OULUN AMMATTIKORKEAKOULU

Annika Ojala

MAXTECH PUBLIC API

Maxtech Pro -järjestelmän avoin rajapinta

MAXTECH PUBLIC API

Maxtech Pro -järjestelmän avoin rajapinta

Annika Ojala
Opinnäytetyö
Syksy 2020
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Annika Ojala
Opinnäytetyön nimi suomeksi: Maxtech Public API – Maxtech Pro-järjestelmän avoin rajapinta
Opinnäytetyön nimi englanniksi: Maxtech Public API – Public API for Maxtech Pro system
Työn ohjaaja(t): Lasse Haverinen, Jenna Moisejeff
Työn valmistumislukukausi ja -vuosi: Syksy 2020
Sivumäärä: 46

Opinnäytetyö perustuu Max Technologies Oy:n toimeksiantoon, jossa aiheena oli suunnitella ja osittain toteuttaa helposti laajennettavissa oleva Maxtech Pro-järjestelmän avoin rajapinta ja rajapinnalle dokumentaatio. Rajapinnan kautta järjestelmän dataa voidaan hyödyntää sisäisessä tai kolmannen osapuolen ohjelmistokehityksessä.

Työssä rajapintaan kehitettiin REST-arkkitehtuuria käyttäen tietojen lukutoiminnallisuus sekä organisaatiokohtainen autentikointi, jolla todennetaan rajapinnan käyttäjä ja rajataan tietoihin pääsy. Lisäksi rajapinnalle tehtiin interaktiivinen dokumentaatio OpenAPI-määrittelyä ja ReDoc-dokumentaatiotyökalua käyttäen.

Tuloksena syntyi toimivat rajapinta- ja dokumentaatiopohjat jatkokehitystä varten. Ennen rajapinnan avaamista asiakkaille rajapintaan kehitetään oleellisimmat osat valmiiksi.

Asiasanat: REST, ohjelmistorajapinta, OpenAPI

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

Author: Annika Ojala
Title of thesis: Maxtech Public API – Public API for Maxtech Pro system

Supervisor(s): Lasse Haverinen, Jenna Moisejeff
Term and year when the thesis was submitted: Autumn 2020
Pages: 46

The purpose of this thesis was to plan and partially implement easily expandable open interface and documentation for Maxtech Pro system. Through the interface, system data can be utilized internally or by third-party software development.

In the thesis, reading functionality was developed for application programming interface using REST-architecture. Organization-specific authentication was also developed for the interface, which authenticates the API user and restricts access to the data. In addition, interactive documentation was made for API using OpenAPI specification and the ReDoc documentation tool.

The result was functional and expandable API and documentation for further development. The most essential parts of the API will be developed before opening API for customers.

Keywords: REST, application programming interface, OpenAPI

ALKULAUSE

Työn tilaajana toimi Max Technologies Oy. Työn tilaajan edustaja toimi Jenna Moisejeff, ohjaavana opettajana Lasse Haverinen sekä kielenohjaajana Tuula Hopeavuori.

Erityisesti haluan kiittää työn tilaajan edustajana toiminutta Jenna Moisejeffiä sekä omaa rakasta perhettäni opinnäytetyön aikana saadusta tuesta ja kannustuksesta. Lisäksi kiitän ohjaavana opettajana toiminutta Lasse Haverista annetuista vinkeistä ja ohjauksesta, työkavereita opinnäytetyöhön saadusta avusta sekä kielenohjaajana toiminutta Tuula Hopeavuorta raportin kielellisestä ohjauksesta.

Oulussa 20.9.2020

Annika Ojala

SISÄLLYS

1 JOHDANTO

Työn tilaajana toiminut Maxtech (Max Technologies Oy) on suomalainen työnhallinta- ja paikannuspalveluita tarjoava yritys, joka on toiminut jo lähes kolmetoista vuotta. Opinnäytetyön aiheena oli suunnitella ja osittain toteuttaa helposti laajennettavissa oleva Maxtech Pro -järjestelmän rajapinta ja rajapinnalle dokumentaatio. Rajapinnan kautta yrityksen asiakkaat voivat hyödyntää järjestelmään kerättyä dataa omassa raportoinnissa, tilastoinnissa ja omien työkalujen kehittämisessä.

Opinnäytetyön tehtävänä oli toteuttaa Maxtech Pro -järjestelmän rajapintaan REST-arkkitehtuuria käyttäen tietojen lukutoiminnallisuus. Lukutoiminnallisuuden kautta yrityksen asiakkaat voivat hakea kaikkia organisaation tapahtumia ja tärkeimpiä tietoja lukumuodossa. Hakua olisi mahdollisuus myös suodattaa.

Lukutoiminnallisuus tuli toteuttaa PHP-kielellä ja tukemaan pelkästään POST-operaatiota yrityksen olemassa olevan API-rakenteen takia. Rajapintaan tulevat pyynnöt ja pyyntöjen vastaukset tuli käsitellä JSON-muotoisena. Pyyntöjen käsittelyssä tuli validoida pyynnön mukana tulleet syötteet rajapinnan väärinkäytön estämiseksi. Rajapintaan tuli kehittää myös organisaatiokohtainen autentikointi, jolla todennetaan rajapinnan käyttäjä ja rajataan tietoihin pääsy.

Tehtävänä oli myös tehdä rajapinnalle dokumentaatio, josta ilmenee tiedon haku-, lisäys- ja muokkaustoiminnallisuudet. Dokumentaatiosta tuli ilmetä rajapinnassa käytettävät parametrit, resurssien kenttien kuvaukset sekä palautettavan tiedon muoto formaatteineen. Rajapinta tuli myös versioida. Dokumentaation esittämiseen tuli tehdä visuaalinen käyttöliittymä.

Tarve asiakkaille avattavasta rajapinnasta tuli yrityksen ja asiakkaiden suunnalta. Yrityksen olemassa olevaa rajapintaa haluttiin laajentaa julkisella rajapinnalla, josta voidaan hyödyntää järjestelmän dataa yrityksen sisäisessä ja kolmannen osapuolen ohjelmistokehityksessä. Rajapinnan kehittäminen on tärkeä osa yrityksen liiketoimintastrategiaa ja platform-ajattelua. Rajapinnoilla mahdollistetaan uusia tapoja kehittää ja tarjota palveluita asiakkaille sekä tehostaa kumppaniver-

koston toimintaa avaamalla toimintoja asiakkaiden hyödynnettäviksi. Rajapinnoilla myös parannetaan omien järjestelmien tiedon saatavuutta ja yhdistettävyyttä muiden järjestelmien kanssa.

Koska opinnäytetyön tekijä on käsitellyt rajapintoja aikaisemmin yritysprojektien, työharjoittelun ja työn ohessa, nähtiin aihe sopivana jatkumona opinnäytetyön aiheena.

2 REST-RAJAPINTA

Ohjelmistorajapinta eli API (Application Programming Interface) mahdollistaa eri palveluiden kommunikaation keskenään reaaliaikaisesti ja automatisoidusti. Rajapintojen avulla organisaatiot voivat rakentaa integraatioita ohjelmistojen välille. Rajapintoja on erilaisia: sisäisiä, avoimia tai avointa dataa tarjoavia rajapintoja ja ne voidaan toteuttaa eri tavoin. Yksi toteutustapa on soveltaa rajapinnassa REST-arkkitehtuuria, joka tässä luvussa käydään läpi lyhyesti.

2.1 REST-arkkitehtuuri

REST (Representional State Transfer) on Roy Fieldingin väitöskirjassa esittelemä järjestelmän resursseihin keskittyvä arkkitehtuurinen tyyli verkkopohjaisten sovellusten suunnitteluun. Fieldingin REST-tyylin määritelmän jättäessä paljon avointa tilaa ja tulkintaa on kuvausta vaikea muuntaa suoraan helpoiksi käytännön ohjeiksi. Tämä johtaa erilaisiin tulkintoihin, mistä seuraa kiistanalaisia keskusteluja mikä tekee verkkopalvelusta ”RESTfulin” ja mikä ei. (1, luku 4.) Täytyy huomata, että termit REST ja RESTful ovat pohjimmiltaan samaa asiaa. Kun API:n kuvaillaan olevan RESTful, API soveltaa REST-arkkitehtuurityylin ominaisuuksia (2).

Leonard Richardson ja Sam Ruby antavat REST-arkkitehtuurityylille määritelmän nimeltään ROA eli resurssisuuntautunut arkkitehtuuri (Resource-Oriented Architecture). Se sisältää seuraavat neljä käsitettä:

1. Resurssit
2. Resurssien nimet (URI:t eli Uniform Resource Identifiers)
3. Resurssin esitystapa
4. Linkit resurssien välillä. (1, luku 4.)

ROA käsittää myös neljä ominaisuutta:

1. Osoitettavuus (Addressability). Jokaisen resurssin tulee olla yksilöllisesti osoitettavissa ainakin yhdellä tunnisteella.

2. Tilattomuus (Stateless). Jokainen pyyntö käsitellään erillään muista. Istuntoja ei tallenneta palvelimen puolelle, koska kaikki palvelimen tarvitsema tieto kulkee pyynnön mukana.
3. Yhdistettävyys (Connectedness). Joskus resurssin representaatio voi sisältää linkkejä toisiin resursseihin.
4. Yhdenmukainen käyttöliittymä (A uniform interface). Pyynnöt ja vastaukset käsitellään määritetyllä tavalla. Palvelimen tulisi ymmärtää, mitä asiakas haluaa palvelimelle sanoa. (1, luku 4.)

Rajapinnan RESTful-luokitteluun käytetään yleisesti Richardsonin kypsyysmallia (Richardson Maturity Model), jossa rajapinta pisteytetään rajapinnan noudattamien rajoitusten mukaan. Richardsonin kypsyysmalli on tosin muokattu versio alkuperäisestä Richardsonin luomasta kypsyysheurestiikasta (The Maturity Heuristic). Richardson kertookin vuonna 2019 pitämässään RESTFest-puheessaan, että noudattamalla jokaista mallin rajoitusta ei saa mallin huipulta palkintoa. Richardsonin mukaan jokaisen rajoituksen noudattamiseen on joko tekninen tai poliittinen syy ja parhaimmassa tapauksessa rajapinnasta saadaan yhteen toimiva, jos kaikkia rajoituksia noudatetaan. (3.)

2.2 Resurssit

RESTful-verkkopalvelut kattavat joukon toisiinsa kytkettyjä resursseja. Resurssit ovat verkkopohjaisten järjestelmien perusrakennuspalikoita ja ne voivat olla kaikkea, mitä internetissä paljastetaan. (4.) Resurssit ovat niin kutsuttuja tietolähteitä todellisen maailman resursseista, kuten varastossa olevista tavaroista, kirjoista, vaatteista ja kartoista (5). Resurssi on siis jotain, joka voidaan tallentaa tietokoneelle ja edustaa bittivirtana, kuten dokumenttina, tietokantarivinä tai algoritmin suorittamana tuloksena (1, luku 4).

Esimerkkeinä resursseista voisi olla seuraavat:

- Ohjelmiston julkaisun versio 1.0.3
- Tiekartta Arkansasin Little Rockista
- Hakemisto meduusoja koskevista resursseista
- Seuraava alkuluku luvun 1024 jälkeen
- Kahden tuttavan välinen suhde

- Luettelo avoimista virheistä virhetietokannassa. (1, luku 4.)

Koska HTTP-protokollaa (Hypertext Transfer Protocol) pidetään ensisijaisena protokollana RESTful-palveluiden rakentamisessa, resurssit tunnistetaan niiden yksilöllisellä resurssitunnisteella URI:llä (6). URI identifioi verkkoresurssin yksilöllisesti ja tekee siitä samalla osoitettavan tai manipuloitavan HTTP:n avulla (4). URI on resurssin nimi ja osoite. URI määrittelee vain yhden resurssin eikä se voi osoittaa useampaan kuin yhteen resurssiin. Jokaisella resurssilla tulee olla vähintään yksi URI. Jokainen ylimääräinen URI tosin laimentaa kaikkien muiden arvon, sillä ei voida varmistaa, että jokainen URI viittaa samaan resurssiin. (1, luku 4.)

