



Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Niklas Räsänen

Salesforcen REST-rajapinta uusien asiakkaiden luomiseen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

28.11.2019

| | |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| Tekijä Otsikko | Niklas Räsänen Salesforce REST-rajapinta uusien asiakkaiden luomiseen |
| Sivumäärä Aika | 25 sivua 28.11.2019 |
| Tutkinto | Insinööri (AMK) |
| Tutkinto-ohjelma | Tietotekniikka |
| Ammatillinen pääaine | Ohjelmistotekniikka |
| Ohjaajat | Lehtori Simo Silander Sovellusarkkitehti Arto Mutanen |
| <p>Insinööritöön tarkoituksena oli kehittää asiakkuuksien hallinnoimiseen käytettävä järjestelmä hyödyntäen REST-rajapintaa Salesforce-alustalla. Siinä käytettiin Salesforce REST -arkkitehtuuria. Uusien asiakkuuksien luominen sekä jo valmiiksi olevien päivitys olivat työn keskeisimmät prosessit. Näitä prosesseja varten luotiin oma hallinnointiluokka, joka teki kokonaan prosessin luonti ja päivitys DML-operaatiokutsut. Myös datan validointi, formatointi sekä muunnokset tehtiin hallinnointiluokassa.</p> <p>Insinööritöössä kerrotaan Salesforcen toiminnasta, REST-rajapintamallista ja projektissa käytetyistä ohjelmointikielistä. Itse projektin toteuttaminen jakautuu seuraaviin alueisiin: projektin suunnitteluun, tietokantaan ja datamalliin, uuden asiakkuuden luomiseen, testaamiseen ja asentamiseen.</p> <p>Insinööritöön tuloksena syntyi järjestelmä, jota voidaan hyödyntää uusien asiakkaiden tallentamiseen ja jo olemassa olevien asiakkaiden tietojen päivittämiseen REST-rajapinnan avulla.</p> | |
| Avainsanat | Salesforce, APEX, SOQL, REST, Rajapinta |

| | |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| Author Title | Niklas Räsänen Salesforce REST Interface for Creating New Customers |
| Number of Pages Date | 25 pages 28 November 2019 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information and Communications Technology |
| Professional Major | Software Engineering |
| Instructors | Simo Silander, Senior Lecturer Arto Mutanen, Senior Application Architect |
| <p>The purpose of this bachelor's thesis was to develop a customer relationship management system using the REST interface on the Salesforce platform using the Salesforce REST architecture. Creating new client accounts and updating existing ones were key processes in the project. A custom Apex handler class was created for these processes, which made all process the creation and update DML operation calls. Data validation, formatting and re-mapping were also done in the handler class.</p> <p>The thesis describes the operation of Salesforce, the REST interface model and the programming languages used in the project. The project implementation itself is divided into the following areas: project design, database and data model, creating, testing and installing a new customer relationship.</p> <p>The result of the study was a system that can be utilized to store new customers and update existing customer information using the REST interface.</p> | |
| Keywords | Salesforce, APEX, SOQL, REST |

Sisälllys

Lyhenteet

| | | |
|-----|-------------------------------------------------|----|
| 1 | Johdanto | 1 |
| 2 | Salesforce.com | 2 |
| 2.1 | Salesforce | 2 |
| 2.2 | Tietokannan rakenne | 3 |
| 2.3 | Objektit | 3 |
| 2.4 | Kentät | 4 |
| 2.5 | Mukautettu metatieto | 5 |
| 3 | Käytetyt ohjelmointi- ja tietokantakyselykielet | 6 |
| 3.1 | Apex | 6 |
| 3.2 | Syntaksi | 6 |
| 3.3 | SOQL | 8 |
| 4 | REST-rajapinta | 9 |
| 4.1 | Asiakas ja palvelin | 9 |
| 4.2 | Tilattomuus | 9 |
| 4.3 | Operaatiometodit sekä niiden käyttö | 10 |
| 4.4 | Salesforce sekä REST | 12 |
| 4.5 | JSON | 13 |
| 5 | Projektin toteutus | 14 |
| 5.1 | Suunnittelu | 14 |
| 5.2 | Tietokanta ja datamalli | 15 |
| 5.3 | Uuden asiakkaan luominen | 17 |
| 5.4 | Testaaminen | 24 |
| 6 | Yhteenveto | 27 |
| | Lähteet | 28 |

