Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Vili-Arttu Ahlgrén

XML-dokumenttien vastaanotto- ja lähetyspalvelu Salesforcessa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto -ja Viestintätekniikka

Insinöörityö

14.3.2019

Tekijä Otsikko Sivumäärä Aika	Vili-Arttu Ahlgrén XML-dokumenttien vastaanotto- ja lähetyspalvelu Salesforcessa 47 sivua 14.3.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto -ja Viestintätekniikan koulutusohjelma
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Juha Kämäri Lehtori Simo Silander

Tämä insinöörityö käsittelee Fluido Oy:n asiakkaan sisäiseen käyttöön tulevaa XML-dokumenttien vastaanotto- ja lähetyspalvelua REST-rajapinnassa. Ohjelman tarkoitus on kyetä lukemaan dokumenttien sisältö ja tuottaa niistä selkeästi luettavissa oleva kokonaisuus erikseen mukautetulle käyttöliittymälle Salesforce-alustalla. Dokumentteihin lähetetään vastaus asiakkaan toimesta. Dokumentin vastaanottaja pitää voida valita tapauskohtaisesti.

Aihe valittiin sen tarjoaman monipuolisuuden takia. Sen suunnittelussa kävi ilmi, että se on tarkoitus tuottaa yhden ohjelmoijan voimin, ja se tulee kattamaan laajasti datamallin suunnittelua, tietokannan hallintaa, käyttöliittymän rakentamista ja kompleksien koodiluokkien tuottamista. Lisäksi se tarjoaa mahdollisuuden tutustua moniin Salesforce-alustan ominaisuuksiin, joihin ei ollut aiempaa kokemusta.

Asiakkaalla on ollut entuudestaan käytössään sisäisesti rakennettu ohjelma, joka lukee ja lähettää viestejä. Asiakkaan siirtyessä Salesforcen käyttöön he haluavat myös siirtää kyseisen integraation kulkemaan Salesforcen kautta. Lisäksi halutaan rakentaa huomattavasti paranneltu käyttöliittymä asiakkaan asiakaspalvelijoiden käyttöön. Tämä toteutetaan Salesforcen Lightning Component -käyttöliittymäkehystä hyväksi käyttäen.

Insinöörityössä käsitellään, kuinka Salesforce-alustalla luetaan ja kirjoitetaan XML-viestejä, kuinka niitä lähetetään ja vastaanotetaan REST-rajapinnassa sekä Lightning Component -käyttöliittymäkehyksen käyttöä. Lisäksi sivutaan tiedon käsittelyä sekä perustoimintojen käyttöä Salesforce-alustalla.

Avainsanat	CRM, Pilvipalvelut, Salesforce, XML, APEX, REST

Author	Vili-Arttu Ahlgrén
Title	XML reading and writing message -service in Salesforce
Number of Pages	47 pages
Date	14 March 2019
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Juha Kämäri, Senior Lecturer Simo Silander, Senior Lecturer

This thesis is about a Fluido customer case with internal XML-document receiving and sending service program in a REST-interface. The purpose of this program is to read incoming documents and read them into an easily manageable custom user interface on Salesforce platform. A response to incoming message is created by the customer. The recipient of outbound message must be enabled for choosing in case-to-case basis.

This subject was chosen for its diversity. The program was planned to be built by one developer and during design it become clear that it entails broad scale of data model planning, database control, user interface building and producing complex code classes. Additionally, it offers an opportunity to dive into yet unknown parts of the Salesforce-platform.

The customer had an internal software for message handling, but when moving it's CRM business handling to Salesforce this functionality was to be integrated along with it. Additionally, the customer wants a well upgraded user interface for its representatives. This is produced by using Salesforce Lightning Component Framework.

In this thesis I will go into detail how to read and write XML-files in Salesforce, how to receive messages into a REST Api, how to send them to a given receiver and details on how user interface was created by using Lighting Component Framework. In addition, basic Salesforce functionality and data structure is explained.

Keywords	CRM, Cloud Services, Salesforce, XML, APEX, REST

Sisällysluettelo

Lyhenteet

API	Application Programming Interface. Ohjelmistorajapinta, jonka avulla ohjelma/ohjelmisto voi keskustella muiden ohjelmien kanssa.
CRM	Customer Relationship Management, asiakkuudenhallintajärjestelmäohjelmisto, jossa säilytetään ja hallitaan tietoja asiakkaista.
DML	Data Manipulation Language, Salesforcen tiedon tallennus- ja muokkauskieli.
HTML	HyperText Markup Language, webohjelmoinnissa yleisesti käytetty kieli.
SFDC	Salesforce.com, yleisesti käytetty nimitys Salesforce-alustasta.
SOQL	Salesforce Object Query Language, Salesforcen oma SQL-tyyppinen kieli tiedonhaulle tietokannasta.
XML	Extensible Markup Language, merkintäkielen standardi, jolla kuvataan tietoa ja/tai sen rakennetta.
HTTP	Hypertext Transfer Protocol, selaimissa ja WWW-palvelimissa käytetty tiedonsiirto protokolla.
APEX	Salesforcen käyttämä Java-pohjainen ohjelmointikieli.
REST	Representational State Transfer, HTTP-Protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen.

1 Johdanto

Tässä työssä esitellään Salesforce-pilvipalveluun kehitetyn XML-viestien vastaanotto -ja lähetyspalvelun toteuttaminen REST-rajapinnassa Fluido Oy:n asiakkaalle. Salesforce on CRM-tuote, eli asiakkuudenhallintajärjestelmä. Pilvipohjaisena palveluna Salesforce mahdollistaa venyvän kehitysalustan ja on iso valttikortti, kun ohjelmistoon halutaan tehdä muutoksia. Työ on erittäin ohjelmointipainotteinen ja esittelee runsaasti esimerkkikoodeja tuotetusta ohjelmasta.

Asiakkaalla oli olemassa oma viestien luku- ja kirjoitusjärjestelmä. Asiakkaan siirtyessä Salesforcen käyttöön asiakashallintansa osalta myös kyseinen palvelu haluttiin siirtää kulkemaan Salesforcen kautta. Samalla haluttiin toteuttaa huomattavia parannuksia asiakaspalvelijoiden käyttöliittymään.

Tämä työ esittelee, kuinka kyseinen palvelu voidaan toteuttaa Salesforcella. Työssä perehdytään REST-arkkitehtuuriin, XML-viestirakenteen lukemiseen ja kirjoittamiseen, Salesforcen tietokantakutsuihin (SOQL), Salesforce Lightning Component -käyttöliittymäkehykseen sekä Salesforcen tarjoamiin valmiisiin ominaisuuksiin ja niiden konfigurointiin. Lisäksi käsitellään, miten mahdollisia virhetilanteita hallitaan järjestelmän sisällä. Projektissa testaus toteutettiin käyttäjälähtöisesti, ja kooditestaus tuotettiin Salesforcen asettamien minimivaatimusten mukaan. Täten tässä työssä ei perehdytä kooditestauksen esittelyyn syvällisellä tasolla.

Projektin haaste on REST-rajapintaan tulevan XML-viestin koko ja rakenne. Viesti sisältää henkilödataa, joka on luettu siihen käsin kirjoitetuilta kentiltä käyttöliittymältä. Tämä tarkoittaa, että viestien sisällöt saattavat vaihdella hyvin paljon toistensa välillä. Tämä oli otettava tarkasti huomioon viestien vastauksia lähetettäessä, sillä tietty data luettiin lähtevään viestiin suoraan sisään tulleesta viestistä, ja sen pitää lähtiessään olla validia, jotta jälleen sen vastaanottaja kykenee viestin lukemiseen.

2 Salesforce

Tämä työ on rakennettu kokonaan pilvipalvelu Salesforcen päälle. Salesforce on asiakkuudenhallintajärjestelmä (CRM, Customer Relationship Management), jonka takana on vuonna 1999 perustettu Salesforce.com-yhtiö. Se on maailman suosituin CRM-alusta ja vuonna 2018 sen liikevaihto olikin 10,84 miljardia Yhdysvaltain dollaria [1].

Salesforcen käyttö voidaan skaalata yritysten vaatimusten mukaan. Yritykset voivat hallita järjestelmässä haluamaansa tietoa esimerkiksi tuotteistaan, markkinoinnista, asiakkaistaan, asiakaspalvelusta tai itse mukautetusta tiedosta. Se tarjoaa eri tarkoituksiin keskittyneitä, mutta ei vain niihin rajoittuneita pilvipalveluita, kuten Marketing Cloudin markkinointiin tai Sales Cloudin myyntiin. Alusta toimi aluksi vain PaaS (Platform as a Service) -periaatteella, eli Salesforce vastaa alustan ohjelmistosta ja sen toiminnan kannalta välttämättömistä laitteista, mutta on sittemmin siirtynyt täysmittaiseksi SaaS (Software as a Service) -palveluksi. Ohjelmistokehitys Salesforcelle on mahdollista Force.com-alustan avulla, ja se vaatii vain internetyhteyden.

Salesforce käyttää ohjelmoinnissa useita omia ohjelmointikieliään, jotka perustuvat muihin ohjelmointikieliin, mutta on räätälöity vastaamaan paremmin Salesforcen tarkoituksia. Salesforcen eri instansseja käytetään kehitykseen, testaukseen ja itse tuotteen varsinaiseen käyttöön. Kehitys- ja testausympäristönä käytetään termejä hiekkalaatikko (engl. Sandbox) ja itse varsinaisesta käytössä olevasta tuotteesta termiä tuotanto (engl. Production). Näistä instansseista puhutaan organisaatioina. Salesforce.com lyhennetään muotoon SFDC.

2.1 Objektit

Salesforcen datamalli perustuu objekteihin (sObject – Salesforce Object), jotka sisältävät tarpeelliset tiedot halutusta datasta. Jokaista objektia käytetään tietyn tyyppisen tiedon säilyttämiseen. Ilmentymää objektista kutsutaan tietueeksi. [2.]

Salesforce tarjoaa useita vakio-objekteja. Näitä objekteja ovat esimerkiksi Tili (engl. Account) ja Tapaus (engl. Case). Kaikilla objekteilla on tiettyjä vakiokenttiä, kuten nimi, luomisaika ja viimeksi objektia muokanneen henkilön nimi. Vakiokenttiä ei voi objekteilta

poistaa, mutta uusia mukautettuja kenttiä voidaan lisätä, muokata ja poistaa tarpeen mukaan. Vakio-objekteja ei voida poistaa.

Vakio-objektien lisäksi voidaan luoda omia mukautettuja objekteja vastaamaan järjestelmän datamallin vaatimuksia. Luonnin jälkeen mukautetut objektit toimivat Salesforcen sisällä kuten vakio-objektit. Mukautetut objektit saavat samat vakiokentät kuin vakio-objektitkin, eikä niitä voi poistaa objektilta. Mukautetut objektit voidaan tarpeen mukaan poistaa. Mukautettuja objekteja voi luoda rajallisen määrän riippuen organisaation Salesforce-versiosta, ja tämä on otettava huomioon suunnitellessa tietokannan rakennetta.

Kaikille objekteille voidaan antaa useita käyttötarkoituksia erittelemällä ne toisistaan objektikohtaisilla tietuetyypeillä (engl. Recordtype). Esimerkiksi tileille voidaan määritellä tietuetyypiksi henkilötili tai yhtiötili. Tietuetyypillä voidaan määritellä muun muassa, mitkä valikkokenttien arvot kyseisellä tyypillä on valittavissa.

Objekteilla on näkyvän nimen lisäksi oma rajapinta-nimi (engl. API nami), jota on käytettävä viitatessa objektiin koodin puolella. Mukautettujen objektien nimen perään lisätään automaattisesti ”__c” -pääte. Mukautettua objektia luodessa on parhaan käytännön mukaista asettaa rajapintanimi mahdollisimman lähelle objektin käyttöliittymänimeä. Malli työhön liittyvän mukautetun objektin luomisesta esitetään luvussa 5.2.3. [2.]

2.2 Kentät

Objekteille voidaan lisätä valmiiden kenttien lisäksi mukautettuja kenttiä. Näitä voivat olla muun muassa tekstikentät, numerokentät, valikkokentät, kaavakentät tai viitteet toisiin objekteihin. Mukautetut kentät seuraavat samaa nimeämistyyliä kuin mukautetut objektit. Ne saavat erillisen rajapintanimen, jolla sitä voi kutsua tai muokata koodissa. Myös kenttien rajapintanimi on aina ”__c” -loppuinen.

2.3 Chatter

Chatter on Salesforcen tarjoama ajantasainen yhteistyösovellus. Käyttäjät voivat sen kautta keskustella toisilleen, ryhmissä tai projektien hallinnassa. Jokaisella näistä on

oma syöte-kenttä, johon käyttäjät voivat jättää kommentteja. Objekteille voidaan asettaa Chatter-syöte aktiiviseksi, jolloin käyttäjät voivat kommentoida yksittäisen tietueen syötteeseen. [3.]

Chatter käyttää vastaavia merkintöjä kuin monet muut keskustelupalvelut, kuten hymiöitä, @-merkintää, hyperlinkkejä ja kuvien latausta. Syötteisiin voi suoraan kommentoida, tykätä tai jakaa. Chatter-viestejä voidaan luoda Apex-luokkien kautta käyttämällä Salesforcen tarjoamaa ConnectApi-luokkaa. Näin organisaatiossa voidaan luoda täysin mukautettuja Chatter-toimintoja. [4]

3 Ohjelmointi Salesforce-ympäristössä

3.1 Apex

Apex on Salesforcen sisäinen ohjelmointikieli, joka pohjautuu pitkälle Javaan, ja täten onkin vahvasti tyypitetty olio-ohjelmointiin perustuva kieli. Apexilla kuitenkin tuotetaan ohjelmistoa, joka toimii vain Salesforce.com-alustan palvelimilla. Se mahdollistaa halutun logiikan lisäämisen järjestelmän tapahtumiin, esimerkiksi tietokantakutsuihin napin painalluksesta. Sillä voidaan myös kutsua objektikohtaisia laukaisimia (engl. Trigger). [5.] Laukaisimille määritellään, laukeavatko ne ennen tai jälkeen, kun tietuetta luodaan, poistetaan tai päivitetään. Määrittelyn mukaan kutsutaan Apex-luokkaa, joka toteuttaa ohjelmoidut operaatiot. Apexilla voidaan luoda myös REST- ja SOAP-rajapintoja, joihin voidaan yhdistää ulkoisten palveluntarjoajien kutsuja ja viestejä. [6.] REST ja SOAP kutsuista on lisää luvussa 4.

3.1.1 Syntaksi

Koska Apex on Java-pohjainen ohjelmointikieli, sen syntaksi on myös hyvin samantyyppinen. Apexissa käytetään hyväksi primitiivisiä datatyyppejä esimerkiksi String, Boolean, Integer, listoja, hajautustauluja ja mukautettuja luokkia sekä järjestelmän valmiita luokkia.