Richardsonin ja Rubyn mukaan resurssien URI:n tulisi olla kuvaava ja sillä tulisi olla selkeä ennakoitavissa oleva rakenne. Tämä ei ole REST-arkkitehtuurityylin ehdoton sääntö, mutta kuvaavan ja selkeän rakenteen ansiosta asiakkaiden on helppo käyttää palvelua tuntemalla palvelun URI-tunnisteen rakenne. (1, luku 4.)

Esimerkiksi resurssien kissa ja koira tulisi löytyä poluista

- /animal/cat
- /animal/dog

eikä poluista

- /search/cat
- /information-of/dog.

Noutaessa palvelimelta resurssia voi palvelin palauttaa pyydetyn resurssin lisäksi tietoja muista liittyvistä resursseista. Nämä muut resurssit ovat pyydetyn resurssin objekteja. (1, luku 4.)

2.3 HTTP-protokolla

HTTP tukee kahden tyyppisiä viestejä, joita ovat pyynnöt ja vastaukset. Pyynnössä palvelimen asiakas lähettää palvelimelle pyynnön resurssista. HTTP-

pyyntö koostuu pyynnön vastaanottavan päätepisteen URL-osoitteesta, ylätunnisteista ja HTTP-metodeista. Pyyntö voi sisältää myös viestin rungon, jossa rajapintaan voidaan lähettää parametrejä. HTTP-pyynnön metodit ovat seuraavat:

- GET hakee sisältöä tietystä resurssista.
- POST toimittaa tiedot palvelimelle käsittelyä varten. POST-metodia käytetään usein tiedon muokkaamiseen, lisäämiseen tai muiden palvelimen määrittelemien operaatioiden suorittamiseen. POST-pyynnöissä viestin runko sisältää yleensä dataa, jonka palvelimen asiakas tarjoaa palvelimelle.
- PUT lähettää tietoja palvelinresurssin päivittämiseksi. PUT-metodilla korvataan sisältö tai luodaan uusi sisältö puuttuvan sisällön tilalle.
- DELETE lähettää poistettavan resurssin viittauksen.
- Muita pyyntötyyppejä ovat HEAD, OPTIONS ja PARTIAL, mutta niitä harvoin käytetään REST-arkkitehtuurissa. (7; 2.)

POST-, GET-, UPDATE- ja DELETE-metodit vastaavat tiedon käsittelyyn tarvittavia CRUD (Create, Read, Update, Delete) -operaatioita.

HTTP-pyynnön saadessaan palvelin palauttaa vastauksen. Vastauksessa asiakkaalle annetaan tietoa siitä, miten palvelin on käsitellyt pyynnön. Vastaus sisältää HTTP-tilakoodin, ylätunnisteen ja resurssin. (2.)

HTTP tarjoaa viisi pääluokkaa tilakoodeille:

- 1xx: Informoiva: pyyntöviesti vastaanotettu ja prosessia jatketaan.
- 2xx: Onnistunut: toiminta vastaanotettiin, ymmärrettiin ja hyväksyttiin onnistuneesti.
- 3xx: Uudelleenohjaus: lisätoimia vaaditaan pyynnön loppuun saattamiseksi.
- 4xx: Palvelimen asiakkaan pyynnössä virhe: pyyntö sisältää huonon rakenteen tai pyyntöä ei voida täyttää.
- 5xx: Palvelimen virhe: palvelin epäonnistui täyttämään oikeassa muodossa olevaa pyyntöviestiä. (2.)

Palvelimen asiakkaan REST-rajapintaan lähettämään pyyntöön tulisi vastata yhdellä tilakoodilla. Virheellinen tilakoodin käyttö johtaa harhaan ja estää asiakassovelluksia prosessoimasta vastauksia oikein. Esimerkiksi tilakoodi 500 tulisi ainoastaan palauttaa silloin, kun palvelimen puolella on virhe. Jos palvelimen asiakas on muotoillut pyynnön väärin, olisi tilakoodi 400 sopivampi. (2.)

2.4 Rajapinnan autentikointi

Todennuksella eli autentikoinnilla tunnistetaan rajapinnan käyttäjä ja sillä tarjotaan turvattu käyttäjäkohtainen sessio. Todennuksella estetään myös roskapostia ja väärinkäyttöä. (8.)

Todennukseen voidaan käyttää API-avainta, jonka asiakas tarjoaa API-palveluita käyttäessään. API-avaimen on tarkoitus olla salaisuus, jonka vain asiakas ja palvelin tietävät. (8.) API-avaimeen perustuva tunnistautumista pidetään turvallisena vain käytettynä yhdessä toisen turvallisuusmekanismin kanssa, kuten HTTPS:n (Hypertext Transport Protocol Secure) ja SSL:n (Secure Sockets Layer) kanssa. (9.)

API-avaimen avulla rajapinnan käyttöä voidaan seurata. Seuranta mahdollistaa muun muassa mahdollisten virhetilanteiden tallentamisen sekä rajapinnan käytön rajaamisen niin, että rajapinnan käyttäjä voi tehdä pyyntöjä tietyn verran tietyn ajan sisällä. Suuremmista pyyntömääristä käyttäjältä voidaan jopa pyytää maksua. (9.)

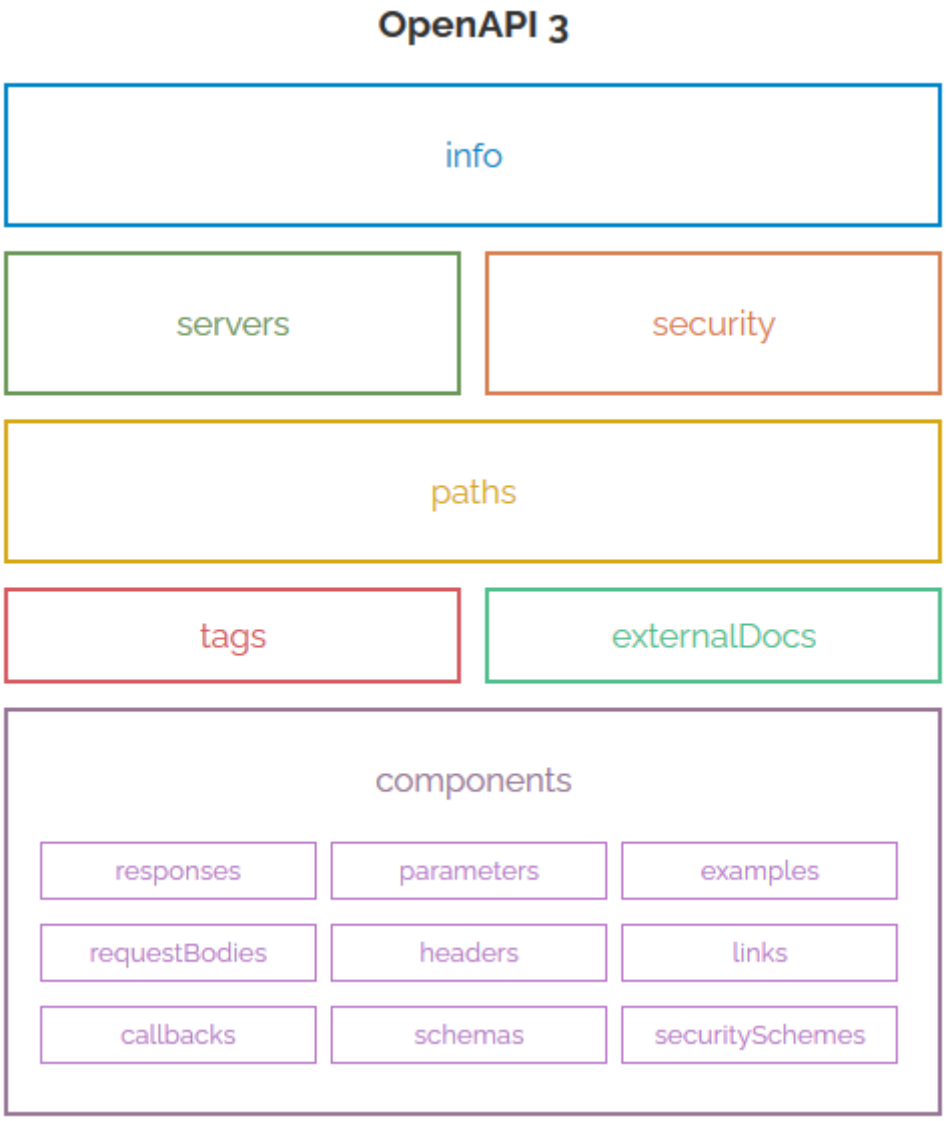
Autentikointia ei tule sekoittaa auktorisointiin, jossa tunnistetulle käyttäjälle annetaan käyttöoikeudet tiettyihin palveluihin ja järjestelmiin.

2.5 Rajapinnan dokumentointi

Jotta rajapinnan käyttäjä osaisi käyttää rajapintaa itsenäisesti, tulee rajapinta dokumentoida. Dokumentointi on tärkeää tehdä riittävällä tarkkuudella, jotta rajapinnan käyttö olisi mahdollisimman vaivatonta ja käyttäjän ei tarvitsisi kysyä lisätietoja rajapinnan toimittajalta. Dokumentaatiossa tulisi pyrkiä ajantasaisuuteen ja virheettömyyteen. Dokumentaatiota tulisi myös päivittää aina, kun rajapinnan toiminnallisuutta muutetaan tai päivitetään. (2.)

Rajapintojen määrittelyssä voidaan käyttää OpenAPI-määrittelystandardia, joka on alun perin johdettu Smartbear Softwaren OpenAPI-aloitteelle vuonna 2015 lahjoittamasta Swagger-määrittelystä. OpenAPI 3.0 on ensimmäinen OpenAPI-aloitteen virallinen julkaisu lahjoituksen jälkeen. Määrittelyllä luodaan RESTful-käyttöliittymä kartoittamalla kaikki siihen liittyvät resurssit ja toiminnot. OpenAPI-määrittely voidaan kirjoittaa joko JSON- (JavaScript Object Notation) tai YAML-muotoisena (YAML Ain't Markup Language). (10.)

OpenAPI 3.0 -määrittely on jäsennelty kuvassa 1 näkyviin osiin.



KUVA 1. OpenAPI 3.0 -määrittely jäsenneltynä eri osiin (11)

Info-osio sisältää rajapintaan liittyvät metatiedot. Info-osion on tarkoitus antaa rajapinnan käyttäjälle korkeatasoinen kuvaus rajanpinnan toiminnasta. (12.)

Servers-osio sisältää tietoja API-palvelimen sijainnista. OpenAPI 3.0 -määritelmä tukee useiden palvelimien määrittelemisen. (12.)

Security-osiossa määritellään rajapinnassa käytetyt suojausmekanismit. Suojausmekanismit määritellään myöhemmin esiteltävässä Components-osiossa. (12.)

Paths-osiossa määritellään rajapinnan päätepisteet ja vastaavat HTTP-metodit. Jokaisen metodin alle määritellään yksityiskohtaisesti pyyntöön tarvittavat parametrit, mahdollinen pyynnön runko ja palautettavat vastaukset. Lisäksi voidaan määritellä esimerkiksi polun lyhyt kuvaus ja tagi. (12.)

Tags-osio on luettelo tageista rajapinnan dokumentaation hallintaa varten. Tageja voidaan käyttää polkujen loogiseen järjestelyyn. (12.)

ExternalDocs-osiossa voidaan määritellä muut ulkoiset dokumentaatiot. (12.)

Components-osio sisältää joukon erilaisia uudelleenkäytettäviä komponentteja. Uudelleenkäytettävät komponentit voivat olla esimerkiksi vastauksia, parametrejä, esimerkkejä tai skeemoja eli syötettävien ja lähetettävien tietotyyppien määrittelyjä. Näihin komponentteihin voidaan viitata missä tahansa polussa käyttämällä $ref-viittausta. (12.)

Dokumentaation käyttöliittymän tekemiseen voidaan käyttää esimerkiksi ReDoc-dokumentaatiotyökalua, jolla saadaan OpenAPI-määrittelystä luotua interaktiivinen käyttöliittymä. ReDoc perustuu avoimeen lähdekoodiin ja on täysin ilmainen.

