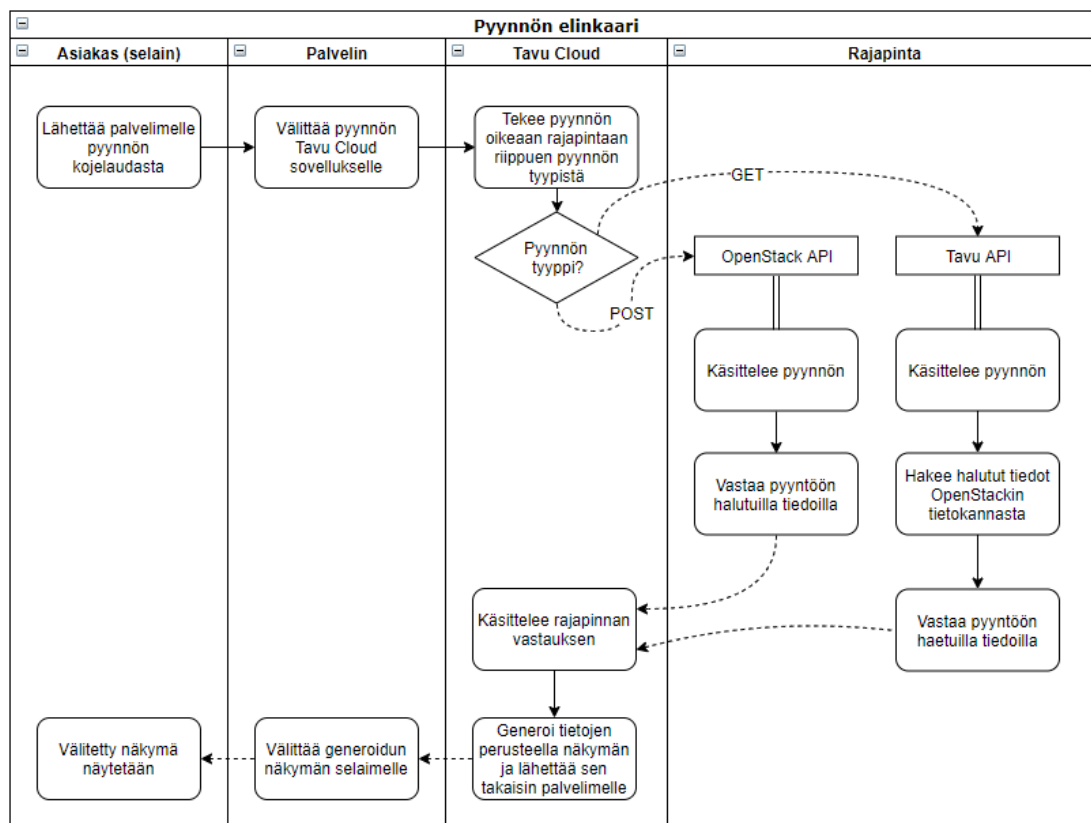
3.4.1 Pyynnön elinkaari

Tiedonhaku asiakkaan virtuaalisista resursseista tehdään suorittamalla http-pyyntöjä OpenStackin tarjoamaan web-rajapintaan. Käyttäjä aloittaa sivunlatauksen siirtymällä esimerkiksi osoitteeseen <https://panel.tavu.io/>. Tämä aloittaa kyselyn palvelimelle, joka saa palvelinsovelluksen hakemaan kyseisen käyttäjän virtuaalipalvelinten tiedot. Palvelimen päässä tehdään kysely OpenStackin rajapintaan. OpenStackin rajapinta vastaa kyselyyn joko asiakkaan virtuaalipalvelinten tiedolla tai virhevastauksella. Tämä vastaus tarkistetaan ja jäsennellään sellaiseen muotoon, että sitä on helpompi käsitellä ohjelmakoodissa. Kuvassa 5 nähdään edellä selitetty prosessi kaaviona.



Kuva 5. Pelkistetty kaavio pyynnön elinkaaresta.

Tulevaisuudessa, haluttu elinkaari muuttuisi get-pyyntöjen osalta niin, että get-pyyntöissä pyydetty tiedot haettaisiin suoraan OpenStackin tietokannasta. Kuvassa 6 on esitelty kaavio suunnitellusta pyynnön elinkaaresta, josta nähdään, että rajapinnalle tulevat pyynnot suoritetaan eri tavalla, riippuen pyynnön tyypistä. Get-pyyntöjen tiedonhaku suoritetaan OpenStackin tietokannasta, ja post-pyyntöjä lähetetään edelleen OpenStack-rajapinnalle.



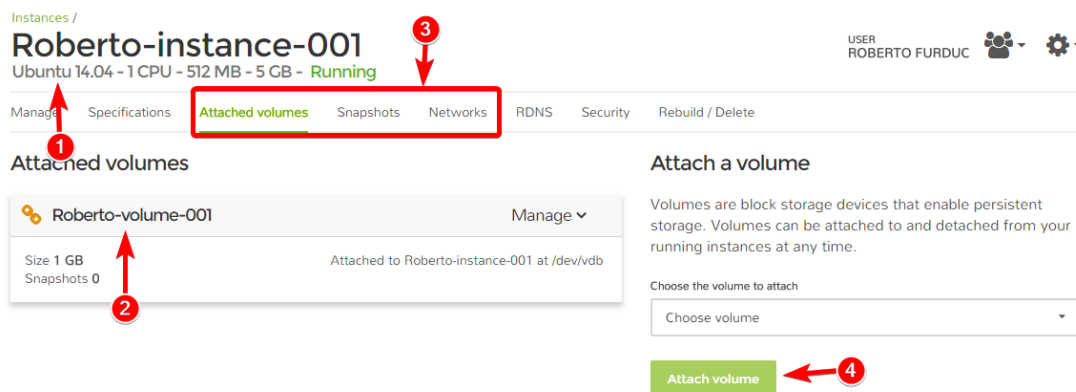
Kuva 6. Pyyntöjen uusi elinkaari.

Kojelauta-sivun tapauksessa tiedonhaku on suoraviivaista. OpenStackin rajapinnalta pyydetään listausta kaikista käyttäjän virtuaalipalvelimista. Tästä vastauksesta saadaan kaikki tarpeellinen tieto esille kojelaudan listaukseen. Ongelmallisempi tilanne syntyy kun käyttäjä haluaa nähdä virtuaalipalvelimensa hallintasivun. Seuraavassa luvussa tarkastellaan, miksi virtuaalipalvelimen hallintasivulla haettavien tietojen kysely on ongelmallisempaa kuin kojelaudalle haettavien tietojen kysely.