Lyhenteet

| | |
|------|------------------------------------------------------------------------------------------------|
| SF | Salesforce-sanan lyhenne. |
| SaaS | Software as a Service. Termi ohjelmiston kauppaamisesta palveluna. |
| CRM | Customer Relationship Management. Asiakkuudenhallinta. |
| Apex | Salesforcen kehittämä ohjelmointikieli, joka on hyvin samankaltainen kuin Java. |
| SOQL | Salesforce Object Query Language. Salesforcen oma tietokantakyselykieli. |
| SQL | Structured Query Language. Standardoitu kyselykieli. |
| DML | Data Manipulation Language. Salesforcen datanmuokkauskieli. |
| SOAP | Simple Object Access Protocol. Viestintäprotokolla tiedon välitykseen. |
| API | Application Programming Interface. Ohjelmointirajapinta. |
| REST | Representational State Transfer. Verkkopalveluarkkitehtuuri internetrajapintojen toteutukseen. |
| JSON | JavaScript Object Notation. Tiedon välitykseen käytettävä dataformaatti. |

1 Johdanto

Työn aiheena on Salesforce-pilvipalveluun kehitetyn asiakkuuksien luonti-ja-päivitysmahdollisuuden toteuttaminen REST-rajapinnassa Fluido Oy:n asiakkaalle. Koska pilvipohjainen palvelu mahdollistaa ketterän kehityksen, Salesforce soveltuu loistavasti asiakkaan tarpeisiin. Salesforcen yksi tärkeimmistä tuotteista on CRM-järjestelmä, eli asiakkuuksienhallintajärjestelmä.

Insinööritö käsittelee finanssialan vaatimuksia täyttävää kehitystä Salesforce-ympäristössä. Finanssialan vaatimuksiin kuuluvat joustava, ketterä sekä luotettava tuotekehitys. Työssä perehdytään Salesforcen datamalliin, ohjelmointikieleen, REST-rajapintaan sekä niiden hyödyntämiseen tämän projektin toteutuksessa. Työssä toteutetaan uuden asiakkuuden luonti, päivitys sekä hallinta Salesforcen tarjoamia standardeja sekä mukautettuja ominaisuuksia hyödyntäen.

Työn alussa käydään lyhyesti läpi Salesforce.com-yrityksen rakennetta, Salesforce-alustaa ja sen eri komponentteja. Tämän jälkeen perehdytään Salesforcen käyttämiin ohjelmointi- ja tietokantakyselykieliin. Seuraavaksi perehdytään työssä käytettävään REST (Representational State Transfer) -arkkitehtuuria ja Salesforcen soveltuvuutta REST-rajapintojen toteutukseen sekä JSON-formaattiin. Lopuksi työssä käydään läpi järjestelmän toteutus vaihe vaiheelta ja tehdään yhteenveto työstä.

2 Salesforce.com

Salesforce.com on amerikkalainen maailmanlaajuisesti toimiva pilvipohjainen asiakkuudenhallintajärjestelmiä (CRM, Customer Relationship Management) tarjoava yritys, jonka perusti vuonna 1999 Marc Benioff. Yrityksen ensimmäinen toimisto sijaitsi Benioffin kotona Telegraph Hillissä, San Franciscossa. Yrityksessä oli Benioffin lisäksi kolme henkilöä töissä: Parker Harris, Frank Dominguez ja Dave Moellenhoff. Heidän tavoitteenaan oli tarjota ohjelmistoja SaaS (Software as a Service) -palveluina. Alusta asti yrityksen mottona oli ”No Software”, eli estää niin sanottujen Legacy-järjestelmien syntyminen. [1.] Legacy-järjestelmä on yleisesti sovellus tai muu tekninen ratkaisu, joka vaatii paljon ylläpitoa sekä voi vaatia monimutkaisia muokkauksia ja paikkauksia. Usein järjestelmää ei voida tehdä riittävän joustavaksi, tehokkaaksi tai turvalliseksi. [2.]