3 RAJAPINNAN TOTEUTUS

Yrityksen olemassa olevaa rajapintaa haluttiin laajentaa julkisella rajapinnalla, josta voidaan hyödyntää järjestelmän dataa yrityksen sisäisessä ja kolmannen osapuolen ohjelmistokehityksessä. Opinnäytetyön alkuvaiheilla opinnäytetyön painopistettä siirrettiin rajapinnan määrittelystä ja suunnittelusta rajapinnan toteutukseen, sillä rajapinnan kehitystä oli ehditty jo aloittamaan. Opinnäytetyön tarkoituksena oli laajentaa rajapintaa lisäämällä toiminnallisuus tietojen lukemista varten. Tietojen lukemiseen haluttiin myös suodattamismahdollisuus. Rajapinnan tietojen lisäys- ja muokkaustoiminnallisuudet lisättiin asiakkaiden tarpeista opinnäytetyön ulkopuolella.

Rajapintaan toteutettiin lukutoiminnallisuuden lisäksi organisaatiokohtainen autentikointi. Autentikoinnilla tarkistetaan käyttäjäoikeudet ja rajoitetaan tiedon hakua koskemaan vain kyseistä organisaatiota. Autentikointiin tarkoitetuille avaimille luotiin oma näkymä, josta organisaation avaimia voidaan tarkastella ja lisätä.

Yrityksen rajapinnalle tehtiin myös dokumentaatio, joka sisältää opinnäytetyössä tehdyn lukemistoiminnallisuuden sekä olemassa olevat lisäys- ja muokkaustoiminnallisuudet. Rajapinnan dokumentaatiosta tuli ilmetä rajapintaan tehtävien pyyntöjen parametrit, kenttien kuvaukset sekä palautettavan tiedon muoto formaatteineen eli mitä rajapinnasta palautetaan ja missä muodossa. Dokumentaatio esittämiseen haluttiin interaktiivinen käyttöliittymä.

3.1 Tietojen lukurajapinta ReadApi

Rajapintaan toteutettiin tietojen lukemista varten ReadApi, jota kautta halutun resurssin tietoja voidaan hakea. Tietokannan tietojen luku on toteutettu vain POST-tyyppisellä rajapintapyynnöllä yrityksen olemassa olevan API-rakenteen takia. ReadApi toteutettiin verkkokehitykseen soveltuvalla yleiskäyttöisellä PHP-ohjelmointikielellä ja rajapinta käsittelee JSON-muotoista dataa.

ReadApi on rajapinnan komponentti, joka käsittelee rajapintaan tulevat tietojen lukupyynnöt. Rajapintapyynnön URI:ssä annetaan tietojen lukemiseen viittaava

toiminnallisuus *read* ja haluttu resurssi muodossa /read/{resurssi}. Tietojen lukemiseen viittaava toiminnallisuus *read* tehtiin POST-tyyppisen API-rakenteen takia, jotta pyynnöstä saadaan selville haluttu toiminnallisuus. URI:ssa annettava *read* toiminnallisuus ohjaa pyynnön ReadApille, joka käsittelee pyynnössä annetun resurssin, mahdollisen ID:n ja parametrit.

Kaikki pyynnössä annetut tiedot validoidaan rajapinnassa eli annettujen tietojen oikeellisuus tarkistetaan väärinkäytön estämiseksi. Tällainen väärinkäyttö voi olla esimerkiksi SQL-injektio, jossa hyökkääjä hyödyntää järjestelmän heikkoutta murtautuakseen järjestelmän sisälle käyttäen SQL-kyselyitä.

Rajapinnasta voidaan myös hakea tietty resurssi resurssin ID:llä, jolloin pyynnön URI on muotoa /read/{resurssi}/{id}. Tällöin rajapinnasta palautetaan vain kyseisen ID:n omaavan resurssin tiedot.

Resurssin eri kenttien hakua varten annetaan JSON-muotoisen pyynnön rungossa Array-tyyppinen fields-parametri, jossa haettavat kentät ilmoitetaan merkkijonoina Arrayn sisällä, kuten kuvassa 2. Fields-parametri oli alun perin vapaaehtoinen parametri, mutta se asetettiin toistaiseksi pakolliseksi rajapinnasta palautettavien tietojen rajaamiseksi. Tähän tullaan keksimään ratkaisu jatkokehityksessä.

```
{
    "fields": [
        "ID",
        "StartDatetime"
        ]
}
```

KUVA 2. Pyynnön rungossa annettavassa fields-parametrissä annetaan haettavat kentät

Rajapinnasta haettavia tietoja on mahdollisuus myös suodattaa esimerkiksi hakemalla tietyllä aikavälillä. Suodattaminen tapahtuu antamalla parametrinä filters, jossa Arrayn sisällä annetaan objektina haluttu kenttä ja sen arvo, kuten kuvassa 3. Vastauksena rajapinnasta palautuu kyseiseen hakuehtoon sopivat resurssit.

KUVA 3. Rajapinnasta haettavia tietoja voidaan suodattaa esimerkiksi tietyllä aikavälillä antamalla haluttu hakuehto filters-parametrissä

Asiakkaat voivat rajapinnan kautta hakea esimerkiksi viikonlopun aikana tehdyt työtunnit tai tällä hetkellä käynnissä olevat ajotapahtumat. Lain vaatimia henkilöstösuunnitelmia varten rajapinnasta voidaan hakea esimerkiksi henkilöstön sukupuoli, ikä ja työn muoto.

3.2 Autentikointi

Opinnäytetyössä autentikointitavaksi päätettiin yrityksessä API-avain ja se annetaan jokaisen rajapintapyynnön yhteydessä. API-avaimille tehtiin erillinen rajapinta-avainten näkymä, jossa voidaan lisätä ja tarkastella lisättyjä API-avaimia. API-avain lisätään rajapinta-avainten näkymässä lisäysnappia painamalla, jolloin API-avaimeksi generoituu ja tallentuu tietyn pituinen satunnainen ja ainutlaatuinen merkkijono. Avaimen generoimisen yhteydessä tarkistetaan, onko kyseinen merkkijono jo olemassa järjestelmässä. Jos merkkijono on jo olemassa, generoidaan uusi avain ja tarkistus tehdään uudestaan. API-avaimia ei voida jälkeenpäin muokata, jotta avain pysyy ainutlaatuisena.

Todennäköisyys generoida jo olemassa oleva avain on minimaalinen, mutta tarkistuksella varmistetaan avaimen ainutlaatuisuus. Avaimen ainutlaatuisuus varmistettiin vielä lisäämällä tietokantaan asetus UNIQUE, jolloin tietokantaan ei voida tallentaa kahta samaa arvoa. Avaimen ainutlaatuisuus on erityisen tärkeää, jotta eri asiakkaat eivät pääse vahingossakaan toisen asiakkaan tietoihin käsiksi.

Rajapinta-avaimia voi tarkastella vain asiakkaiden rajapinta-avaimiin oikeutetut käyttäjät. Rajapinta-avainten näkymässä API-avaimet näytetään listanäkymänä lisäämis- ja muokkausaikoineen.

API-avain annetaan jokaisen rajapintapyynnön yhteydessä antamalla avain pyynnön rungossa, kuten kuvassa 4. Kuvassa olevaa API-avainta on muunneltu tietoturvan vuoksi eikä se ole validi.

KUVA 4. Rajapintapyynnön yhteydessä annettava API-avain

3.3 Dokumentointi

Opinnäytetyön yhtenä tavoitteena oli luoda julkiselle rajapinnalle dokumentaatio, jotta rajapinnan käyttäjä osaa käyttää rajapintaa ilman erillistä apua. Dokumentaatiosta tuli ilmetä rajapinnassa käytettävät parametrit, resurssien kenttien kuvaus sekä palautettavan tiedon muoto formaatteineen.

Alun perin tavoitteena oli tehdä dokumentaatio aikatapahtumille, mutta se jouduttiin muuttamaan opinnäytetyön keskivaiheilla henkilöstön ja ajoneuvojen dokumentoimiseen. Dokumentoinnin järjestykseen vaikuttivat asiakkaiden tarpeet. Dokumentaatio luotiin Henkilöstö- ja Ajoneuvo-resursseille ja niiden luku-, lisäys- ja muokkaustoiminnallisuuksille. Henkilöstö-resurssi sisältää organisaation henkilöstön tiedot, kuten henkilön etu- ja sukunimen, sukupuolen, syntymäajan, henkilönumeron, työsuhteen alkamis- ja loppumisajankohdan sekä työsuhteen tyypin. Ajoneuvo-resurssi sisältää organisaation ajoneuvojen tiedot, kuten ajoneuvon nimen, rekisterinumeron, mallin, hinnan sekä ostopäivän ja -paikan. Lisäys- ja muokkaustoiminnallisuudet kuuluvat rajapinnassa SaveApi-komponentin alle, joka käsittelee rajapintaan tulevat tietojen lisäys- ja muokkauspyynnöt.

Dokumentoinnin määrittelyyn valikoitui OpenAPI-standardi ja dokumentaation esittämiseen ReDoc-työkalu. OpenAPI-standardi valikoitui vahvan dokumentaation ja laajan tuen perusteella. ReDoc valikoitui ilmaisuuden, muokattavuuden ja

selkeän ulkoasun takia. Dokumentaation luomista kyseisellä työkalulla oli yrityksessä testattu jo aikaisemmin, mutta testitiedostoa laajempaan käyttöön kyseinen dokumentaatio ei ollut vielä päässyt.

3.3.1 Määrittelytiedostojen luominen

OpenAPI 3.0 -määrittely kirjoitettiin käsin, sillä opinnäytetyötä tehdessä ei löytynyt sopivaa ilmaista dokumentaatiotyökalua, joka tukisi PHP-kieltä.

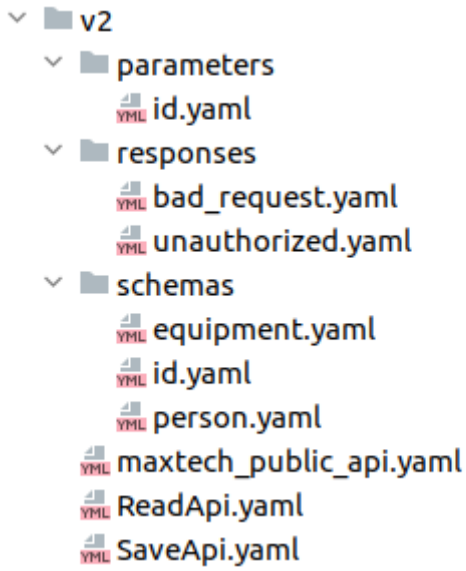
Ensimmäiseksi määrittelyn teossa luotiin OpenAPI-määrittelyn juuritiedosto (kuva 5). Yleensä juuritiedoston nimi on openapi.yaml. Tässä tapauksessa juuritiedoston nimeksi tuli maxtech_public_api.yaml, jotta sisäinen ja ulkoinen dokumentaatio saadaan eroteltua toisistaan. Juuritiedostossa määriteltiin juuritason elementit eli OpenAPI-määrittelyn versio, rajapinnan tiedot, rajapintapalvelimen URL, rajapinnan autentikaatiomenetelmät, resurssien polut sekä komponentit ja tagit.

```yaml
openapi: 3.0.3
info:
  title: Maxtech Public API
  version: '1.0.0'
servers:
  - url: '/api/v2.1'
security:
  - PublicApiKey: []

paths:
  /read/personnel/:
    $ref: './ReadApi.yaml#/personnel'
  /read/personnel/{id}:
    $ref: './ReadApi.yaml#/personnelById'
  /read/equipment/:
    $ref: './ReadApi.yaml#/equipment'
  /read/equipment/{id}:
    $ref: './ReadApi.yaml#/equipmentById'
  /add/equipment/:
    $ref: './SaveApi.yaml#/addEquipment'
  /edit/equipment/{id}:
    $ref: './SaveApi.yaml#/editEquipment'

components:
  securitySchemes:
    PublicApiKey:
      type: apiKey
      name: public_api_key
      description: >
        Api key must be given with every API request. Api key is given in the request body.

tags:
  - name: Person
  - name: Equipment
```

KUVA 5. OpenAPI-määrittelyn juuritiedoston rakenne

Määrittelyn purkaminen osiin

Rajapinnan OpenAPI-määrittely purettiin omiin YAML-tiedostoihin ja kansioihin ylläpidettävyyden ja selkeyttämisen takia (kuva 6). Kansiot on jaoteltu uudelleenkäytettävien osien eli komponenttien mukaan. OpenAPI-määrittelyssä toistuvat parametrit eroteltiin parameters-kansioon, yleiset vastaukset responses-kansioon ja resurssien skeemat schemas-kansioon. Jokainen parametri, vastaus ja resurssi on eroteltuna erillisiin YAML-tiedostoihin. Osiin purkamisessa tuli ottaa OpenAPI-määrittelyn rajoitukset huomioon.