3.4.2 Virtuaalipalvelimen hallintasivu

Kuten aiemmin mainittu, hallintapaneelin on tarkoitus helpottaa asiakkaan elämää. Virtuaalisten resurssien on oltava helposti ja kätevästi hallittavissa sekä ymmärrettävissä käyttöliittymästä käsin. Haluan kiinnittää lukijan huomion alla olevassa kuvassa 7 numeroituihin kohtiin:

1. Asiakkaalle näytetään virtuaalipalvelimen levykuvan nimi.
2. Asiakkaalle näytetään virtuaalipalvelimeen kiinnitettyinä olevien tallennuslaitteiden tiedot.
3. Hallintasivu on jaettu välilehtiin, joiden takaa löytyy lisää tietoa esimerkiksi:
 - a. Virtuaalipalvelimeen kiinnitetystä tallennuslaitteista.
 - b. Virtuaalipalvelimesta otetuista tilannevedoksista.
 - c. Virtuaalipalvelimen verkoista.
4. Helppokäyttöisyyden vuoksi haluttiin, että virtuaalipalvelimeen on mahdollista kiinnittää lisää tallennuslaitteita suoraan virtuaalipalvelimen hallintasilvulta.



Kuva 7. Virtuaalipalvelimen hallintasivu.

Ymmärtääksemme miksi tämän sivun rakentaminen on ongelmallista, tarkastellaan OpenStackin rajapinnasta palautuvaa vastausta pyydettyäessä virtuaalipalvelimen tietoja. Vastausta tarkastellessa huomataan, että OpenStackin rajapinta ei palauta kaikkea tietoa mitä sivulla halutaan näyttää.

Kuten esimerkistä 2 näemme, rajapinta palauttaa virtuaalipalvelimeen liittyvistä tiedoista, levykuva (image), konfiguraatio (flavor), liitetyt tallennuslaitteet (os-extended-

volumes:volumes_attached), lähinnä pelkkiä ID arvoja. Syy tähän saadaan selville, kun mietitään hieman, miten resurssien tietoja yleensä säilytetään tietokannoissa.

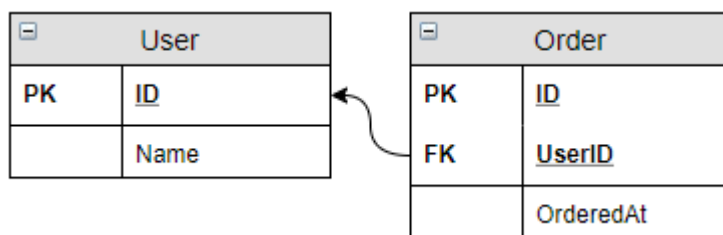
```

1.  "server": {
2.    "image": {
3.      "id": "00ed37ab-7a47-45b3-9419-543a40b3c3f2",
4.      "links": [
5.        {
6.          "href": "http://internal.os.lo-
cal:8774/4c8db2c916ac4cd78185a6258c5ac713/images/00ed37ab-7a47-45b3-9419-
543a40b3c3f2",
7.          "rel": "bookmark"
8.        }
9.      ]
10.    },
11.    "flavor": {
12.      "id": "2257932d-1de5-45d7-ba34-3d513b927b15",
13.      "links": [
14.        {
15.          "href": "http://internal.os.lo-
cal:8774/4c8db2c916ac4cd78185a6258c5ac713/flavors/2257932d-1de5-45d7-ba34-
3d513b927b15",
16.          "rel": "bookmark"
17.        }
18.      ]
19.    },
20.    "id": "4d517658-7619-4b3a-aebc-7ea34268285e",
21.    "os-extended-volumes:volumes_attached": [
22.      {
23.        "id": "ced938f9-fb0b-46f9-b609-2e33a8ae4f28"
24.      }
25.    ]
26.  }

```

Esimerkki 2. OpenStackin rajapinnasta palautunut vastaus, vastauksesta on leikattu pois tietoa mikä ei ole tärkeää tarkastelun kannalta.

Yleisesti ottaen, tietokannoissa oleville riveille on määriteltä primääriavain, joka on uniikki ja jonka avulla voidaan viitata tiettyyn tietokannassa sijaitsevaan riviin. Tämä primääriavain-kenttä nimetään usein ”ID”:ksi. Kun riviin liittyy lisää tietoa, joka löytyy toisesta taulusta, siihen on helppo viitata liittyvän rivin ID-kentän arvolla. Tämän avulla on helppo rakentaa tietokantakysely jonka avulla haetaan liittyvän rivin tiedot eri taulusta. Kuvassa 8 on esitetty yksinkertainen yhteys Order ja User nimisten taulujen välillä. Kuvasta näkyy, että Orderilla ei ole muuta tietoa User resurssista, kuin Userin ID, joten jos halutaan hakea Orderissa olevan Userin Name attribuutti, voitaisiin käyttää hyväksi Orderin UserID kentässä olevaa arvoa.



Kuva 8. Yksinkertainen tietokantakaavio.

Koska rajapintojen tarjoamat tiedot ovat usein tietokannassa, ja näin on tässäkin tapauksessa, voidaan edellä tehtyjen tarkastelujen avulla ymmärtää, miksi OpenStackin rajapinnan vastauksessa virtuaalipalvelimeen liittyvistä tiedoista palautuu pitkälti pelkkiä ID arvoja. Rajapinnan tarjoamat tiedot ovat usein suoraan tietokannasta löytyvät tiedot.

OpenStack ei hae virtuaalipalvelimeen liittyvistä tiedoista muita tietoja, kuin ID arvon, joka on tallennettuna virtuaalipalvelin taulun tietoihin, koska lisätietojen hakemiseen jouduttaisiin suorittamaan useampi, tai raskaampi tietokantakysely. Yleiseen tai julkiseen käyttöön tarkoitetut rajapinnat toteutetaan usein näin, jotta vältytään mahdollisesti turhilta kyselyiltä tietokannasta ja jotta tiedonhaku olisi nopeampaa. Kuten REST-arkkitehtuurimallia tarkastellessa selvisikin, niin tämä on täysin arkkitehtuurimallia seuraava käytäntö, ja malli tarjoaa tälle ongelmalle ratkaisun HATEOAS:än muodossa.

