Eetu Rouhiainen

Rajapintapalveluiden virtualisointi MuleSoft-alustalla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

21.5.2025

Tiivistelmä

Tekijä:	Eetu Rouhiainen
Otsikko:	Rajapintapalveluiden virtualisointi MuleSoft-alustalla
Sivumäärä:	38 sivua
Aika:	21.05.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mobile Solutions
Ohjaajat:	Manager Integraatiot ja robotiikka Markus Petman Lehtori Matti Peltoniemi

---

Insinöörityössä tutkittiin MuleSoft-alustan työkaluja ja niiden mahdollista hyödyntämistä rajapintapalveluiden virtualisoinnissa. Lisäksi selvitettiin, mitä hyötyjä rajapintapalveluiden virtualisoinnilla voidaan saavuttaa muun muassa kehitystyön nopeuttamisessa, järjestelmäriippuvuuksien vähentämisessä ja testauksen tehostamisessa. Insinöörityössä rakennettiin myös rajapintojen virtualisointipalvelu käyttäen MuleSoft-työkaluja. Vakuutusalan yrityksellä, jolle insinöörityö tehtiin, on menossa suuri järjestelmähanke, ja uuden rajapintojen virtualisointipalvelun on tarkoitus auttaa hankkeen kehitys- ja testausprosesseissa. Yrityksellä oli ennestään jo rakennettu palvelu rajapintapalveluiden virtualisointiin, mutta se ei pystynyt enää vastaamaan hankkeen tarpeisiin, joten migraatio uuteen palveluun oli tarpeellinen.

Uuden palvelun tuli kyetä palauttamaan oletusvastauksia ja simuloimaan virhetilanteita. Tarkoituksena oli siirtyä täysin MuleSoft-alustalle ja käyttää sen kyvykkyyksiä mahdollisimman paljon. MuleSoft-alusta tarjosi tähän Mocking Service -palvelun. Työssä testattiin ja tutkittiin Mocking Service -palvelun kyvykkyyksiä.

Palvelu vaati kuitenkin paljon mukautettua logiikkaa, joka rakennettiin Mule-sovellukseen Anypoint Studio -työkalun avulla. Valmis palvelu on ollut hyödyllinen kehitys- ja testaustyössä yrityksen järjestelmähankkeessa. Palvelusta on ollut hyötyä myös alkuperäisten käyttötarkoitusten ulkopuolella. Tällainen käyttötarkoitus on esimerkiksi järjestelmähankkeen tuotantokoodin varmistaminen. Rajapintojen virtualisoinnista on ollut niin paljon hyötyä, että yrityksessä harkitaan niiden käyttöönottoa kaikessa kehityksessä, joissa rajapintapalveluita tarvitaan.

Avainsanat: MuleSoft, ohjelmointirajapinta, palveluvirtualisointi

---

Abstract

Author:	Eetu Rouhiainen
Title:	API virtualization on the MuleSoft platform
Number of Pages:	38 pages
Date:	21 May 2025
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Mobile Solutions
Supervisors:	Markus Petman, Manager Integrations and Robotics Matti Peltoniemi, Senior Lecturer

---

The final year project investigated the tools of the MuleSoft platform and their utilization in API service virtualization. Additionally, it explored what benefits service virtualization could accomplish in speeding up the development work, reducing system dependencies and enhancing testing efficiency. The final project also included building an API service virtualization application using tools provided by the MuleSoft platform. The insurance company for which the project was conducted is carrying out a major system project, and the new API virtualization service is intended to assist in the project's development and testing processes. The company already had an API virtualization service for this, but it could not meet the requirements of the system project anymore, making migration to a new service necessary.

The new service needed to be able to return default responses and simulate error cases. The goal was to transition entirely to the MuleSoft platform and leverage its capabilities as much as possible. The Mocking Service of the MuleSoft platform was used for this purpose. The capabilities of the Mocking Service were tested and investigated as part of the final project.

However, the service required a significant amount of custom logic, which was built into a Mule application using the Anypoint Studio tool. The completed service has proved to be useful in the development and testing work of the company's system project. The service has also proved beneficial beyond its original use cases. For example, in the process of verifying the system project's code. The developed API service virtualization application has been so beneficial that the company is considering adopting API service virtualization in all developments where API services are needed.

Keywords: MuleSoft, API, Service virtualization

Sisällys

Lyhenteet

JSON: *JavaScript Object Notation* on formaatti, jota käytetään datan tallentamiseen ja liikuttamiseen.

XML: *Extensible Markup Language* on merkintäkieli, joka tarjoaa säännöt minkä tahansa tiedon määrittelemiseksi.

YAML: *Yet Another Markup Language* on luettavaksi tarkoitettu tietojen merkintäkieli.

RAML: *RESTful API Modeling Language* on YAML-kieleen perustuva kieli staattisten sovellusliittymien kuvaamiseen. Sitä käytetään yleisesti vain RESTful-rajapintojen kehitykseen.

HTTP: *HyperText Transfer Protocol* on sääntöjärjestelmä tietojen siirtämiseen internetissä.

CRM: *Customer Relationship Management* on järjestelmä asiakassuhteiden hallintaan ja kehittämiseen.

1 Johdanto

Tämä insinöörityö käsittelee MuleSoft-alustaa ja yleisesti rajapintapalveluiden virtualisointia. Työssä tutkitaan MuleSoft-alustaa ja sen työkaluja. Työssä käydään tarkasti läpi rajapintapalveluiden virtualisointia ja sen tuomia hyötyjä sovellusten ja ohjelmistojen kehitysprosesseissa. Työn tarkoitus on lisätä ymmärrystä rajapintapalveluiden virtualisoinnin kannattavuudesta ja siitä, miten virtualisointi on mahdollista MuleSoft-alustan tarjoamien työkalujen avulla. Työssä käytetään yleistä nimitystä rajapinta, mutta sillä viitataan ohjelmointirajapintoihin.

Insinöörityössä rakennetaan rajapintapalveluiden virtualisointipalvelu, jonka tavoitteena on tarjota uudenlainen toimintaympäristö vakuutusalan yrityksen uudelle järjestelmähankkeelle. Palvelun on tarkoitus simuloida oikeiden rajapintapalveluiden toiminnallisuutta ja vähentää riippuvuuksia reaaliaikaisista tuotantojärjestelmistä ohjelmistokehityksen ja testauksen aikana. Valmiina palvelun pitäisi nopeuttaa kehitysprosesseja ja tuoda uusia toiminnallisuuksia yrityksen järjestelmähankkeen kehitykseen ja testaukseen. Mikäli palvelu osoittautuu hyödylliseksi, virtualisoidut rajapintapalvelut voidaan ottaa käyttöön jokaisessa rajapintoja tarvitsevassa ohjelmistokehitysprojektissa. Lisäksi insinöörityö sisältää rakennetun palvelun vaikutusten arviointia.

2 Rajapintapalveluiden virtualisointi

Brajesh De [1] kutsuu kirjassaan rajapintapalveluita ohjelmistojen välisiksi silloiksi, joihin on asetettu ohjeistuksia, miten siltaa käyttävät ohjelmistot käyttäytyvät. Rajapintapalveluiden tarkoitus on paljastaa osa yrityksen liiketoimintaa tai jokin resurssi kehittäjille, jotka voivat rakentaa logiikkaa rajapintapalvelun pohjalta. Esimerkiksi Google Maps tarjoaa rajapintapalvelun, josta voi löytää kartalta tietyn pisteen, jota kehittäjä voisi käyttää jossain oman sovelluksen toiminnallisuudessa. RESTful-rajapintapalvelut, joihin tämä insinöörityö liittyy, ovat HTTP-protokollaa käyttäviä rajapintapalveluita. Kyseiset rajapintapalvelut käyttävät yleensä jotain neljästä HTTP-protokolla metodista, jotka ovat GET, POST, PUT ja DELETE. Näitä eri metodeja voidaan käyttää datan hakemiseen, lisäämiseen tai muuttamiseen. RESTful-rajapintapalvelut välittävät dataa JSON- tai XML-muodossa. [1.]

GET-metodi on kaikista eniten käytetty HTTP-protokolla, ja sitä käytetään datan pyytämiseen rajapintapalvelusta. POST-metodia käytetään tiedon lähettämiseen rajapintapalveluun, ja POST-metodin avulla luodaan uutta dataa rajapintapalveluun yhdistettyihin järjestelmiin. PUT-metodia käytetään vanhan datan korvaamiseen uudella ja DELETE-metodia on tarkoitus käyttää datan poistamiseen. POST- ja PUT-metodeilla tehtäviin kutsuihin täytyy lisätä runko, joka sisältää uuden tai korvattavan datan. [1.]

Rajapintapalveluiden virtualisoinnilla simuloidaan oikeiden rajapintapalveluiden toiminnallisuuksia. Usein organisaatioiden sisäiset rajapintapalvelut kutsuvat toisiaan, jolloin rajapintapalveluiden virtualisoinnista on usein hyötyä. Rajapinnan virtualisoinnilla voidaan saada esimerkkivastauksia kutsumalla virtualisoitua rajapintapalvelua, vaikka kyseisen oikean rajapintapalvelun kehitys olisi vielä kesken. [2.]