KUVA 6. OpenAPI-määrittelyn rakenne on purettuna kansioihin ja tiedostoihin, jolloin määrittelyssä toistuvat osat löytyvät yhdestä ja samasta paikasta

Pyyntöjen polut eroteltiin toiminnallisuuden mukaan ReadApi.yaml- ja SaveApi.yaml-tiedostojen alle. ReadApi.yaml viittaa opinnäytetyössä tehtyyn ReadApi-komponenttiin, joka sisältää lukutoiminnallisuuden. SaveApi.yaml viittaa rajapinnassa olevaan SaveApi-komponenttiin, joka sisältää lisäys- ja muokkaustoiminnallisuudet. Lisäys- ja muokkaustoiminnallisuudet laitettiin OpenAPI-määrittelyssä samaan tiedostoon niiden samankaltaisen vastausrakenteen takia.

Polkujen runko

ReadApi.yaml-tiedostossa on määriteltynä tietojen lukemispyyntöjen polut. Esimerkkinä on kuva 7, jossa on määriteltynä henkilöiden haun polku /read/personnel/. Määrittelystä voidaan nähdä, että henkilöiden hakuun tehtävä pyyntö on POST-tyyppinen ja sillä voidaan hakea henkilöstön tiedot. Pyyntö sisältää viestin rungon sekä palautuvat vastausvaihtoehdot.

```yaml
# Read Api

# Person
personnel:
  post:
    summary: Get persons
    tags:
      - Person
    requestBody:
      $ref: '#/components/requestBodies/person_request_body'
    responses:
      '200':
        description: Success
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/person_response'
      '400':
        $ref: './responses/bad_request.yaml'
      '401':
        $ref: './responses/unauthorized.yaml'
```

KUVA 7. Henkilöiden haku OpenAPI-määrittelyssä

Pyynnön runko on jokaisessa tietojen lukemispyynnössä sama lukuun ottamatta fields-parametrissä annettavia kenttiä eli skeemoja, joiden takia pyynnön runko piti tehdä jokaiselle resurssille erikseen (kuva 8). Sama tilanne on palautuvassa vastauksessa. Ideaalitilanne olisi ollut yksi rungon määritelmä, jossa fields-parametrin skeema määriytyisi polun mukaan discriminator-objektin avulla. Discriminator-objekti on erottelija, jolla ilmoitetaan vaihtoehdoista tietyn arvon mukainen skeema. Tällaiseen polun mukaan erotteluun OpenAPI-määrittelyssä ei kuitenkaan ollut vielä tukea. Rajapinnassa oli tuki muillekin parametreille kuin fields-parametrille, mutta niitä ei ehditty opinnäytetyön ajan puitteissa lisäämään OpenAPI-määrittelyyn.

KUVA 8. Fields-parametrin runko määriteltynä joka resurssille erikseen

Resurssien skeematiedostot

Koska resurssien skeemat toistuivat määritelmässä monta kertaa, eroteltiin skeemat omiin YAML-tiedostoihin. Näissä YAML-tiedostoissa on määriteltynä kaikki kyseisen resurssin kentät. Tällä tavoin kaikki kentät löytyvät yhdestä paikasta ja kenttiin on helpompi tehdä muutoksia.

Kuvassa 9 on omaan YAML-tiedostoon määritelty Henkilöstö-resurssin kentät. Tiedostossa ensimmäisenä näkyy person_items -niminen Array, joka sisältää Henkilöstö-resurssin kentät string-muotoisina. Tätä käytetään määrittelyssä pyynnön rungossa silloin, jos pyynnössä syötetään vain kentän nimi eli haetaan tietoa. Seuraavana olevaa person_fields-nimistä Objektia käytetään vastausten

määritelmissä ja pyynnön rungossa silloin, kun lisätään tai muokataan tietoja. Rajapintaan tulevissa lisäys- ja muokkauspyynnöissä kentän nimen lisäksi kentälle annetaan arvo eli tällöin tieto annetaan objektina.

```yaml
# Person

# Fields in array of strings
person_items:
  type: array
  items:
    type: string
    enum:
      - ID
      - FirstName
      - LastName
      - EmployeeNumber
      - EmploymentStart
      - EmploymentFinished
      - EmploymentRelationshipType
      - Sex
      - DateOfBirth
  example:
    - ID
    - FirstName
    - LastName
    - EmployeeNumber
    - EmploymentStart
    - EmploymentFinished
    - EmploymentRelationshipType
    - Sex
    - DateOfBirth

# All person fields
# (used when an individual field does not need to be defined as required)
person_fields:
  type: object
  allOf:
    - $ref: './person.yaml#/first_name'
    - $ref: './person.yaml#/last_name'
    - $ref: './person.yaml#/employee_number'
    - $ref: './person.yaml#/employment_start'
    - $ref: './person.yaml#/employment_finished'
    - $ref: './person.yaml#/employment_relationship_type'
    - $ref: './person.yaml#/sex'
    - $ref: './person.yaml#/date_of_birth'
```

KUVA 9. Henkilöstö-resurssin kentät määriteltynä omassa YAML-tiedostossa

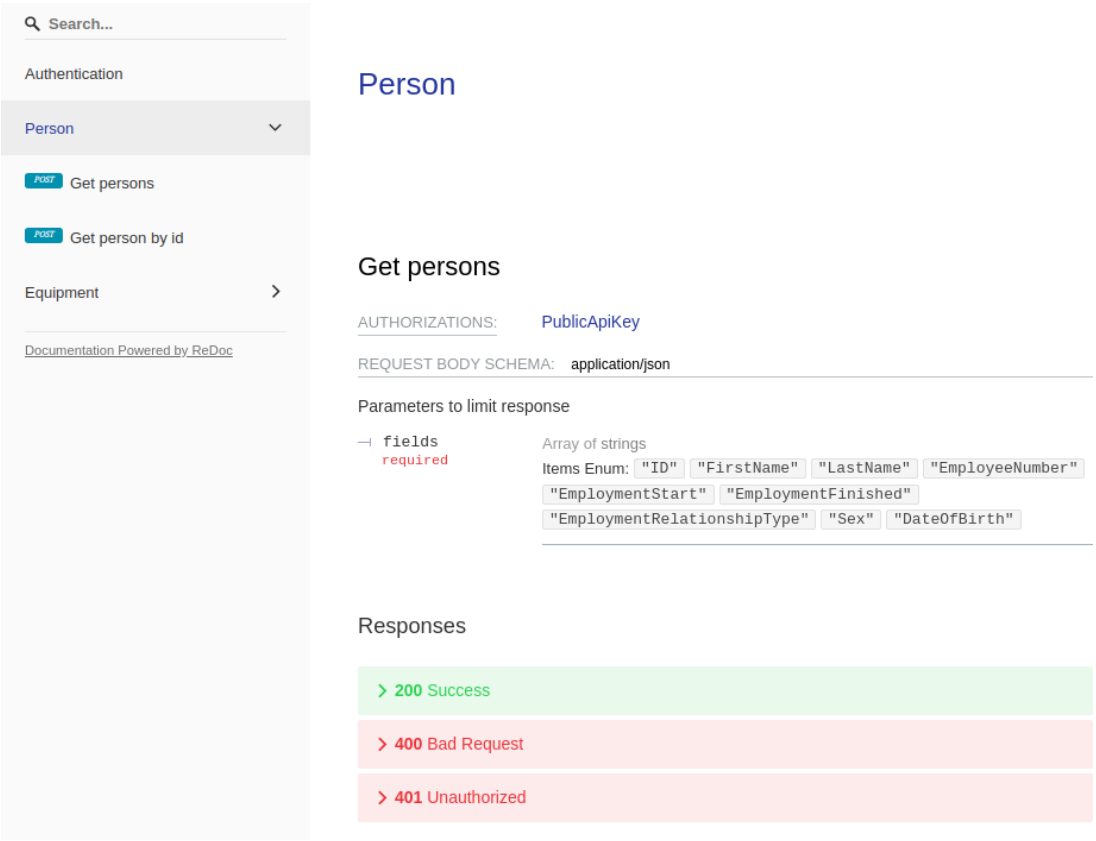
Kuvassa 9 kaikki Henkilöstö-resurssin kentät on listattu samaan person_fields-objektiin. Kun kaikki kentät on listattu samaan objektiin, voidaan määrittelyssä viitata pelkästään person_fields-objektiin ilman, että jokainen kenttä tulisi määrittelyssä kirjoittaa erikseen. Person_fields-objektissa jokaiseen kenttään viitataan $ref-viittauksella. Jokainen viitattu resurssin kenttä on tiedoston lopussa määriteltynä erikseen nimen mukaan, kuten kuvassa 10. Tällaisen toteutuksen taustalla oli tarve asettaa yksittäinen kenttä pakolliseksi. Määrittelyssä resurssin kentät piti luetteloida viittaamalla erikseen jokaiseen resurssiin, mutta tällä ratkaisulla riittää pelkkä viittaus person_fields-objektiin sekä pakolliseksi asetettavaan kenttään. Pakollisuus voidaan näin ylikirjoittaa eikä resurssin kenttiä tarvitse joka kerta erikseen luetteloida.

```
43  # All Person fields separately
44  first_name:
45    type: object
46    properties:
47      FirstName:
48        type: string
49        example: 'Elli'
50
51  last_name:
52    type: object
53    properties:
54      LastName:
55        type: string
56        example: 'Esimerkki'
```

KUVA 10. Henkilöstö-resurssin kentät määriteltynä erikseen kentän nimen mukaan

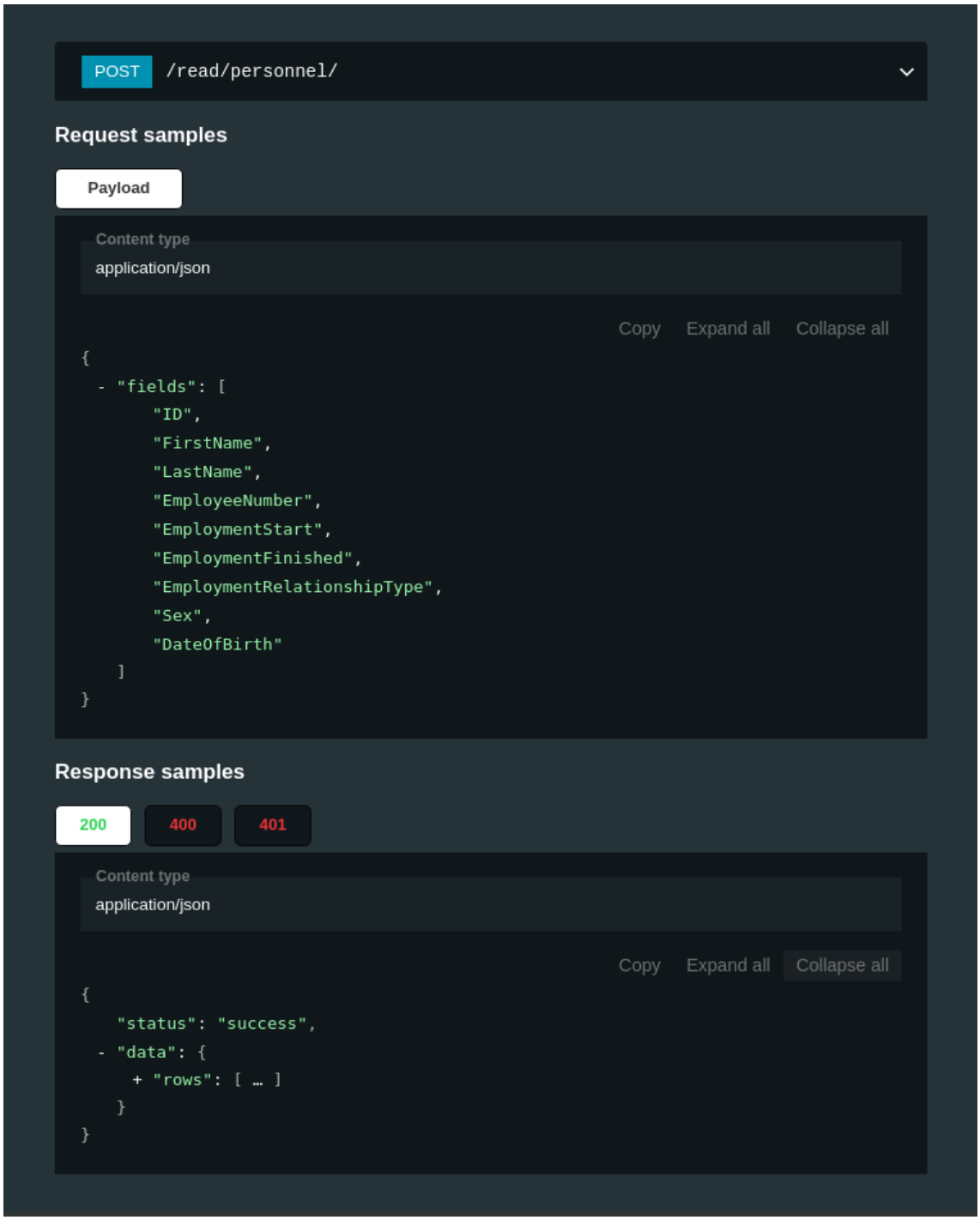
3.3.2 Dokumentaation käyttöliittymä

Dokumentaationäkymä tehtiin avautumaan uuteen ikkunaan, jolloin sen tarkasteleminen ja käyttäminen on helpompaa ja selkeämpää. Dokumentaationäkymässä resurssien nimet ovat pääotsikkona ja pääotsikoiden alla näytetään resurssien eri toiminnot (kuva 11). Toimintoja painamalla päästään tarkastelemaan toimintoon tehtävän pyynnön sekä rajapinnasta tulevan vastauksen muodot.