Suurin erottava tekijä Apexin ja Javan välillä on Apexissa mukana kulkevat Salesforcen objektit. Alla on esimerkki Apex-syntaksista, muuttujien määrittelystä ja objektien kutsumisesta.

```
// Objekti muuttujia
Account account = new Account();
CustomObject__c = new CustomObject();

//Muita muuttujia
String str = 'Demo String';
Integer I = 1;
Boolean example = true;

// Kokoelmia
List<String> stringList = new List<String>();
Map<String, Integer> stringToIntMap = new Map<String, Integer>();

// Käyttäjän määrittelemä luokka
ExampleClass eClass = new ExampleClass();

// Järjestelmän määrittelemiä luokkia
HttpRequest req = new HttpRequest();
Date dt = system.Today();
```

Esimerkkikoodi 1. Muuttujien pohjustus Apexilla

3.1.2 Apex-koodin testaus

Salesforce ei suoranaisesti vaadi yksikkötestejä sisältäviä testiluokkia erikseen jokaiselle Apex-luokalle. Sen sijaan tuotanto ympäristössä Salesforce vaatii 75 % koodin testikattavuuden yhteensä kaikista luokista. Näin ollen yhden luokan testikattavuus voi olla esimerkiksi 100 % ja toisen 50 %, jotta luokat voidaan viedä tuotanto-ympäristöön. On kuitenkin parhaan käytännön toimintamallin mukaista tuottaa kattavat testiluokat jokaiselle ohjelmoidulle luokalle. [7.] Alla on kaksi esimerkkiä tavoista kirjoittaa testiluokka.

```
// Luokan alkuun lisätään annotaatio @isTest
@isTest
private class testiLuokka{

      // Testiluokalle voidaan antaa pohjustus
      @testSetup
      static void testSetup(){
      Integer testInteger = 5;
      }

      // Testimetodin kirjoitustapa @isTest
      @isTest
      Static void testFunction(){

            System.assertEquals(5, testInteger);
```

```
    }
    // Testmetodin kirjoitustapa testMethod
    Static testMethod void testMethod(){
    System.assertEquals(5, testInteger);
    }
}
```

Esimerkkikoodi 2. Apex-testiluokka

Testimetodeilla ei kuitenkaan suoranaisesti voi testata ulkoisia Web service -kutsuja. Niitä varten luodaan erillinen HTTP-kutsun jäljitelmä (engl. Mock), jolla emuloidaan ulkoiseen palvelimeen lähetetyn kutsun vastausta. [8.]

3.2 XML-tiedoston lukeminen ja kirjoitus Apexilla

Salesforce tarjoaa järjestelmän määrittelemät luokat XML-tiedoston lukemiseen ja kirjoittamiseen.

3.2.1 Document-luokka

Rajapintaan saapunut XML-viesti on primitiivisessä String-merkkijonon muodossa. Document-luokan load(xml)-metodilla luetaan viesti rakenteelliseen XML-muotoon. Tämän jälkeen siitä voidaan ottaa sen ensimmäinen elementti getRootElement-metodilla, jonka jälkeen sitä voidaan alkaa parsia XmlNode-luokan avulla. Vaihtoehtoisesti voitaisiin käyttää myös XmlStreamReader-luokkaa. [9.]

Kirjoittaessa uutta ulospäin lähtevää viestiä käytetään hyväksi XmlStreamWriter-luokkaa. Tämän luokan metodeilla kirjoitetaan viesti XML-muodossa. Viestiä lähettäessä se muutetaan Document-luokan metodilla ”toXmlString” takaisin String-merkkijonoksi, jossa se lähetetään vastaanottavalle osapuolelle.

3.2.2 XmlNode-luokka

XmlNode-luokalla voidaan käsitellä XML-viestiä yksi solmu (engl. Node) kerrallaan. Solmusta voidaan tutkia muun muassa sen attribuutteja, nimeä, sen ylä- ja alatason solmuja, nimiavaruutta ja muita elementtejä. Näin jokainen solmu yksi kerrallaan läpi parsien voidaan XML-viestistä tutkia sen sisältö. [10.]

3.2.3 XmlStreamWriter-luokka

Kyseistä luokkaa käytetään XML-viestien kirjoittamiseen. Sen avulla voidaan kirjoittaa tarkka XML-rakenne, jossa rakenteeseen määritetään tarvittavat attribuutit, ylä- ja alatason solmut, nimiavaruus ja tekstiosuus. [11.]

3.3 Salesforce Object Query Language SOQL

SOQL on Salesforcen sisäinen tietokannan kyselykieli. Sitä käytetään tiedon hakemiseen Salesforce-palvelimelta Apex-luokilla. Se osaa sijoittaa palautetun tiedon esiteltyyn muuttujaan sopivaksi.

SOQL on SQL (Structured Query Language) -tyyppinen kyselykieli, jonka syntaksi jäljittelee sen toimintaa. Salesforcen sisäisen objektirakenteen takia sen kyselyissä ei käytetä JOIN-operaatioita. Sen sijaan Salesforcen tietokannassa viittaukset toisiin objekteihin määritellään hakusuhde (engl. Lookup Field) -kenttien avulla. Esimerkiksi mukautetulle objektille voidaan luoda suhde tiliin, jolloin tämän objektin tietueilla on määritelty siihen liittyvä tili. Mikäli objektilla on suhde toiseen objektiin, sitä voidaan hyväksikäyttää tietokanta hakua tehdessä. Hakusuhteella hakiessa tietokanta palauttaa vain objektin tunnuksen. Mikäli halutaan kenttien arvoja, käytetään haussa relaatioviittausta “__r” ja kentän nimeä esimerkkikoodi 3:n osoittamalla tavalla. [12.]

```
CustomObject cObject = [SELECT Id, Name, Account__r.Name, Account__r.Exam-
ple_Field__c FROM CustomObject WHERE Name = ‘test Name’];
```

Esimerkkikoodi 3. SOQL-tietokantakysely, jossa mukautetulta objektilta haetaan omien kenttiensä lisäksi siihen liitetyn tilin tietoa.

SOQL-kielen syntaksi seuraa aina samaa kaavaa. Se alkaa SELECT-komennolla, joka kertoo, mitä tietoa aiotaan hakea. Tätä seuraa kenttien nimet, joiden tietoa halutaan käsitellä. Sen jälkeen määritellään, mistä tieto haetaan merkitsemällä FROM-komento, jonka perään lisätään objekti, josta tietoa halutaan hakea. Tämän rungon ympärille rakentuvat kaikki SOQL -kutsut, joita voi kuitenkin tarkentaa useilla komennoilla. [12.]

Hakujen suodatusta hoidetaan WHERE-komennolla. Tällä voidaan määritellä, millaisista objekteista halutaan saada tietoa. Esimerkkikoodissa 3 haetaan tietoa vain objektilta CustomObject, jonka Name-kentän arvo on ’test Name’. Mikäli tietokannasta löytyy useita tietueita, joiden haetun kentän arvo on sama, täytyy ne tallentaa listaan. Haettaessa vain tiettyä tietuetta täytyy hakuehdot asettaa siten, että ne kohdistuvat vain haluttuun tietueeseen. Tämä olisi jokin uniikki arvo, kuten tietueen tunniste, eli Id-arvo. Alla muutama esimerkki erilaisista SOQL-kyselyistä. [12.]

```
SELECT Id, Name, exampleField__c FROM Account
SELECT Id, Name FROM Account WHERE Name = ‘Test Name’
List <CustomObject> objects = [SELECT Id, Name, Street__c FROM CustomObject
WHERE Street__c = ‘Test Street’];
SELECT testField__c FROM CustomObject WHERE Id IN: objects
```

Esimerkkikoodi 4. SOQL-kyselyitä.

SOQL-kielestä puuttuu villikortti-haku, mikä tarkoittaa, että useiden kenttien tietoa hakiessa kutsun lause saattaa muodostua hyvinkin pitkäksi. Tällöin koodin asiallinen rivitys on tärkeää sen luettavuuden kannalta. [12.]

3.4 Data Manipulation Language DML

Siinä missä SOQL-kieltä käytetään tiedon hakemiseen ja lukemiseen, DML-kieltä käytetään tiedon kirjoittamiseen. Tietueita tallennetaan yksitellen tai ryhmissä. Salesforce on rajoittanut yksittäisten suorituskohtaisen DML-operaatioiden määrän 150 kappaleeseen, mutta tietueen transaktioita voidaan suorittaa 10 000. Näin Salesforce suosittelee, että tietueita päivitetään aina joukoissa, mikäli mahdollista. [13.]

DML-kutsuja tehdään insert-, update-, upsert- tai delete-komennoilla. Insert-komennolla lisätään tietokantaan uusi tietue, update-komennolla päivitetään jo olemassa olevaa

tietuetta, delete-komennolla poistetaan olemassa oleva tietue. Upsert-komento on insert- ja update-komennon yhdistelmä, jolla voi joko luoda uuden tai päivittää olemassa olevan tietueen. Käytettäessä upsert-komentoa ohjelma tarkistaa tietueen uniikin tunnisteen kautta, onko kyseessä päivitys tai uuden tietueen luonti. Mikäli uniikkia tunnistetta ei löydy, luodaan uusi tietue. Mikäli se löytyy, päivitetään haluttuihin kenttiin asetetut arvot.

```
Account account = new Account(Name = 'Test Account');
Insert account;
account.Name = 'Name change';
SteetNumber__c = 12;
update account;
```

Esimerkkikoodi 5. DML-kutsujen käyttö.

Tietueita tallentaessa on otettava huomioon, mitä tietoa ollaan tallentamassa. Esimerkkikoodissa 5 tallennetaan kenttään StreetNumber__c uusi arvo, jonka oletetaan olevan numeroarvo. Mikäli kenttä odottaa merkkijonoa arvokseen, kyseinen objektin päivitys keskeytyy, ja se aiheuttaa virheen. Lisäksi objektilla saattaa olla asetettu mukautettuja vahvistussääntöjä, jotka estävät tiettyjen kenttien muokkauksen.

3.5 Lightning-käyttöliittymäkehikko

Lightning-komponentteja kehitetään käyttämällä Lightning Component Framework -käyttöliittymäkehikkoa. Komponentit edustavat itsenäisiä kokonaisuuksia, jotka ovat uudelleenkäytettävissä Salesforcen Lightning Experience -näkymän Lightning -sivuilla (engl Lightning Page). Käyttöliittymäkehikko sisältää valmiita peruskomponentteja ja mahdollistaa täysin mukautettujen applikaatioiden ja komponenttien luonnin. [14.]

3.5.1 Lightning-komponentit

Itsenäisinä kokonaisuuksia komponentit voivat olla yksinkertaisia painikkeita, tai ne voivat koostaa suuria sovelluksia. Komponentit koostuvat HTML-pohjaisesta näkymästä, CSS-tyylimääritelmästä, JavaScript-ohjaimesta ja JavaScript-apuluokasta. Lightning-komponentit perustuvat Aura-kehykseen, joka piirtää komponenttien elementit selaimeen HTML DOM -elementteinä. HTML- ja CSS-koodit määrittävät käyttöliittymästä

käyttäjälle näkyvän visuaalisen osuuden. JavaScript-luokat tuottavat toiminnallisuuden tapahtumille ja attribuuteille. Koska komponentit perustuvat Aura-kehykseen, sen nimiavaruuden merkintöjä voidaan käyttää hyväksi. Esimerkiksi aura:if-nimiavaruuden merkinnällä voidaan jokin osa komponenttia näyttää käyttäjälle vain, kun tietty vaatimus täytetään.

```
<aura:component controller="ExampleController">
     <aura:attribute name='Name' type='String'/>
     <aura:attribute name='say' type='String' default='Your name is: '/>
     <div id='testDiv' class='slds-m-horizontal--medium card>
     <ui:inputText label='Name' value="{!v.Name}"/>
     <p>{!v.say} {!v.Name}<p>
     </div>
</aura:component>
```

Esimerkkikoodi 6. Lightning-komponentti, jossa käyttäjä voi antaa nimensä, jonka komponentti näyttää

Komponentin toiminnallisuus jaetaan JavaScript-ohjaimen ja apuluokkien kesken. Näkymäosiosta kutsutaan vain ohjainluokkaa, joka tarpeen mukaan käyttää apuluokkien metodeja hyödyksi. Apuluokkien metodit ohjelmoidaan mahdollisimman monikäyttöisiksi.

Komponentit voivat keskustella toistensa kanssa tapahtumien (engl. Lightning Event) avulla. Haluttaessa komponenttiin määritellään, mikä käyttöliittymän tapahtuma laukaisee toisen tapahtuman. Tapahtumat voivat kuljettaa arvoja laukaisijalta kuuntelijalle. Esimerkiksi tiettyä nappia painaessa voidaan käyttöliittymällä näyttää haluttu teksti. [15.] Lisäksi järjestelmä tarjoaa valmiita tapahtumia, joita ohjelmoidessa voidaan hyödyntää, kuten näkymän päivitys, tai käyttäjän ohjaamisen tietylle sivulle napin painalluksesta.

```
helloWorld : function(component, event){
     var event = component.get("ExampeEvent");
          event.setParams({
          "message" : "Hello World"
          });
     event.fire();
}
```

Esimerkkikoodi 7. Parametrillisen tapahtuman laukaiseminen

3.5.2 Näkymän ja palvelimen keskustelu

Palvelinpuolella olevia Apex-luokkia voidaan käyttää hyväksi Lightning-komponenttien toiminnallisuudessa. Komponentin controller-attribuuttiin esitetään haluttu ohjainluokka

esimerkkikoodissa 6 esitellyllä tavalla. Kyseinen luokka ei saa olla privaatti sekä luokasta haettavan metodin pitää olla staattinen ja merkitty @AuraEnable-annotaatiolla.[16.]

Apex-luokalla voidaan tehdä esimerkiksi tietokantakutsuja tai DML-operaatioita käyttöliittymästä lähetetyn parametrin mukaan. Seuraavana ovat esimerkit Apex-luokan kutsumisesta palvelimen puolelta.

```
callApex : function(component, event){
      var action = component.get("c.testFunction");
      action.setParams({ name : "test Name"
      )};

      action.setCallback(component,
            function(response){
                  var state = response.getState();
                  if (state === 'SUCCESS'){
                        component.set("v.testField", response.getReturnValue());
                  else{
                        // Tarvittavat toimenpiteet
                  }
                  }
      )};
      $A.enqueueAction(action);
}
```

Esimerkkikoodi 8. Käyttöliittymän metodi kutsuu palvelimen metodia parametrillä

Esimerkkikoodin 8 palvelimen ohjainluokasta kutsutaan testFunction-metodia. Metodin parametri ”name” on tarkalleen sama kuin Apex-luokan parametrissä, kuten koodiesimerkissä 9 on esitetty. Kutsulle asetetaan takaisinkutsu (engl. Callback), johon palvelimelta palautuva arvo sijoitetaan. Koska palvelimelle tehdessä kutsuja on aina mahdollisuus yhteysongelmiin, toteutuksessa yritetään asettaa kenttään ”testField” palautettu arvo vain, kun palvelimen vastaus on onnistunut (engl. Success).

```
@AuraEnabled
Public static String returnValue(String name){
      return 'Hello ' + name + '!';
}
```

Esimerkkikoodi 9. Palvelinpuolen Apex-metodi, jota voidaan kutsua Lightning-komponentista.