Rajapintapalveluiden virtualisoinnin on tarkoitus nopeuttaa rajapintapalveluiden kehitystä. Virtualisoitu rajapintapalvelu palauttaa vastauksia, jotka on ennalta

määritetty sille kehitteillä olevalle rajapinnalle, jota virtualisointi koskee. Kun rajapintapalvelun kehitystyö on valmis, voidaan järjestelmiin, jotka kutsuvat virtualisoitua rajapintapalvelua, yksinkertaisesti päivittää oikean rajapintapalvelun verkko-osoite. Näin ollen oikeaa rajapintapalvelua ja sitä kutsuvaa järjestelmää on pystytty kehittämään yhtäaikaisesti, ja tämä nopeuttaa koko kehitysprosessia. [2.]

Rajapintapalveluiden virtualisoinnista on hyötyä myös testaajille. Virtualisoitu rajapintapalvelu jäljittelee oikean rajapintapalvelun toimivuutta, jolloin myös testaajat voivat kokeilla sitä eri järjestelmissä ja varmistaa toimivuuden ennen oikean rajapintapalvelun valmistumista. Tällä pyritään siihen, että mahdolliset ongelmat löydetään ennen oikean rajapintapalvelun kehitystyön alkua. Virtualisoiduissa rajapintapalveluissa pitää aina ottaa huomioon se, että niissä ei yleensä ole mitään oikeaa prosessointia tai logiikkaa, vaan ne palauttavat esimerkkivastauksia. Ne eivät näin ollen koskaan voi korvata oikeita rajapintapalveluratkaisuja. [3.]

Taulukko 1 osoittaa, että virtualisoidun rajapintapalvelun ei ole tarkoitus korvata oikeaa rajapintapalvelua, vaan auttaa kehityksessä. Rajapintapalveluiden virtualisointi auttaa testaus- ja kehitysprosesseissa nopeuttaen kehitystä ja varmistamalla, että testausdata on aina saatavilla. Virtualisoiduissa rajapinnoissa ei yleisesti ole tarvetta turvata dataa, koska se ei käsittele esimerkiksi oikeaa henkilödataa. Data ei siis vaadi samoja vaatimustenmukaisuustoimenpiteitä kuin oikea data. Virtualisoituja rajapintapalveluita ei myöskään ole tarve suojata tästä syystä, vaan usein monilla kehittäjillä on niihin vapaa pääsy. Virtualisoidut rajapintapalvelut usein vastaavat kutsuihin nopeasti ja ennustettavasti, koska ne eivät käsittele tietoja, vaan palauttavat vain vastauksia. [3.]

Taulukko 1. Oikean ja virtualisoidun rajapintapalvelun erot. [3.]

	Virtualisoitu rajapinta-palvelu	**Oikea rajapintapalvelu**
Datan käsittely	Simuloitua esimerkki-dataa	Oikean datan käsittelyä, tallentamista ja hake-mista
Verkko	Lokaali tai testausympä-ristö	Verkkojen, tyypillisesti Internetin kautta
Vastausaika	Tasainen	Saattaa muuttua järjes-telmän ruuhkan mukaan
Data	Muuttumatonta esimerk-kidataa	Oikeaa muuttuvaa dataa
Testaus ja kehitys	Käytetään testauksessa ja kehityksessä	Käytetään tuotannossa
Virheiden simulointi	Voidaan simuloida vir-heitä	Ei voida simuloida virhe-tilanteita
Turvallisuus	Ei turvattu	Turvallisuusvaatimuksia datalle

3 MuleSoft-alusta

MuleSoft-alusta on rajapintapalveluiden hallitsemisen helpottamiseen rakennettu alusta. MuleSoft-alusta on tehty tietojen ja järjestelmien integroinnin, sekä työnkulkujen ja prosessien automatisoinnin helpottamiseksi. MuleSoft-alusta sisältää useita eri työkaluja, joista käyttäjä voi valita itselleen sopivat, joiden avulla käyttäjät voivat saavuttaa tavoitteensa. MuleSoft-alustaa voidaan käyttää kaikkiin rajapintapalveluihin liittyviin prosesseihin, esimerkiksi turvallisuus-, hallinto- ja vaatimustenmukaisuustoimenpiteisiin. [4.]

MuleSoft-alustan perusti vuonna 2006 Ross Mason, ja hänen näkemyksensä oli rakentaa alusta, jonka tarkoitus olisi helpottaa rajapintapalveluiden rakentamista ja hallintaa [5]. MuleSoft-alusta oli avoimeen lähdekoodiin perustuva eli sen lähdekoodit olivat ennen kaikkien saatavilla [5]. Vuodesta 2018 alkaen MuleSoft-alustan on omistanut Salesforce, ja MuleSoft-alusta siirtyi pois avoimesta lähdekoodista [5]. Salesforce on perustettu vuonna 1999, ja tarjoaa CRM-teknologiaa [6]. Nykyään Salesforce panostaa paljon CRM-teknologian parantamiseen tekoälyn avulla [6].

3.1 Rajapintajohtoinen liitettävyys

Rajapintajohtoinen liitettävyys on lähestymistapa, jossa rajapintapalveluita rakennetaan uudelleenkäytettävyys ja tarkoituksenmukaisuus edellä. Rajapintajohtoinen liitettävyys on MuleSoft-kehityksen keskiössä ja sen kulmakivi. Rajapintajohtoinen liitettävyys lähestymistavassa uskotaan vahvasti siihen, että rajapintapalvelut ovat nykyisten yrityksien kulmakivi ja palvelut kuuluisivat rakentaa siten, että samoja rajapintoja voidaan käyttää monissa eri prosesseissa. Kuvassa 1 näkyy, miten yksi applikaatio käyttää kolmea eri rajapintapalveluketjua puhelinsovelluksessa. Samoja ketjua voidaan käyttää esimerkiksi verkkosovelluksessa. Rajapintajohtoisen liitettävyyden on tarkoitus vähentää organisaatioiden rajapintapalveluita, ja täten helpottaa niiden ylläpitämistä ja hallintaa. [7.]

-

Kuva 1. Kuvasta nähdään, miten eri rajapintapalveluita voidaan yhdistää yhteen mobiilisovellukseen, kun noudatetaan rajapintajohtoisen liitettävyyden periaatteita. [7.]

Jokaisella rajapintapalvelulla pitää olla tarkoitus, johon liittyy tekninen tai liiketoiminnallinen tarve. Rajapintapalveluista on tarkoitus tehdä automatisoituja ja organisoida ne ominaisuuksilla, jotka tekevät niistä yhteensopivia moniin eri prosesseihin. Rajapintapalveluiden kyky integroida ulkoiset ja sisäiset toiminnot on keskeinen tekijä liiketoiminnan ketteryyden varmistamisessa. Tarkoituksena on, että kun tulee uusi tarve rajapintapalvelulle, voidaan käyttää jo olemassa olevia rajapintapalveluita sen sijaan, että rakennettaisiin uusia. Näin säästyy aikaa ja resursseja. [7.]

Rajapintajohtoisessa liitettävyydessä on kolme eri tasoa rajapintapalveluille. Tarkoituksena on se, että rajapintapalveluita voidaan siten käyttää eri prosesseissa. Nämä kolme tasoa on nimetty järjestelmä-, prosessi- ja kokemusrajapinnoiksi. Kokemusrajapinnat ovat korkein taso, ja ne antavat lopulliset vastaukset prosesseille, jotka kutsuvat rajapintapalvelua. Prosessirajapinnoissa dataa muutetaan eri muotoihin, joita käytetään kokemustasolla ja kokemusrajapinnat

kutsuvat prosessirajapintoja. Järjestelmärajapinnat kutsuvat yrityksen omia tai ulkoisten kumppanien järjestelmiä, ja järjestelmärajapintoja kutsuvat prosessirajapinnat. [7.]

3.2 C4E

C4E eli englanniksi *Center for Establishment* on monialainen tiimi, jonka perustamista organisaatioihin MuleSoft suosittelee. Tiimin tarkoitus on tuotteistaa, julkaista ja kerätä erilaisia resursseja, joita voidaan käyttää MuleSoft-kehityksessä tai sen ulkopuolella. C4E-ajattelutavan on tarkoitus korvata yleinen CoE-ajattelutapa eli englanniksi *Center of Expertise*. [8.]

C4E- ja CoE-ajattelutapojen keskeiset erot löytyvät kahdesta eri näkökulmasta. CoE-ajattelutavassa rajapintapalveluita rakennetaan projektikohtaisesti, kun taas C4E-ajattelutavassa pyritään rakentamaan rajapintapalvelut siten, että voidaan uudelleen käyttää vanhoja rajapintapalveluita uusissa projekteissa. C4E-ajatellutavassa pyritään siihen, että usea taho yrityksen sisällä voi käyttää rajapintapalveluita, eikä niitä ole keskitetty vain yhden tahon käyttöön. [8.]

3.3 Anypoint-alusta

Anypoint-alusta sisältää MuleSoft-työkaluja, joita käytetään rajapintapalveluiden hallintaan, rakentamiseen ja suojaamiseen [9]. Anypoint-alusta tarjoaa verkkoportaalin, jonka tarkoitus on koota suurin osa MuleSoftin työkaluista yhteen verkkoportaaliin [9]. Tämän insinöörityön osalta tärkeitä työkaluja ja ympäristöjä ovat Design Center, Exchange, Anypoint studio, CloudHub ja API Manager.