KUVA 11. Resurssien nimet ja toiminnot dokumentaation näkymässä

Rajapinnan pyynnöistä ja vastauksista näytetään myös esimerkit (kuva 12). Esimerkeistä rajapinnan käyttäjä näkee, miten rajapintaan tehdään pyyntöjä ja mitä rajapinnasta pitäisi palautua vastauksena. Rajapinnan käyttäjä voi kopioida esimerkit ja käyttää niitä omissa pyynnöissään.



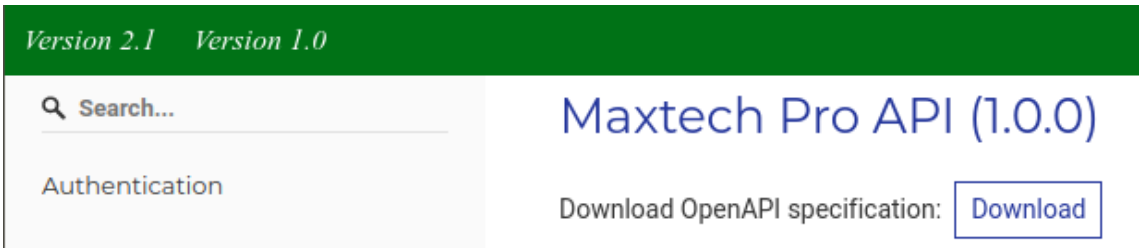
KUVA 12. Esimerkit pyynnöstä ja vastauksista dokumentaationäkymässä

Käyttöliittymän tyyliä olisi ollut mahdollista muuttaa ylikirjoittamalla ReDocin teeman kuvan 13 osoittamalla tavalla. ReDocin teeman ylikirjoittamisen kautta pystyy muuttamaan muun muassa käyttöliittymän värejä ja fontteja.

```
Redoc.init(api.url, {
 hideDownloadButton: true,
 theme: { colors: { primary: { main: \'#53b75a\' } } }
});
```

KUVA 13. Käyttöliittymän tyyliä voidaan muuttaa ylikirjoittamalla koodissa ReDoc-teemaa

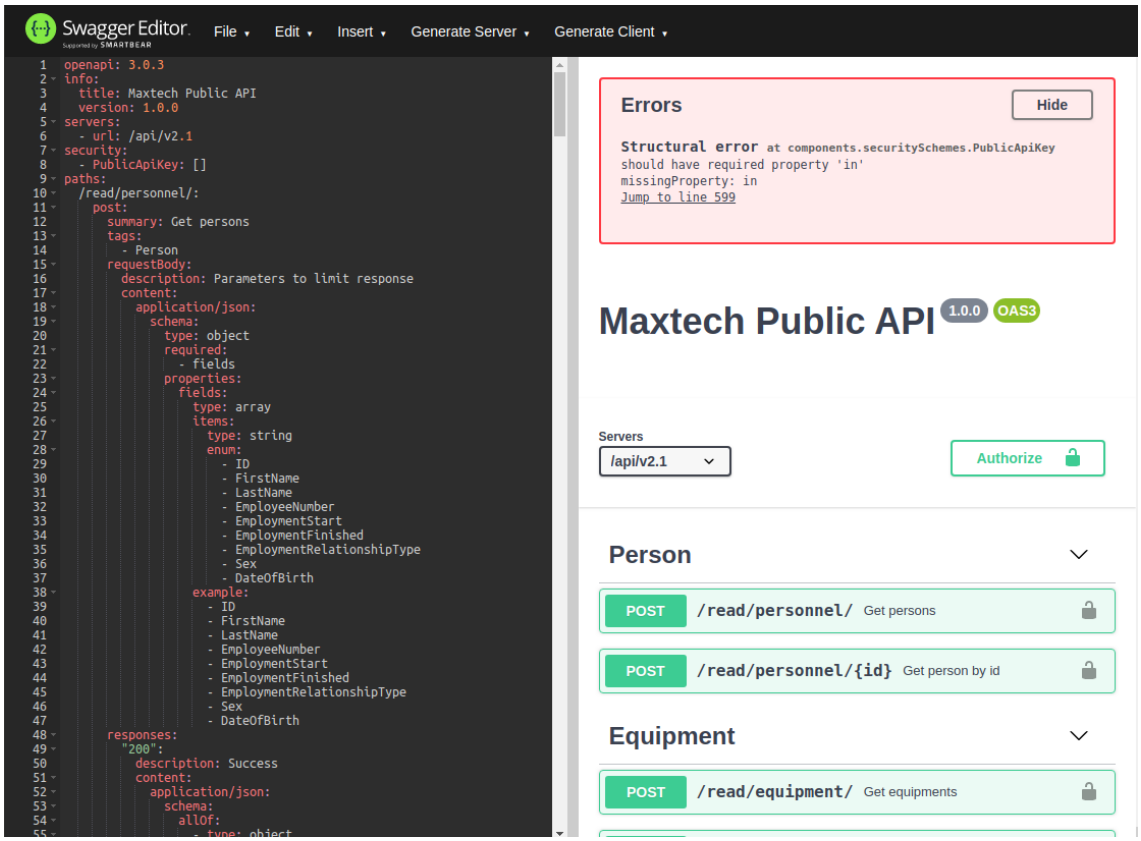
Työssä haluttiin myös selvittää versioinnin näkökulmasta, onko käyttöliittymään mahdollista lisätä eri rajapintaversioiden OpenAPI-määrittelyitä. Kuvassa 14 käyttöliittymään kokeiltiin lisätä versiopainikkeita, joiden kautta voidaan tarkastella mahdollisten vanhempien ja tulevien rajapintaversioiden OpenAPI-määrittelyitä. Kokeilussa ReDoc mahdollisti näkymän ulkopuolelle lisättävän palkin painikkeineen, joista saadaan painettua eri rajapintaversioiden OpenAPI-määritelmä auki. Versiopainikkeita ei tässä tapauksessa vielä tarvittu, joten ne jätettiin vielä dokumentaationäkymästä pois.



KUVA 14. ReDoc mahdollistaa versiopainikkeiden lisäämisen ja siten eri rajapintaversioiden OpenAPI-määrittelyn avaamisen

3.3.3 OpenAPI-määrittelyn oikeellisuuden testaaminen

OpenAPI-tiedoston oikeellisuus testattiin ilmaisella Swaggerin editorilla (kuva 15). Swaggerin editoriin vaadittiin yksittäinen YAML-tiedosto, josta löytyy kaikki OpenAPI-määrittelyn osat.



KUVA 15. OpenAPI-tiedoston oikeellisuuden testaaminen Swaggerin editorilla

Editori ei siis osaa käsitellä eri tiedostoihin purettua määrittelyä, vaan määrittelyn tulee olla kokonaisuudessaan yhdessä tiedostossa. Tiedostojen yhteen liittämisessä käytettiin apuna speccy-työkalua, joka selvittää eri tiedostoissa olevan OpenAPI-määrittelyn yhteen tiedostoon kuvassa 16 annetulla komennolla. Komennossa annetaan YAML-tiedosto, joka sisältää OpenAPI-määrittelyn version (testi_api.yaml) ja ulostulevan tiedoston nimi (test.yaml).

```
$ speccy resolve testi_api.yaml -o test.yaml
```

KUVA 16. Komento, jolla liitetään OpenAPI-määrittelyn tiedostot yhteen

Editori antaa virheilmoituksen, jos OpenAPI-määrittelyssä on virhe, kuten kuvassa 15 näkyy. Tämä virheilmoitus jätettiin huomioimatta, sillä OpenAPI-määrittelyn ‘in’-ominaisuuden vaihtoehdoissa, ”query”, ”header tai ”cookie”, ei ole sopivaa arvoa tilanteeseen, jossa API-avain annetaan viestin rungossa.

3.3.4 Kenttien selitykset

Rajapinnan dokumentaatioon oli tarvetta saada myös jokaisen resurssin kentille selitykset eli ohjeet sekä suomeksi että englanniksi, jotta rajapinnan käyttäjä ymmärtäisi paremmin mihin kenttiä käytetään. Sopivin paikka ohjeiden sijainnille olisi resurssin otsikon eli tagin alla, jonka jälkeen tulisivat loogisesti resurssin toiminnot kenttineen.

Ohjeiden tekemiseen testattiin kahta eri työkalua: Swagger-PHP:tä ja PHP-OpenAPI:a, joista Swagger-PHP ei toiminut eikä sopinut kyseisen ongelman ratkaisemiseen. Testisana saatiin järjestelmässä olevaa käännösfunktiota ja PHP-OpenAPI-työkalua käyttäen suomennettuna OpenAPI-määrittelyn joukkoon. Ongelmana PHP-OpenAPI-työkalussa oli se, että työkalun ohjeet olivat epäselvät ja olemattomat. Toiminnallisuus selvisi käymällä läpi lähdekoodia ja kokeilemalla kuten kuvassa 17. Määritelmän viittaus $ref hakee onnistuneesti aikaisemmin tehtyjen polkujen määritelmät dokumentaatioon esille. Työkalu oli kuitenkin työläs ja huonosti dokumentoitu, joten se jätettiin toistaiseksi dokumentaatiosta pois.