2000-luvulla yritys lanseerasi kaksi yrityksen tärkeintä tapahtumaa: Dreamforce ja Ohana, jotka vaikuttavat edelleen yrityksen toimintaan, työntekijöiden hyvinvointiin sekä yrityksen imagoon [1]. Dreamforcen neljä päivää kestävässä tapahtumassa oli viime vuonna 171 000 osanottajaa ja yli 10 miljoonaa ihmistä seurasi tapahtumaa livenä. Tapahtuman tarkoituksena on oppia, jakaa, yhdistyä, tuottaa ja raivata uusia polkuja yhdessä. [3.] Ohana on yrityksen sisäinen tapahtuma, jossa on tarkoitus pukeutua sekä koristella työpaikka Havaiji-tyyliseksi. [1.]

2.1 Salesforce

Salesforce oli 2016 vuoden suosituin CRM-alusta 26 %:n markkinaosuudella ja 6,3 miljardin CRM-tuotteen liikevaihdolla. [5.] SF:n tarjoamiin pilvipalveluihin kuuluvat Marketing Cloud, Service Cloud, Community Cloud, IoT Cloud, Health Cloud ja Sales Cloud. Marketing Cloudin avulla voidaan kehittää markkinoinnin automaatio- ja analyytiikkaohjelmistoja, joita voidaan hyödyntää esimerkiksi sähköposti-, mobiili-, sosiaaliseen ja verkkomarkkinointiin. Service Cloud on asiakassuhteiden hallinta -alusta (CRM) asiakaspalvelulle ja -tuelle. Se perustuu yrityksen CRM-ohjelmistoon myynnin ammattilaisille. Community Cloud on sosiaalinen alusta, joka on suunniteltu yhdistämään ja helpottamaan viestintää organisaation työntekijöiden, kumppaneiden ja asiakkaiden välillä. IoT Cloud on alusta, joka on tarkoitettu tallentamaan ja käsittelemään esineiden Internet (IoT) -

dataa. Health Cloud on alusta, joka on tehty terveydenhuollon potilassuhteiden hallintaratkaisuksi.

Tämän työn kannalta näistä oleellisin on Salesforce Sales Cloud. Sen on CRM-alusta, jonka tarkoituksena on hallita myyntiä, markkinointia ja asiakaspalvelua niin yritykseltä yritykselle (B2B) kuin yritykseltä asiakkaalle (B2C). [4.] Sales Cloud on täysin omien toiveidensa mukaan muokattavissa oleva tuote, joka on tarjottu SaaS-palveluna nettiselain- sekä mobiilikäyttöön. Sales Cloud tuo kaikki asiakastiedot yhteen integroituna alustana, joka sisältää markkinoinnin, liidin eli tuotteesta kiinnostuneet henkilöt, myynnin, asiakaspalvelun, myynnin analytiikan ja tarjoaa oikeuden tuhansiin ohjelmistoihin AppExhangerin kautta. [6.] AppExchange on Salesforceen tarjoama markkinapaikka asiakkaiden, ohjelmistojen ja partnereiden kehittämiin sovelluksiin. [7.]

2.2 Tietokannan rakenne

Salesforcen datamalli on objektipohjainen. Tietokannan rakennetta voi verrata relaatio-tietokantaan, jossa yksi taulu on Salesforceissa objekti. Objektin kentät ovat kolumneja, tietueet ovat rivejä taulussa. Tietokannassa tarvittavat tiedot pyritään jakamaan taulukoihin eli objekteihin siten, että yksi tieto tallennetaan vain yhteen paikkaan. Salesforcen tietokannassa tallennetaan myös tieto siitä, miten eri objektit liittyvät toisiinsa. [8.]

2.3 Objektit

Salesforcessa on valmiiksi tehtyjä objekteja (esim. Tili). Näitä kutsutaan vakio-objekteiksi (standard objects). Vakio-objekteille voi lisätä tai niitä voi poistaa kenttiä, mutta itse objektia ei voi poistaa.

Halutessa voidaan luoda omia objekteja, joita kutsutaan mukautetuksi objekteiksi (custom objects). Näiden modifiointimahdollisuudet ovat vakio-objekteja laajemmat, sekä mahdollinen objektin poisto on sallittua.

Mukautetun objektin nimi täytyy kuitenkin olla uniikki. Objektin rajapintanimi (API Name) on myös uniikki, se sisältää päätteen ”__c”. Rajapintanimi on koodissa käytettävä nimi

mukautetulle objektille. Vakio-objekteilla rajapintanimi on sama kuin objektin nimi, esimerkiksi "Tili". [9.]