Design Center on MuleSoft-alustan kehitysympäristö, joka sisältää API Designer -työkalun. API Designer -työkalulla on tarkoitus luoda RAML-tiedostoja, joita käytetään rajapintapalveluiden määrittelemiseen. Kun RAML-tiedosto on luotu rajapintapalvelulle Design Center -kehitysympäristössä, julkaistaan se Anypoint Exchange -työkaluun. Kun Design Center -kehitysympäristössä luodaan RAML-tiedosto rajapintapalvelulle, MuleSoft luo automaattisesti testaamista varten

rajapintapalvelun. Uuden rajapintapalvelun kehitystyö alkaa yleensä API Designer -työkalusta. [10.]

Exchange-palvelu on uudelleenkäytettävien rajapintamäärittelyjen ja komponenttinen kirjasto. Kaikki API Designer -työkalulla tehdyt rajapintamäärittelyt julkaistaan Exchange-palveluun, jolloin niitä voidaan käyttää organisaation eri prosesseissa. Exchange-palvelu sisältää myös organisaation erilaisia resursseja, joita käytetään prosesseissa. Exchange-palvelu sisältää myös MuleSoft-alustan ja sen kumppanien luomia resursseja, joita voidaan käyttää organisaatioiden prosesseissa. Kaikki rajapintapalveluiden rakentamiseen tarvittava on Exchange-palvelusta helposti löydettävissä, ja käyttöoikeuksia voi samoin pyytää Exchange-palvelun kautta. Exchange-palvelussa voidaan testata myös näitä komponentteja ja rajapintapalveluiden määrittelyjä Exchange-palvelun omassa verkkoportaalissa. [11.]

Anypoint Studio -työkalu on Eclipse-ohjelmointiympäristöön perustuva työkalu [12]. Eclipse on avoimeen lähdekoodiin perustuva ohjelmointityökalu, joten sen käyttö on ilmaista ja työkalun muokkaaminen omiin tarpeisiin on sallittua [13]. Eclipse on visuaalinen työkalu, jota yleensä käytetään Java-koodikielellä tehtävään kehitykseen [13]. Eclipse kehitettiin Java-kehityksen helpottamiseksi, mutta sitä voidaan käyttää myös eri koodikielillä tehtävään kehitykseen [13].

Anypoint Studio -työkalua käytetään Mule-sovellusten kehittämiseen. Mule-sovellukset ovat integraatioratkaisuja, joiden tarkoitus on lisätä rajapintapalveluihin logiikkaa [12]. Mule-sovellukset koostuvat eri osista, joista kerrotaan tarkemmin luvussa 4. Mule-sovellukset ovat MuleSoft-kehityksen keskiössä [12]. Muista Anypoint-alustan työkaluista eroten Anypoint Studiota ei ole mahdollista käyttää verkkoportaalissa [12].

CloudHub on integraatioympäristö, jossa Mule-sovellukset käynnistetään ja suoritetaan. CloudHub tarjoaa paljon erilaisia työkaluja Mule-sovellusten seuraamiseen kuten Runtime manager. Runtime managerilla käyttäjä voi seurata Mule-sovellustaan, vaikka virheiden tai kutsujen määrän osalta verkkoportaalissa. [14.]

API Manager tarjoaa ominaisuuksia rajapintapalveluiden suojaukseen ja hallintaan. API Managerin kautta rajapinnoille voi asettaa erilaisia sääntelyitä tai esimerkiksi rajoittaa pääsyn rajapintaan vain tietyille käyttäjille. API Managerin kautta on myös mahdollista seurata rajapintapalvelun käyttöä kuten kutsujen tai virheiden määrää. Kun uusi rajapintapalvelu kehitetään, luodaan API-Manager-työkaluun uusi rajapintapalvelu, joka yhdistetään uuteen rajapintapalveluun. [15.]

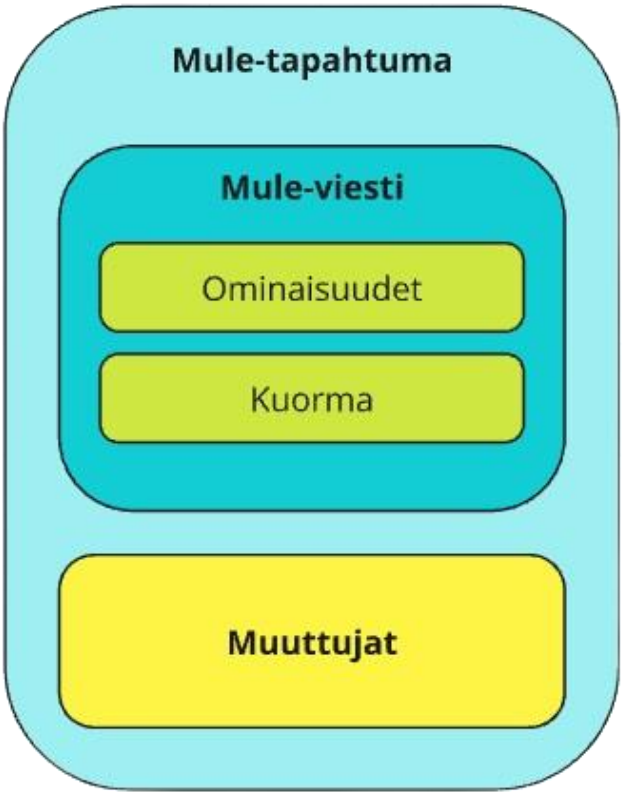
4 Mule-sovellukset

Mule-sovellukset ovat sovelluksia, joita rakennetaan järjestelmäintegraatioiden suorittamiseksi. Mule-sovellusten on tarkoitus lukea dataa sisäisistä tai ulkoisista järjestelmistä ja käsitellä sitä. Mule-sovellukset voivat muuttaa datan eri muotoihin, ja rakenteisiin sekä lähettää datan järjestelmiin, joissa sitä käytetään. Mule-sovellukset toimivat MuleSoftin suorituspalvelussa. [16.]

4.1 Mule-sovelluksen rakenne

DataWeave on MuleSoftin kehittämä koodikieli. DataWeaven tarkoitus on suorittaa erilaisia dataan liittyviä muutoksia Mule-sovelluksessa. DataWeaven avulla voidaan muokata tai noutaa dataa Mule-tapahtumasta. DataWeaven avulla voidaan esimerkiksi muokata datan muotoa XML-muodosta JSON-muotoon. [17.]

Mule-tapahtumat sisältävät tietoa, jota voidaan käyttää Mule-sovelluksessa. Mule-tapahtumat sisältävät muuttujat ja Mule-viestin, ominaisuudet ja kuorman. Kuvassa 2 esitetään Mule-tapahtuman rakenne. Mule-tapahtumat kulkevat flow-tapahtumissa, ja Mule-tapahtumien sisällä on kaikki tiedot, mitä Mule-sovellus vastaanottaa tai lähettää. Muuttujat ovat Mule-tapahtuman osa, johon voidaan asettaa Mule-sovelluksen sisällä dataa, ja tätä dataa voidaan käyttää jossain myöhemmin, ennen kuin vastaus palautetaan kutsuneeseen järjestelmään. [18.]



Kuva 2. Kuva Mule-tapahtuman rakenteesta. [18.]

Jos kutsuva järjestelmä on käyttänyt POST-metodia, niin kuorma sisältää kutsun mukana tulleen datan. Tätä dataa voidaan käsitellä Mule-sovelluksessa. GET-metodia käyttävät kutsut eivät tuo uutta dataa kuormaan, mutta usein tällöin sovellus noutaa jostain muusta järjestelmästä dataa, ja data asetetaan kuormaksi ja palautetaan kutsuvaan järjestelmään. Kuorma on usein se, mitä palautetaan kutsuvaan järjestelmään. Ominaisuudet-osio sisältää kutsun kuljetuskerroksia, parametreja ja muita kutsuun liittyviä tietoja. Esimerkkinä tällaisesta tiedosta on päivämäärä, jolloin kutsu on saapunut rajapintapalveluun. Ominaisuuksia myös usein käytetään Mule-sovelluksissa eri käyttötarkoituksiin. [19.]

Flow’t ovat Mule-sovelluksien tärkeimpiä rakennusosia. Mule-sovellukset käsittelevät dataa erilaisten liittimien ja moduulien avulla, jotka sijaitsevat aina Flow’n sisällä. Flow’t jaetaan kahteen eri ryhmään flow tai subflow. Mule-sovellukset voivat koostua yhdestä tai useasta flowsta, mutta yleensä niitä on sovelluksissa

useita. Flow sisältää jonkin lähteen esimerkiksi HTTP listener -komponentin, joka ottaa vastaan rajapintakutsun. Flow sisältää myös virheidenhallintaosion, johon on mahdollista lisätä logiikkaa virheiden varalle. Subflow’n ero flow’hun on se, että sillä ei ole lähdettä eikä virheidenhallintaa, vaan sitä kutsutaan Mule-sovelluksen sisällä, ja se käyttää kutsuvan flow’n virheidenhallintaa. [20.]