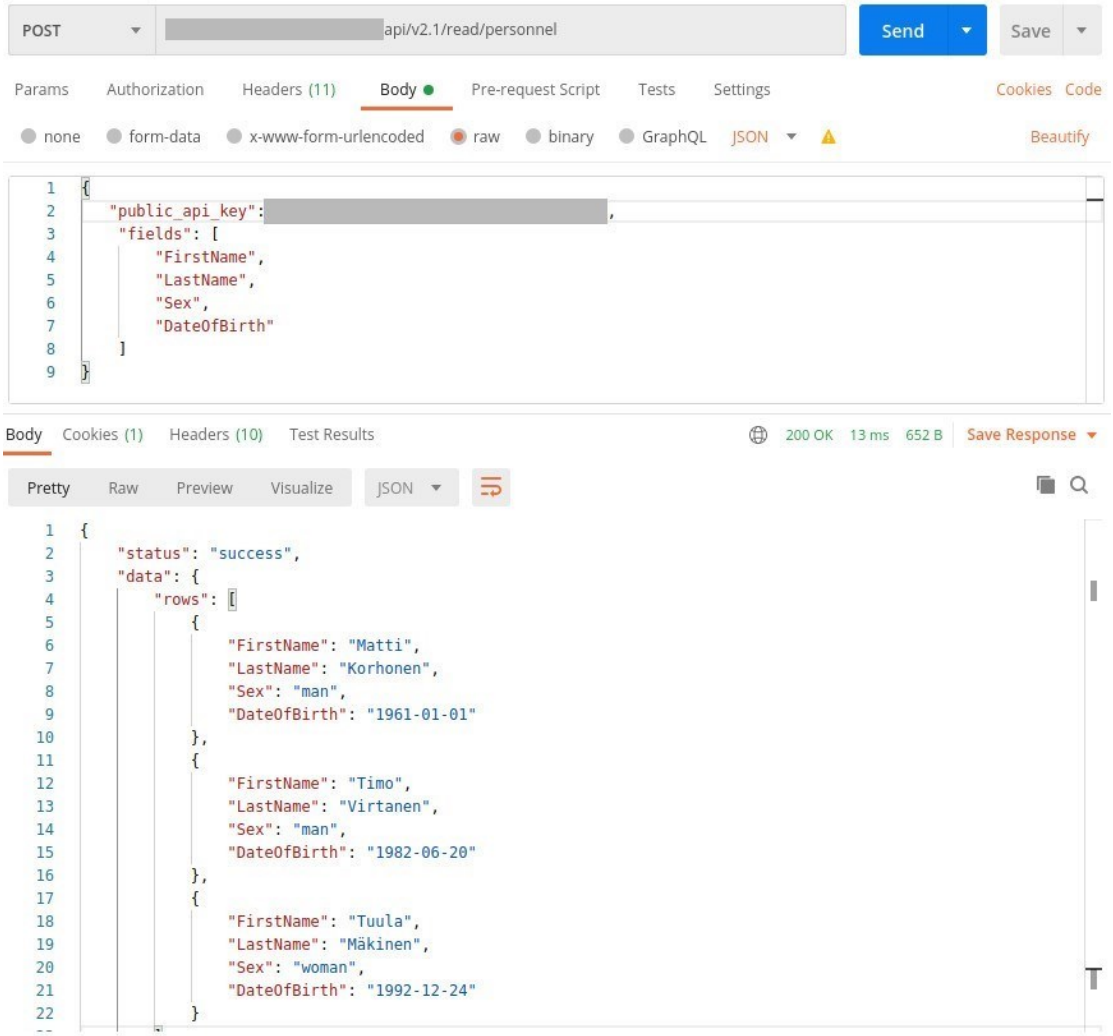
```
$openapi = new OpenApi( [
    'openapi' => '3.0.3',
    'info' => new Info( [
        'title' => 'Test API',
        'version' => '1.0.0'
    ] ),
    'paths' => [
        '/read/personnel' => new PathItem( [
            '$ref' => './ReadApi.yaml#/personnel'
        ] ),
        '/read/personnel/{id}' => new PathItem( [
            '$ref' => './ReadApi.yaml#/personnelById'
        ] )
    ],
    'tags' => [
        new Tag( [
            'name' => 'Person',
            'description' => TranslateUtil::translate( native: 'Personnel information' )
        ] ),
        new Tag( [
            'name' => 'Equipment'
        ] )
    ]
] );
```

KUVA 17. Resurssien ohjeiden testaamista PHP-OpenAPI-työkalulla

3.4 Rajapinnan testaaminen

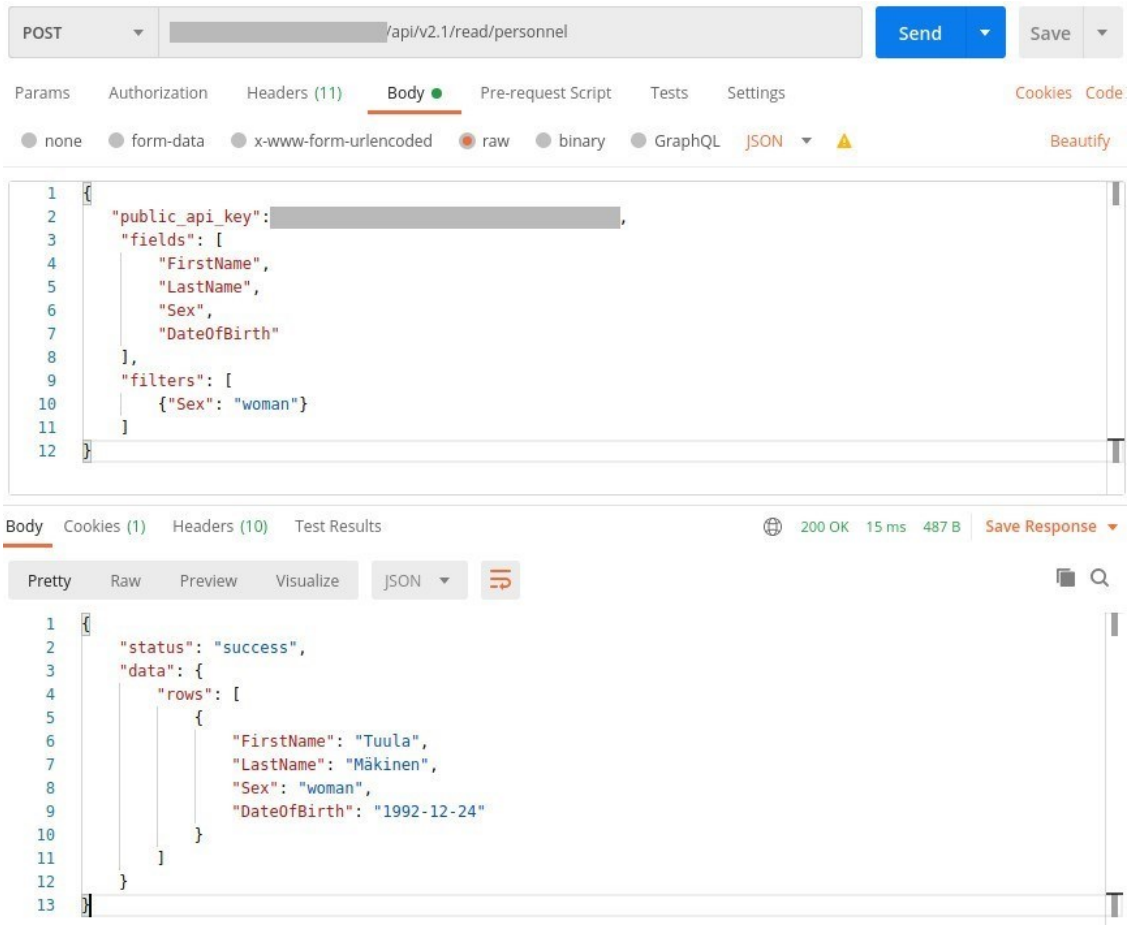
Koska automatisoitua testaamista ei tässä opinnäytetyössä ehditty tekemään, testattiin rajapinnan toiminnallisuutta dokumentaation edetessä manuaalisesti. Dokumentaatiota tehdessä jokainen kenttä testattiin ja tietojen oikeellisuus tarkistettiin järjestelmän tietokannasta. Testaamisessa käytettiin Postman-työkalua, jolla voidaan testata rajapinnan toimintaa tekemällä rajapintapyyntöjä erilaisilla arvoilla.

Rajapinnan toiminnallisuuksia testattiin eri tavoin ja tässä esitellään muutama keksitty asiakastapaus, joissa testiympäristöstä haetaan kuvitteellisia resursseja. Ensimmäisessä tapauksessa (kuva 18) rajapinnasta haetaan lain vaatimia henkilöstösuunnitelmia varten organisaation henkilöstöön kuuluvien henkilöiden etu- ja sukunimi, sukupuoli ja syntymäaika. Henkilöstösuunnitelmaa varten haettavia tietoja voivat olla toteutuneet määräaikaisuudet sekä ikä- ja sukupuolijakauma. Rajapinnan vastauksessa ”status” kertoo pyynnön onnistumisesta ja ”data” sisältää onnistuneessa pyynnössä yhden tai useamman palautetun arvon. Tässä tapauksessa rajapinnasta saatiin onnistuneesti haettua halutut Henkilöstö-resurssin kentät.



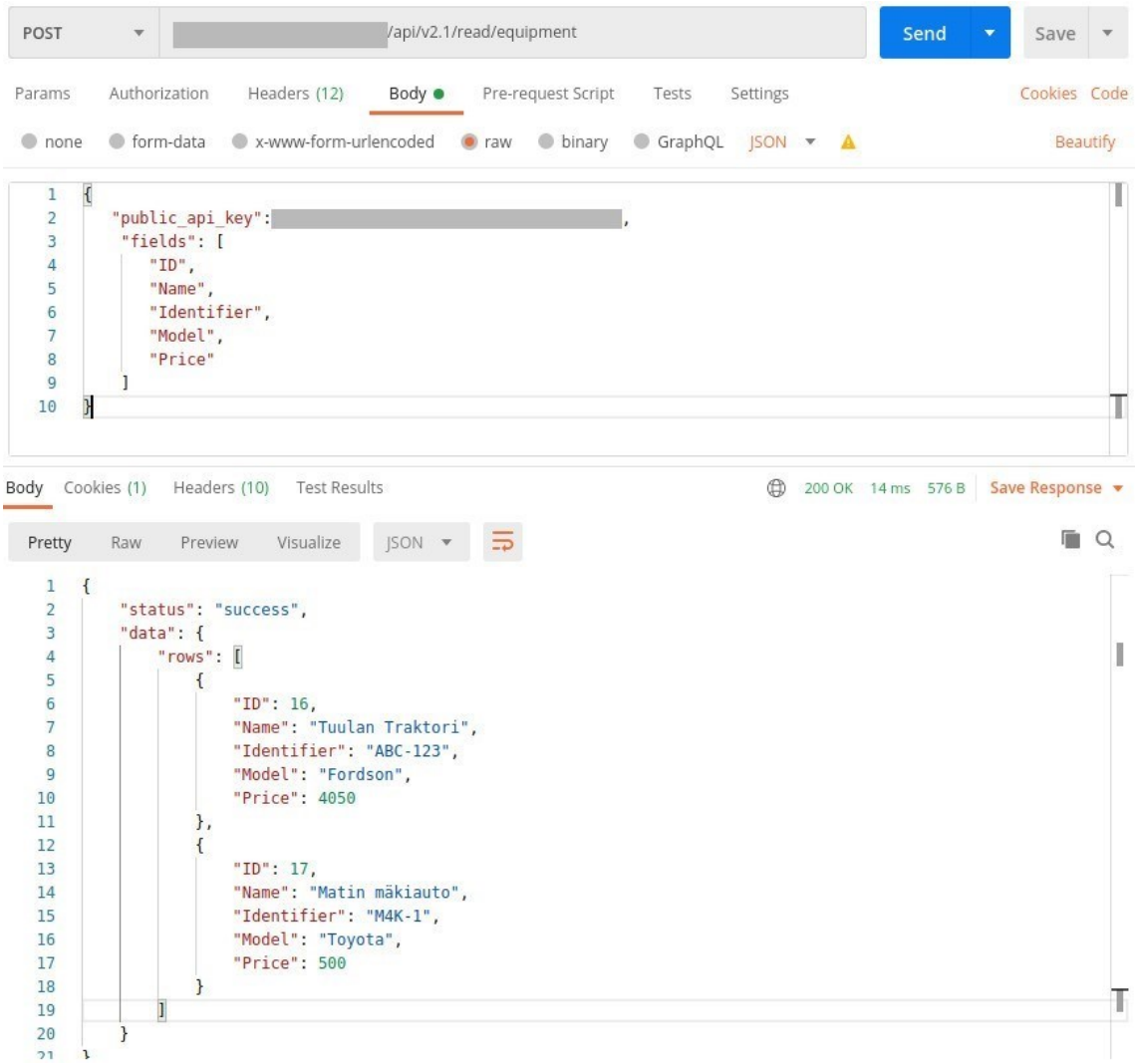
KUVA 18. Rajapinnasta palautuu onnistuneesti henkilöstön henkilöiden etu- ja sukunimi, sukupuoli ja syntymäaika

Rajapinnasta haettavaa henkilöstöä voidaan myös suodattaa esimerkiksi sukupuolen perusteella. Kuvassa 19 haetaan suodattamalla henkilöstöstä vain naispuoliset henkilöt.