3.5.3 Lightning-tyylit SLDS

Lightning-komponentit voivat käyttää sisäänrakennettua Salesforce Lightning Design Systemiä (SLDS) luomaan yhdenmukaisia ja Lightning Experience -näkymän näköisiä

mukautettuja näkymiä. Se sisältää muun muassa ennalta määriteltyjä tyylejä ja CSS-luokkia. SLDS on Salesforcen ylläpitämä, ja käytettävissä kaikissa Salesforce-organisaatioissa. Tyylittelyn käytön esimerkki on esitelty esimerkkikoodi 6:ssa.

4 Verkkopalvelut

Verkkopalveluksi kutsutaan palveluita, joissa tietoa liikutetaan kahden tai useamman päätteen välillä internetin yli. Palveluiden välillä kulkevien viestien lähettämiselle ja lukemiselle on olemassa useita protokollia. Näistä Salesforcessa tavanomaisimmat ovat SOAP (Simple Object Access Protocol) ja REST. Yleisimmät näissä liikkuvat dokumenttityypit ovat XML ja JSON (JavaScript Object Notation). Useimmiten SOAP-rajapinnoissa kulkevat viestit ovat XML-viestejä, ja REST-rajapinnoissa kulkee JSON-viestejä. Poikkeuksia kuitenkin on, kun esimerkiksi palveluntarjoajat päivittävät rajapintansa käyttämään REST-rajapintaa, mutta siellä kulkevaa viestiä ei päivitetä XML:stä JSON:ksi. [17.]

Tämä työ on yksi niistä poikkeuksista, jossa asiakas on päivittänyt oman rajapintansa REST-pinnaksi, mutta toisen ulkoisen palveluntarjoajan viestit kulkevat edelleen XML-muodossa. Näin ollen tässä työssä keskitymme REST-rajapintaan, ja kuinka siinä XML-viestit kulkevat ja mitä hankaluuksia se tuo mukanaan.

4.1 XML

XML eli Extensible Markup Language on kehitetty 1998. Sen tärkein tarkoitus oli kuljettaa tietoa selkeässä strukturaalisessa muodossa elementti-tagien alla. Se on kirjasto, jossa jokainen tieto on asetettu sille kuuluvan elementin alle. Numero 10 kahden järjestelmän välillä ei merkitse mitään, mutta jos voidaan tarkentaa, mitä kyseinen numero tarkoittaa, esimerkiksi tietyn käyttäjän kirjautumisten määrä, voidaan tarkkaa dataa alkaa lähettää järjestelmien välillä. [18.]

Koska XML ei ole rajoitettu yhden protokollan alle, on tärkeää, että kumpikin järjestelmä osaa lukea kyseistä viestiä, eli tietää minkälaisten elementtien alla luettava tietoa säilytetään.

Useat applikaatiot ovat todella datakeskeisiä, jolloin XML-dokumentit voivat kasvaa hyvinkin laajoiksi. Näin ollen koko XML-dokumentit tallentamisen sijaan monet palvelut tyytyvät ottamaan viestin vastaan, parsimaan sen haluttuihin tauluihin ja objekteihin ja tallentamaan kyseisen taulun tai objektin tietokantaan. [18.]

4.1.1 Syntaksi

XML-syntaksi on pohjimmiltaan yksinkertainen. Sitä on melko helppo muokata tarpeen mukaan, kunhan viestiä lukevat applikaatiot pysyvät muutoksen mukana. XML-dokumentti voidaan aloittaa prologilla. Tähän voidaan kirjata muun muassa tekstin merkistöstandardi ja XML-version numero. Mikäli prologi kirjataan, mitään merkkejä ei saa olla ennen sitä. Tätä seuraa juurielementti. Juurielementti on ainoa pakollinen osa XML-dokumenttia, kaikki muut osat tulevat juurielementin sisälle.

4.1.2 Elementit ja attribuutit

XML-dokumentin päärakenne koostuu elementeistä ja niitä tarkentavista attribuuteista. Elementtejä dokumentissa voi olla rajattomasti, mutta ylimmällä tasolla ne ovat lopulta kaikki juurielementin lapsia. Esimerkkikoodissa 10 on esitelty XML-dokumentin rakennetta, jossa on mukana prologi. Esimerkissä prologin jälkeen on juurielementti ja kaikki elementit ovat sen sisällä. Elementtejä kuvataan alku- ja loppumerkinnöillä. Elementtien merkintöjä kutsutaan tageiksi (engl. Tag). Nimi-tagillä on attribuuttina ”hetu”, jolle on asetettu arvo. Tällä voidaan tarkentaa, kenen henkilön dataa käsitellään. [19.]

```
<?xml version=”1.0” encoding=”UTF-8”?>
<root>
     <nimi hetu=”010101-0101”>
          <Etunimi>Esimerkki</Etunimi>
          <Sukunimi>Merkkinen</Sukunimi>
     </nimi>
</root>
```

Esimerkkikoodi 10. XML-esimerkki, jolla on attribuutti.

4.1.3 Nimiavaruusmäärittelyt

Elementtien aloitusmerkintöihin voidaan lisätä nimiavaruusmäärittely. Niiden tarkoitus on erottaa toisistaan samannimiset elementit ja attribuutit. Monet järjestelmät käsittelevät useita uniikkeja Id- ja nimi-tagejä, jolloin nimiavaruuden määrittely on tarpeen. [20.]

4.2 REST

REST (engl. Representational State Transfer) on arkkitehtuurimalli, jonka tarkoitus on auttaa ohjelmointirajapintojen (engl. Application programming interface, API) keskustelua keskenään. Sen kautta eri ohjelmat voivat vaihtaa tietoja ja tehdä pyyntöjä internetin välityksellä. Internetin levittäytyessä huomattiin, että olemassa olevat protokollat käyvät isojen datamäärien käsittelyssä hankalaksi. Yksi ratkaisu ongelmaan saapui Roy Fieldingin väitöskirjassa, jossa hän esitteli uuden datanvälitysarkkitehtuurimallin, RESTin. [21.]

4.2.1 RESTin vahvuudet

RESTin suurin vahvuus tulee sen tilattomuudesta. Tilattomuuden tarkoitus on antaa asiakkaalle mahdollisuus pitää session tila hallussa jatkuvasti. Tilattomassa viestissä kutsun suorittamiseen tarvittava tieto on sisällytettävä viestiin, jonka asiakas lähettää palvelimelle. Kutsun suorittaminen ei ole riippuvainen siitä, onnistuivatko tiedolle tarkoitetut operaatiot palvelimella. Tilattomuus parantaa protokollan luotettavuutta, skaalautuvuutta ja näkyvyyttä. Protokolla on luotettavampi, sillä mahdollisista virheistä on verrattain paremmat mahdollisuudet palautua. Se skaalautuu tehokkaasti, sillä tilan ylläpitäminen vaatii ohjelmistolta huomattavasti resursseja sen kuormittuessa useiden viestien samanaikaisesti vastaanottamisesta. Lisäksi tilattomuuden kautta vastaanottava osapuoli saa aina kysyjän sisällön ja tarkoituksen sen sisältämän datan perusteella. [21.]

Tilattomuuden kannalta on ehdotonta, että asiakkaan puolella ylläpidetään välimuistia. Tämä tarkoittaa, että kyselyn vastaukseen merkitään, tallennetaanko se välimuistiin. Mikäli uusi vastaava kysely lähetetään rajapintaan, voidaan välimuistiin tallennettua vastausta käyttää uudelleen hyväksi. Tämä nopeuttaa rajapinnan toimintaa, kun jokaisella kutsulla ei välttämättä tarvitse ladata tietokannasta dataa uudestaan sen tallentuessa

välimuistiin. Välimuistin käytössä on kuitenkin huonona puolena se, että välimuistissa oleva vastaus voi ehtiä vanhentua tietokannan dataan verrattuna, eikä näin ollen vastauksena palauta uusinta tietoa.

4.2.2 RESTin operaatiot

Operoidessa REST-rajapintojen kanssa käytetään HTTP-standardin merkintää kertomaan operaation tarkoitus. Seuraavassa ovat tavanomaisimmat operaatiot.

Get-metodilla haetaan tietoa tietokannasta.

Post-metodia käytetään tiedon hakemiseen tai päivittämiseen.

Patch–metodia käytetään määriteltyjen taulujen ja kenttien päivittämiseen.

Delete-meteodia käytetään tiedon poistamiseen tietokannasta.

4.2.3 RESTin toteutus Salesforcessa

Salesforcen sisäänrakennettu REST-standardi tarjoaa yksinkertaisen tavan integroida ulkoiset järjestelmät siihen. Salesforce myös mahdollistaa Apex-pohjaisten REST-luokkien luomisen ja Apex-koodin upottamisen REST-rajapinnan toimintaan.

Apex-luokan määrittely REST-luokaksi tapahtuu lisäämällä luokan alkuun @RestResource-annotaatio. Näin mahdollistetaan ulkoisten kutsujen ohjautuminen tämän luokan toimintaan. Luokalle luodaan metodi vastaamaan kutsuun. Tämä merkitään operaatiota vastaavalla annotaatiolla, esimerkiksi @HttpPost. Lisäksi Salesforce tarjoaa järjestelmän sisäisen staattisen luokan RestContext. Tämä luokka mahdollistaa kyselyn sanoman tarkistelun, ja voi asettaa vastauksen tietoja, tärkeimpänä kummastakin sanoman HTTP-statuskoodi ja tarvittavat ilmoitukset statukseen liittyen.

```
@RestResource('esimerkkiURL/*')
Global class ExampleRestApi{
      @HttpPost
      global static responseClass doPost(){
            RestRequest req = RestContext.request;
```

```
            RestResponse res = RestContext.response;

            try{
                  exampleParser(request);
            }catch(XmlExeption x){
                  System.debug(x.getMessage());
      }
res.statusCode = 200;
response.success = true;
response.message = 'success';
return response;
}
```

Esimerkkikoodi 11. Apex-luokan ja sen sisältävän HttpPost-metodin määritys sekä vastaus REST-kyselyyn

Salesforce rajoittaa Apex REST-rajapintoja siten, että se sallii vain yhden kutakin HTTP-metodia. Näin kahta @HttpPost-metodia ei voida määrittää yhteen REST-rajapintaan. [22.]

5 Toteutus

Työssä toteutettu ohjelmisto on kykenevä vastaanottamaan tietyn standardin mukaisia XML-dokumentteja ja luomaan niistä järjestelmään niin mukautettuja kuin standardiobjekteja, jotka on linkitetty toisiinsa hakusuhde-kenttien avulla. XML-dokumenteissa liikkuu henkilödataa. Henkilödataa lähetettäessä kaikki arkaluontoinen data suojataan. Salesforce tarjoaa jokaiselle kentälle erikseen mahdollisuuden suojaukseen, joka voidaan aktivoida kentän asetuksista luonnin tai muokkauksen yhteydessä. Kyseisiin viesteihin voidaan vastata vastaavan rakenteen omaamalla XML-dokumentilla, johon keskeinen data haetaan järjestelmän objektien kenttien arvoista. Tätä varten kehitettiin mukautettu käyttöliittymä. Lisäksi tarvittaessa objektin tietueita voidaan luoda käsin käyttöliittymältä sekä lähettää niitä haluttuun osoitteeseen ilman aiemmin saapunutta viestipyyntöä.

Työ toteutettiin tuottamalla REST-rajapinta, Apex-luokkia palvelimen puolelle käsittelemään tietokantakutsut sekä XML-dokumentin parsiminen, rakentaminen ja viestin lähettäminen. Käyttöliittymä toteutettiin yhdistämällä Salesforcen tarjoamia Lightning-sivuja ja niihin aseteltuja mukautettuja ja standardiobjekteja.

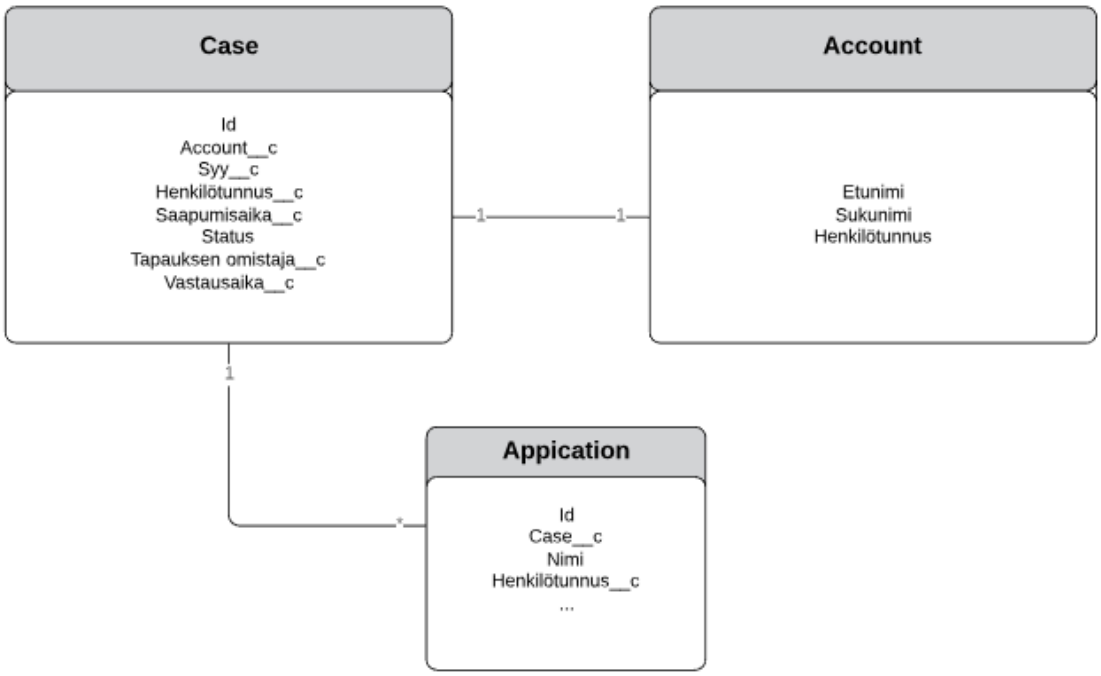
REST-rajapintaan saapuva XML-dokumentti sisältää tarkan strukturaalisen rakenteen, mutta on iso dokumentti, jossa on paljon parsittavaa dataa. Jokaiselle tallennettavalle datalle luotiin oma kenttä mukautetulla objektilla.

Tässä luvussa käydään läpi työ vaiheittain yleisellä tasolla siten, että esitellään käytetyt Salesforcen asetukset, Apex-luokkien toiminta, käyttöliittymän toiminta sekä testiluokkien toiminta.

5.1 Tietokannan datamalli

Tietokannan datamallissa hyödynnetään järjestelmän valmiita standardeja objekteja sekä yhtä mukautettua objektia. Datamallissa hyödynnetyt objektit ovat tilit ja tapaukset sekä täysin mukautettu objekti, jota kutsumme nimellä Application__c. Tapaukset voivat olla esimerkiksi Salesforcea käyttävän osapuolen asiakkaiden kysymyksiä, palautteita tai tukipyyntöjä.