4.2 Virheidenhallinta

Mule-sovelluksissa virhetilanteet jaetaan kahteen kategoriaan: järjestelmä- ja viestivirheisiin. Järjestelmävirheitä ei voida käsitellä Mule-sovelluksessa, vaan ne käsitellään Mule-sovelluksen ulkopuolella. Järjestelmävirheen tapahtuessa Mule kirjaa lokiin virheen ja lähettää tiedotteen järjestelmään, jos sellainen on asetettu vastaanottamaan tiedotteita. Yhteysvirhetilanteissa Mule yrittää myös palauttaa yhteyden. [21.]

Jos Mule-sovelluksessa tapahtuu virhetilanne Flow-tasolla, silloin virhetilannetta kutsutaan viestivirheeksi. Viestivirheitä voidaan käsitellä käyttämällä Mulen omaa oletusvirheenkäsittelijää, tai viestivirhe voidaan käsitellä itse Flow’n omassa virheenkäsittelijässä. Flow’n omasta virheenkäsittelijästä on myös mahdollista kutsua toista Flow’ta, joka sisältää logiikkaa virheidenkäsittelyyn. Oletusvirheenkäsittelijää käytettäessä virhetilanteet kirjataan lokiin ja Flow’n käsittely pysähtyy ja kutsuvaan järjestämään palautetaan virheviesti. Esimerkkikoodissa 1 on esimerkki lokista virhetilanteen 404 sattuessa. [21.]

```
ERROR 2021-01-19 18:48:50,392
  [[error-handlers-example].http.requester.HTTP_Request_configura-
tion.11 SelectorRunner]
  [processor: error-handlers-example/processors/0; event: 1d3baf11-
5aa0-11eb-b96f-a483e7abe2b5] org.mule.runtime.core.internal.excep-
tion.OnErrorPropagateHandler:
********************************************************************
**********
Message               : HTTP GET on resource 'http://jsonplace-
holder.typicode.com:80/somebadrequest' failed: not found (404).
Element               : error-handlers-example/processors/0 @ error-
handlers-example:error-handlers-example.xml:15 (Request)
Element DSL           : <http:request method="GET" doc:name="Request"
config-ref="HTTP_Request_configuration" path="/somebadre-
quest"></http:request>
Error type            : HTTP:NOT_FOUND
FlowStack             : at error-handlers-example(error-handlers-exam-
ple/processors/0 @ error-handlers-example:error-handlers-exam-
ple.xml:15 (Request))

  (set debug level logging or '-Dmule.verbose.exceptions=true' for
everything)
```

Esimerkkikoodi 1. Virheloki virhetilanteessa 404. [21.]

Loki sisältää tietoa virhetilanteesta. Message eli viesti sisältää kuvauksen virhetilanteesta. Element eli elementti sisältää tiedon XML-elementistä missä virhetilanne on tapahtunut ja Element DLS sisältää saman tiedon DLS-muodossa, jonka tarkoituksena on osoittaa elementin XML-rakenne, joka johti virhetilanteeseen. Error type sisältää virheen tyypin ja Flowstack sisältää flow’n nimen, jossa virhetilanne on tapahtunut. Esimerkkikoodissa 2 esitetään vastaus, joka palautetaan järjestelmään, joka kutsui kyseistä rajapintapalvelun päätepistettä. [21]

```
HTTP GET on resource 'http://jsonplaceholder.typicode.com:80/some-
badrequest'
failed: not found (404).
```

Esimerkkikoodi 2. Rajapintapalvelun palauttama viesti [21].

5 Rajapintapalveluiden virtualisointi MuleSoft-alustalla

MuleSoft tarjoaa rajapintapalveluiden virtualisointiin Mocking Service -palvelun. Mocking Service -palvelu luo automaattisesti virtualisoidun rajapintapalvelun, kun käyttäjä tekee RAML-määrittelyn API Designer -työkalussa. Mocking Service -palvelun avulla voidaan testata kaikkia luotuja rajapintapalvelun päätepisteitä. Mocking Service -palvelu tarjoaa myös työkaluja rajapintapalvelun testaamiseen. [22.]

Rajapinnan määrittelyvaiheessa RAML-tiedostoon on mahdollista lisätä esimerkkivastauksia [23]. Esimerkkivastaukset lisätään RAML-tiedostoon rajapintapalvelun päätepisteen alle, joko *example*- tai *examples*-nimityksiä käyttäen riippuen siitä, halutaanko lisätä yksi vai useampi esimerkkivastaus [23]. Esimerkkivastaukset voidaan lisätä suoraan rajapintapalvelun päätepisteen määrittelyn alle tai voidaan käyttää fragmentteja, joihin viitataan määrittelyssä [23]. Fragmentit ovat tiedostoja, jotka sisältävät koodia. Mocking Service -palvelu käyttää automaattisesti esimerkkivastauksia, kun sitä kutsutaan. Jos esimerkkivastauksia on useita, täytyy ne nimetä eri nimillä, jotta Mocking Service -palvelu palauttaa oikean vastauksen käytettäessä MS2-Example-kuljetuskerrosta rajapintaa kutsuttaessa. Esimerkkikoodeissa 3, 4 ja 5 näkyy esimerkkidatan lisäys RAML-tiedostoon, fragmentin, joka sisältää asiakastiedoston rakenteen ja JSON-tiedoston, joka sisältää lopullisen vastauksen.

```
/asiakas:
  get:
    description: Hakee asiakasdataa
    responses:
      200:
        body:
          application/json:
            type: !include asiakas.raml
            example:
              asiakas: !include asiakas.json
```

Esimerkkikoodi 3. Esimerkki asiakaspäätepisteen määrittelystä.

```
#%RAML 1.0 DataType
type: object
properties:
  nimi:
    type: string
    example: "Mikko Mock"
```

Esimerkkikoodi 4. Esimerkki *asiakas.raml*-fragmenttitiedostosta.

```
{
  "nimi": "Mikko Mock"
}
```

Esimerkkikoodi 5. Esimerkkidatatiedosto *asiakas.json*, jonka data palautetaan kutsujalle.

Mocking Service -palveluun on rakennettu sisäinen mahdollisuus käyttää dynaamista dataa. Tämä onnistuu asettamalla kutsun kuljetuskerroksiin MS2-Example-kuljetuskerros [24]. MS2-Example-kuljetuskerroksen avulla voidaan valita eri esimerkkivastaukista, joihin on viittaus RAML-määrittelyssä haluttu vastaus [24]. Dynaaminen data on usein tarpeellista, koska rajapintapalvelut palauttavat usein eri vastauksia. Dynaamisella datalla tarkoitetaan, että rajapintapalvelu voi palauttaa saamasta päätepisteestä eri vastauksia.

Mocking Service -palvelu tarjoaa mahdollisuuden virhetilanteiden simuloinnille asettamalla kuljetuskerrokseen MS2-Status-Code. Tätä voidaan hyödyntää, kun halutaan selvittää, miten Mocking Service -palvelua kutsuva järjestelmä käyttäytyy virhetilanteissa. Esimerkiksi jos asettaa MS2-Status-Code-arvoksi 404, saa vastauksen ”Ei löydy”. MS2-Delay-kuljetuskerros antaa mahdollisuuden lisätä viivettä pyyntöön. Viivettä voi lisätä enintään 10 000 millisekuntia. Viiveen avulla voidaan seurata, miten kutsuva järjestelmä käyttäytyy, kun vastaus tulee myöhemmin kuin on oletettu. [24.]

6 Mock Experience API

6.1 Uuden palvelun vaatimukset ja määrittely

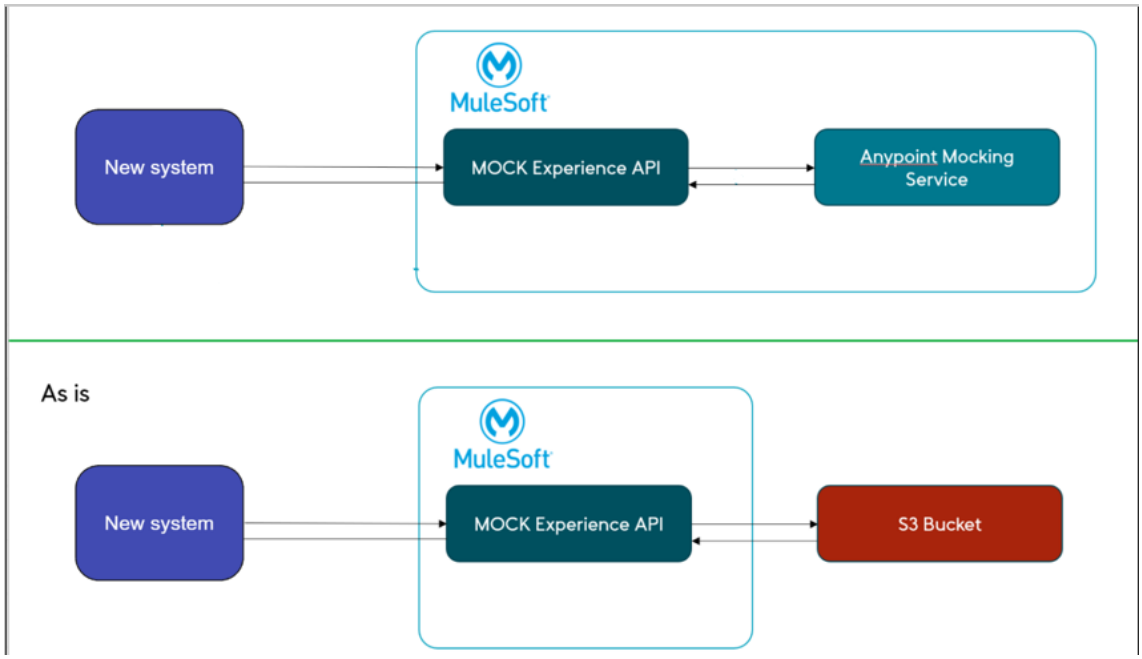
Vakuutusalan yrityksellä, jolle projekti tehtiin, on meneillään suuri järjestelmähanke, ja siinä ilmeni tarve virtualisoidulle rajapintapalvelulle. Järjestelmähankkeen tarpeisiin oli jo ennestään rakennettu palvelu käyttäen Amazonin S3-palvelua ja MuleSoft-alustaa, mutta tämä palvelu ei enää pystynyt vastaamaan hankkeen tarpeisiin, joten migraatio uuteen palveluun oli tarpeellinen. S3-palvelu on Amazonin omistama datan säilytyspalvelu [25]. Kaksi pääsyytä migraatiolle olivat tarve oletusvastauksille ja virhetilanteiden simuloimiselle. S3-palvelusta haluttiin myös luopua ja käyttää niin paljon MuleSoft-alustan tarjoamia ominaisuuksia kuin mahdollista.