3.5 Ongelman analysointia

Virtuaalipalvelimeen liittyvien tietojen puuttuminen OpenStackin rajapinnan vastauksista johtaa kuitenkin ongelmaan, jossa sivunlataus hidastuu, sillä OpenStackin rajapintaan joudutaan tekemään useampi kysely kaikkien haluttujen tietojen saamiseksi. Ongelma ei olisi niin suuri, jos tietoja kyseltäisiin suoraan tietokannasta joka sijaitisi samalla palvelimella kuin sovellus, sillä kyselyjen välinen latenssi olisi minimaalinen. Todellisuudessa kuitenkin kyselyt kulkevat internetin läpi OpenStack-rajapinnalle, joka lisää latenssia helposti jopa 100 millisekuntia per kysely, kun taas kyselyt suoraan tietokannasta suoriutuvat usein muutamissa millisekunnissa tai alle, riippuen kyselyn monimutkaisuudesta.

Kuvassa 9 on otettu kuvankaappaus virtuaalipalvelimen hallintasivulla tehtävistä kyselyistä OpenStackin rajapintaan. Kuvasta nähdään, melko selkeästi, että kyselyihin kuluva aika on kohtuuton (n. 600ms), varsinkin kun otetaan huomioon, että kyseessä on pelkästään tiedonhakuun kuluva aika, eikä siihen ole vielä sisällytetty sivun HTML-koodin generointiin tai muuhun prosessointiin kuluva aikaa. Lisäksi, kuvakaappaus on vain yhdestä sivunlatauksesta, eikä ota huomioon, että joskus kyselyt saattavat suoriutua hitaammin, riippuen palvelimeen ja OpenStackiin kohdistuvista kuormista.

| | duration (ms) | from start (ms) | query time (ms) |
|--|---------------|-----------------|-----------------|
| GET http://dev102.shellit.fi:80/instances/007... | 22.5 | +0.0 | 4 sql 0.7 |
| Executing action: show | 102.3 | +20.0 | 17 sql 15.6 |
| Net::HTTP GET /v2.1/4c8db2c916ac4cd78... | 0.5 | +40.0 | |
| Net::HTTP GET /v2.1/4c8db2c916ac4cd7... | 130.9 | +41.0 | |
| Net::HTTP GET /v2/4c8db2c916ac4cd7818... | 0.4 | +182.0 | |
| Net::HTTP GET /v2/4c8db2c916ac4cd781... | 38.5 | +183.0 | |
| Net::HTTP GET /v2.1/4c8db2c916ac4cd78... | 0.5 | +222.0 | |
| Net::HTTP GET /v2.1/4c8db2c916ac4cd7... | 103.5 | +223.0 | |
| Rendering: layouts/_messages | 5.7 | +332.0 | |
| Rendering: layouts/_flash_message | 2.6 | +337.0 | |
| Net::HTTP GET /v2/4c8db2c916ac4cd7818... | 0.4 | +340.0 | |
| Net::HTTP GET /v2/4c8db2c916ac4cd781... | 34.9 | +341.0 | |
| Net::HTTP GET /v2.0/networks | 0.6 | +381.0 | |
| Net::HTTP GET /v2.0/networks | 54.0 | +381.0 | |
| Net::HTTP GET /v2.0/subnets | 0.5 | +436.0 | |
| Net::HTTP GET /v2.0/subnets | 40.9 | +437.0 | |
| Net::HTTP GET /v2.0/ports?device_id=007... | 0.4 | +479.0 | |
| Net::HTTP GET /v2.0/ports?device_id=00... | 30.0 | +479.0 | |
| Net::HTTP GET /v2.1/4c8db2c916ac4cd78... | 0.5 | +512.0 | |
| Net::HTTP GET /v2.1/4c8db2c916ac4cd7... | 37.5 | +512.0 | |
| Net::HTTP GET /v2.1/4c8db2c916ac4cd78... | 0.5 | +551.0 | |
| Net::HTTP GET /v2.1/4c8db2c916ac4cd7... | 44.4 | +552.0 | |
| Net::HTTP GET /v2.0/security-groups | 0.5 | +625.0 | |
| Net::HTTP GET /v2.0/security-groups | 28.2 | +625.0 | |

Kuva 9. OpenStack-rajapintaan lähtevät kyselyt ja niiden kestot.

Vaikka tässä tarkasteltiin vain ongelmallisinta sivua, niin sama ongelma esiintyi myös muilla sovelluksen sivuilla. Seuraavassa kappaleessa tarkastellaan mahdollisia ratkaisuja ongelmaan, ja minkälaiseen ratkaisuun loppujenlopuksi päädyttiin.

3.5.1 Mahdollisia ratkaisuja

Ongelma todettiin tärkeäksi käyttäjäkokemuksen kannalta, käyttäjähän pitävät nopeasti toimivista sovelluksista. Aluksi ongelman ratkaisemiseksi yritettiin moniajaa

kyselyitä OpenStackin rajapinnalle. Moniajolla tarkoitetaan usean pyynnön näennäistä samanaikaista suoritusta. Tällaisessa tilanteessa ohjelma lähettäisi kaikki kyselyt näennäisesti samanaikaisesti, eikä peräkkäisesti, poistamalla tarpeen odottaa yhden pyynnön valmistumista ennen kuin seuraava pyyntö voidaan lähettää. Ratkaisu kuitenkin koettiin hieman purkkaratkaisuksi ja hienostumattomaksi. Lisäksi sitä ei koettu helposti ylläpidettäväksi ja ratkaisua testatessa huomattiin outoja ilmiöitä, missä pyynnot eivät aina lähteneet matkaan tai vastaukset jäivät saamatta. Tästä syystä ongelmaa alettiin miettimään uudelleen.

Seuraavaksi mietittiin tietojen haun siirtoa käyttöliittymän puolelle. Kuvasta 6 nähtiin, miten sovelluksen käyttöliittymä on toteutettu. Virtuaalipalvelimen hallintasivu on jaettu osiin välilehtien avulla. Tämä antaisi mahdollisuuden hakea tietyn välilehden tiedot vasta kun käyttäjä klikkaa välilehteä, tai jopa taustalla, kun käyttäjä on siirtynyt virtuaalipalvelimen hallintasivun päävälilehteen. Testatessa kuitenkin huomattiin ratkaisun vaikuttavan käyttäjän kokemaan sovelluksen nopeuteen negatiivisesti, sovellus ei enää tuntunut yhtä nopealta käyttäjän silmissä, kun välilehteä vaihtaessa sivulle laddattiin lisää tietoa lennosta. Tämän lisäksi, asiakassovellukseen olisi pitänyt siirtää ohjelmalogiikkaa, kuten resurssien kiintiötarkistuksia. Tämä olisi johtanut koodin kahdentumiseen ja liian suureen määrään koodin refaktorointia, varsinkin kun seuraavana listalla oleva ratkaisu tuntui paremmalta.