Toiminnallisuudessa kaikki on kiinnitetty tapauksiin. Kun REST-rajapinnan kautta uusi viesti saapuu tietokantaan, sen datan perusteella luodaan uusi tapaus tai se kiinnitetään olemassa olevaan tapaukseen dokumentin niin vaatiessa. Tapaukselle kiinnitetään tili XML-dokumentin henkilödatan perusteella. Mikäli kyseiselle henkilölle on jo tili tietokannassa, se liitetään tapaukseen hakusuhde-kenttään. Mikäli ei ole, luodaan uusi tili ja se liitetään tapaukseen. Tapaukselle luodaan myös uusi Application-objektin tietue. Applicationille on määritelty kenttä, johon siihen liittyvä tapaus kiinnitetään. Tämä kenttä on määritelty pakolliseksi kentäksi Application-objektin tietuetta luodessa, eikä sitä voida poistaa objektilta.



Kuva 1. Datamalli. Applicationilla on huomattavasti enemmän kenttiä kuin on tarpeen kuvata

Tilit kiinnittyvät tapauksiin henkilötunnuksen kautta. Tapauksella ei kuitenkaan ole pakollista olla tiliä kiinni. Tämä on siltä varalta, että tietokantaan saapuu puutteellinen XML-dokumentti, joka kuitenkin halutaan kyetä lukemaan järjestelmään.

5.2 Asetukset

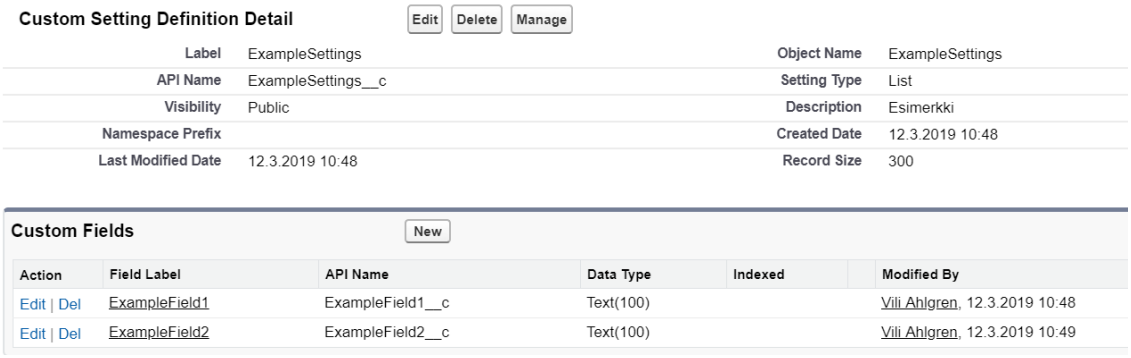
5.2.1 REST-rajanpinnan käyttäjä

Organisaatioon luotiin yksi erillinen käyttäjäprofiili, jolle asetettiin oikeus käyttää tuotettua REST-rajapintaa. REST-rajapintaan yhteyttä ottaessa yhteydenottaja tarjoaa kirjautumistunnuksensa, jonka onnistuessa profiilin käyttäjä voi ottaa yhteyden REST-rajapinnan osoitteeseen (engl. URL, Uniform Resource Locator).

5.2.2 Mukautetut asetukset

Mukautetut asetukset ovat samantyyppisiä kuin mukautetut objektit. Niille voidaan asettaa kenttiä kuin objekteille. Kaikki mukautettujen asetusten data on tallennettu välimuistiin, joten useita tietokantakutsuja niitä varten ei tarvita. Mukautetut asetukset ovat organisaatiokohtaisia, eikä niitä voi kuljettaa organisaatioiden välillä.

Projektiin luotiin mukautettu asetus järjestelmästä ulospäin lähteviä viestejä varten. Kun se on kirjattu organisaation mukautettuihin asetuksiin, vältytään koodissa yhteyttä ottaessa erikseen käsin kirjoittamasta yhteydenoton päätepistettä (engl. Endpoint) eikä vaadittavia sertifikaatteja, vaan ne voi hakea mukautetusta asetuksesta. Mikäli niihin tulee muutoksia, voi muutoksen hoitaa järjestelmän ylläpitäjä, eikä koodiin tarvitse tehdä muutoksia.



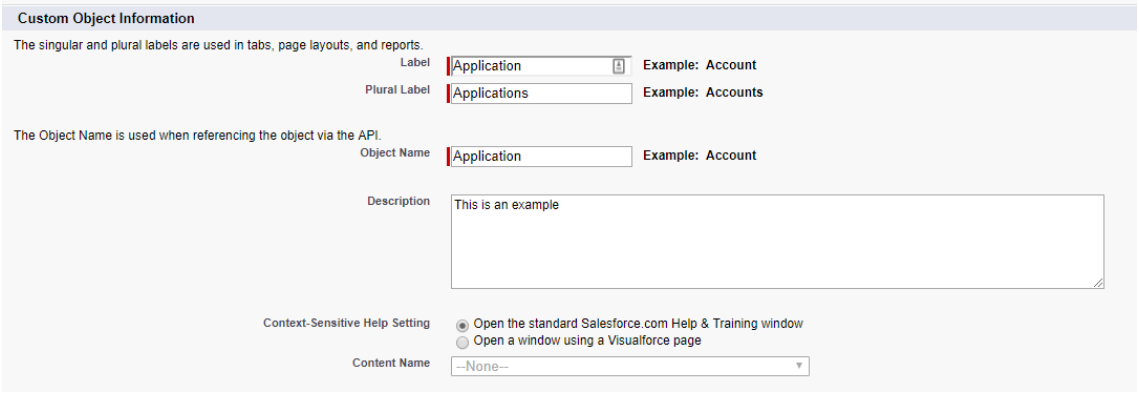
Kuva 2. Mukautetut asetukset

Manage-napin alta käyttäjä pääsee luomaan uuden mukautetun asetuksen, jossa on kyseiselle asetukselle määritetyt kentät. Kenttiä voidaan poistaa tai lisätä tarvittaessa. Tätä voidaan hyväksikäyttää esimerkiksi, jos vastaanottajalla on useita testiosoitteita erinäisiin testitarkoituksiin.

5.2.3 Mukautettu objekti

Mukautettu objekti Application__c luotiin säilyttämään kaikki data REST-rajapintaan saapuvasta XML-dokumentista. Objekti sisältää teksti-, numero-, hakusuhde-, kaava- ja päivämääräkenttiä. Kenttiin sisällytetään henkilödataa, kuten nimet, henkilötunnus, kotiosoite ja runsaasti muuta henkilökohtaista dataa. Kenttiin tallennetaan myös jokaisen

XML-dokumenttiin omat uniikit dokumenttinumerot sekä lähettäjän ja vastaanottajan tekninen tunnusnumero OID (Object Identifier).



Kuva 3. Mukautetun objektin luominen

Application-objektille luotiin kolme tietuetyyppiä, jotka määrittävät objektin tilan ja mitä Lightning-sivun näkymää käyttäjälle objektilta näytetään:

- Pyyntö viestillä

  Yhtiön asiakas lähettää viestin, johon pyytää yhtiön vastausta. Käytetään saapuneen viestin näkymää, jossa kaikkien kenttien muokkaus on estetty, sillä alkuperäinen viesti haluttiin säilyttää ennallaan.

- Pyynnön vastaus

  Ennen kuin asiakaspalvelija vastaa pyyntöviestiin, vastauksen luonnos tallennetaan pyynnön vastauksena tietokantaan. Näin vastausta ei tarvitse lähettää heti, vaan sitä pääsee muokkaamaan tarvittaessa.

- Pyynnön vastaus (Lähetetty)

  Kun vastaus on lähetetty, tietueen tietuetyypiksi asetetaan ”Pyynnön vastaus (Lähetetty)”. Tällöin vastaus on lähetetty, eikä sitä pääse enää muokkaamaan.

5.3 Saapuvan XML-dokumentin struktuuri

XML-dokumentti koostuu kahdesta osasta. Sen alussa on ylätunniste (engl. Header), jonka sisään kirjataan data lähettäjästä, vastaanottajasta sekä ajasta, jolloin dokumentti on lähetetty. Data niin lähettäjästä kuin vastaanottajastakin sisältää kummankin OID-numeron, kontaktin nimen ja identifikaatiotunnuksen, joka määrittää viestin tyypin.

Ylätunnisteen alla on juurielementti, jonka lapsielementeissä kuljetetaan haluttu data. Juurielementtiin sijoitetaan myös halutut nimiavaruuden määritelmät. OID-tunnus toistuu kummassakin dokumentin osassa, sillä dokumentit kulkevat useiden ulkoisten välittäjien kautta, jotka ohjaavat dokumentin kulkua oikeaan osoitteeseen. Standardia käyttäessä jotkin ohjaavat osapuolet lukevat vain ylätunnistetta, ja viestin saapuessa määränpäähän voidaan lukea niin ylätunnistetta kuin juurielementtiäkin. Kokonaisuudessaan saapuvan XML-dokumentin pituus on keskimäärin noin 1000-2000 riviä.

5.3.1 XML-dokumentin ylätunniste

Esimerkkikoodi 12 kuvaa ylätunnisteen rakenteen. Niin Sender- kuin Receiver-tagien arvona on numerosarja. Tämä numerosarja on kyseisen kontaktin OID-numero. ContactTypeIdentifier-tagi kuvaa lähettäjän tai vastaanottajan tyyppiä, jotka on erikseen standardissa määritelty. DokumentIdentification-tagin alla oleva data on kaikki itse dokumenttiin liittyvää tietoa. Standard -tagi kertoo, mitä standardia tässä dokumentissa käytetään. TypeVersion-tagi kertoo standardin versionumeron.

```
<StandardBusinessDocumentHeader>
    <HeaderVersion>1.0</HeaderVersion>
    <Sender>
        <Identifier Authority="Esim">1.1.222.333.110.12345</Identifier>
        <ContactInformation>
            <Contact>Lähettäjän nimi</Contact>
            <ContactTypeIdentifier>AA</ContactTypeIdentifier>
        </ContactInformation>
    </Sender>
    <Receiver>
        <Identifier Authority="Esim">1.1.222.555.110.9876</Identifier>
        <ContactInformation>
            <Contact>Vastaanottajan nimi </Contact>
            <ContactTypeIdentifier>BB</ContactTypeIdentifier>
        </ContactInformation>
    </Receiver>
    <DocumentIdentification>
        <Standard>Standardin nimi </Standard>
        <TypeVersion>1.1</TypeVersion>
```

```
            <InstanceIdentifier>1.1.222.333.110.12345.89237893678367
                </InstanceIdentifier>
            <Type>Viesti</Type>
            <CreationDateAndTime>2019-01-17T11:41:03</CreationDateAndTime>
        </DocumentIdentification>
    </StandardBusinessDocumentHeader>
```

Esimerkkikoodi 12. XML-dokumentin ylätunniste

InstanceIdentifier on viestin uniikki tunnistenumero. Se koostetaan lähettäjän OID-numerosta + lähettäjän sisäisen järjestelmän uniikista numerosta. Tämän tagin arvoa käytetään mahdollisissa virhetapauksissa tunnistamaan, missä dokumentissa virhe on tapahtunut. Type kertoo dokumentin tyypin ja CreationDateAndTime kertoo dokumentin lähetysajan.

5.3.2 XML-dokumentin juurielementti

Juurielementissä on pohjustettu xmlns- ja xmlns:voc-nimiavaruudet. Tämän dokumentin parsimiseen ne eivät kuitenkaan vaikuta. Juurielementin tunniste ”Id”-tagin attribuuteista löytyy ”extension”, joka merkitsee dokumentin juoksevaa numeroa, ja ”root”, jossa on lähettäjän OID-tunnus. EffectiveTime-tagi kertoo dokumentin luontiajan. Kyseisessä standardissa code-tagin attribuutissa näkyvä ”displayName” on aina sama arvo kuin sen alta löytyvä title-tagin arvo. Esimerkkikoodi 13:ssa näkyvät muut arvot ovat standardiin kuuluvia staattisia arvoja.

```
     <Juurielementti xmlns="urn:b19-org:v3" xmlns:voc="urn:b19-org:v3/voc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <id extension="25678312" root="1.1.222.333.110.12345"/>
    <code code="164" codeSystem="9.9.9.9.9.9.9" displayName="Viestipyyntö"/>
    <title>Viestipyyntö</title>
    <effectiveTime value="20181214"/>
    <confidentialityCode code="N" codeSystem="2.16.840.1.113883.5.25"/>
    <setId extension="25678312" root="1.1.222.333.110.12345"/>
    <versionNumber value="1"/>
```

Esimerkkikoodi 13. XML-dokumentin juurielementti

5.3.3 XML-dokumentin toistuvat osat

Itse luettava data dokumentilta löytyy toistuvien component-tagien alta. Component-kokonaisuuksilla on aina selkeä rakenne esimerkkikoodin 14 mukaisesti.

```
<component>
     <section>
          <code code="9" codeSystem="1.2.246.537.6.12.2004.164" codeSys-
temName="Viestipyyntö" displayName="Osoite"/>
          <title>Osoite</title>
          <text>
               <paragraph>
                    <content>Kauppakatu 6, Helsinki 00560</content>
               </paragraph>
          </text>
          <entry>
               <observation moodCode="EVN" classCode="OBS ">
                    <code code="9" codeSystem="1.2.246.537.6.12.2004.164"
codeSystemName="Viestipyyntö" displayName="Osoite"/>
                    <value xsi:type="AD" use="HP">
                         <streetName>Kauppakatu</streetName>
                         <houseNumber>6</houseNumber>
                         <city>Helsinki</city>
                         <postalCode>00560</postalCode>
                    </value>
               </observation>
          </entry>
     </section>
</component>
```

Esimerkkikoodi 14. Component-tagin ja sen lapsielementtien rakenne

Component-tagia aina seuraa Section-tagi. Sitä seuraa Code-tagi, jonka attribuutti ”code” on dokumentin lukemisen kannalta tärkeä. Jokainen luettava component-tagi merkitään juoksevalla ”code”-tagin numeroinnilla. Tämä tarkoittaa, että sama tieto, tässä tapauksessa osoite, on aina ”code = ’9’”. Tätä tietoa käytetään hyväksi myöhemmin XML-dokumenttia parsiessa tietokannan Application-objektin tietueen kenttiin. Arvo codeSystem-tagissa on standardin määrittämä staattinen numero. Arvo codeSystemName -tagissa kuvaa dokumentin tyyppiä. Sisäänpäin tullessaan se on ”Viestipyyntö”, mutta viestiin vastatessa järjestelmästä tyyppi on ”Vastaus”.

Tätä seuraa aina seuraava rakenne:

- Title – kuvaa mikä data kyseessä
- Text – sisältää lapsielementtejä
- Paragraph – sisältää lapsielementtejä
- Content – sisältää luettavan datan.