Objektin tietuetyyppi (Recordtype) määrittelee, mitä valikkotyyppisiä arvoja on käytössä tietyllä tyyppillä ja mitä sivuasettelua (Page Layout) käytetään. Jokaisella objektilla on pää tietuetyyppi, mikä on standardi jokaiselle uudelle tietueelle. Mikäli lisätään useita tietuetyyppejä, käyttäjän tulee valita uudelle tietueelle tyyppi sitä luodessa. Koodissa tietuetyyppejä käytetään esimerkiksi tietuetyypin tunnisteiden avulla eli id:n avulla (esimerkkikoodi 1).

```
RecordType recordType = [SELECT Id,DeveloperName FROM RecordType WHERE SubjectType = 'Tili' AND DeveloperName = 'UusiTietueTyyppi'];

Tili uusiTili = new Tili(Name = 'EsimerkkiTili', RecordTypeId = recordType.Id);

insert uusiTili;
```

Esimerkkikoodi 1. Tietuetyypin haku sekä käyttö uuden Tilin luonnin yhteydessä.

Sivuasetteluja voidaan määritellä objekteille eri käyttötarkoitusten mukaan. Objektien kenttiä, nappeja, mukautettuja linkkejä ja objektiin kuuluvia listoja voidaan piilottaa tai näyttää esimerkiksi tietuetyypin perusteella.

Objekteille voi lisätä vahvistussääntöjä (Validation Rule), jotka vahvistavat tiedon oikeellisuuden objektiokohtaisesti. Vahvistussäännön ehdot määritellään kaavakentällä, jonka palautusarvot ovat "tosi" tai "epätosi". Vahvistussäännöt estävät tiedon tallentamisen, mikäli säännön ehdot tulevat toteen. Käyttäjän määrittelemä virheilmoitus annetaan käyttäjälle "tosi"-arvolla. [10.]

2.4 Kentät

Mukautettuja kenttiä voi lisätä objekteille, kuten edellä mainitut mukautetut objektit. Myös kentät saavat myös uniikin rajapintanimen, joka sisältää päätteen "__c". Uusien kenttien tyyppi valitaan vaatimuksien mukaisesti. Näitä tyyppieitä ovat esimerkiksi teksti, numero, valintaluettelo, hakusuhde ja päiväys. [11.]

2.5 Mukautettu metatieto

Mukautettujen metatietotyyppien (Custom Metadata Types) avulla pystytään määrittelemään uudesti käytettävä toiminnallisuus, joka määrittelee käyttäytymisen metatietotyyppin perusteella. Esimerkiksi eri maille voi lisätä eri metatietoa, joita voidaan hakea eri kooditoteutuksissa maan perusteella. Näin ollen koodia ei tarvitse muokata. Jos arvoja tarvitsee muokata, niin niitä muokataan metatiedossa. Tietorakenteeltaan mukautetut metatiedot muistuttavat mukautettuja objekteja: niitä voi luoda, muokata ja poistaa. Tietueita ja metatietoja voi ottaa käyttöön toisesta Salesforce-organisaatiosta toiseen pakettien avulla eli esimerkiksi testi ympäristöstä viedä toiseen tuotanto ympäristöön. [12.]

Luodaan Tili-objektille Tilin Arvot metadata, jonka avulla voidaan hakea maakohtainen Tilin alkuarvo. Kuvasta 1 nähdään luodut arvot ja maakohtainen esimerkki kuvasta 2.

Tilin Arvot

Näytä: Kaikki ▼ [Luo uusi näkymä](#)

| Toiminto | Otsikko ↑ | Tilin Arvo Nimi |
|--------------------------------------------------|------------------------------|-----------------|
| Muokkaa Poista | Norjan Arvo | Norjan_Arvo |
| Muokkaa Poista | Ruotsin Arvo | Ruotsin_Arvo |
| Muokkaa Poista | Suomen Arvo | Suomen_Arvo |

Kuva 1. Tilin Arvot metatietotyyppit.

Tilin Arvo

[« Palaa luetteloon: Tilin Arvot](#)

Lisätiedot: Tilin Arvo

[Muokkaa](#) [Poista](#) [Kloonaa](#)

| | |
|-----------------|-------------|
| Otsikko | Suomen Arvo |
| Tilin Arvo Nimi | Suomen_Arvo |
| Arvo | 1 000,00 |

Kuva 2. Suomen Arvo metatietotyyppinä.