Oletusvastauksilla tarkoitetaan tässä kontekstissa sitä, että rajapintapalvelua kutsuttaessa pitäisi aina palautua jokin ennalta määritelty vastaus. Vanhassa palvelussa jos haettua dataa ei löytynyt, niin palautettiin virheviesti. Tästä huolimatta virhekoodia ei palautunut. Tämä johtui siitä, että jos kyseistä tiedostoa ei ollut S3-palvelussa, niin rajapintapalvelu nouti sieltä tiedoston, jossa oli virheviesti, ja palautti sen kutsujalle. Uuden palvelun tuli pystyä simuloimaan oikeita virhetilanteita ja palauttamaan oletusvastauksia.

Virhetilanteiden simulointiin MuleSoft-alusta tarjoaa Mocking Service -palvelun avulla kyvykkyyden käyttäen MS2-Status-Code-kuljetuskerrosta. Tämä kyvykkyys oli myös yksi syy Mocking Service -palvelun käytölle uudessa toteutuksessa. Kuljetuskerroksen avulla oli mahdollista rakentaa logiikkaa virhetilanteiden simuloinnille.

Vanha palvelu toimi käyttämällä Mule-sovellusta S3-palvelun ja kutsuvan järjestelmän välissä. Kun kutsuvasta järjestelmästä tuli kutsu, Mule-sovellus nouti JSON-datatiedoston S3-palvelusta ja palautti sen. Kuvassa 3 esitetään erot

uuden ja vanhan palvelun järjestelmäarkkitehtuurista. Ylempänä on uusi palvelu ja alempana vanha.



Kuva 3. Uuden ja vanhan palvelun järjestelmäarkkitehtuurien erot.

Uuden palvelun vaatimuksissa piti päästä eroon S3-palvelusta. S3-palvelu oli ongelmallinen käyttöoikeuksien vuoksi suuressa järjestelmähankkeessa. Käyttöoikeuksia ei ollut S3-palveluun niitä tarvitsevilla henkilöillä ja niiden hankkiminen oli välillä haastavaa. Uuden palvelun avulla ei ole tarvetta käyttää muita kuin MuleSoft-alustan työkaluja, mikä vähensi käyttöoikeuksien tarvetta. S3-palvelusta luopuminen myös vähensi tarvetta palvelun päivitykselle, koska S3-palvelun käyttöavaimia ei tarvinnut enää päivittää tietyin väliajoin Experience Mock API -palveluun.

6.2 Palvelun suunnittelu

Palvelun laajuus oli 51 rajapinnan päätepistettä, jotka liittyivät kahdeksaan eri järjestelmään. Vanhassa toteutuksessa päätepisteitä oli 37, eli projekti koostui vanhojen päätepisteiden migraatiosta ja 14 uuden päätepisteen rakentamisesta. Projektin aluksi tutkittiin Mocking Service -palvelun ominaisuuksia. Ensin tutustuttiin ohjaaviin kuljetuskerroksiin ja niiden mahdollisiin hyötyihin. Mocking

Service -palvelun ohjaavista kuljetuskerroksista todettiin, että osa niistä toimisi erinomaisesti projektissa, varsinkin MS2-Example ja MS2-Status-Code. MS2-Delay-kuljetuskerroksen 10 sekunnin rajoitus teki siitä Experience Mock API -palvelun osalta turhan, koska 10 sekunnin viiveen lisäys kutsuun ei olisi tuottanut juurikaan hyötyä testaajille tai kehittäjille.

Mocking Service -palvelua testattiin kuormituksen alla. Tarkoituksena oli saada tietää, kuinka monta kutsua Mocking Service -palvelu kestää sekunnissa. Kuormitusta testattiin Postman-työkalulla, joka on työkalu rajapintapalveluiden rakentamiseen ja testaukseen [26]. Postman tarjoaa rajapintapalveluiden testaukseen ominaisuuden nimeltä Runner [27]. Runner-ominaisuudella voidaan lähettää useita rajapintakutsuja nopeasti rajapintapalveluihin [27]. Taulukko 3 esittää kuormitustestauksen tulokset.

Taulukko 3. Kuormitustestauksen tulokset.

Kutsua/s	Vasteaika (ms)	Virhetilanteet (%)
5,6	79	0
17,96	73	0,11
27,67	118	17,5

Kuormitustestauksessa testattiin kahta Mocking Service -palvelun päätepistettä samaan aikaan. Huomattiin, että yli 20 kutsua sekunnissa aiheutti paljon virheitä Mocking Service -palvelussa, ja kutsut palauttivat virhekoodia 503, joka viittaa palvelun kaatumiseen eli Mocking Service -palvelu ei pystynyt käsittelemään

kaikkia saapuvia kutsuja. Testausta jatkettiin noin viidellä kutsulla sekunnissa, jolloin virhetilanteita ei tapahtunut. Testausta jatkettiin lisäämällä aina yksi kutsu sekunnissa ja lopulta havaittiin, että noin 18 kutsua sekunnissa aiheutti hieman virhetilanteita. Experience Mock API -palvelun suhteen todettiin, ettei tämä tulisi todennäköisesti olemaan ongelma, koska palvelu ei tulisi olemaan kovin suuren kuormituksen alla. Palvelun suunnittelu jatkui suunnittelemalla kaksi eri vaihtoehtoa palvelun toimivuudelle. Suunnitelmat vietäisiin liiketoimelle, joka vastaa niiden hyväksynnästä, jolloin mahdollisesti yksi niistä hyväksyttäisiin ja projekti voitaisiin aloittaa.

Ensimmäinen suunnitelma olisi ollut nopein ja helpoin toteuttaa. Suunnitelmana oli tehdä kaikille päätepisteille yhtenäinen ratkaisu, jolloin millään päätepisteellä ei olisi ollut mukautettua toiminnallisuutta. Tämä ratkaisu olisi käyttänyt Mocking Service -palvelun ohjaavia kuljetuskerroksia suoraan uudesta järjestelmästä, jolloin esimerkkivastaukset tai virhetilanteet tilattaisiin ohjelmistohankkeen järjestelmästä Mule-sovelluksen sijaan. Mule-sovellus taas olisi toiminut vain välikappaleena hankkeen järjestelmän ja Mocking Service -palvelun välillä ilman räätälöityä logiikkaa. Tätä suunnitelmaa ei kuitenkaan valittu, koska järjestelmähankkeeseen ei haluttu järjestelmään rakentaa tarvittavaa toiminnallisuutta kuljetuskerroksien lisäämiseen.

Toisena esitetty suunnitelma valittiin projektiin. Suunnitelmassa jokaiselle päätepisteelle rakennettaisiin oma logiikka, mikä tarkoitti huomattavasti suurempaa manuaalista työtä kuin ensimmäisessä suunnitelmassa ja teknisen velan lisääntymistä pitkällä aikavälillä, koska jokainen päätepiste tulisi olemaan erilainen. Tämä johtaa siihen, että pienetkin muutokset, jotka koskevat montaa päätepistettä, pitäisi tehdä jokaiselle päätepisteelle erikseen. Ohjaavat kuljetuskerrokset lisättäisiin Mule-sovelluksen sisään ja niihin pitäisi kehittää logiikka, jolla niihin viitataan RAML-määrittelyssä.