Content voi kuitenkin olla tyhjä, mikäli lähettävä osapuoli ei ole täyttänyt vastaavaa kenttää lähettäessään XML-dokumentin. Rakenne jatkuu seuraavasti:

- Entry – sisältää lapsielementtejä
- Observation – sisältää attribuuteissa componentin luokkamääritykset
- Code – sisältää saman datan kuin aiemmin esiintyessään
- Value – sisältää saman datan kuin content mahdollisesti jäsennellyssä muodossa. Lisäksi nimiavaruuskoodi määrittää, millaista dataa on kyseessä.

Component-tageja voi olla sisäkkäin useita. Esimerkiksi yksi component-tagi voidaan kirjoittaa title-tagiin asti, jossa otsikoksi mainitaan ”Henkilötiedot”. Tämän alla voi olla useita esimerkkikoodin 14 tapaisia kokonaisuusia, jotka kukin kertovat muun muassa nimen, henkilötunnuksen ja osoitteen.

5.3.4 Ongelmat XML-dokumentin lukemisessa

Suurin ongelma dokumentissa on sen mahdolliset epäjärjestelmällisyydet. Asiakas voi saada dokumentteja useiden eri kontaktien kautta, ja jokainen kontakti on rakentanut oman XML-dokumentin erikseen. Vaikka standardia pitäisi seurata tarkasti tehokkaan datansiirron kannalta, niin ei kuitenkaan aina ole.

Testauksia tehdessä kävi ilmi, että jotkut kontaktit mahdollistivat useiden kenttien tyhjäksi jättämisen. Esimerkiksi osoitteen kanssa sen sijaan että kirjoitettaisiin koodiesimerkki 14 mallin mukaan streetName, houseNumber, city ja postalCode erillisille kentille, saattaa kahden kentän tiedot olla yhden kentän alla, ja toinen kenttä olla tyhjä kuten koodiesimerkissä 15.

```
<streetName>Kauppakatu 6</streetName>
<houseNumber></houseNumber>
<city>Helsinki 00560</city>
<postalCode></postalCode>
```

Esimerkkikoodi 15. Virheellisesti kirjattu osoitetieto

Ongelmaksi tämä muodostuu, kun asiakas haluaa saada osoitetiedot omille kentilleen omassa järjestelmässään. Sama tapaus olisi myös dokumenttia koskevan henkilön nimien kanssa. Koska emme voi olla varmoja, mille kentälle data tulee, ratkaisuna käytämme datan parsimisessa aina content-tagin arvoa. Tämä kenttä saa arvonsa Value-tagin kaikki kentät yhdistettynä, kuten esimerkkikoodissa 14. Tällöin tieto tallennetaan yhteen kenttään, jossa se on kokonaisuudessaan tallessa. Tällöin jotkin usean tagin omaavat tietokokonaisuudet voivat Salesforcen käyttöliittymän näkymässä jäädä heikommin jäsennellyiksi, mutta voimme olla varmoja, että data löytyy objektilta.

5.4 REST-rajapinta

Projektin suunnittelu aloitettiin määritelmästä, että XML-dokumentti tuodaan REST-rajapintaan, ja se parsitaan läpi luoden uusi mukautettu Application-objekti. REST-rajapinta palauttaa onnistuneen 200-statuksen, mikäli dokumentin parisinta onnistuu. Muussa tapauksessa annetaan järjestelmän palauttama virhe. Tätä varten luotiin Apex-luokka, jolle luokan alussa määritetään uniikki osoite, jolla sitä kutsutaan. Luokka pohjustetaan REST-rajapinnaksi @RestResource(urlMapping=’/viestiRestApi/v1/*) -annotaatiolla. Tähän osoitteeseen yhteyden ottaa asiakkaalle määritelty profiili, jolla on oikeus kyseiseen luokkaan.

```
@HttpPost
global static MLResponse doPost() {
    RestRequest req = RestContext.request;
    RestResponse res = RestContext.response;

    MLResponse response = new MLResponse();
    res.addHeader('Content-Type','text/xml;charset=utf-8');

    try{

        try{
            //Parse received XML string
            String reqString = req.requestBody.toString();
            XMLUtils.handleXMLParser(reqString);
        }catch (
                MLException ghex){
            throw new MLException(ghex.getMessage());
        }
        response.success = true;
        res.statusCode = 200;
        response.message = '200: Success ';
    }catch (MLException gex){
        res.statusCode = 400;
        response.success = false;
        response.message = gex.getMessage();
```

```
        }catch (Exception e){
            res.statusCode = 500;
            response.success = false;
            response.message = 'Unexpected exception. '+ e.getMessage() + ' '
+ e.getStackTraceString();
        }
        return response;
```

Esimerkkikoodi 16. REST-rajapinnan palauttama vastaus

Jäsennystä varten luotiin XMLUtils-luokka, jonka metodi handleXMLParser ottaa parametrissä XML-dokumentin merkkijonona vastaan.

5.4.1 XML-dokumentin jäsennys

Jäsennyksessä käytetään Salesforcen Dom-luokkaa, johon XML-dokumentti voidaan tarjota merkkijonona.

```
Dom.Document dom = new Dom.Document();
dom.load(xml);
Dom.XmlNode node = dom.getRootElement();
```

Esimerkkikoodi 17. Dokumentin pohjustus Dom-luokan avulla

Dom-luokan getRootelement-metodi etsii dokumentin juurielementin. Tämän jälkeen koko XML-dokumentti on käsiteltävissä juurielementin lapsielementteinä.

5.4.2 Rekursiivinen jäsennys

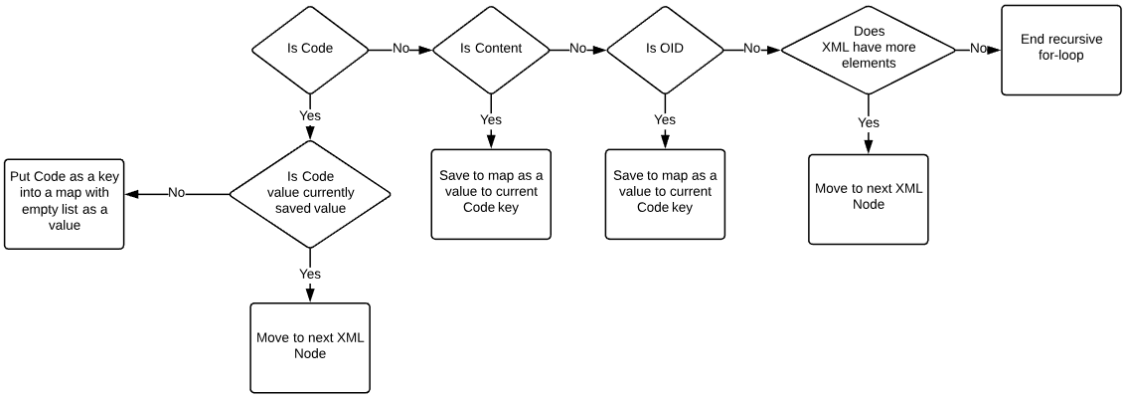
Jäsennyksessä käytetään hyväksi for-silmukan rekursiivisuutta. Sillä tarkoitetaan, että metodin lopussa se kutsuu itseään niin kauan, kuin sille on määritelty. Rekursion kanssa työskennellessä on oltava hyvin tarkka, että for-silmukka ei jää silmukkaan ikuisesti.

```
public static void parseXMLNodes(Dom.XmlNode node){

     switch on node.getName(){
          when 'code'{
               parseCodeNode(node);
          }
          when 'content'{
               parseContentNode(node);
          }
          when 'OID'{
               parseOIDNode(node);
          }
```

```
        }

        for(Dom.XmlNode childNode : node.getChildElements()){
            parseXMLNodes(childNode);
        }
}
```

Esimerkkikoodi 18. Rekursiota toteuttava metodi, joka parsii kaikkien XML-solmujen läpi.

Silmukassa tutkitaan jokaisen elementin nimi. Tiedetään, että standardin mukaan jokaisella tietoryhmällä on oma koodinumeronsa, joten data voidaan jaotella niiden mukaan. Silmukka tallentaa koodinumeron ja sen alla tulevan content-tagin arvon hajautustauluun (engl. Map) avain-arvo-pariksi. Tämä toistuu koko dokumentin rakenteen yli siten, että hajautustauluun tallentuivat kaikki löydetyt code-tagin arvot ja niiden content-tagien arvot.



Kuva 4. XML-jäsentimen toiminta

Kuva 4 esittää XML-jäsentimen logiikan. Jäsennin lopettaa itsensä kutsumisen, kun kaikki XML-dokumentin elementit on käsitelty.

5.4.3 Tapauksen linkitys

On mahdollista, että sisään tuleva XML-dokumentti on päivitystieto edelliseen, jo yhtiön vastaamaan viestipyyntöön. Jokaiseen vastaukseen kirjataan erillinen tapausnumero, jonka perusteella päivitystiedot voidaan yhdistää järjestelmässä alkuperäiseen tapaukseen. Tapausnumero on XML-dokumentissa mukana ainoastaan, jos yhtiö on jo vastannut viestipyyntöön.

5.4.4 Application-tietueen luonti

Jäsennyksen jälkeen luodaan uusi Application-objektin tietue. Sille toteutetaan SOQL-kysely, jossa tutkitaan, löytyykö tietokannasta jo viestipyyntö, jolla on dokumentissa tullut tapausnumero. Jos sellainen löytyy, liitetään siihen liittyvä tapaus myös uuteen tietueeseen. Jos sellaista tapausta ei löydy, luodaan uusi tapaus esimerkkikoodin 5 tavoin.

Uudelle tietueelle asetetaan kenttien arvot hajautustauluun tallennettujen arvojen perusteella. Hajautustaulun läpi iteroidaan ja sen avaimella haettu arvo sijoitetaan sitä vastaavaan kenttään.

```
public static String codeMapping(String str){
      String stringToReturn;

      Map<String, String> codeMap = new Map<String, String>{
                ’1’=>'Tunnus__c',
                '2'=>'Nimi__c',
                '3'=>'Henkilötunnus__c',
                '4'=>'Tapausnumero__c'
      };
      return stringToReturn;
}
```

Esimerkkikoodi 19. Hajautustausta haetaan avainta vastaavan kentän arvo.

Hajautustaulun avaimella haetaan sen avainta vastaava tietokannan kentän nimi. Kentän nimi tallennetaan merkkijono-muuttujaan. Uuteen Application-tietueeseen sijoitetaan hajautustaulun avaimen arvo, ja sille kuuluva kenttä. Koko hajautustaulu käydään for-silmukassa läpi niin, että kaikki avain-arvo-parit lisätään uuteen Application-tietueeseen.

```
Application__c application = new Application__c();
for(String key : Map.keySet()){
           //Fetch field value by it's key (Code) (follows XML example Codes)
           String codeString = codeMapping(key);

           //Get field value string
           List<String> fieldStringList = tvMap.get(key);

           String fieldString;
           for(String str : fieldStringList){
                if(fieldString == nulL || String.isNotBlank(fieldString)){
                     fieldString = String.join(fieldStringList,', ');
                }
           }
           // Put Field-Value pair to Application object
           if(codeString != null && String.isNotBlank(codeString)){
                application.put(codeString, fieldString);
           }
```

```
}
```

Esimerkkikoodi 20. For-silmukassa iterointi codeMapping-metodin avulla ja datan asetus uuteen application-tietueeseen.

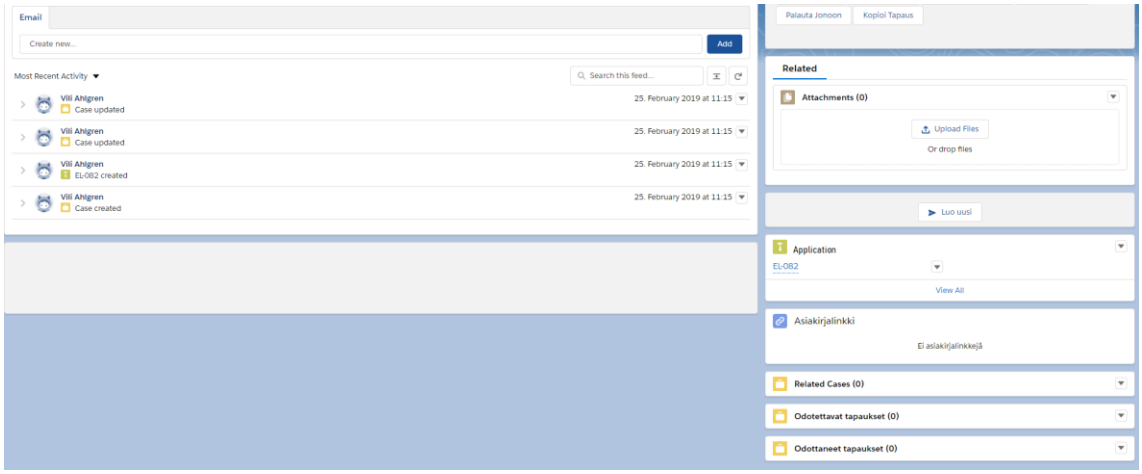
Nyt tietueen tapaukseksi voidaan asettaa aiemmin todettu jo olemassa oleva tai juuri luotu tapaus. Application tietueen tietuetyypiksi asetetaan ”Pyyntö viestillä”. Mikäli luotiin uusi tapaus, sille asetetaan dokumentista parsittu henkilötunnus.

Viimeiseksi tapaukseen liitetään tili. Tileille luotiin globaalisti uniikki kenttä Henkilötunnus__c, eli koko organisaatiossa kahdella tietueella ei voi olla tässä kentässä samaa arvoa. XML-dokumentista parsitulla henkilötunnuksella suoritetaan SOQL-kysely tarkistamaan, löytyykö tietokannasta tiliä kyseisellä henkilötunnuksella. Mikäli tili löytyy, se linkitetään tapauksen Account__c hakusuhde -kentän arvoksi. Mikäli tiliä ei löydy, luodaan uusi tili parsitun nimen ja henkilötunnuksen perusteella ja linkitetään se tapaukseen.

5.5 Käyttöliittymä

Tämä luku käsittelee käyttöliittymän toiminnan asiakaspalvelijan näkökulmasta. Lightning Componenttien tarkempaan toimintaan perehdytään luvussa 5.6.