Koodissa nämä arvot voidaan hakea, esimerkiksi `QualifiedApiName`. Esimerkkikoodissa 2 haetaan lista Tilin Arvoja ja Tilin Arvo metadatalistasta otetaan `get`-metodia hyödyntäen Suomen arvo, joka sijoitetaan `defaultArvo` integeriin ja palautetaan.

```
List<Tilin_Arvo__mdt> creditLimit;
Integer defaultArvo;
creditLimit = [SELECT QualifiedApiName FROM Tilin_Arvo__mdt];
defaultArvo = Integer.valueOf(creditLimit[0].get('Suomen_Arvo'));
return defaultArvo;
```

Esimerkkikoodi 2. Suomen Tilin Arvot-metadatan hakeminen.

3 Käytetyt ohjelmointi- ja tietokantakyselykielet

3.1 Apex

Salesforcen kehittämä Apex on oliopohjainen vahvasti tyypitetty ohjelmointikieli, jossa käytetään javamaista syntaksia, mutta Apexilla toteutettuja ohjelmia voidaan käyttää vain Salesforcen-ympäristössä. Java- ja C#-ohjelmointikieliä muistuttavan Apexin koodi on ytimekästä ja helppoa kirjoittaa.

Apexin avulla voidaan lisätä kustomoitua liiketoimintalogiikkaa useimpiin järjestelmän tahtumiin, kuten nappien toiminnallisuuksiin, Apex-laukaisimiin (Apex Triggers) sekä Visualforce-sivujen kontrollereihin. Visualforce on web-kehityskehys, jonka avulla voidaan rakentaa käyttöliittymiä mobiili- ja työpöytäsovelluksille. [13.]

Apex-laukaisimien avulla pystytään suorittamaan mukautettuja toimintoja ennen tai jälkeen, kun Salesforce tietueita on luotu, muokattu tai poistettu.

Apexin sisäänrakennettuja toiminnallisuuksia ovat DML-operaatiot (Data manipulation language), SOQL (Inline Salesforce Object Query Language) ja SOAP (Simple Object Access Protocol). [13.]

3.2 Syntaksi

Apexissa on Javasta tuttuja primitiivisiä datatyppejä kuten Integer, Boolean, String, koelmia (Collections), listoja, luoteltuja tyyppejä (enum). Uuden muuttujan esittely tapahtuu datatyyppin ja muuttujan nimen esittelyllä. [14.] Esimerkkikoodissa 3 käydään läpi erityyppisten muuttujien esittelyä.

```
// Objektit
Contact kontakti = new Contact();
MukautettuObjekti__c omaObjekti = new MukautettuObjekti();

// Primitiiviset datatyypit
Integer vuosiluku = 2019;
Boolean onTosi = true;
String kommentti = 'Tämä on kommentti';

// Kokoelmat
List<Integer> vuosiluvut = new List<Integer>();
public enum vuodenAjat{Talvi, Kevät, Kesä, Syksy};
Map <String, String> kissat = new Map<String,String>{ 'pieni' => 'matti',
'suuri' => 'katti'};

// Käyttäjän ja järjestelmän määrittelemät Apex luokat
mukautettuLuokka testiLuokka = new mukautettuLuokka();
Http http = new Http();
```

Esimerkkikoodi 3. Erityyppisten muuttujien esittely.

Javasta poiketen Apex sisältää sObject (Salesforce Object) -tyypin, joka esittää mukautettuja objekteja tai Salesforcen standardiobjekteja, kuten Contact-objektia. SObjecteja voi myös esitellä geneerisesti eli yleisesti, samankaltaisesti kuin SOAP API -metodikutsussa tehdään esimerkkikoodi 4:ssä. Koodissa uuden sObjectin tulee sisältää objekti-tyyppi, jotta esimerkiksi listaa voidaan käydä läpi kuten esimerkkikoodissa 4.

```
sObject s = new Contact();
DescribeSObjectResult describeSObjectResult = connection.describeSObject("Contact");
List<Contact> contacts = [SELECT Id,Name FROM Contact ];
for (Contact c : contacts){
    System.debug('Contact name : ' + c.Name);
}
```

Esimerkkikoodi 4. Geneerisen sObjektin esittely.

sObject esittää Salesforcen tietokannassa olevaa tietokantaobjektia. Sen avulla voidaan toteuttaa tietokantaoperaatioita, kuten "insert", "update" ja "delete".