6.3 RAML-tiedoston päivitys

Palvelun ohjelmointi tehtiin päätepiste kerrallaan. Tässä insinöörityössä selostetaan yhden päätepisteen ohjelmointi, koska kaikki päätepisteet noudattivat

suunnilleen samaa kaavaa. Palvelun ohjelmointi alkoi RAML-määrittelyn päivittämisellä. Vanhan palvelun RAML-määrittelyyn täytyi lisätä tarvittavat esimerkkivastaukset. Esimerkkikoodissa 6 on */customers*-päätepiste. *Customers*-päätepisteen tarkoitus on simuloida oikean rajapinnan toimivuutta siten, että se palauttaa JSON-tiedoston, joka sisältää asiakkaan tietoja.

```
/customers:
  /{ssn}:
    uriParameters:
      ssn:
        type: string
        description: "Civil registration code for the private person"
        pattern: "^[0-3]\\d[01]\\d{3}[+\\-A]\\d{3}[0-9A-FHJ-NPR-Y]$"
        example: "180991-4949"
    type:
      common.readCollectionResourceType:
        readResponseType: satResponse.SatCustomerResponse
        readResponseExample: !include /examples/SAT/get-custom-
ers/090999-999U.json
    get:
      displayName: Get customer information by SSN
      description: This mock API endpoint returns basic information of
private person from SAT (Suomen Asiakastieto).
      responses:
        200:
          body:
            application/json:
              examples:
                default: !include /examples/SAT/get-customers/090999-
999U.json
                010187-9341: !include /examples/SAT/get-custom
ers/010187-9341.json
                020275-951X: !include /examples/SAT/get-custom-
ers/020275-951X.json
                010200A9618: !include /examples/SAT/get-custom-
ers/010200A9618.json
```

Esimerkkikoodi 6. *Customers*-päätepiste, johon on lisätty esimerkkivastaukset.

Päätepistettä kutsuttaessa täytyy verkko-osoitteeseen lisätä myös henkilötunnus. Esimerkkikoodissa 6 tähän viitataan */{ssn}*-osiolla. Henkilötunnusta käytettiin myös tämän kyseisen päätepisteen datan nimityksissä, jolloin datan nimitykset voidaan erottaa toisistaan. Esimerkkikoodissa 6 *pattern*-osiota käytetään tarkistamaan henkilötunnuksen muoto, ja *pattern* palauttaa virheen, jos henkilötunnus ei ole oikeassa muodossa päätepistettä kutsuttaessa. Muissa päätepisteissä henkilötunnuksen sijasta päätepisteessä toimi, joku muu tunniste. Esimerkiksi */address*-päätepisteessä tunnisteena toimi katuosoite esimerkiksi *Kaivokatu_1_00100_ Helsinki.*

Get-osio esimerkkikoodissa 6 viittaa päätepisteen HTTP-metodiin. Tämän alle lisätään nimi päätepisteelle ja kuvaus, jotka näkyvät sitten Exchange-työkalussa, kun RAML-määrittely julkaistaan. Osion alta löytyy myös *responses* eli vastaukset-osio. Vastaukset-osion alussa määritellään koodi, joka palautetaan kutsuvaan järjestelmään. Tämän päätepisteen käyttäessä GET HTTP -metodia, palautetaan koodi 200. Seuraavaksi määriteltiin palautettavan datan muoto, joka oli JSON. *Examples*-osion alle on määritelty kaikki esimerkkidata, jota päätepiste palauttaa.

Esimerkkikoodissa 7 on esimerkki JSON-datasta 010187–9341, jonka tunnisteena toimii henkilötunnus. Joissain päätepisteissä ei ollut tarvetta dynaamiselle datalle, vaan yksi vastaus oli riittävä. Tällaiset päätepisteet yleensä vastaavat geneerisellä ”viesti vastaanotettu” -viestillä.

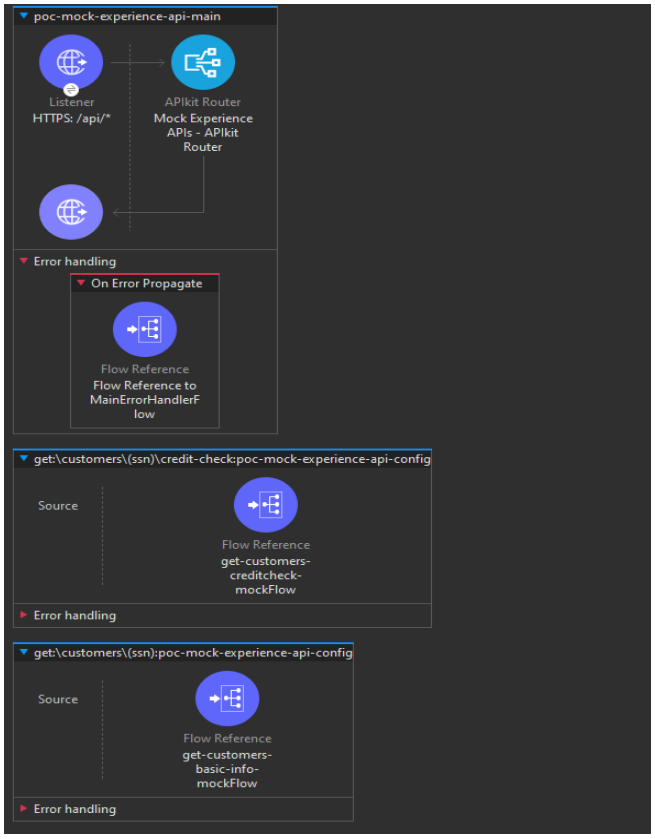
```
{
    "customer": {
        "customerType": "private",
        "socialSecurityNumber": "010187-9341",
        "firstName": "Testi",
        "lastName": "Testinen",
        "addresses": [
            {
                "code": "01",
                "type": "Osoite",
                "street": "Mannerheimintie",
                "postalCode": "00100",
                "city": "Helsinki"
            }
        ]
    }
}
```

Esimerkkikoodi 7. Esimerkki JSON-datasta.

6.4 Päätiedoston muutokset

Kun RAML-määrittely oli päivitetty, voitiin siirtyä ohjelmoimaan Anypoint Studio -sovelluksella. Mock Experience API -palvelu rakentuu pääasiassa kahdesta eri XML-tiedostosta *poc-mock-experience-api.xml* ja *api-impl.xml*. *Poc-mock-experience-api.xml* on päätiedosto, joka ohjaa kutsun oikeaan päätepisteeseen rajapintapalvelun sisällä, ja *api-impl.xml* sisältää päätepisteet ja niiden logiikan.

Kuvassa 4 havainnollistetaan vanhan *poc-mock-experience-api.xml*-tiedoston rakennetta.



Kuva 4. Ote Anypoint Studio -työkalusta. Ote sisältää osan poc-mock-experience-api.xml-tiedostosta.

Kuvassa 4 *poc-mock-experience-api-main* flow'n sisällä listener-komponentti, joka vastaanottaa kutsun rajapintapalveluun. Tämän jälkeen APIkit-router-komponentti ohjaa kutsun oikeaan päätepisteeseen kutsun perusteella. Virhetilanteissa flow käyttää yrityksen omaa virhetilanteiden käsittelijää, jota tässä insinöörityössä ei esitellä. Kuvasta 4 löytyy myös kaksi flow'ta, joihin APIkit-router-komponentti viittaa. Kun kutsu on ohjattu oikeaan päätepisteeseen, alta löytyvät flow't ohjaavat ne oikeaan päätepisteeseen *api-impl.xml*-tiedostossa.

Poc-mock-experience-api-tiedostoon päätettiin lisätä mahdollisuus tilata virhetilanteita käyttämällä kuljetuskerroksia, vaikka tätä toiminnallisuutta ei pitänyt

alkuperäisessä suunnitelmassa toteuttaa. Syy tämän toiminnallisuuden rakentamiselle oli kuitenkin se, että kaikkiin päätepisteisiin ei voitu lisätä logiikkaa virhetilanteiden tilaamiselle. Virhetilanteiden tilaaminen päätepistekohtaisesti vaatii tässä toteutuksessa aina jonkin tunnisteen. Tällaisena tunnisteena esimerkiksi */customers*-päätepisteessä toimii henkilötunnus, kun taas */address*-päätepisteessä, joka palauttaa katuosoitteita tunnisteena, toimi katuosoite. Kuva 5 havainnollistaa uuden palvelun päätiedoston rakenteen ja toimintaa.

Kuva 5. Ote flow'sta, joka käsittelee kaikki rajapintapalvelun kutsut.

Kuvassa 5 on huomattavia eroja kuvan 4 vanhan palvelun toteutukseen verrattuna. Flow'n alussa on kaksi Set Variable -komponenttia, jotka tallentavat

muuttujia Mule-viestiin. Ensimmäinen tallentaa HTTP-metodin, jolla rajapinta-palvelua on kutsuttu muuttujaan nimellä *httpMethod*. Toinen tallentaa verkko-osoitteen muuttujaan nimeltä *urlPath*. Näitä käytetään HTTP request -komponenteissa myöhemmin.

Muuttujien tallentamisen jälkeen on Choice-reititin. Choice-reitittimeen voidaan lisätä kaksi tai useampia mahdollisia työnkulkuja. Choice-reititin käy reitit ylhäältä alas ja reitteihin on asetettu lausekkeita, ja reitti valitaan sen perusteella, joka palauttaa arvon tosi. Tässä Choice-reitittimessä tarkistetaan, onko kuljetuskerros MS2-Status-Code läsnä. Esimerkkikoodissa 8 on kyseinen lauseke.

```
attributes.headers.'MS2-Status-Code' != null
```

Esimerkkikoodi 8. Koodirivi oikean reitin valitsemiseksi.