Viimeisenä ajateltiin, että tiedonhakua saadaan nopeutettua ottamalla suoraan yhteys OpenStackin tietokantaan, ja hakemalla tiedot omilla SQL-kyselyillä. Ratkaisua testaamalla todettiin, että ottamalla suora yhteys OpenStackin tietokantaan nopeutti tiedonhakua huomattavasti verrattuna rajapinnan käyttöön. Lisäksi tällä tavoin saatiin haettua kaikki tarvittavat tiedot joissakin tilanteissa jopa yhdellä tietokantakyselyllä, ja vaikka joissakin tapauksissa piti suorittaa useampikin tietokantakysely, niin ratkaisun avulla tiedonhaku oli silti nopeampaa kuin rajapintaa käyttämällä.

Tämän ongelman analyysin lopputuloksena päätettiin tappaa kaksi kärpää yhdellä iskulla, ja yhdistää ongelman ratkaisu rajapinnan kehitykseen. Rajapintaa kehitettäessä tiedonhaku tultaisiin suorittamaan suoraan OpenStackin tietokannasta eikä OpenStack-rajapintaa käyttämällä.

4 TOTEUTUS

Kuten aikaisemmin mainittu, rajapintaa lähdettiin toteuttamaan Ruby on Rails -ohjelmistokehyksellä. Railsin valintaan vaikutti vahvasti sen ydinfilosofiat sekä sen resurs-sipohjainen lähestymistapa verkkosovelluksiin. Lisäksi kehittäjien tuttuus ohjelmisto-kehityksen kanssa oli vahva motivaatio sen valintaan.

Rajapinta tulisi saataville vain yrityksen asiakkaille, jotka ovat luoneet tilin Tavu Cloud -hallintapaneeliin. Rajapintaan tarvittiin siis tapa todentaa käyttäjien identi-teetti. Lisäksi, käyttäjät saattoivat myös omistaa tai olla osana hallintapaneelissa luo-tuja tiimejä. Tästä syystä tarvitaan merkitsemistapa, jolla merkataan, halutaanko ky-seisen pyynnön suorittamat toiminnot suorittaa tiimin resursseille, vai asiakkaan hen-kilökohtaisen tilin resursseille. Merkkaamisen lisäksi rajapinnassa tarvitsi myös ti-meihin kohdistuvia pyyntöjä tehdessä, tarkastaa, että pyynnön lähettäneellä käyttäjällä on oikeus suorittaa kyseinen toiminto tiimin resursseille.

4.1 Virheiden käsittely

Virheiden käsittely on rajapinnan käytön kannalta tärkeää, sillä käyttäjän on tiedettävä, johtuuko virhe palvelimesta vai itsestä. Tästä syystä virheviestien on oltava tarkkoja ja kuvaavia. Virheiden käsittely päätettiin toteuttaa ExceptionHandler-moduulin avulla. Kyseinen moduuli on vastuussa poikkeuksien nappaamisesta. Poikkeuksen na-pattuaan, moduuli muotoilee nousseen poikkeuksen perusteella sopivan virheviestin ihmisiä tähden, sekä koneellista käsittelyä tähden virhetyypin.

4.2 Reititys

Railsin resursseihin pohjautuva reititys sopii mainiosti rajapinnalle, sillä käyttäjän vir-tuaalisia resursseja, kuten palvelimia, tallennuslaitteita ja niin edelleen, voidaan hel-posti ajatella resurssikokoelmina. Laajennettavuuden ja ymmärrettävyyden vuoksi, OpenStack-resurssit päätettiin kaikki laittaa yhden nimiavaruuden, VPS, alle. Ni-miavaruuden lisäksi resurssit jaettiin ryhmiin palvelutyypin mukaan. Esimerkiksi pal-velimet kuuluvat laskentaryhmään, kun taas portit ja aliverkot kuuluvat

verkotusryhmään. Ryhmittelyn lisäksi jokaisen resurssin päätepisteet versioitiin, jolloin päivityksiä tehtäessä, voidaan nostaa resurssin versionumeroa ja samalla säilyttää vanhan version toimivuus. Näin myös päivittämättömät asiakasovellukset toimivat mahdollisimman pitkään. Esimerkissä 3 esitellään routes-tiedoston rakennetta.

```

1. namespace :os, path: 'vps' do
2.   namespace :compute do
3.     namespace :v1 do
4.       resources :instances
5.     end
6.   end
7.
8.   namespace :networking do
9.     namespace :v1 do
10.      resources :subnets
11.      resources :ports
12.    end
13.  end
14. end

```

Esimerkki 3. Projektin routes.rb tiedoston sisältöä.

Reitityksen lopputuloksena oli seuraavanlainen URL-rakenne:
 https://api.tavu.io/vps/palvelu/versio/resurssityyppi, esimerkiksi
 https://api.tavu.io/vps/compute/v1/instances.

4.3 Todentaminen

Todentaminen päätettiin toteuttaa yksinkertaisella tavalla. Tietokantaan generoidaan asiakkaan pyynnöstä API-avain, joka yhdistetään asiakkaan tiliin. Asiakas lähettää pyynnön mukana API-avaimen otsikkotiedossa. Esimerkissä 4 on curl'in avulla toteutettu esimerkkipyyntö.

```

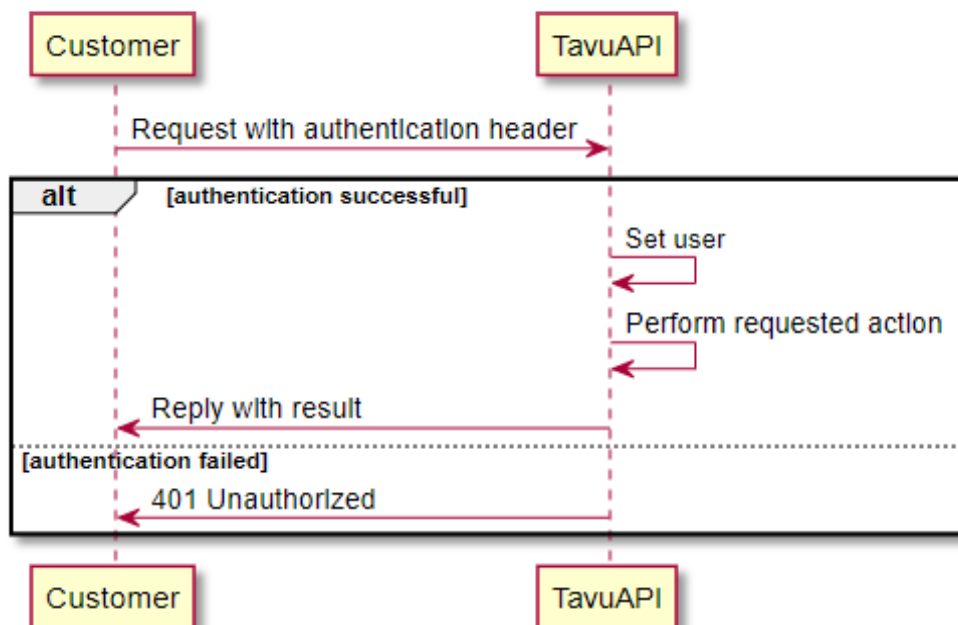
1. curl "https://api.tavu.io/vps/compute/v1/instances" -H 'Authorization: api-avain'