Salesforcen järjestelmällä on valmiiksi määriteltyjä standardimetodeja sObjektille, eikä sille voi suoraan tehdä omia mukautettuja metodeja. Näitä standardimetodeja ovat esimerkiksi "addError", jonka avulla voidaan tietylle tietueelle antaa virheviesti laukaisimessa ja estää DML-operaatiota tapahtumista, "get(fieldName)", jonka avulla palauteaan tietty kenttä määrittelemällä "fieldName". [15.]

3.3 SOQL

Salesforce Object Query Language on Salesforceen oma tietokannan kyselykieli, jota käytetään tietyn tiedon hakemiseen Salesforce-organisaatiosta. Salesforce-organisaatio on kokonaisuus, joka koostuu organisaatiokohtaisista käyttäjistä, tiedoista ja automaatiosta esimerkiksi Apex-laukaisimista. SOQL on samanlainen tietokanta ja kyselykieli kuin SQL muutamaa poikkeusta lukuun ottamatta. Kuten SQL:ssä SOQL:ssä "SELECT"-sanalla valitaan kentät, jotka halutaan palauttaa. "FROM"-sanalla valitaan määritellyn lähdeobjekti ja "WHERE"-sanalla määritellään ehdot rivien valitsemiseen lähdeobjektista. SOQL ei kuitenkaan tue kaikkia SQL:ssä olevia "SELECT"-toimintoja, kuten JOIN-operaatioita, jokerimerkin käyttöä kenttähaussa tai laskutoimituksien käyttöä.

Objektien väliset viittaukset toimivat Salesforceissa hakusuhdekentän (Lookup Field) avulla. Esimerkiksi Opportunity (Mahdollisuus) -standardiobjektilla on hakusuhdekenttä Account (Tili), jonka avulla Opportunity voidaan liittää tietylle Accountille. Jos halutaan tietää Account-nimi Opportunity-objektilta, se voidaan tehdä yhdellä SOQL-kyselyllä hyödyntäen objektien suhdetta (koodiesimerkki 5).

```
Opportunity opp = [SELECT Id, Name, Account.Name, CustomExampleObject__r.Name  
FROM Opportunity WHERE Name = 'Test' ORDER BY Name ASC];  
System.debug('Opportunity Name: ' + opp.Name);  
System.debug('Account Name: ' + opp.Account.Name);  
System.debug('Custom Object Name: ' + opp.CustomExampleObject__r.Name);
```

Esimerkkikoodi 5. SOQL-tietokantakysely, joka sisältää viittauksen objektiin, jolla on hakusuhde.

Kuten esimerkkikoodi 5:ssä esitetään, mahdollisuusobjektilta voidaan hakea mahdollisuusobjektin nimi- ja hakusuhdekentän avulla Tilin nimi sekä mukautetun objektin nimi. Mukautetun objektin kyselyssä tulee käyttää "__r"-päättettä, kun haetaan hakusuhdekentän avulla tiettyä kenttää toiselta objektilta. Standardiobjektin kentän haussa riittää pelkkä objektin rajapintanimi.

4 REST-rajapinta

REST on lyhenne sanoista Representational State Transfer. Ohjelmointirajapinnan käsite on laaja ja sen käyttötarkoituksia on useita. Tässä luvussa kerrotaan REST-rajapinnan ominaisuuksista.

4.1 Asiakas ja palvelin

Asiakkaan ja palvelimen toteutus voidaan tehdä yksilöityneesti, toisen tietämättä toisesta. Eli asiakkaan puolen koodia voidaan muokata vaikuttamatta palvelin puolen koodiin ja palvelimen koodia voidaan muokata vaikuttamatta asiakkaan puolen toteutukseen.

Molempien osapuolien tulee tietää lähetysformaatti, jotta koodit voidaan pitää modulaarisina ja erikseen. Kun käyttöliittymäkysymykset ja tietojen varastointiin liittyvät kysymykset pidetään erillään, parantuu käyttöliittymän joustavuus ja skaalautuvuus. Tämä mahdollistaa myös molempien kehittämisen erillisinä komponentteina. [16.]