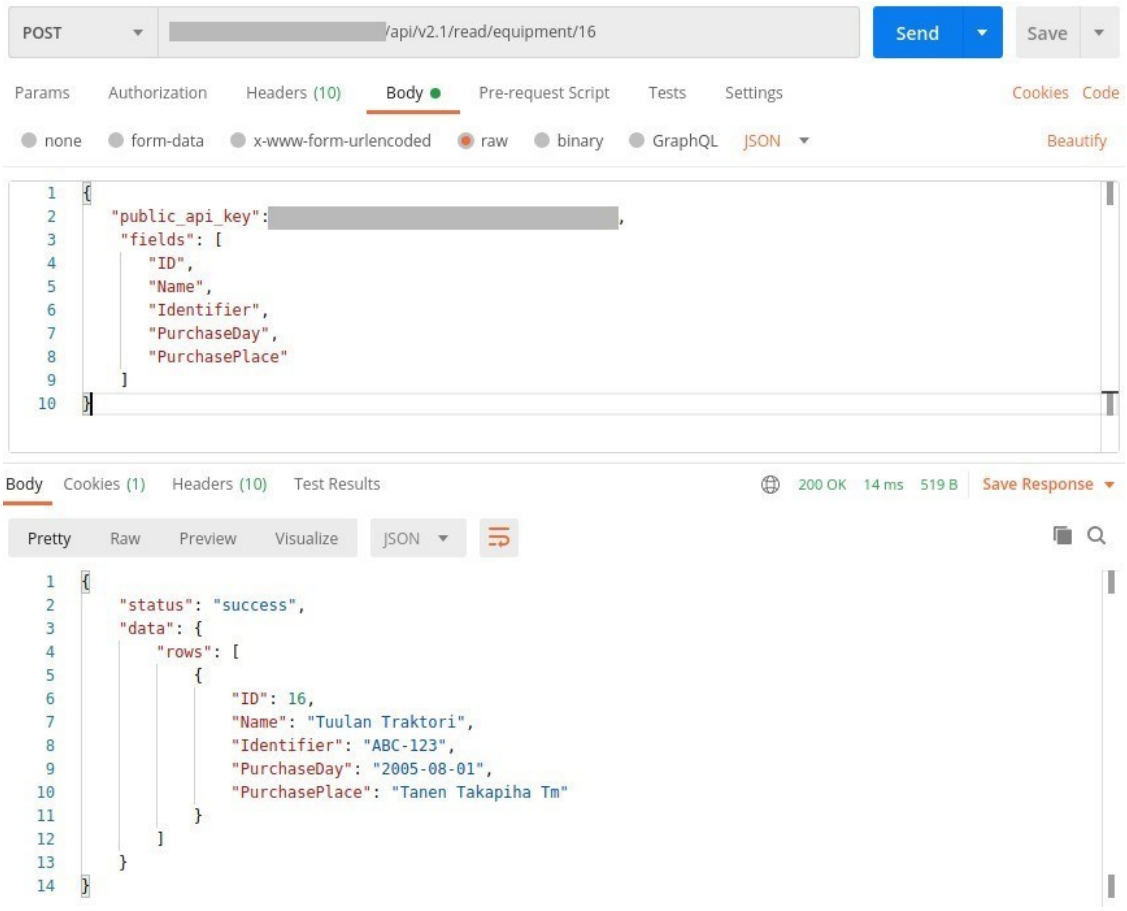
KUVA 19. Henkilöstöstä voidaan suodattamalla hakea esimerkiksi naispuoliset henkilöt

Toisessa tapauksessa haettiin järjestelmästä organisaation kaikkien ajoneuvojen ID, nimi, rekisterinumero, malli ja hinta (kuva 20). Ajoneuvoja organisaatiolla on kaksi.



KUVA 20. Organisaation kaikkien ajoneuvojen haku rajapinnasta

Resursseja voidaan hakea myös pelkästään ID:n perusteella. Kuvassa 21 haetaan tarkempia ajoneuvon ostotietoja ajoneuvosta, jonka ID on 16.



KUVA 21. Rajapinnasta haetaan tarkempia ostotietoja ajoneuvosta, jonka ID on 16

4 TULOKSET

Opinnäytetyön tehtävänä oli suunnitella ja osittain toteuttaa helposti laajennettavissa oleva Maxtech Pro -järjestelmän rajapinta ja rajapinnalle dokumentaatio. Rajapintaan tehtävästä dokumentista tuli ilmetä parametrit, kenttien kuvaukset ja palautettavan tiedon muoto formaatteineen. Rajapinta tuli myös versioida.

Työ aloitettiin hankkimalla tietoa REST-arkkitehtuurista ja rajapinnoista kokonaisuutena. Rajapintoihin perehtymisen ja suunnittelun jälkeen aloitettiin rajapinnan toteuttaminen. Tehdessä rajapintaan lukutoiminnallisuutta tehtiin samaan aikaan opinnäytetyön ulkopuolella lisäys- ja muokkaustoiminnallisuudet. Alun perin tarkoitus oli aloittaa lukutoiminnallisuuden tekemisellä, mutta asiakkaiden tarpeiden takia jouduttiin rajapintaa aloittamaan lisäys- ja muokkaustoiminnallisuuksilla. Lukutoiminnallisuutta tehdessä rajapintaa testattiin eri tiedoilla ja varmistettiin, että lukutoiminnallisuus palauttaa tiedot oikein. Koska yrityksellä oli jo olemassa olevaa rajapintarakennetta sisäisten rajapintojen myötä, julkisen rajapinnan versioinnissa käytetään versiota 2.1. Versiointi otettiin kuitenkin huomioon dokumentaatiossa, johon on mahdollista lisätä rajapinnan tulevat versiot poistamatta vanhoja.

Rajapintaan tuli myös toteuttaa organisaatiokohtainen autentikointi, jolla käyttäjä todentaa identiteettinsä rajapinnan käyttöä varten. Rajapinnasta haetut tiedot, joihin oikeudet riittävät, palautettaan asiakkaalle lukumuodossa.

Autentikoinnin teossa autentikointimenetelmä oli yrityksessä päätetty, joten vertailuun ja tekemiseen ei mennyt paljoa aikaa. Rajapinta-avaimessa tärkeintä oli avaimen ainutlaatuisuus, joka työssä varmistettiin moneen otteeseen. Rajapinta-avaimiin ei myöskään kuka tahansa saanut päästä käsiksi, joka varmistettiin erottamalla rajapinta-avaimet aktivoitavaksi osaksi.

Alun perin työhön valittiin dokumentoitavaksi objektiksi Tapahtuma-resurssi, mutta se täytyi työn edetessä vaihtaa Henkilöstö- ja Ajoneuvo-resursseihin Tapahtuma-resurssin monimutkaisuuden takia. OpenAPI-määrittelyä Tapahtuma-resurssille ehdittiin tehdä jo, joten OpenAPI-määrittelyn tekemiseen meni suun-

niteltua enemmän aikaa. Koska OpenAPI-määrittelyn runko oli jo Tapahtuma-resurssin myötä valmiiksi rakennettuna, oli resurssin vaihtaminen suhteellisen helppoa. Henkilöstö-resurssista jätettiin toistaiseksi lisäys- ja muokkaustoiminnallisuudet pois, sillä ajateltiin, että henkilöstön tietojen muokkaaminen rajapinnan kautta ei välttämättä tässä vaiheessa ole vielä järkevää. Jotta työssä tehtävään dokumentaatioon saataisiin lisättyä OpenAPI-määrittely myös lisäys- ja muokkaustoiminnallisuuksille, otettiin toiseksi resurssiksi Ajoneuvo-resurssi. Dokumentointiin valittujen resurssien järjestyksessä huomioitiin asiakkaiden tarpeet.

Dokumentaation käyttöliittymän monipuolisuuteen, selkeyteen ja yleiseen ilmeeseen oltiin tyytyväisiä. Dokumentaation käyttöliittymän väritys oli tarkoitus vaihtaa yrityksen värimaailman mukaiseksi, mutta dokumentaatiossa käytetty ReDoc ylikirjoitti käyttöliittymässä aiemmin lisätyt yrityksen omat CSS-tyylit, joten tyylin muokkaamiseksi olisi pitänyt ylikirjoittaa ReDocin teemaa. Tyylit päätettiin pitää toistaiseksi alkuperäisinä, jotta vältyttäisiin ylimääräisen tyylin ylläpitämiseltä.

Rajapintaan oli tarkoitus toteuttaa myös automatisoitu testausmenetelmä, jolla testattaisiin rajapinnan toiminnallisuus säännöllisin väliajoin. Automatisoidulla testauksella varmistettaisiin, että rajapinta toimii niin kuin sen haluttiin toimivan ja huomattaisiin odottamattomat muutokset. Koska dokumentaation ja siihen kuuluvan OpenAPI-määrittelyn tekemiseen meni suunniteltua enemmän aikaa, ei automatisoidun testauksen tekemistä ehditty työssä aloittamaan.

5 JATKOKEHITYS

Dokumentaatiota aiotaan jatkossa laajentaa resursseiltaan kattavammaksi. Jatkokehityksessä otetaan huomioon asiakkaiden tarpeet ja käytössä olevat resurssit. Jatkokehityksessä tärkeää on ottaa huomioon rajapinnan ja dokumentaation ylläpidettävyys jatkossa.

Rajapinnassa oli jo olemassa useampi tietojen suodatus- ja rajausmahdollisuus. Tässä opinnäytetyössä otettiin huomioon vain kenttien rajaus eli fields-parametri, joka aiemmin mainittuna luvussa 3.2 muutettiin pakolliseksi, sekä filters-parametri, jolle pitäisi jatkokehityksessä tehdä dokumentaatio. Rajapintaa olisi hyvä laajentaa offset-parametrilla, jolla voidaan rajata rajapinnasta palautuvien tulosten määrää. Offset-parametri määrittää, mistä arvosta alkaen tuloksia palautetaan. Offset-parametri toimii parhaiten limit-parametrin kanssa, joka määrittää palautettavien tulosten määrän.

Fields-parametrin pakollisuus tulisi myös korvata niin, että rajapinnasta saataisiin haettua kaikki kentät ilman fields-parametriä. Tätä varten tarvittaisiin rajapinnan puolelle määrittelyt, mitä resursseista palautetaan.

Rajapinta-avaimille tulisi tehdä myös käyttöoikeusrajoitukset rajapintaan. Rajapinta-avaimille siis määriteltäisiin, mihin toiminnallisuuksiin (luku, kirjoitus ja muokkaus) ja resursseihin avaimella on oikeus.

Rajapinnan autentikointiin voisi myös pohtia OAuth 2.0:n lisäämistä, jos sille nähdään tarvetta. OAuth 2.0:n käyttöönotto vaikutti haastavalta, joten siihen täytyisi paremmin perehtyä. OAuth 2.0 on kuitenkin laajasti käytössä oleva autentikointimenetelmä, joka toisi rajapinnalle lisäarvoa.

Dokumentaatioon toivottiin mahdollisuutta generoida OpenAPI-määrittely automaattisesti eli resurssien kentät, polut ja vastaukset generoituisivat automaattisesti OpenAPI-määritelmäksi PHP-kielellä tehdyn rajapinnan koodeissa. Tällaiseen automaattiseen generoimiseen voisi jossain määrin soveltua opinnäyte-

työssä testattu PHP-OpenAPI-työkalu. Automaattinen OpenAPI-määrittelyn generoiminen voisi säästää yrityksen ja ohjelmistosuunnittelijoiden aikaa, mutta voi aiheuttaa virheitä ilman huolellista tarkistamista ja testaamista.

Rajapintaan ja dokumentaatioon voisi tehdä myös toiminnallisuuden, jossa organisaatiokohtaisesti näytetään vain organisaatiolle oikeutetut ja tarpeelliset resurssit. Organisaation kannalta turha tieto karsiintuisi ja rajapinnan käyttö nopeutuisi huomattavasti, kun dokumentaatiossa näkyisi vain organisaatiolle tarpeelliset resurssit. Tämä tosin voisi vaatia yritykseltä liikaa resursseja ja aikaa, kun jokaisen organisaation tarpeet tulisi erikseen selvittää.

Rajapintaan oli tarkoitus toteuttaa myös automaattinen testausmenetelmä, jolla olisi säännöllisesti testattu rajapinnan toiminnallisuutta. Testausmenetelmän toteutus tehdään opinnäytetyön ulkopuolella jatkokehityksessä. Rajapinnan testaamisen lisäksi voitaisiin automatisoida dokumentaation oikeellisuuden tarkistus, jossa ilmoitettaisiin OpenAPI:n ja palautettavien vastausten välillä olevat erot.