```

Esimerkki 4. Esimerkkipyyntö Tavun rajapintaan, jossa todentamista varten lähetetään API avain otsikkotiedoissa.

Palvelimen puolella Authorization-otsikkotiedosta parsitaan tämä API-avain ja varmennetaan, että se on olemassa. Jos käyttäjän antama API-avain löydetään tietokannasta, sovellus asettaa pyynnön ajaksi API-avaimeen liittyvän asiakkaan aktiiviseksi, jotta voidaan tarvittaessa kätevästi hakea asiakkaan tiedot pyynnön aikana. Jos

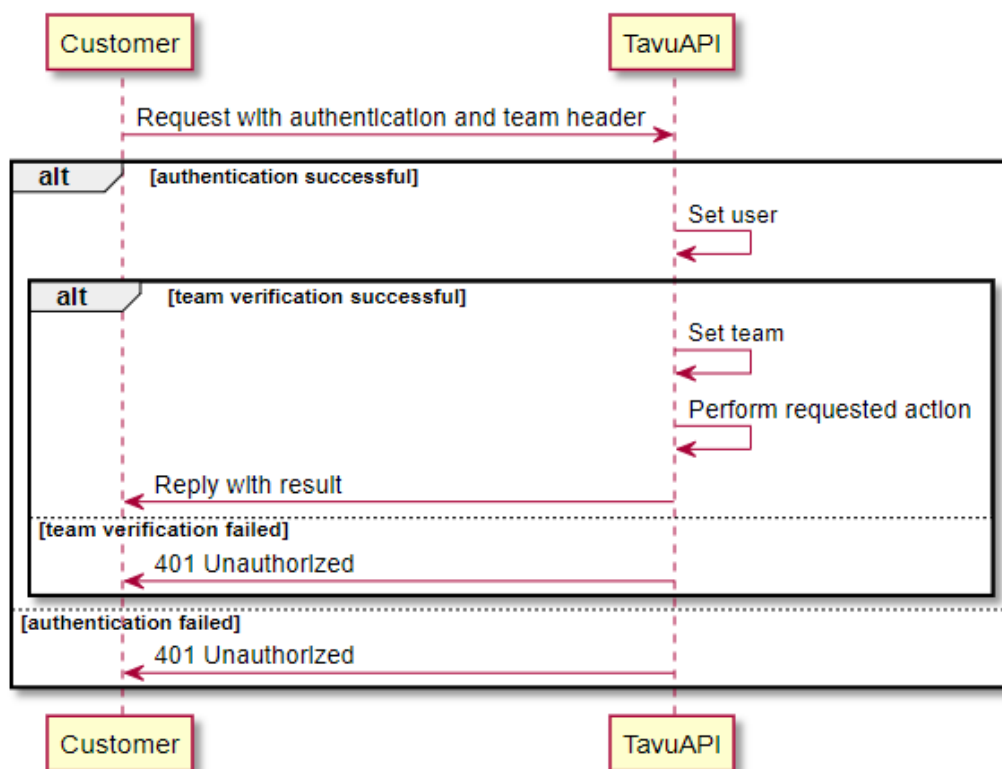
käyttäjän antamaa API-avainta ei löydy, sovellus vastaa HTTP:ssä määritellyllä 401 Unauthorized vastauksella. Esimerkissä 5 on esitetty edellä mainittu prosessi sekvenssikaaviona.



Esimerkki 5. Pelkistetty sekvenssikaavio rajapinnan autentikaatioprosessista.

4.3.1 Tiimit

Kuten aiemmin mainittu, niin hallintapaneelissa on myös mahdollista luoda tiimi, joissa käyttäjä voi olla omistaja tai jäsen. Tämä toiminnallisuus haluttiin myös tuoda rajapintaan yrityksiä varten. Käyttäjä voi pyytää rajapintaa suorittamaan toiminnon tiimille antamalla X-Team-Id-otsikkotiedon arvona halutun tiimin ID:n. Palvelin varmistaa, että käyttäjä on tiimin jäsen tai omistaja, ja että käyttäjällä on tarvittavat oikeudet suorittaa haluttu toiminto. Varmistuksien onnistuessa toiminto suoritetaan, ja epäonnistuessa vastataan 401 Unauthorized vastauksella. Esimerkissä 6 näytetään prosessi sekvenssikaaviona.



Esimerkki 6. Esimerkkipyyntö Tavun rajapintaan autorisaatio- ja tiimi id-otsikkotiedoilla.

4.4 Ohjaimet

Ohjaimet päätettiin pitää yksinkertaisena. Pyrittiin siihen, että ohjaimet sisältäisivät mahdollisimman vähän logiikkaa. Tästä syystä ohjaimet koostuvat lähinnä index, show, create, update ja destroy metodeista (joitain poikkeuksia lukuun ottamatta), joiden sisällä suoritetaan kutsuja ohjainta vastaavalle mallille. Lisäksi ohjaimet sisältävät usein metodeja, joiden avulla sallitaan parametreja resurssien luontia ja päivitystä varten. Lähes kaikki ohjaimet seuraavat samaa kaavaa. Seuraavissa aliluvuissa tarkastellaan kaikille ohjaimille yhteiseksi tarkoitettuja metodeja sisältävää apumoduulia, sekä kaavaa, jonka mukaan kaikki ohjaimet pyrittiin toteuttamaan.