Jos kuljetuskerros on läsnä, tallennetaan kuljetuskerroksen arvo muuttujaan nimeltä *errorHeader* ja tehdään kutsu Mocking Service -palveluun HTTP request -komponentilla. HTTP request -komponentti mahdollistaa ulkoisten HTTP-palveluiden kutsumisen Mule-sovelluksesta [28]. HTTP-request-komponentti sisältää konfiguraation, jossa verkko-osoite Mocking Service -palveluun rakennetaan esimerkkikoodin 9 mukaisesti. Esimerkkikoodissa 9 tärkein osuus on viittaus *vars.urlPath*-muuttujaan, joka asetettiin flow’n aikaisemmassa vaiheessa. Muut osat verkko-osoitteessa ovat viittauksia konfiguraatiotiedostoon, joka sisältää ohjeet Mocking Service -palvelun kutsumiseen ja muuta usein käytettävää tietoa palvelussa. HTTP request -komponenttiin on lisätty kuljetuskerros MS2-Status-Code, jonka arvo on *vars.errorHeader*, joka asetettiin aikaisemmin.

```
HTTPS://${mocking_service.host}${mocking_service.basePath#vars.url-
Path}
```

Esimerkkikoodi 9. Mocking Service -palvelun verkko-osoitteen rakennus HTTP request -komponentissa.

Kun HTTP Request -komponentti tekee MS2-Status-Code-kuljetuskerroksella pyynnön Mocking Service -palveluun, rajapinnasta palautuu virhetilanne.

Virhetilanne riippuu annetusta virhekoodista, mutta esimerkiksi jos asetetaan kuljetuskerroksen arvoksi 400 saadaan esimerkkikoodi 10:n mukainen vastaus, joka sisältää tietoa simuloidusta virhetilanteesta. Transform Message-komponentti flow'ssa on lisätty vain varmuuden vuoksi. Kun virhetilanne tapahtuu HTTP request -komponentissa, Mule kutsuu virheidenkäsittelijää ja Transform Message -komponentti ei aktivoidu. Jos jostain syystä Transform Message -komponentti aktivoituu, palauttaa se viestin, joka kertoo virhetilanteen simuloinnin epäonnistuneen. Kun virhetilannetta ei ole tilattu, APIkit-reititin ohjaa kutsun oikeaan päätepisteeseen ja päätepisteet löytyvät *api-impl-tiedostosta*.

```
{
    "appName": "API",
    "errorCode": "400",
    "errorMessage": "x-correlation-id not present",
    "correlationId": "dd960082-abbf-47c2-8c65-f18eeb35848b",
    "timeStamp": "2019-09-30T15:27:49.274Z"
}
```

Esimerkkikoodi 10. Esimerkki virhetilannevastauksesta.

6.5 Yksittäisen päätepisteen toteutus

Tässä luvussa esitellään Mock Experience API -palvelun */customers*-päätepiste. Kuva 6 havainnollistaa */customers* flow'n rakenteen. Flow'n alussa asetetaan kaksi muuttujaa ja asetetaan log-context muuttujaan. Nämä ovat yrityksen käytäntöjä, jotka eivät vaikuta palvelun toiminnallisuuteen. Tämän jälkeen Mule-viestistä haetaan parametri, joka viittaa henkilötunnukseen, ja tallennetaan se muuttujaan *ssnFile* "Set SSN" -nimisen komponentin avulla. Seuraavaksi tallennetaan uusi muuttuja *errorSsnList* "Error SSN list" -nimetyllä komponentilla, joka hakee määrittelytiedostosta listan henkilötunnuksia, jotka viittaavat johonkin virhekoodiin. Tämän jälkeen asetettiin Choice-reititin, joka tarkistaa onko henkilötunnus, jota on käytetty kutsussa, osa "Error SSN list" -muuttujia. Jos kyseinen henkilötunnus on määritelty virhetilanteen tilaamiseen, ohjataan virhetilanteiden simulointi flow'lle. Virhetilanteiden tilaamisesta lisää luvun 6.5 lopussa.

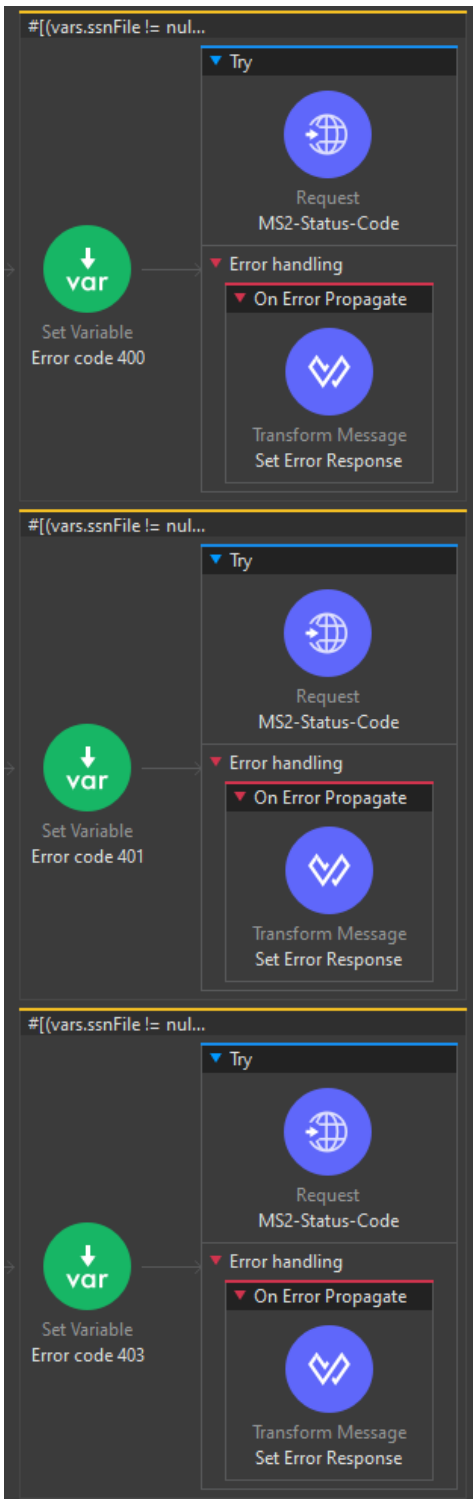
Kuva 6. */customers* flow'n visuaalinen rakenne Anypoint Studio -työkalussa

Kun jatketaan samassa flow'ssa, seuraavaksi löytyy Until Successful -osio. Until Successful -osioon on mahdollista lisätä uudelleenyritystoiminnallisuutta. Kyseinen osio ottaa kaksi eri muuttujaa, jotka ovat yritysten määrä ja aikaväli uudelleenyrittämisten välillä. Kaikki flow't tässä palvelussa käyttävät kolmea yritystä, ja aikaväli yritysten välillä on 6 sekuntia. Syy tämän toiminnallisuuden lisäämiselle oli kuormitustestauksessa löydetty mahdollinen ongelma 503-virhetilanteiden varalle. Kun 503-virhetilanne tapahtuu Mocking Service -palvelussa, yrittää Until Successful -osio kutsua Mocking Service -palvelua vielä kolmesti.

Seuraavaksi flow'sta löytyy Try-osio. Try-osion tarkoitus on mahdollistaa toiminnan jatkaminen, jos HTTP request -komponentti palauttaa virheen. Jos molemmat HTTP request -komponentit Try-osion sisällä palauttavat virhetilanteen, aktivoituu Until Successful -osio. "MS2-Example"-niminen HTTP request -komponentti kutsuu Mocking Service -palvelua ja lisää kutsun MS2-Example-kuljetuskerrokseen aiemmin tallennetun henkilötunnuksen muuttujasta *ssnFile*. Jos kyseisellä henkilötunnuksella ei ole määritelty esimerkkidataa, RAML-tiedostoon palautuu virhetilanne. Virhetilanteen sattuessa Try-osio jatkaa omaan virhetilanteidenkäsittelijään. Experience Mock API -palvelussa oli suunniteltu niin, että virhetilanteen sattuessa tehdään uusi kutsu Mocking Service -palveluun, mutta MS2-Example-kuljetuskerrokseen lisätäänkin arvo "default". Tällöin Mocking Service -palvelu palauttaa aina saman vastauksen, koska RAML-tiedostossa kaikille päätepisteille on lisätty "default"-niminen esimerkkidata. Tämä

toiminnallisuus täyttää palvelulle asetetun vaatimuksen oletusdatan palautuksesta, koska aina palautetaan oletusvastaus paitsi, jos Mocking Service -palvelussa on häiriö. Lopuksi Transform Message -komponentti varmistaa vielä, että vastaus palautetaan JSON-muodossa ja lopullinen vastaus vielä lisätään myös lokiin. Flow’n loppuessa Mule-sovellus palauttaa vastauksen kutsuneeseen järjestelmään.