Rajapintaan tulisi tehdä lisäksi lokitus eli lokitietojen tallennus, jossa jokainen rajapintaan tehty pyyntö tallennetaan. Lokituksella väärinkäytöt ja asiakkailla ilmenneet virheet saataisiin tallennettua tarkastelua ja selvittelyä varten. Lokitietojen avulla voidaan tarkastella virheen ajankohtaa, tapahtumaa ja tekijää. Esimerkiksi käyttötuen työntekijät voivat käyttää lokitietoja hyödyksi selvitellessä asiakkailla tapahtuneita virheitä. Lokitusta voidaan käyttää myös rajapinnan käytön hinnoittelussa hyväksi. Hinnoittelun perusteena voitaisiin käyttää rajapintaan tehtyjen pyyntöjen määrää ja ilmaiseksi voitaisiin tarjota tietty määrä pyyntöjä.

Rajapintaa varten olisi hyvä tehdä myös tarkistuslista, mistä voidaan tarkistaa tarvittavat toimenpiteet, jos rajapintaan tehdään muutoksia. Tarkistuslistasta voidaan tarkistaa, mitä esimerkiksi rajapinnassa, dokumentaatiossa ja testauksessa tulee ottaa huomioon ja minne kaikkialle tulee tehdä muutoksia.

Dokumentaation käyttöliittymää voitaisiin muokata myöhemmin yrityksen värimaailman mukaiseksi. Fontit ja väritys muutettaisiin yrityksen tyylin mukaiseksi, kun löydetään järkevä ratkaisu muokata tyyliä ilman ylimääräistä ylläpitoa.

Tehtyä rajapintaa hyödyntämällä oli tarkoitus kehittää asiakkaiden käyttöön raportointityökalu, jota kautta asiakkaat saisivat muodostettua halutunlaisia raportteja valitsemillaan kentillä ja hakuehdoilla. Tämä raportointityökalu siirtyi jatkokehitykseen.

6 POHDINTA

Idea kehittää avoin Maxtech Public API -rajapinta opinnäytetyönä tuli toimeksiantajalta. Aihe kuulosti mielenkiintoiselta ja haastavalta opinnäytetyöaiheelta, jolla voisi syventää jo olemassa olevaa tietoa ja jolla olisi yrityksen kannalta suurta kehityksellistä merkitystä. Aiempaa kokemusta rajapinnoista oli mobiilikehityksen puolelta, jossa yrityksen sisäiseen palvelinrajapintaan lisättiin mobiilisovelluksen tarvitsemia ominaisuuksia. Itsessään tieto rajapinnan syvemmästä toimivuudesta oli jäänyt vähäiseksi, joten opinnäytetyöaihe antoi oivan mahdollisuuden oppia rajapinnoista enemmän ja syvällisemmin. Vaikka rajapinnoista oli jo aiempaa kokemusta, opittiin opinnäytetyön aikana erittäin paljon uutta ja vahvistettiin vanhaa tietoa.

REST terminä oli ennestään vieras ja sen opiskeluun ja ymmärtämiseen meni paljon aikaa. REST on erittäin monitulkintainen aihe ja toissijaisissa lähteissä Roy Fieldingin väitöskirjassa esiteltyä REST-arkkitehtuuria oli tulkittu eri tavoin. Fieldingin väitöskirja oli vaikeaa luettavaa, joten väitöskirjan lisäksi tässä opinnäytetyössä käytettiin muita lähteitä kokonaisuuden hahmottamiseksi. Työssä haluttiin ottaa kantaa myös Richardsonin kypsyysmalliin, johon opinnäytetyötä tehdessä törmättiin monta kertaa. Kypsyysmalli on kuitenkin muokattu versio alkuperäisestä Richardsonin The Maturity Heuristic ja sen oikeellisuus alkuperäiseen lähteeseen on myös monitulkintainen. Opinnäytetyössä tehty rajapinta vaikuttaisi olevan pääosin RESTful, mutta edellä mainittujen monitulkintojen takia tässä opinnäytetyössä ei oteta rajapinnan RESTful-määrään sen tarkemmin kantaa.

Tietoa hankkiessa mietintää aiheuttivat myös rajapinnassa käytetyt POST-metodit. Voiko rajapinta olla REST, jos kaikki pyynnöt tehdään POST-metodilla? Kysymykseen ei löytynyt selkeää vastausta, mutta rajapinnan voidaan jossain määrin sanoa olevan RESTful, kunhan POST-metodien käyttö on dokumentoitu. Suotavaa olisi käyttää CRUD-metodeja, jotta rajapintaa voidaan täysin kutsua REST-rajapinnaksi. Tosin, kuten kappaleessa 2.1 mainitaan, mitään palkintoa ei saada noudattamalla orjallisesti kaikkia REST-arkkitehtuurin rajoituksia.

Opinnäytetyön aikana pyrittiin miettimään rajapintaa kokonaisuutena, johon myös kuului suunniteltu raportointityökalu. Raportointityökalu käyttäisi toteutettua rajapintaa ja antaisi asiakkaille uusia mahdollisuuksia hakea räätälöity raportti. Raportointityökalua varten asiakkailta kyseltiin toiveita ja tarpeita, joita hyödynnettiin opinnäytetyön teossa ja priorisoinnissa. Toteutettu rajapinta antaa asiakkaille myös muita mahdollisuuksia hyödyntää järjestelmän dataa.

Opinnäytetyön tekemisessä painopiste siirtyi dokumentaation puolelle, mihin panostettiin myös eniten aikaa. Rajapinnasta kiinnostuneita asiakkaita oli paljon ja dokumentaatio on rajapinnalla pakko olla, joten dokumentaation teko muuttui opinnäytetyön aikana prioriteettilistalla ykköseksi.

Dokumentaatiota tehdessä tuli miettiä myös loppukäyttäjää eli kehittäjää, joka dokumentaation perusteella käyttää rajapintaa. Dokumentaation tulisi olla tarpeeksi tarkka ja ajan tasalla eikä dokumentaatio saisi sisältää virheitä. Kirjoittaessa OpenAPI-määrittelyä käsin, opittiin dokumentaation kirjoittamisesta todennäköisesti enemmän kuin OpenAPI-määritelmiä automaattisesti tekeviä dokumentaatiosovelluksia käyttämällä. Käsin kirjoittamisessa on suurempi virheiden mahdollisuus, varsinkin jos dokumentoitavan tiedon määrä on suuri.

Suurimpaan osaan tavoitteista päästiin, vaikka dokumentaation tekemiseen menikin yllättävän paljon aikaa. Aikaa vievää dokumentaatiossa oli OpenAPI-määrittelyn tiedostorakenteen rakentaminen siten, että samat osat toistuvat mielellään vain kerran ja ylläpitäminen olisi mahdollisimman helppoa. Esimerkkejä OpenAPI-määrittelyn pilkkomisesta löytyi erittäin vähän ja pääosin rakenne syntyi itse kokeilemalla ja päättelemällä.

Vaikka rajapintaan jäi kokonaisuudessaan kehityskohteita, saatiin rajapintaan ja dokumentaatioon tehtyä hyvät pohjat jatkokehitystä varten. Ennen rajapinnan avaamista asiakkaiden käyttöön kehitetään rajapintaan ensin rajapinnan toiminnan kannalta tarpeelliset osat, kuten automatisoitu testaus ja lokitus. Jos rajapinta avataan asiakkaiden käyttöön liian aikaisin, isot rakenteelliset muutokset aiheuttavat ylimääräisiä rajapinnan versioimisia ja ylimääräistä työtä kehittäjille. Lisättyjä toiminnallisuuksia on työlästä ottaa pois, sillä käyttäjiä tulisi informoida

muutoksesta ja käyttäjille tulisi antaa tarpeeksi aikaa muuttaa omia järjestelmiään, joissa rajapintaa on käytetty. Rajapinnan käyttäjillä tuskin on tarpeeksi resursseja ja mielenkiintoa seurata rajapinnan dokumentaatiota päivittäin.

Kokonaisuutena olen tyytyväinen lopputulokseen. Rajapinnasta saatiin toimiva, vaikkakin fields-parametri jouduttiin muuttamaan pakolliseksi. Rajapinta-avaimet toimivat oikein sekä avaimen kaksoiskappaleen luomisen estämiseksi tehtiin mahdollisimman paljon varotoimenpiteitä. Rajapinta-avaimesta ja rajapinta-avainten näkymästä tehtiin selkeä kokonaisuus, johon voitaisiin vielä selkeyttämiseksi lisätä mahdollisuus asettaa avaimelle nimi. Dokumentaatiosta ilmenee, että rajapinnassa käytetään API-avainta ja pyynnöissä POST-metodeja. Resurssit ovat selkeästi eroteltuna ja esimerkkien avulla käyttäjän on helppo aloittaa rajapinnan käyttö.

OpenAPI-määrittelyn rakenteeseen olenkin erityisen tyytyväinen, sillä OpenAPI käsitteenä oli täysin uusi ja esimerkkejä pilkotusta OpenAPI-määrittelystä oli turhauttavan vähän. Kaikki mahdolliset tavat estää toistuvaa koodia tehtiin tämän opinnäytetyön aikana määrittelyn hyväksi. Harmi, että dokumentaation käyttöliittymän värimaailmaa ei saatu muutettua, mutta sillä ei ole käytön ja toimivuuden kannalta merkitystä.

LÄHTEET

1. Richardson, Leonard – Ruby, Sam 2007. RESTful Web Services. Sebastopol, CA: O'Reilly Media, Inc. Saatavissa: https://learning.oreilly.com/library/view/restful-web-services/9780596529260 (vaatii käyttöoikeuden). Hakupäivä 28.2.2020.

2. Tarkowska, Aleksandra - Carvalho-Silva, Denise – Cook, Charles E. – Turner, Edd – Finn, Robert D. – Yates, Andrew D. 2018. Eleven quick tips to build a usable REST API for life sciences. PLOS Computational Biology. Saatavissa: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006542. Hakupäivä 13.2.2020.

3. Richardson, Leonard 2015. Five In Five Talk, REST Fest 2015: What Have I Done? Video. Saatavissa: https://vimeo.com/139971856. Hakupäivä 22.8.2020.

4. Robinson, Ian - Parastatidis, Savas - Webber, Jim 2010. REST in Practice. Sebastopol, CA: O'Reilly Media, Inc. Saatavissa: https://learning.oreilly.com/library/view/rest-in-practice/9781449383312 (vaatii käyttöoikeuden). Hakupäivä 25.2.2020.

5. Nyström, Mikael – Karlsson, Daniel - Eneling, Martin - Chen, Rong – Örman, Håkan 2013. Applying representational state transfer (REST) architecture to archetype-based electronic health record systems. BMC Medical Informatics and Decision Making. Saatavissa: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3694512. Hakupäivä 26.2.2020.

6. Kobusińska, Anna – HsienHsu, Chieng 2018. Towards increasing reliability of clouds environments with RESTful web services. Future Generation Computer Systems vol. 87. S. 502–513. Saatavissa: https://www.sciencedirect.com/science/article/pii/S0167739X17316709. Hakupäivä 26.2.2020.

7. Axelson, Jan 2003. Embedded ethernet and internet complete. Madison: Lakeview Research LLC. Saatavissa: https://ebookcentral-proquest-

com.ezp.oamk.fi:2047/lib/oamk-ebooks/reader.action?docID=175243 (vaatii käyttöoikeuden). Hakupäivä 13.2.2020.

8. Vijayakumar, Thurupathan 2018. Practical API Architecture and Development with Azure and AWS: Design and Implementation of APIs for the Cloud. Apress. Saatavissa: https://learning.oreilly.com/library/view/practical-api-architecture/9781484235553/html/460096_1_En_5_Chapter.xhtml (vaatii käyttöoikeuden). Hakupäivä 14.9.2020

9. River, Stone 2017. Starting with REST API's. O'Reilly Media, Inc. Video. Saatavissa: https://learning.oreilly.com/videos/starting-with-rest/100000006A0463 (vaatii käyttöoikeuden). Hakupäivä 13.2.2020.

10. SmartBear Software. 2020. API Resources. Saatavissa: https://swagger.io/resources/open-api/. Hakupäivä 15.5.2020.

11. Koberger, Gregory 2017. A Visual Guide to What's New in Swagger 3.0. Saatavissa: https://blog.readme.com/an-example-filled-guide-to-swagger-3-2/. Hakupäivä 10.9.2020

12. SmartBear Software. 2020. OpenAPI Specification. Saatavissa: https://swagger.io/specification/. Hakupäivä 10.9.2020