4.4.1 Response-apumoduuli

Response-apumoduuli perii ohjainten vastausten kirjoittamisen helpottamiseen ja selkeyttämiseen luotuja apumetodeja. Esimerkissä 7 tarkastellaan json_response-

apumetodia. Metodi sisältää kutsun Railsissä määriteltyyn metodiin render, jolle annetaan avain-arvo-pari, joka sisältää arvot json ja status avaimille. Json-avaimen arvoksi asetetaan vastauksen keho. Kehon arvoksi annetaan avain-arvo-pari, jonka Rails automaattisesti muuttaa JSON-muotoon. Status-avaimen arvoksi asetetaan vastauksen status-otsikkotiedon arvo. Tässä tapauksessa oletusarvoksi on valittu :ok, jonka Rails automaattisesti kääntää 200 OK otsikkotiedoksi.

```
1. def json_response(object, status = :ok)
2.   render json: object, status: status
3. end
```

Esimerkki 7. json_response-apumetodin lähdekoodi.

Esimerkissä 8 tarkastellaan respond_unauthorized-apumetodia, jossa on vastauksen luomiseksi käytetty hyväksi aiemmin määriteltyä json_response-apumetodia. Tästä metodista voidaan nähdä, miten json_response-apumetodia voidaan käyttää hyödyksi koodin selkeyttämiseksi. Kyseistä metodia käytetään hyödyksi, esimerkiksi käyttäjätodentamisen epäonnistuessa.

```
1. def respond_unauthorized(message = I18n.t('unauthorized.general'))
2.   json_response({ message: message }, :unauthorized)
3. end
```

Esimerkki 8. respond_unauthorized-apumetodin lähdekoodi.

Apumoduuli perii lisäksi muita vastauksiin tarkoitettuja apumetodeja, mutta kaksi edellä mainittua metodia ovat työn kannalta tärkeimmät.

4.4.2 Resurssien haku

Index- ja show-metodit vastaavat tiedonhausta. Index-metodin avulla käyttäjä voi hakea tietoa resurssikokoelmista, ja show-metodin avulla käyttäjä voi hakea tietoa yksittäisestä resurssista. Metodit vastaavat siis CRUD-akronyymin read-osiosta.

Metodeissa kutsutaan kyseisen ohjaimen mallia vastaavia luokkametodeja. Luokkametodit pyrittiin toteuttamaan noudattamalla Railsistä tuttua nimeämiskäytäntöä, eli all-niminen metodi vastaa resurssikokoelman hausta, ja find-niminen metodi vastaa yksittäisen resurssin tietojen hausta. Lopuksi ohjain palauttaa json_response-

apumetodin avulla mallin luokkametodin palauttaman arvon. Tämä sama kaava toistuu pääasiallisesti jokaisessa rajapinnan ohjaimessa, parantaen koodin luettavuutta ja yhdenmukaisuutta. Parempi luettavuus ja yhdenmukaisuus taas helpottavat sekä soveluksen ylläpitoa että laajennettavuutta.

4.4.3 Resurssien luonti, päivitys ja poisto

Create-metodi vastaa uusien resurssien luonnista. Käyttäjä lähettää Tavu-rajapinnalle pyynnön JSON-muodossa, joka sisältää parametreja. Nämä parametrit rinnastetaan resurssien attribuutteihin, esimerkiksi asiakas voi uutta tallennuslaitetta luodessaan määrittellä sille haluamansa koon size-parametrin avulla. Ohjaimen vastuulla on määrittää sallitut parametrit, jolloin mallin ei tarvitse huolehtia siitä, kutsutaanko sen initialisointimetodia sallituilla parametreilla vai ei.

Kaava, jota kaikkien ohjainten luonti-, päivitys- ja poistometodit pyrkivät seuraamaan, voisi kuvailla näin:

1. Ohjain kutsuu sitä vastaavan mallin initialisointimetodia asiayhteyteen sopivilla parametreilla.
2. Mallin initialisointimetodin luoman objektin `os_session`-attribuutti asetetaan, jotta myöhemmin mahdollisesti tehtävät OpenStack-rajapintapyynnot tehdään oikean käyttäjän tiedoilla. `Os_session`-attribuutti pitää siis sisällään käyttäjän todentamiseen tarvittavia tietoja.
3. Ohjain kutsuu asiayhteydestä riippuen initialisoidun objektin `save!` – tai `destroy`-metodia. `Save!` -metodia kutsutaan luonnissa ja päivityksessä, kun taas `destroy`-metodia kutsutaan resurssia poistaessa. Molemmat metodit suorittavat OpenStack-rajapintapyynnön, jonka vastauksen perusteella palautetaan rajapinnan käyttäjälle sopiva vastaus pyynnön onnistumisesta.

Resurssia päivittäessä ja poistaessa käyttäjän tulee antaa Tavu-rajapinnalle lisäparametrina päivitettävän tai poistettavan resurssin ID. Luontien, päivitysten ja poistojen kohdalla on aina ensin todennettava, että pyynnön lähettäneellä käyttäjällä on tarvittavat oikeudet luoda, muokata tai poistaa resurssi. Lisäksi usein päivitysparametrien

rajoittamiselle on oma apumetodi, sillä kaikki resurssien attribuutit eivät ole aina välttämättä päivitettävissä.

4.5 Mallit

Aina kun järkeväksi nähtiin, mallien kohdalla päätettiin seurata mahdollisimman paljon Railsin jalanjäljissä. Mallin vastuu on olla tiukasti sidoksissa tietokantaan. Yhden mallin tulisi kuvata yhtä tietokannassa olevaa taulua. Malleihin voidaan lisätä toiminnallisuuksia, joiden avulla voidaan luoda, päivittää ja poistaa rivejä kyseisestä tietokantataulusta. Malleja voidaan ajatella eräänlaisena rajapintana tietokannan ja soveluksen välillä. Näiden operaatioiden lisäksi, mallin vastuulla on validoida sille annetut tiedot, sekä ylläpitää yhteyksiä muiden mallien välillä.

4.5.1 OsTools

OsTools-apumoduuli peritään jokaisessa mallissa. Apumoduuli sisältää kokoelman metodeja, joiden avulla koodista saadaan yhdenmukaisempaa, ja joiden avulla vähennetään koodin kahdennusta. Avaan seuraavaksi lyhyesti työn kannalta tärkeitä apumoduulin perimiä apumoduuleita:

- `OsResponseParser`. Apumoduuli, joka sisältää metodeja OpenStack-rajapinnan vastauksien käsittelyyn ja jäsenyykseen.
- `OsValidationTools`. Apumoduuli, jonka määrittelemien metodien avulla helpotetaan attribuuttien validointia. Sovelluksesta tunnistettiin kaikki yleisimmät attribuutit, kuten resurssien nimet, ja luotiin niiden validointiin yleisiä validointimetodeja, joiden avulla saadaan helposti validoitua käyttäjien rajapinnalle lähettämiä arvoja.
- `OsDbTools`. Sisältää apumetodeja, joiden avulla luodaan tietokantayhteys OpenStackin tietokantaan.