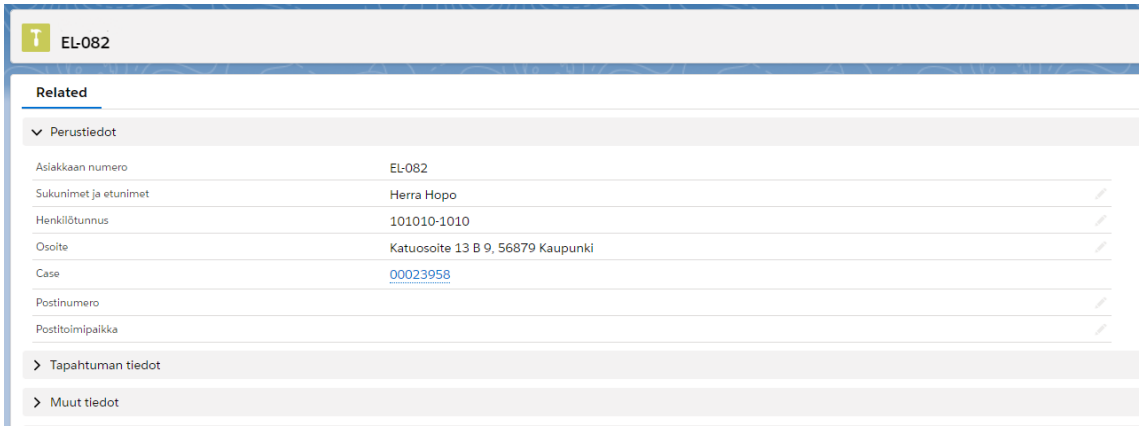
Asiakkaalla on olemassa mukautettu tapausten käsittelyjono. Kun uusi tapaus luodaan, se lisätään jonoon ja siitä tehdään ilmoitus asiakaspalvelijalle. Asiakaspalvelija ottaa tapauksen käsittelyyn jonosta. Tapauksella on mukautettu Lightning sivu -käyttöliittymä, jossa viestipyyntö näkyy.



Kuva 5. Kuvaus tapauksen näkymästä.

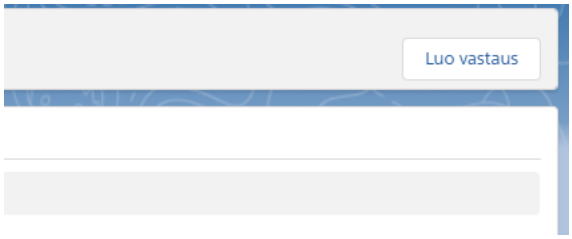
Tapauksen näkymässä saapuneet Application-tietueet näkyvät sen oikeassa laidassa. Tässä näkyvät myös luodut luonnokset ja vastauksena lähetetyt Applicationit (kuva 5).

Viestipyyntöön vastaus luodaan avaamalla kyseinen tietue auki tapauksen käyttöliittymältä (kuva 5). Tietue avautuu näkymään, jossa sen kentille on XML-jäsentimen toimesta luotu siihen kuuluvat tiedot. Kyseinen näkymä on määritelty Lightning-sivulla näkymään vain, kun tietueen tietuetyyppi on ”Pyyntö Viestillä” (kuva 6).



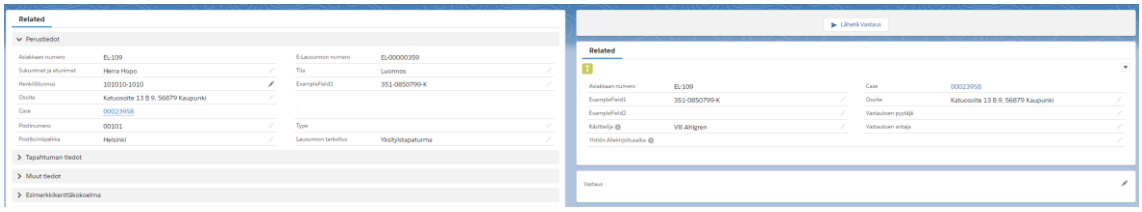
Kuva 6. Application-tietueen näkymä, johon on parsittu dataa.

Vastaus viestipyyntöön luodaan mukautetun Lightning Componentin avulla, jossa toiminnallisuus on liitetty ”Luo vastaus” -napin painallukseen (kuva 7). Napista painamalla kutsutaan Apex-luokkaa newViestiController luvussa 3.5.2 esitetyllä tavalla.



Kuva 7. Luo vastaus nappi Application-objektin tietueen näkymässä

Napin toiminta avaa käyttäjälle uuden näkymän, johon on kopioitu tiedot kyseiseltä tietueelta. Uusi avattu tietue tallennetaan luonnin yhteydessä tietokantaan Application-objektin tietueena. Tämän tarkoitus on luoda pohja vastaukselle, jonka yhtiö lähettää viestipyyntöön. Uuden tietueen tietuetyypiksi asetetaan ”Pyynnön vastaus”. Tämän tietuetyypin tiettyjä teknisiä kenttiä asiakaspalvelijalla on oikeus muokata vastauksen vaatimusten mukaiseksi. Pyyntöä käsitellessään asiakaspalvelija näkee tarpeellisen datan myös kentiltä, joita sillä ei ole oikeus muokata.



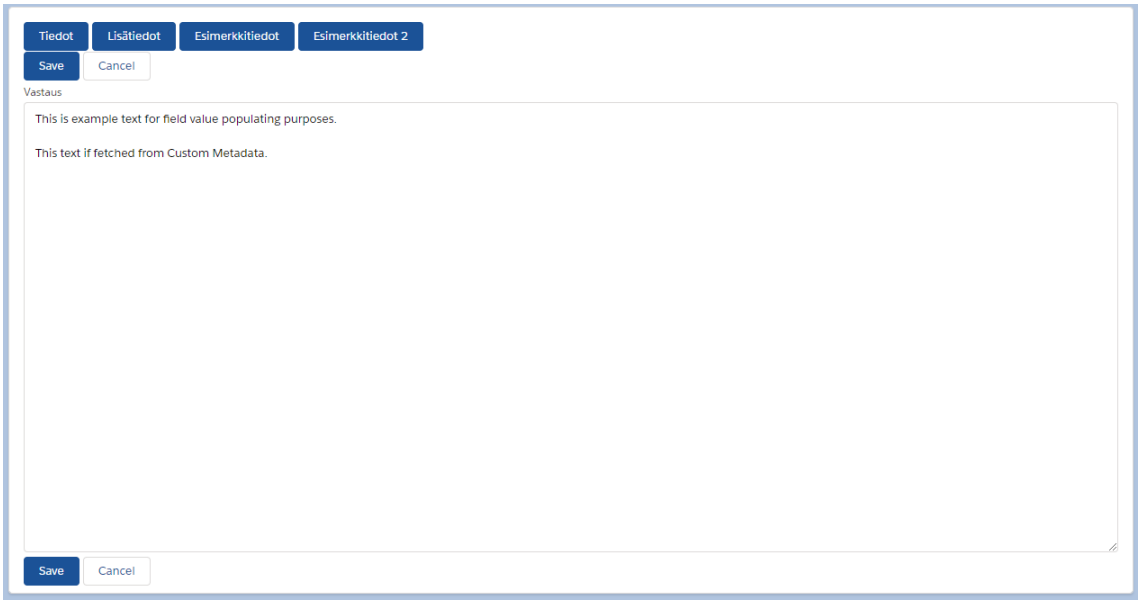
Kuva 8. Viestin vastauksen käyttöliittymä.

Käyttöliittymällä vasemmalle asettuvat kentät, joita asiakaspalvelijalla ei ole oikeutta muokata. Tämä on toteutettu kahdella Lightning-standardikomponenteilla. Tabs-komponentti toimii kenttien pohjana, sen sisään on upotettu Record Detail -komponentti, joka näyttää tietueen kentät sille asetetun sivupohjustuksen mukaan (kuva 8).

Käyttöliittymän oikealle asetettiin kolme komponenttia. Ylimpänä SendVastaus – mukautettu Lightning komponentti, jonka tehtävä on koostaa yhtenäinen XML-dokumentti tietueen kenttien arvoista ja lähettää se haluttuun osoitteeseen. Komponentin toiminta tarkemmin luvussa 5.6. Keskellä on vastaavasti kuin käyttöliittymän vasen puoli, Tabs, jonka sisällä on Record Detail. Tähän Record Detail -komponenttiin on käytetty Salesforcen tarjoamaan vaihtoehtoa upottaa standardi lisätoiminnallisuus sen sisään. Kom-

ponentissa käytetään Quick Action -toimintaa, joka on määritetty päivittämään käsiteltävä tietue. Toiminnallisuudelle on määritetty, mitkä kentät se näyttää, eli minkä päivitys komponentin kautta mahdollistetaan. Näin rajoitetaan asiakaspalvelijaa tekemään muutoksia vain kenttiin, joihin sillä on oikeus (kuva 8).

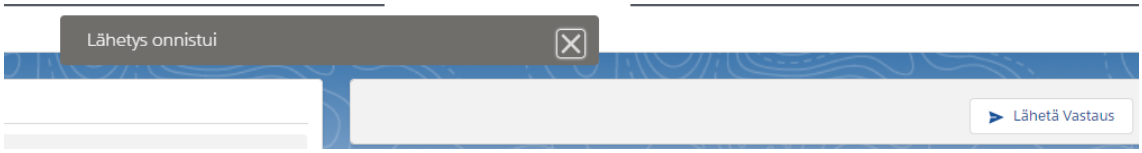
Käyttöliittymän oikean puolen alimpana on mukautettu komponentti ReplyField. Komponentti sisältää syöttökentän (engl. Input Field), johon se sijoittaa Vastaus__c -kentän arvon. Kenttään kirjataan yhtiön tarjoama vastaus. Yhtiön vastaukset voivat olla pitkiä, joten toteutuksessa otettiin huomioon kentän vaativa tila ja käyttöliittymän sujuva toiminta, kun dataa on paljon. Komponenttiin lisättiin neljä nappia, joita painamalla komponentin syöttökenttä voidaan esitäyttää valmiiksi määritellyllä tekstillä (kuva 9).



Kuva 9. Vastauskenttä, jossa on neljä nappia kentän esitäyttöä varten.

Tekstipohjat ovat tallennettuina tietokannan mukautettuun metadataan. Mukautettu metadata toimii hyvin samantyylisesti kuin luvussa 5.2.2 esitellyt mukautetut asetukset. Mukautetun metadatan käytön helpottava puoli on, että se voidaan viedä organisaatiosta toiseen. Näin ollen sitä voidaan kuljettaa hiekkalaatikoiden ja tuotannon välillä vaivattomasti. Järjestelmänvalvojilla on pääsy mukautettuun metadataan, joten koodia ei tarvitse muuttaa, jos asiakas haluaa tehdä muutoksia tekstipohjiin.

Asiakaspalvelijan suoritettua tarpeelliset muutokset tietue tallennetaan Save-painikkeesta. Kun vastaus on valmis lähettäväksi, painetaan ”Lähetä vastaus” -nappia (kuva 7). Onnistuneessa viestin lähetyksessä järjestelmä tuottaa ilmoitusviestin siitä asiakaspalvelijan näkymään (kuva 9). Mikäli lähetyksessä tulee ongelma, se tuotetaan ilmoitusviestiin.

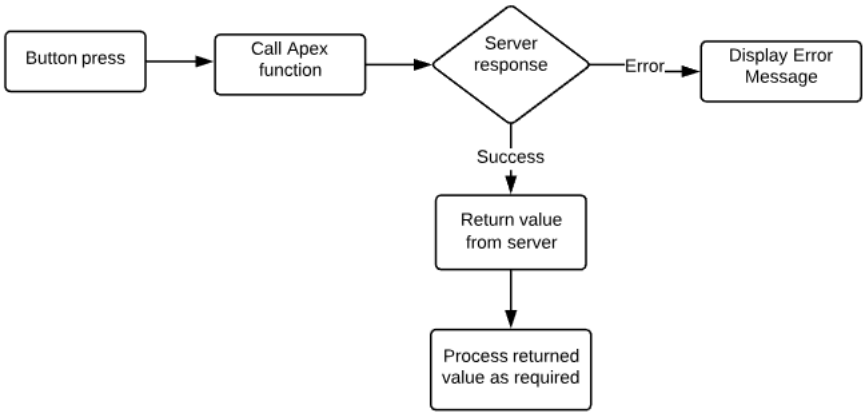


Kuva 10. Onnistunut lähetys.

Ilmoitusviestit on toteutettu ”Lähetä vastaus” -painikkeen mukautetun komponentin toimintaan.

5.6 Lightning-komponenttien toiminta

Projektissa pyrittiin käyttämään Salesforcen tarjoamia standardikomponentteja mahdollisimman paljon, jotta muutoksien teko käyttöliittymän toimintaan olisi mahdollisimman yksinkertaista. Lopulta projektiin luotiin neljä mukautettua Lightning-komponenttia.



Kuva 11. Painikkeen kautta tehdyn palvelinpuolen kutsunnan tapahtumakulku

Mukautettujen komponenttien painikkeet tekevät vaatimusten mukaisia kutsuja Salesforcen palvelimelle rakennetuille tietokantaa käsitteleville metodeille.

5.6.1 Luo uusi -komponentti

Luo vastaus -komponentti sijaitsee tapausten näkymässä (kuva 5). Sen tarkoitus on luoda uusi, tyhjä vastaus. Tämä on tarpeen tapauksissa, missä viestipyyntö asiakkaalle ei ole tullut REST-rajanpinnan kautta, vaan esimerkiksi puhelimitse tai sähköpostilla.

```
crateNewViesti: function(c,e){

      var recId = c.get('v.recordId');
      var urlEvent = $A.get("e.force:navigateToURL");

      var action = c.get("c.newVastaus");
      action.setParams({ casString : recId});

      action.setCallback(c, function(response) {
        var state = response.getState();
        if (state === 'SUCCESS'){
                urlEvent.setParams({
                  "url": "/"+response.getReturnValue()
                });
             urlEvent.fire();
        } else {
             console.log(state + ' ' + response.getError());
        }
        });

        $A.enqueueAction(action);
}
```

Esimerkkikoodi 21. Apex-luokan kutsu palvelimelta ja navigointi osoitteen perusteella.

Komponenttiin sijoitettu nappi kutsuu ohjaimestaan metodia, joka kutsuu palvelimen Apex-luokkaa koodiesimerkin 21 mukaisesti. Apex-metodi ottaa vastaansa Id-parametrin, johon ohjaimen metodi syöttää auki olevan tapauksen tunnisteen. Apex-metodi tekee tietokantakyselyn tapauksen tunnisteella ja hakee sieltä asiakkaan nimen ja henkilötunnuksen. Luodaan uusi Application-objektin tietue, jolle annetaan tapaukselta haetut arvot. Lisäksi tietueelle annetaan kovakoodattuja arvoja, jotka pysyvät lähes aina samana. Näitä ovat esimerkiksi viestiin vastaava yhtiö, eli vastauksen tekijä, ja yhtiön osoite. Uusi tietue lisätään tietokantaan insert-operaatiolla. Lopuksi metodi palauttaa uuden, tietokantaan lisätyn tietueen tunnisteen paluuviestissä ohjaimeen. Ohjain sijoittaa palautetun tunnisteen Aura-kehyksen navigateToUrl-metodiin parametrina. Tämä vie käyttäjän uuden tietueen käyttöliittymän näkymään (kuva 6).

5.6.2 Luo vastaus -komponentti

Luo vastaus -komponentti sijaitsee Application-objektin tietueen näkymän oikeassa yläreunassa (kuva 7). Sen avulla luodaan vastaus auki olevaan viestipyyntöön. Komponentin toiminnan tarkoitus on luoda kopio auki olevasta tietueesta. Luodulle kopiolle annetaan luvussa 5.5 esitetyt oikeudet.