4.2 Tilattomuus

REST-arkkitehtuurin avulla palvelimen ja asiakkaan ei tarvitse tietää toistensa tiloja, minkä ansiosta molemmat ymmärtävät kaikki vastaanotetut kutsut, vaikka eivät näkisi vanhoja kutsuja.

REST-järjestelmän rajoituksena on se, että järjestelmät ovat vuorovaikutuksissa vain resurssien standarditoimintojen kautta. Resurssit kuvaavat mitä tahansa objektia, dokumenttia tai asiaa, jota tallennetaan tai lähetetään muille palveluille. Näitä esimerkiksi ovat Apex luokat, jotka voidaan esitellä Rest-resurssina annotaatiolla @RestResource.

REST-järjestelmän rajoituksen avulla sovelluksien luotettavuus, suorituskky sekä skaalautuvuus parantuu, koska komponentteja voidaan hallita, päivittää ja käyttää uudelleen vaikuttamatta koko järjestelmään. [16.]

4.3 Operaatiometodit sekä niiden käyttö

REST-arkkitehtuurissa asiakas lähettää kutsun, jolla vastaanotetaan tai muokataan resursseja. Palvelin vastaa näihin kutsuihin.

Asiakkaan lähettämä kutsu yleisesti sisältää http-verbin, joka määrittelee mitä, operaatiota suoritetaan, otsikon, jonka avulla voidaan välittää tietoa kutsusta, polku resurssille ja valinnainen viesti sisältäen dataa.

HTTP-verbejä on neljä perusverbiä, joiden avulla ollaan vuorovaikutuksissa resurssien kanssa (taulukko 1).

Taulukko 1. HTTP-metodit ja vaikutukset.

| VERBI | VUOROVAIKUTUS |
|---------------|----------------------------------------------------|
| GET | Noudetaan dataa tietokannasta (yksittäinen/lista). |
| POST | Tiedon tallentaminen / päivitys. |
| PUT | Tiedon tallentaminen / päivitys. |
| DELETE | Tiedon poistaminen. |

GET-metodilla noudetaan dataa tietokannasta joko yksittäisenä tietueena tai sitten listana. Esimerkkikoodissa 6 haetaan tietokannasta lista eläimiä ja käydään lista objekti kerrallaan läpi ja tulostetaan yksittäinen eläinobjekti. Tietokannan osoite määritellään

kutsuun "request.setEndPoint()", metodi määritellään "request.setMethod()" ja vastaus otetaan haltuun kutsun http.send-metodin joka sisältää kutsun palautusarvolla.

```
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndPoint('https://esimerkki.com/eläimet');
request.setMethod('GET');
HttpResponse response = http.send(request);
if (response.getStatusCode() == 200) {
    //Success
    Map<String, Object> results = (Map<String, Object>) JSON.deserializeUn-
typed(response.getBody());
    List<Object> eläimet = (List<Object>) results.get('eläimet');
    for (Object eläin: eläimet) {
        System.debug(eläin);
    }
}
```

Esimerkkikoodi 6. GET-metodin avulla noudetaan lista eläimiä ja tulostetaan ne yksitellen.

POST-metodin avulla pystytään tallentamaan dataa suoraan tietokantaan. Seuraavassa esimerkissä esimerkkikoodissa 7 tallennetaan poro-niminen eläin eläimet-taulukkoon. Kutsun otsikossa määritellään palvelimelle kutsun formaatti, joka on esimerkissä JSON. Kutsun sisällössä määritellään itse poroeläin: "name": "poro".

```
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndPoint('https://esimerkki.com/eläimet');
request.setMethod('POST');
request.setHeader('Content-Type', 'application/json;charset=UTF-8');
request.setBody('{"name": "poro"}');
HttpResponse response = http.send(request);
if (response.getStatusCode() != 201) {
    //Error message
} else {
    //Success message
}
```

Esimerkkikoodi 7. POST-metodilla tallennetaan uusi eläin eläimet-taulukkoon.

PUT-metodi eroaa POST-metodista jo olemassa olevan datan päivitystä tehtäessä. POST-metodi päivittää samaan tietueeseen, mutta PUT-metodi ylikirjoittaa koko tietueen (esimerkkikoodi 7).

```
Http h = new Http();
HttpRequest req = new HttpRequest();
req.setEndPoint('https://esimerkki.com/eläimet');
req.setMethod('PUT');
req.setBody('{"name": "kirahvi"}');
```



```

HttpResponse res = null;
res = h.send(req);
system.debug(res.getBody());

```

Esimerkkikoodi 8. PUT-metodin avulla uuden kirahvi eläimen luonti.