Apumoduulien perimisen lisäksi, OsTools-moduulissa määritellään metodeja, jotka todettiin jokaisessa mallissa toistuviksi. Esimerkiksi, jokaisesta mallin edustamasta resurssista tarvitsee pystyä hakemaan resurssikokoelman tiedot, sekä yksittäisen resurssin tiedot, joten toteutettiin yleiset `all-` ja `find-` metodit. `All-` metodin vastuulla on kutsua

jokaisessa mallissa toteutettua `_load_all`-privaattimetodia, joka suorittaa SQL-kyselyn OpenStackin tietokantaan ja palauttaa kysytyt tiedot. All-metodi käy palautetut resurssikokoelman tiedot läpi resurssi resurssilta, ja initialisoi jokaisen resurssin tiedoilla objektin, ja lisää initialisoidun objektin palautettavien objektien kokoelmaan. Find-metodi toimii samalla kaavalla, mutta objektikokoelman sijaan se palauttaa vain yhden objektin.

4.5.2 Attribuutit ja `as_json`

OpenStackin tietokannassa olevia attribuuttien nimiä ei välttämättä haluttu käyttäjille suoraan paljastaa, tai joissakin tapauksissa tietokannassa olevat attribuuttien nimet olivat sekavia. Joissakin tapauksissa myös malleille tarvittiin attribuutteja sisäiseen käyttöön, esimerkiksi logiikan ohjaamiseen, ja näitäkään ei haluttu käyttäjille paljastaa. Tästä syystä malleille kehitettiin tapa uudelleennimetä ja rajoittaa Tavun rajapinnassa palautettavia attribuutteja. Alla olevassa esimerkissä 9 nähdään tallennuslaitteen mallista otetut attribuuttimääritelmät. Muiden mallien attribuuttimääritelmät seuraavat samaa kaavaa:

- `OS_ATTRS`. OpenStackin tietokannasta tulevat attribuutit.
- `API_ATTRS`. Tavun rajapinnan palauttavat attribuutit.
- `INTERNAL_ATTRS`. Sisäisesti käytettävät attribuutit.

```
1. OS_ATTRS = %w(id created_at status project_id attach_status display_name in
  stance_uuid display_description mountpoint attach_time snap-
  shot_id).freeze
2.
3. API_ATTRS = %w(id name size created_at status description snapshot_id snaps
  hots).freeze
4.
5. INTERNAL_ATTRS = %w(os_session errors).freeze
```

Esimerkki 9. Tallennuslaitemallin attribuuttimääritelmät.

Kun objekti annetaan Railsin `render` kutsun `json`-avaimen arvoksi, niin Rails kutsuu automaattisesti objektin `as_json`-metodia. Tästä syystä jokaiselle mallille toteutettiin `as_json`-metodi, jonka avulla hallittiin mitä attribuutteja rajapinnan vastauksissa palautetaan. Esimerkissä 10 nähdään kyseisen metodin toiminta. Except avainsanalla annettu kokoelma poistetaan muodostetusta avain-arvo-parista. Esimerkissä olevan

toteutuksen avulla vastauksesta siis poistetaan kaikki attribuutit, joita ei löydy API_ATTRS kokoelmasta.

```
1. def as_json(options = { })
2.   super(except: [(OS_ATTRS - API_ATTRS), *INTERNAL_ATTRS])
3. end
```

Esimerkki 10. as_json-metodin lähdekoodi.

Attribuuttien uudelleennimeäminen toteutettiin omassa metodissaan, jota vuorostaan kutsutaan jokaisen mallin initialisointimetodissa. Esimerkissä 11 nähdään, että esimerkiksi tallennuslaitteiden kohdalla OpenStack-tietokannasta löytyvät display_name ja display_description-attribuutit haluttiin uudelleennimetä name ja description-nimiksi.

```
1. def initialize(attributes = { }, slim: false)
2.   setup_attributes unless slim
3. end
4.
5. def setup_attributes
6.   @description = self.display_description unless @description
7.   @name        = self.display_name unless @name
8. end
```

Esimerkki 11. Esimerkkitoteutus, jonka avulla attribuutit uudelleennimetään initialisoinnin aikana.

4.5.3 SQL-kyselyt

Kuten luvussa 3 mainittiin, rajapinnan toteutuksen yhteydessä päätettiin ratkaista myös hallintapaneelissa tiedonhakua haittaava ongelma SQL-kyselyiden avulla. Ongelman ratkaisemiseksi malleihin toteutettiin _load_all- ja _load_one-metodit, joiden vastuulla oli suorittaa tiedonhaku suoraan OpenStackin tietokannasta. Näihin metodeihin myös viitattiin luvussa 4.4.1, kun kerrottiin OsTools-apumoduulin toiminnasta.

_load_all-metodi vastaa resurssikokoelman hakemisesta. Sitä kutsutaan käyttäjän OpenStack projekti ID:llä, jonka jälkeen metodi ottaa yhteyden oikeaan tietokantaan, ja palauttaa tietokantakyselystä palautuneen tiedon. Esimerkissä 12 nähdään esimerkkitoteutus metodista.

```
1. def self._load_all(project_id)
2.   client = cinder_client
```

```

3.   volumes_data = cli-
    ent.query("SELECT * FROM volumes WHERE volumes.project_id = '#{client.escap
e(project_id)}' AND volumes.deleted = 0")
4.   client.close
5.   return volumes_data
6. end

```

Esimerkki 12. Esimerkki, miten resurssikokoelman hakeminen OpenStackin tietokannasta voitaisiin toteuttaa.

Toisin kuin `_load_all`, `_load_one`-metodi vastaa yhden tietyn resurssin tietojen hausta. Tietokantayhteyden muodostamisen lisäksi, metodiin on lisätty yksi rivi, joka nostaa poikkeuksen, jos käyttäjän antamalla tiedoilla ei löydetty resurssia.

```

1. def self._load_one(project_id, id:)
2.   client = cinder_client
3.   volume_data = cli-
    ent.query("SELECT * volumes WHERE volumes.project_id = '#{client.escape(pro
ject_id)}' AND volumes.id = '#{client.escape(id)}' AND volumes.deleted = 0"
    ).first
4.   client.close
5.   raise TavuApiErrors::ResourceNotFound.new(id, self.new(slim: true)) un-
    less volume_data
6.   return volume_data
7. end

```

Esimerkki 13. Yksittäisen resurssin tiedonhaun lähdekoodi.