Komponentti käyttää vastaavaa toiminnallisuutta ohjaimessaan kuin esimerkkikoodissa 21. Palvelimen Apex-luokka saa parametrinä auki olevan tietueen tunnisteen, ja luo uuden tietueen sen kenttätietojen perusteella. Tämä on esitetty esimerkkikoodissa 22.

```
public static Application__c getViestit (Id id){

      Application__c applicationClone = new Application__c();
      Application__c app = [SELECT
                  Nimi,
                  Henkilötunnus__c,
                  Case__r.Id
      FROM Application__c
      WHERE Id =: id];

      Application__c applicationClone = app.clone(false, true, false, false);

      return applicationClone;
}
```

Esimerkkikoodi 22. Tietueen kopion luonti Apex-metodilla

Tietueen kenttien arvot haetaan parametrin Id-arvolla. Luodaan uusi tietue, jolle asetetaan clone-metodin avulla kenttien arvot. Lopuksi Apex-metodi palauttaa ohjaimelle luodun tietueen tunnisteen, ja se käyttää navigateToUrl-metodia siirtämään käyttäjän uuden tietueen näkymälle.

5.6.3 Vastaus-komponentti

Vastaus-komponentti luotiin, kun sen vaatimukset ylittivät Salesforcen vakiokenttien tarjoamat mahdollisuudet. Komponentti koostuu mukautetusta syöttökentästä, neljästä tekstin esitäyttöpainikkeesta sekä tallennus- ja keskeytä -painikkeista. Kenttä näyttää automaattisesti arvokseen Vastaus__c -kentän arvon. Tallenna-painike asettaa syöttökentän kyseisen arvon kentän arvoksi.

Komponentti käyttää hyväkseen aura:if -lausetta, jonka avulla on toteutettu komponentin painikkeiden ja kentän muokkauksen avautuminen. Painikkeet tulevat esille vain, kun kenttä avataan muokattavaksi painamalla kynä-ikonia kentän vierellä.

```
<aura:attribute name="inEdit" type="Boolean" default="false" />
<lightning:card>
     <div id="div" class="slds-m-horizontal--medium card">
          <aura:if isTrue="{!v.inEdit}">
               <lightning:recordEditForm objectApiName="{!v.sObjectName}" re-
cordId="{!v.recordId}" onsubmit="{!c.editState}">
                    <lightning:button type="button" class="btns"
aura:id="Tiedot" label="Tiedot" onclick="{!c.callMetaData}" variant="brand"/>
```

Esimerkkikoodi 23. aura:if-lauseella toteutettu syöttökentän ja painikkeen näyttäminen käyttöliittymällä

Komponentin ohjaimeen toteutettiin metodi, joka asettaa komponentin inEdit -attribuuttiin tarkoituksen mukaan joko todeksi tai epätodeksi. Tämä määrittää, näkyykö syöttökenttä ja sen painikkeet.

Painikkeista haetaan mukautetusta metadatasta painikkeen tunnisteen mukaan. Jokainen painike määritettiin painalluksesta kutsumaan ohjaimen metodia callMetadata kuten esimerkkikoodissa 23. Tämä metodi kuuntelee millä painikkeen tunnisteella sitä kutsutaan, ja sen mukaan tekee Apex-luokan kutsun.

```
callMetadata: function(component,e,h){
     var buttonId = e.getSource().getLocalId();

     if(buttonId === "Tieto"){
          var action = component.get("c.tieto");
     }else if(buttonId === "Lisatieto"){
          var action = component.get("c.lisatieto");

     action.setCallback(this, function(response) {
               var state = response.getState();
               if( state == "SUCCESS"){
                    component.set("v.fields", response.getReturnValue());
                }else{
                    console.log('There was a problem : '+response.getEr-
ror());
               }
     });
     $A.enqueueAction(action);
}
```

Esimerkkikoodi 24. Painike kutsuu Apex-luokkaa painikkeen tunnisteen mukaan.

Apex-luokka tekee tietokantakutsun parametrin mukaan, ja palauttaa sieltä saadun mukautetun metadatan merkkijonona. Palautettu merkkijono asetetaan kentän arvoksi.

Koska koko komponentti tehtiin mukautettuna, myös siihen lisätyt kentän arvon tallennus- ja muokkauksen keskeytys -painikkeet jouduttiin tuottamaan erikseen. Tallennuksessa kentän uusi arvo lähetetään ohjaimen kautta Apex-metodiin, joka suorittaa tietueelle update-toiminnon. Tämän jälkeen kutsutaan Aura-kehyksen tarjoamaa refreshView-metodia, joka päivittää näkymän käyttäjälle uudella kentän arvolla. Keskeytä-painike toteuttaa vain näkymän päivityksen ja asettaa komponentin inEdit-attribuutin arvoksi epätosi.

5.6.4 Lähetä vastaus -komponentti

Lähetä vastaus -komponentti sisältää vain painikkeen, jonka tarkoitus on koostaa auki olevasta tietueesta XML-dokumentti ja lähettää se asetettuun osoitteeseen. XML-dokumentin koostaminen on selitetty tarkemmin luvussa 5.7.

Komponentti kutsuu Apex-luokkaa, jonka metodit rakentavat XML-dokumentin. Ennen metodin kutsua se tarkistaa, onko yhteydenoton kannalta kriittiset kentät täytetty. Näitä ovat Vastauksen_pyytäjä__c ja Pyytäjän_OID__c, joista jälkimmäinen on kaavakenttä, joka täyttää itsensä ensimmäisen perusteella. Komponentti esittää kuvan 11 virheilmoituksen jommankumman kentän arvon puuttuessa.

Kuva 12. Viestin pyytäjän tai OID-tunnuksen puuttumisen virheilmoitus.

Mikäli XML-dokumentti koostetaan ja lähetetään oikein, mutta lähetyksessä ilmenee jokin virhe, ilmoitetaan asia kuvan 12 tapaisella virheviestillä. Tällöin viestin sisällöksi sisällytetään vastauksessa saatu HTTP-tila ja viestin mukana tullut viesti. Tämä tilanne voi olla esimerkiksi, jos vastaanottajan palvelin on epätoiminnassa. Onnistuneen lähetyksen jälkeen komponentti ilmoittaa viestissä onnistumisesti (kuva 10).

5.7 XML-dokumentin koostaminen ja lähettäminen

Ulospäin lähetettävä XML-dokumentti on rakenteeltaan hyvin samantyyppinen kuin REST-rajapintaan saapunut dokumentti. Siihen ei kuitenkaan käytetä kaikkia tietueen kenttien arvoja, ja on täten huomattavasti lyhyempi. XML-dokumentin koostamista varten luotiin CallOut-luokka, jonka metodit rakentavat eri osat dokumentista ja lähettävät sen valitulle vastaanottajalle. Koska dokumentissa on useita toistuvia kohtia, kuten esitetty esimerkkikoodissa 14, niiden kirjoitusta varten suunniteltiin yksittäisiä metodeja, jotta maksimoidaan dokumentin kirjoittamisen automatisointi ja minimoidaan käsin kirjoitettu osuus.

5.7.1 Kirjoituksen syntaksi

Kirjoittamiseen käytettiin Salesforcen tarjoamaa XmlStreamWriter-luokkaa. Lähtökohtaisesti luokka tarjoaa metodit XML-dokumentin kirjoittamiseen, mutta kirjoitus tapahtuu hyvin pitkälle käsin. Koodiesimerkissä 25 on esitetty, kuinka koodiesimerkin 14 näköinen ylätunniste kirjoitetaan sender-tagiin asti.

```
XmlStreamWriter writer = new XmlStreamWriter();

writer.writeStartDocument('UTF-8','1.0');
writer.writeStartElement(null, 'StandardBusinessDocument', null);
writer.writeAttribute(null, null, 'xmlns',
'http://www.unece.org/cefact/namespaces/StandardBusinessDocumentHeader');
writer.writeStartElement(null, 'StandardBusinessDocumentHeader', null);

writer.writeStartElement(null, 'HeaderVersion', null);
writer.writeCharacters('1.0');
writer.writeEndElement();

writer.writeStartElement(null, 'Sender', null);

writer.writeStartElement(null, 'Identifier', null);
writer.writeAttribute(null, null, 'Authority', 'MDS.HUB');
writer.writeCharacters(1.1.222.333.110.12345);
writer.writeEndElement();

writer.writeStartElement(null, 'ContactInformation', null);
writer.writeStartElement(null, 'ContactTypeIdentifier', null);
writer.writeCharacters('AA');
writer.writeEndElement();
writer.writeEndElement();
writer.writeEndElement(); //Close Sender
```

Esimerkkikoodi 25. Ylätunnisteen kirjoitus

Kyseisessä esimerkissä koodirivien kirjoituksen määrä on noin kaksinkertainen tuotettuun XML-dokumenttiin. Lisäksi jokainen avattu XmlStreamWriter-elementti joudutaan sulkemaan käsin. Tämä voi käydä hyvin työlääksi ja todentua erittäin hankalaksi käsitellä, ellei kirjoituksen osia paloitella metodeihin.

5.7.2 Ylätunnisteen kirjoitus

Lähetettävän dokumentin ylätunniste seuraa saapuneen dokumentin rakennetta. Iso osa siihen kuuluvista tageista ja niiden arvoista ovat staattisia, jotka kovakoodattiin dokumentin kirjoituksen yhteyteen.

Kutsuessaan CallOut-luokkaa ”Lähetä vastaus” -komponentti antaa sille parametrinä auki olevan tietueen tunnisteen, jonka perusteella luokka tekee tietokantakyselyn. Tietokantakyselyn yhteydessä tietueelta haetaan kaikki dokumentissa käytettävien kenttien arvot. Näin kenttien arvoja voidaan käyttää suoraan dokumenttia kirjoittaessa. Näihin kenttiin kuuluvat muun muassa ylätunnisteessa tarvittavat lähettäjän nimi ja lähettäjän OID, jotka kirjataan ylätunnisteeseen koodiesimerkin 14 tavoin.

Ylätunnisteeseen kirjataan teknisenä tietona lähettävän osapuolen OID-numero, johon pisteellä eroteltuna lisätään tietueen järjestelmän sisäinen tunnus. Näin dokumentille luodaan kaikkien verkostossa olevien järjestelmien yli kattava uniikki tunnistenumero, jota käsitellään vastaanottajan palauttamissa ApplicationResponse-vastausviesteissä. Vastausviesti on esitelty luvussa 5.7.5.

5.7.3 Juurielementin kirjoitus

Juurielementin osat jakautuvat toistuviin esimerkkikoodissa 14 esitettyihin ”Component” -tagin osiin. Nämä sisäiset tagit jakautuivat Code, Text, Entry ja Value osiin. Jokaisen luomiselle toteutettiin oma metodinsa. Esimerkkikoodi 26 on malli Entry-osuuden kirjoittamisesta.

```
public static void writeEntry(XmlStreamWriter writer, String code, String dis-
playName, String codeSystem, String codeSystemName){
     writer.writeStartElement(null, 'entry', null);
     writer.writeStartElement(null, 'observation', null);
     writer.writeAttribute(null, null, 'moodCode', 'EVN');
```

```
      writer.writeAttribute(null, null, 'classCode', 'OBS');
      writer.writeStartElement(null, 'code', null);
      writeCodeAttributes(writer, code, codeSystem, codeSystemName, display-
Name);
      writer.writeEndElement(); //end code
}
```

Esimerkkikoodi 26. Metodi Entry -tagin kirjoitukselle

Koska Entry-tagin lapsielementti ”Value” on jokaisessa Entryssä erilainen, piti tagien sulkeminen jättää metodien ulkopuolelle. Lisäksi ”Componentin” pohjustukselle kehitettiin oma metodinsa. Component-tagien ajoittaisen päällekkäisyyden takia myös ne suljetaan metodin ulkopuolella. Malli tästä on esimerkkikoodissa 27.

```
startComponent(writer, '40', 'Allekirjoitusaika', 'Allekirjoitusaika', co-
deSystem2004, ’Esimerkki');
      writeTextElement(writer, todayFinnishString, null);
      writeEntry(writer, '40', 'Allekirjoitusaika', codeSystem2004, ‘Esimerk-
ki');
      writeValue(writer, 'TS', effectivetime, null, null);
      writer.writeEndElement(); // end observation
      writer.writeEndElement(); // end entry
      writer.writeEndElement(); //end section
      writer.writeEndElement(); //end component 40
```

Esimerkkikoodi 27. Allekirjoitusaika-tagin kirjoitus dokumenttiin.

Esimerkkikoodissa kirjoitetaan Allekirjoitusaika-tagin Component-osuus, jossa kutsutut metodit kirjoittavat omat osuutensa. Lopuksi jokainen tagi suljetaan käsin kutsumalla writeEndElement-metodia. Näin kaikki juurielementin alle sijoittuvat component-tagit on rakennettu käsin mahdollisimman paljon metodeja hyväksikäyttäen. XML-dokumentin rakennuksen jälkeen se muutetaan merkkijonoksi käyttäen XmlStreamWriter-luokan getXmlString-metodia.

5.7.4 XML-dokumentin lähetys

Lähetyksessä käytetään Salesforcen tarjoamaa HttpRequest-luokkaa. Vastaanottavaksi osoitteeksi haetaan mukautetuista asetuksista asiakkaalle määritelty osoite. Viestit lähtevät aina asiakkaan reitittimen suuntaan, joka jakaa viestit eteenpäin viestipyynnön tehneen kontaktin OID-tunnuksen perusteella.

Viestiin tulee vastauksena vastauskoodi, kun yhteydenotto vastaanottajaan on tehty. Vastauskoodi palautetaan ”Lähetä viesti” -komponentin ohjaimeen, joka tuottaa käyttöliittymälle sen mukaisen ilmoituksen. Vastauskoodin lisäksi asiakkaalta eteenpäin toimitettu taho palauttaa XML-dokumentin muodossa olevan vastauksen käyttöliittymälle lukemisen onnistumisesta.

5.7.5 ApplicationResponse XML -dokumentti

ApplicationResponse-dokumentti palautetaan järjestelmään aina ulkoisen palveluntarjoajan vastauksena lähetetystä viestistä. ApplicationResponse sisältää tiedon siitä, onko vastaanotetun viestin vastaanotto ja lukeminen ulkoisen palveluntarjoajan näkymälle onnistunut. ApplicationResponse otetaan vastaan samassa REST-rajapinnassa, missä viestipyynnötkin, jossa se ajetaan XML-jäsentimen läpi. Jäsennin käsittelee dokumentin sisällön ja toteuttaa sen vaatimat operaatiot. Näiden määrittely on luvussa 5.7.6.

```
<cac:Response>
     <cbc:ReferenceID>1.1.222.333.110.12345.1015131646710</cbc:ReferenceID>
     <cbc:ResponseCode listID="UNCL4343">RE</cbc:ResponseCode>
     <cbc:Description>Failure from receiving party</cbc:Description>
```

Esimerkkikoodi 28. Ote ApplicationResponsesta.