Päätepistekohtaisten virhetilanteiden tilaamista varten rakennettiin flow, joka sisälsi vain yhden Choice-reititimen. Choice-reititin sisältää mahdollisuuden simuloida 15 eri virhetilannetta. Kuva 7 havainnollistaa sen rakennetta ja sisältää kolme ensimmäistä reittiä. Reitti, jota kutsu kulkee, määräytyy päätepisteen tunnisteen perusteella. Jos tunnisteena toimii henkilötunnus, verrataan sitä määriteltyyn listaan henkilötunnuksia ja Choice-reititin ohjaa kutsun oikeaan reittiin. Tämän jälkeen tehdään kutsu Mocking service -palveluun käyttämällä kuljetuskerrosta MS2-Status-Code, jonka arvo asetetaan jokaisella reitillä erikseen. Tämän jälkeen Mocking service -palvelu palauttaa virhetilanteen ja virhe palautetaan Mock Experience API -palvelua kutsuneeseen järjestelmään. Esimerkiksi jos kutsuu palvelua henkilötunnuksella 090999-969V, palauttaa palvelu virhekoodin 400 ja siihen liittyvän virheviestin, joka on esimerkkikoodissa 10 (luku 6.4).



Kuva 7. Visuaalinen ote Choice-reitittimestä, joka ohjaa virhetilanteen oikeaan reittiin.

6.6 Palvelun testaus

Rajapintapalveluiden toimivuuden varmistaminen on olennainen osa kehitystyötä. Experience Mock API -palvelun testattiin pääasiallisesti käyttämällä Postman-työkalua lokaalissa kehitysympäristössä. Palvelussa päätettiin myös, ettei yrityksen normaalikäytäntöjä testaukseen käytettäisi, koska kyseessä on yrityksen sisäinen palvelu, joka ei koskaan tule tuotantoympäristöön. Tämän takia palvelun testaus oli vähäistä verrattuna muuhun normaaliin kehitystyöhön. Viimeinen testaus tehtiin kehitysympäristössä, jossa palvelu otettiin lopulta käyttöön.

Experience Mock API -palvelu käynnistettiin aina testauksen ajaksi lokaalisti Anypoint Studio -työkalua käyttäen. Postman-työkalussa luotiin jokaiselle päätepisteelle oma kutsu, jota voitiin käyttää päätepisteiden testaamiseen. Jokaiselle päätepisteelle määriteltiin tarvittavat kuljetuskerrokset. POST- ja PUT-metodilla toimiviin kutsuihin lisätiin myös pyynnön runko, joka oli JSON-muodossa. Jokainen päätepiste testattiin aina päätepisteen kehityksen jälkeen lähettämällä päätepisteeseen kutsuja ja tarkistettiin, tuliko virhetilanteita. Kaikki palvelun päätepisteet käyttävät suurin piirtein samaa rakennetta, joten virhetilanteita testauksessa ei juurikaan ilmennyt.

7 Yhteenveto

Insinöörityössä tutkittiin rajapintapalveluiden virtualisointia ja sen mahdollisia hyötyjä. Työssä myös perehdyttiin MuleSoft-alustaan ja siihen, miten sitä voitaisiin hyödyntää rajapintapalveluiden virtualisointiin. Todettiin, että MuleSoft-alusta tarjoaa paljon hyviä työkaluja ja käytäntöjä kaikkeen rajapintapalveluihin liittyvään kehittämiseen ja rajapintapalveluiden käytön hallintaan. MuleSoft-alusta myös sisältää paljon muita työkaluja, jotka eivät olleet oleellisia tämän työn osalta, mutta joihin kannattaa paneutua, jos aikomuksena on siirtyä MuleSoft-alustan käyttöön rajapintapalveluiden osalta.

Experience Mock API -palvelun tarkoitus on nopeuttaa uuden hankkeen kehitys- ja testausprosesseja. Palvelu on mahdollistanut uusia tapoja kehittää ja testata uuden järjestelmähankkeen järjestelmää esimerkiksi virhetilanteiden simuloimisen osalta. Palvelun kehitysprosessissa opittiin paljon varsinkin MuleSoft-alustan Mocking Service -palvelusta ja opittiin hyödyntämään MuleSoft-alustan tarjoamia ominaisuuksia niiden täydellä kyvykkyydellä. Tulevaisuudessa on tarkoitus lisätä Experience Mock API -palveluun lisää päätepisteitä sitä mukaan, kun niille ilmenee tarvetta myös järjestelmähankkeen ulkopuolella. Palvelun kehitys toi esille myös rajapintapalveluiden virtualisoinnin hyötyjä yritykselle.

Tulevaisuudessa yrityksessä otetaan mahdollisesti virtualisoidut rajapintapalvelut mukaan kaikkeen rajapintapalveluita koskevaan kehitykseen. Palvelulle on löytynyt käyttöä myös normaalin kehitys- ja testaustyön ulkopuolella. Palvelua on esimerkiksi käytetty myös tuotantoympäristön toiminnallisuuksien varmistamiseen, joka ei ollut palvelun alkuperäinen käyttötarkoitus. Virtualisoidut rajapintapalvelut ovat osoittautuneet hyödyllisiksi myös ennalta suunnittelemattomissa tilanteissa. Siksi on suositeltavaa harkita niiden käyttöönottoa aina, kun teknologiahankkeissa tarvitaan rajapintaratkaisuja.

Lähteet

1 De, Brajesh. 2023. API Management: An Architect's Guide to Developing and Managing APIs for Your Organization. E-kirja. Apress.

2 What is API mocking? Verkkoaineisto. Postman blog. <https://blog.postman.com/what-is-api-mocking>. Luettu 6.10.2024.

3 What is API Mocking? Definition, Guide, and Best Practices. Verkkoaineisto. Katalon. <https://katalon.com/resources-center/blog/what-is-api-mockings>. Luettu 6.10.2024.

4 Munday, Jess. What is MuleSoft? Verkkoaineisto. <https://www.salesforce.com/mulesoft/what-is-mulesoft/>. Luettu 23.9.2024.

5 Mason, Ross. Ross Mason. Verkkoaineisto. MuleSoft Blog. <https://blogs.mulesoft.com/author/rossmason/>. Luettu 23.9.2024.

6 What is Salesforce? Verkkoaineisto. Salesforce. <https://www.salesforce.com/products/what-is-salesforce/>. Luettu 11.3.2025

7 Choudhary Prashant. What Is API-led Connectivity? The 360 Blog. Verkkoaineisto. <https://www.salesforce.com/blog/api-led-connectivity/>. Luettu 23.9.2024.

8 Frye, Ma-Keba. What Is a Center for Enablement (C4E)? Verkkoaineisto. The 360 Blog. <https://www.salesforce.com/blog/what-is-a-center-for-enablement/>. Luettu 20.2.2025.

9 A powerful and innovative feature set for APIs and integrations. Verkkoaineisto. <https://www.mulesoft.com/platform/anypoint-platform-features>. Luettu 20.2.2025.

10 About Design Center. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/design-center/>. Luettu 2.3.2025.

11 Anypoint Exchange Overview. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/exchange/>. Luettu 2.3.2025.

12 Anypoint Studio. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/studio/latest/:~:text=Anypoint%20Studio%20is%20MuleSoft%E2%80%99s%20Eclipse-based%20integration%20development%20environment,video%20to%20see%20a%20quick%20 overview%20of%20Studio>. Luettu 5.3.2025.

13 Holzner, Steven. 2004. Eclipse. E-kirja. O'Reilly Media.

14 CloudHub Overview. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/cloudhub/>. Luettu 5.3.2025.

15 Anypoint API Manager. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/api-manager/latest/latest-overview-concept>. Luettu 7.3.2025.

16 Mule Application Development. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/mule-runtime/latest/mule-app-dev>. Luettu 7.3.2025.

17 DataWeave Overview. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/dataweave/latest/>. Luettu 7.3.2025.

18 Mule Events. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/mule-runtime/latest/about-mule-event#:~:text=A%20Mule%20event%20contains%20the%20core%20information%20processed,results%20in%20the%20creation%20of%20a%20new%20instance>. Luettu 15.3.2025.

19 Mule Message Structure. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/mule-runtime/latest/about-mule-message>. Luettu 31.3.2025.

20 Flows and Subflows. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/mule-runtime/latest/about-flows>. Luettu 15.3.2025.

21 Error Handlers. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/mule-runtime/latest/error-handling>. Luettu 15.3.2025.

22 Simulate API Calls with the Mocking Service. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/design-center/design-mocking-service>. Luettu 6.10.2024.

23 Guide to Defining Examples in RAML 1.0. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/design-center/design-named-examples>. Luettu 6.10.2024.

24 Add Behavioral Headers to Simulated API Calls. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/design-center/apid-behavioral-headers>. Luettu 6.10.2024.

25 Buckets overview. Verkkoaineisto. Amazon AWS Docs. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Using-Bucket.html>. Luettu 20.3.2025.

26 Postman documentation overview. Verkkoaineisto. Postman documentation overview. <https://learning.postman.com/docs/introduction/overview/>. Luettu 20.3.2025.

27 Test your API using the Collection Runner. Verkkoaineisto. Test your API using the Collection Runner. <https://learning.postman.com/docs/collections/running-collections/intro-to-collection-runs/>. Luettu 20.3.2025.

28 Configure HTTP Request Operation. Verkkoaineisto. MuleSoft Docs. <https://docs.mulesoft.com/http-connector/latest/http-request-ref>. Luettu 22.3.2025.