5 TESTAUS

Rajapinnan päätepisteiden testaus toteutettiin Postman-sovelluksen avulla. Postman-sovellus sallii rajapintapyyntöjen automaattisen testauksen, sekä kokoelmien muodostamisen. Nämä kokoelmat voidaan jakaa usean henkilön kesken, joten useampi kehittäjä pystyi samanaikaisesti testaamaan rajapinnan eri osa-alueita, ja nämä osa-alueet voitiin lopuksi yhdistää yhdeksi yhtenäiseksi kokoelmaksi.

Tiedonhaun nopeutusta testattiin Railsin mini-profiler -lisäosalla. Lisäosa mittaa kaikkien sovelluksen suorittamien pyyntöjen aikaa. Mini-profiler ei ollut kaikista mittausmenetelmistä tarkin, mutta sen avulla nähtiin jo hyvin, kuinka rajapinta nopeuttaa tiedonhakua. Kuvassa 10 nähdään kuvankaappaus uusista hallintapaneelin virtuaalipalvelimen hallintasivulla suoritettavien kyselyjen kestoista. Kuvankaappauksesta nähdään, että kyselyjen kokonaiskesto vähentyi noin 600:sta millisekunnista noin 180:een millisekuntiin.

| | duration (ms) | from start (ms) | query time (ms) |
|--|---------------|-----------------|-----------------|
| GET http://dev102.shellit.fi:80/instances/007... | 13.2 | +0.0 | 3 sql 0.3 |
| Executing action: show | 75.6 | +10.0 | 21 sql 3.9 |
| Net::HTTP GET /vps/compute/v1/instances/... | 0.3 | +29.0 | |
| Net::HTTP GET /vps/compute/v1/instance... | 22.5 | +29.0 | |
| Net::HTTP GET /vps/image/v1/instance-sna... | 0.3 | +55.0 | |
| Net::HTTP GET /vps/image/v1/instance-s... | 7.9 | +56.0 | |
| Rendering: layouts/_messages | 3.6 | +68.0 | |
| Net::HTTP GET /vps/storage/v1/volumes/ | 0.3 | +75.0 | |
| Net::HTTP GET /vps/storage/v1/volumes/ | 7.4 | +75.0 | |
| Net::HTTP GET /vps/networking/v1/network... | 0.2 | +84.0 | |
| Net::HTTP GET /vps/networking/v1/netwo... | 59.7 | +84.0 | |
| Net::HTTP GET /vps/networking/v1/security... | 0.3 | +181.0 | |
| Net::HTTP GET /vps/networking/v1/securit... | 16.1 | +181.0 | |

Kuva 10. Virtuaalipalvelimen hallintasivun kyselyjen kokonaiskesto rajapinnan toteutuksen jälkeen.

6 YHTEENVETO

Projektin lähtökohtana oli uuden, laajennettavissa olevan rajapinnan toteuttaminen olemassa olevan sovelluksen perusteella. Rajapinta toteutettaisiin, jotta yrityksen asiakkaat voisivat ohjelmallisesti hallinnoida virtuaalisia resurssejaan, ja jotta yritys voisi tulevaisuudessa laajentaa rajapinnan toimintaa yrityksen muilla palveluilla. Projekti oli itselleni opettavainen kokemus, sillä vaikka olin ennen rajapintoja käyttänyt, ja vaikka olin tietoinen, että Ruby on Rails -ohjelmistokehyksen reititys perustui resursipohjaiseen malliin, en ollut sisäistänyt kuinka vahvasti se oli sidoksissa REST-arkkitehtuurimalliin.

Toimeksiannon tavoitteet, pieniä poikkeuksia lukuun ottamatta, saatiin kaikki kohdatua. Rajapinnan ja nykyisen hallintapaneelin toiminnallisuudet saavuttivat virtuaalisten resurssien osalta ns. feature-parity:n, eli rajapinnan kautta asiakas pystyy hallinnoimaan virtuaalisia resurssejaan yhtä syvällisesti kuin hallintapaneelinkin kautta. Ongelmallisten tiedonhakujen kestoa saatiin vähennettyä roimasti, joka johti hallintapaneelin nopeutumiseen. Lisäksi rajapintaan saatiin toteutettua yrityksen sisäiseen käyttöön tarvittavia admin-ominaisuuksia, vaikka niistä ei työssä tietoturvan vuoksi kerrottukaan. Lopputuloksena projektista syntyi varteenotettava beetaversio rajapinnasta, jota saadaan tulevaisuudessa helposti laajennettua, ja paranneltua.

Puutteita ja parannettavaakin rajapinnasta jäi. Nimellisenä mainittakoon HATEOAS:n puute. Rajapintaa ei siis teknisesti ottaen voida kutsua täysin REST-arkkitehtuurimallia noudattavaksi. Deadlinen lähestyessä kuitenkin tehtiin päätös, että HATEOAS:n toteutus jätetään myöhemmälle. Tämä rajoittaa rajapinnan koneellista käyttöä, mutta sitä ei koettu varsinaiseksi prioriteetiksi tällä hetkellä.

Kaiken kaikkiaan, projektissa onnistuttiin hyvin, ja toteutettu rajapinta on helpottanut huomattavasti sisäistä käyttöä, sekä uusien palvelujen suunnittelussa.

LÄHTEET

Rails Routing from the Outside In. 2018. Viitattu 16.12.2018. Rails Routing from the Outside In. <https://guides.rubyonrails.org/routing.html>

Getting Started with Rails. 2018. Viitattu 16.12.2018. Getting Started with Rails. https://guides.rubyonrails.org/getting_started.html

RFC2616. 1999. Viitattu 4.12.2018. Hypertext Transfer Protocol -- HTTP/1.1. <https://tools.ietf.org/html/rfc2616>

DrDobbs. 2004. Measuring API Usability. Viitattu 25.3.2018. <http://www.drdobbs.com/windows/measuring-api-usability/184405654>

ProgrammableWeb. 2018. Research Shows Interest in Providing APIs Still High. Viitattu 25.3.2018. <https://www.programmableweb.com/news/research-shows-interest-providing-apis-still-high/research/2018/02/23>

Fielding, T. R. 2000. Architectural Styles and the Design of Network-based Software Architectures. Viitattu 4.12.2018. <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