ApplicationResponse sisältää tekniset tiedot viestin lähetyksestä. ReferenceID-tagi sisältää lähetetyn dokumentin ylätunnisteeseen kirjatun globaalisti uniikin tunnistenumeron. Näin dokumentin sisältävä data voidaan liittää tapaukseen, johon lähetetty viesti liittyy.

5.7.6 ApplicationResponse-toimenpiteet

Jäsennin tutkii esimerkkikoodissa 28 esitetyt ResponseCode- ja Description-tagit. ResponseCodeja on kahdenlaista:

- RE (Rejected) – Viestiä vastaanottaessa tai luettaessa kohdattiin ongelma.
- AP (Accepted) – Viesti vastaanotettiin onnistuneesti.

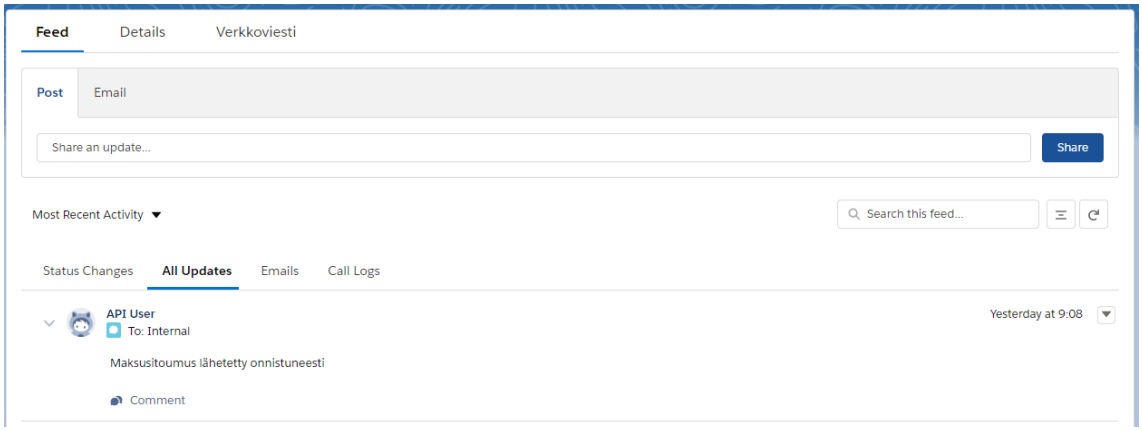
Molemmissa tapauksissa viestin sisällön perusteella luodaan uusi Chatter-viesti tietueeseen liittyvälle tapaukselle. Jos ResponseCode on ”AP”, voidaan todeta, että viesti toimitettiin onnistuneesti perille, eikä lisäoperaatioita tarvita.

Jos ResponseCode on ”RE”, otetaan dokumentin Description -tagin arvo talteen, ja se lisätään Chatter-viestiin. Lähettäneen viestin asiakaspalvelija lisätään Chatter-viestiin @-merkinnällä. Näin virhetapauksissa asiakaspalvelijaa tiedotetaan välittömästi, ja minimoidaan virheen elinkaari.

```
ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
      ConnectApi.MentionSegmentInput mentionSegmentInput
            = new ConnectApi.MentionSegmentInput();
      ConnectApi.MessageBodyInput messageBodyInput
            = new ConnectApi.MessageBodyInput();
      ConnectApi.TextSegmentInput textSegmentInput
            = new ConnectApi.TextSegmentInput();

      messageBodyInput.messageSegments
            = new List<ConnectApi.MessageSegmentInput>();

      if(isApplicationResponse && responseCode == 'RE'){
            mentionSegmentInput.id = userId;
            messageBodyInput.messageSegments.add(mentionSegmentInput);

            textSegmentInput.text = '\n' + Viestin lukeminen epäonnistui. Ota
yhteys ylläpitäjään. Virheviestin sisältö: ' + '\n' + applicationResponseSt-
ring + '\n' + 'Application numero: ' + issueId + '\n' + 'Lähetyksen aika ja
päivämäärä: ' + issueTime + ' ' + issueDate;

            messageBodyInput.messageSegments.add(textSegmentInput);

            feedItemInput.body = messageBodyInput;
            feedItemInput.feedElementType = ConnectApi.FeedElementType.FeedItem;
            feedItemInput.subjectId = caseId;

            ConnectApi.FeedElement feedElement = ConnectApi.ChatterFeeds.post-
FeedElement(null, feedItemInput);
```

Esimerkkikoodi 29. Chatter-viestin luonti Apex-luokassa.

Chatter-viestin luontiin käytetään Salesforcen ConnectApi-luokkaa. Luokan kautta voidaan luoda uusi viestin sisältö ja lisätä siihen @-merkintä esimerkkikoodin 29 esittämällä tavalla. Sisältöelementin feedItemInput-muuttujalle asetetaan viestin sisältö ja kohdetapaus, joka on haettu tietokantakyselyllä ApplicationResponsen palauttaneen uniikin tunnuksen avulla. Kyseinen muuttuja asetetaan postFeedElement-metodin parametriksi, joka tuottaa Chatter-viestin.



Kuva 13. Chatter-viesti tapauksella.

Tätä toiminnallisuutta käytetään yleiseen virheiden tarkasteluun. Vaikka virheviestiä ei saapuisikaan, voi asiakaspalvelija silti varmistaa, onko onnistunut Chatter-viesti luotu. Jos viestiä ei ole, voi virheenvälityksessä olla häiriöitä ja ulkoista palveluntarjoajaa voidaan huomauttaa asiasta mahdollisimman tehokkaan viankorjauksen takaamiseksi.

5.8 Testaus

Ohjelmiston testaus toteutettiin käyttäjäpohjaisesti, eli kun osa ohjelmaa oli kehittäjän arvion mukaan testausvalmis, se siirrettiin kehitysorganisaatiosta (engl. Development, Dev) käyttäjätestauksen organisaatioon (User Acceptance Testing, UAT). Asiakas toteutti testauksen omalla testidatallaan siten, että se kattaa heidän vaatimuksensa ja kaikki mahdolliset käyttötapaukset on käyty läpi. XML-dokumenttien vastaanoton REST -rajapinnan testaukseen käytettiin ulkoisten ohjelmien palvelua, kuten Postman- ja Restlet-ohjelmia. Dokumenttien lähetyksen ja XML-struktuurin tarkistukseen käytettiin RequestBin-verkko-ohjelmaa sekä ilmaisia XML–formatointiverkko-ohjelmia.

Koodin testikattavuuden yli vaaditun 75 % toteamiseksi luotiin testiluokat kaikille käytetyille Apex-luokille.

```
static testMethod void testHandling(){
      XMLUtils.handleXMLParser(xmlString);

      Case testCase = new Case(
          Tyyppi__c = 'Tyyppi',
          Status = 'Saapunut',
```

```
            Henkilötunnus__c = '010376-5115',
            nimi__c = 'testName',
        );
        insert testCase;

        Application__c app = new Application__c(
                Case__c = testCase.Id,
                Henkilotunnus__c = '010376-5115',
                Sukunimet_ja_etunimet__c = 'testName',
                Type__c = 'Tyyppi'
        );
        insert app;

        system.assertEquals(testCase.Henkilötunnus__c, app.Henkilötunnus__c);
        system.assertEquals(testCase.nimi__c, app.Sukunimet_ja_etunimet__c);
        system.assertNotEquals(app.Henkilotunnus__c, '');
    }
```

Esimerkkikoodi 30. Kooditestausta XML-jäsentimelle.

Vaikka kooditestaus on minimaalinen, voidaan sen avulla kuitenkin todeta tiettyjen koodin osien varma toiminta. Esimerkkikoodin 30 tapaisia testimetodeja luodaan eri tapauksille siten, että se kattaa tarpeellisen määrän koodiriveistä. Samalla system.assert-metodeilla voidaan testata, että kentillä on niille kuuluvat arvot.

Käyttöliittymän osia, eli tässä työssä Lightning Componentteja ei testata. Jos niiden kuitenkin niiden ohjaimet käyttävät palvelimen Apex-luokkia hyväkseen, on nämä luokat testattava.

6 Yhteenveto

Tämän insinöörityön tarkoituksena oli rakentaa dynaamisesti toimiva XML-dokumenttien vastaanotto- ja lähetyspalvelu Salesforce.com-alustalle REST-arkkitehtuurimallia käyttäen. Lisäksi toteutettiin helposti lähestyttävä ja dynaaminen käyttöliittymä, joka vastaa asiakkaan asiakaspalvelijoiden tarpeita.

Tekniseltä puolelta hankalin toteutettava oli XML-dokumentin jäsennys siten, että jäsennyksessä ei kestä kauaa verkkoviestien saumattoman liikkumisen takaamiseksi. Tähän tehokas ratkaisu oli jäsennys rekursiivisessa For-silmukassa, joka käyttää sisällään Apexin Switch On -rakennetta keskeyttämään silmukan toiminnan tietyin toimenpiteen tekemiseksi. Myöskään ikuisen silmukan pelotetta ei ollut, kun jäsennyksessä käytiin

XML -dokumenttia juurielementistä hierarkiassa alaspäin kaikki lapsielementit läpi. Silmukka loppuu, kun lapsielementtejä ei enää ole.

XML-dokumentin kirjoittamisen toteutus ei sinänsä tuottanut ongelmia, mutta se haluttiin tuottaa mahdollisimman fiksusti siten, että mahdollisesti projektiin tulevaisuudessa liittyvät kehittäjät ymmärtävät sen toiminnan. XML-dokumentti hyvin pitkälle joudutaan kirjoittamaan käsin, mutta useat metodit lyhentävät koodirivien määrää huomattavasti, ja täten selkeyttävät koodin ymmärtämistä.

Projektin hallinnalliselta puolelta päänvaivaa tuotti kommunikaatio-ongelmat ulkoisten palveluntarjoajien kanssa, erityisesti XML-dokumentin ymmärtämisen suhteen. Massiivisen dokumentin lukemiseen tarjottiin minimaaliset ohjeet, jotka eivät selittäneet useiden näennäisesti satunnaisten numeroarvojen tarkoitusta. Tiedusteluihin asiasta jouduttiin kuluttamaan ylimääräisiä työpäiviä harmillisen paljon. Lisäksi asiakas oli pitkään epätietoinen käyttöliittymän ulkonäöstä, ja mitä siltä haluaa. Tämän takia sen viilaamiseen käytettiin turhan paljon aikaa.

Jatkokehityksenä käyttöliittymälle voisi tehdä pieniä ns. elämän laadun parannus -muutoksia, kuten asettaa mukautettu Vastaus__c -kentän komponentti seuraamaan käyttäjän vertikaalista liikkumista sivulla. Täten vastausta kirjoittaessa ei tarvitsisi rullata sivua edestakaisin, kun halutaan tarkistaa tietoa tietueen alaosista. Koko järjestelmän toteutus olisi ollut huomattavasti helpompi toteuttaa ja tehokkaampaa käyttää, jos REST-rajapintoihin lähetettäisi JSON-dokumentteja. Näiltä dokumenteilta Salesforce osaa dataa suoraan objektin tietueen kentille ilman näin kattavaa ja laajaa jäsentämisoperaatiota. Tämä tietysti tarkoittaisi, että kaikki ulkoiseen palveluun liittyvät tahot joutuisivat päivittämään järjestelmänsä kykeneväksi vastaanottamaan JSON-viestejä, joka voi olla paljon vaadittu.

Työssä opittiin REST-rajapinnan helppoutta sekä miten venyvä ja tehokas arkkitehtimalli se on. Työ sisälsi laajoja osia back- ja front end -ohjelmointia, joka antoi tulevaisuuden vastaavanlaisen projektin suunnittelulle ja ohjelmoinnille todella vahvan pohjan.

Loppupäätelmänä voidaan todeta, että ohjelmisto toteuttaa sille asetetut tavoitteet ja asiakas on tyytyväinen tarjottuun tuotteeseen. Salesforce taipuu asetettuihin vaatimuksiin mallikkaasti ja tarjoaa käyttäjälleen miellyttävän käyttökokemuksen.

Lähteet

1 Salesforce – Investor Relations. Verkkodokumentti https://investor.salesforce.com/about-us/investor/investor-news/investor-news-details/2018/Salesforce-Announces-Record-Fourth-Quarter-and-Full-Year-Fiscal-2018-Results/default.aspx Luettu 09.03.2019.

2 SOAP API Developer Guide – Objects Basics and Concepts. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.api.meta/api/sforce_api_objects_concepts.html Luettu 09.03.2019.

3 Trailhead – Get Started with Chatter. Verkkodokumentti. https://trailhead.salesforce.com/en/content/learn/modules/chatter/chatter_intro Luettu 12.03.2019.

4 Apex Developer Guide – Chatter in Apex. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/connectAPI_overview.htm Luettu 13.03.2019.

5 SOAP API Developer Guide – Objects Basics and Concepts. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.api.meta/api/sforce_api_objects_concepts.html Luettu 09.03.2019.

6 Apex Developer Guide – Apex Triggers. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_triggers.htm Luettu 09.03.2019.

7 Apex Developer Guide – Apex Unit Testing. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_testing_unit_tests.htm Luettu 09.03.2019.

8 Apex Developer Guide – Test Web Service Callouts. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_callouts_wsdl2apex_testing.htm Luettu 09.03.2019.

9 Apex Developer Guide – Dom Namespace. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_namespace_Dom.htm Luettu 10.03.2019.

10 Apex Developer Guide – XmlNode Class. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes_xml_dom_xmlnode.htm#apex_classes_xml_dom_xmlnode Luettu 10.03.2019.

11 Apex Developer Guide – XmlStreamWriter Class. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes_xml_XmlStream_writer.htm Luettu 10.03.2019.

12 SOQL and SOSL Reference – SOQL Salesforce… Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql.htm Luettu 10.03.2019.

13 Apex Developer Guide – Data Manipulation Language. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_dml.htm Luettu 10.03.2019.

14 Lightning Platform. Verkkodokumentti. https://developer.salesforce.com/lightning Luettu 10.03.2019.

15 Lightning Aura Components Developer Guide. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/access_intro.htm Luettu 11.03.2019.

16 Apex Developer Guide – AuraEnabled Annotation. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes_annotation_AuraEnabled.htm Luettu 11.03.2019.

17 The history of REST APIs. Verkkodokumentti. https://blog.readme.io/the-history-of-rest-apis/ Luettu 11.03.2019.

18 A Bried History of XML. Verkkodokumentti. https://ccollins.wordpress.com/2008/03/03/a-brief-history-of-xml/ Luettu 11.03.2019.

19 XML -Elements. Verkkodokumentti. https://www.tutorialspoint.com/xml/xml_elements.htm Luettu 11.03.2019.

20 W3Schools – XML Namespaces. Verkkodokumentti. https://www.w3schools.com/xml/xml_namespaces.asp Luettu 11.03.2019.

21 Representational State Transfer – REST. Verkkodokumentti. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm Luettu 11.03.2019.

22 Apex Developer Guide – Apex REST Methods. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_rest_methods.htm Luettu 11.03.2019.