DELETE-metodilla poistetaan tietueita tietokannasta tietue Id:n avulla kuten esimerkkikoodissa 9 tehdään.

```

Http h = new Http ();

HttpRequest = new HttpRequest ();
HttpRequest.setEndpoint('https://esimerkki.com/api/poista/recordId');
HttpRequest.setMethod('DELETE');
HttpRequest.setBody('');

h.send(httpRequest);

```

Esimerkkikoodi 9. DELETE-metodin avulla suoritetaan tietueen poisto.

4.4 Salesforce sekä REST

Salesforcen standardoidun REST-arkkitehtuurin avulla ulkoiset sovellukset pystyvät käyttämään koodia ja sovellusta. Tämä toteutetaan esimerkiksi määrittelemällä Apex-luokan alkuun "@RestResource", joka kertoo, että luokka on REST-resurssi. Samalla tavalla luokan metodit voidaan määritellä esimerkiksi "@HttpGet", joka paljastaa, että metodi on REST-resurssi, jota voidaan kutsua HTTP GET-kutsulla. [18.]

Kuten edellä mainittiin RestResource-annotaatiota, käytetään luokkatasolla, se antaa paljastaa Apex-luokan REST-resurssina. RestResource-annotaatiota käytettäessä on muistettava, että URL-kuvaus on riippuvainen Salesforce-instanssista <https://instance.salesforce.com/services/apexrest/> ja merkkiä * voidaan käyttää kuten esimerkkikoodissa 10.

```

@RestResource(urlMapping='/asiakasURL/v1/*')
global class IntegrationRestApi {
    //REST-Rajapinta luokan koodi
    @HttpGet
    global static ResponseClass doGet(){
        List accounts = [SELECT Id,Name from Account LIMIT 10];
        return accounts;
    }
}

```

Esimerkkikoodi 10. Apex-luokan määrittely REST-rajapinnaksi sekä metodin määrittely HTTP-GET-metodiksi.

Salesforcella on rajoituksia liittyen koodin suoritusajarakajaan sekä SOQL-hakujen määrään. Myös Apex REST-luokilla on rajoituksia kuten kyselyn sekä vastauksen maksimikoko, joka on 6 MB synkroniselle ja 12 MB asynkroniselle kutsulle. Apex REST -rajapinnan rajoituksena on vain yhden HTTP-metodin luonti per luokka, eli rajapinnalle ei voi määrittellä esimerkiksi kahta @HttpGet-metodia. [19.]

Apex REST -metodi voi palauttaa kahta erilaista datatyyppiä: JSON- ja XML-muotoista dataa. Formaattiin viitataan HTTP-otsikossa, joka pystytään määrittelemään Apexin tarjoamaan staattiseen luokkaan RestContext. RestRequest- ja RestResponse-objektit ovat staattisen RestContext-luokan avulla käytössä automaattisesti. Näiden objektien avulla pystytään käsittelemään kyselyn sisältöä, jonka voi käsitellä Apex-luokaksi sekä asettamaan vastauksen sanoman tyyppin ja tilakoodin (esimerkkikoodi 11).

```
RestRequest req = RestContext.request;
RestResponse res = RestContext.response;
res.addHeader('Content-Type', 'application/json');

res.statusCode = 200;
```

Esimerkkikoodi 11. Rest-kysely ja -vastausviittaukset ja käyttö.

4.5 JSON

REST-rajapinnan dataformaatteina toimivat niin JSON kuin XML tai mukautettu ratkaisu. Tässä työssä käytetään JSON-formaattia. JSON on JavaScript-objektinotaatio. JSON on helppolukuista, se on helppo generoida ja jäsentää koneelle. Se on tekstiformaatti, joka on kokonaan riippumaton ohjelmointikielestä. JSON:n on erinomainen syntaksi tiedon tallennukseen ja välitykseen.

Määrittelyksiä on kahdenlaisia. JSON-määrittelyssä voidaan käyttää nimi-arvo-pari-kokemuksia ja järjestettyjä listoja (taulukoita). Nämä ovat universaaleja datatietueita. Kaikki nykyaikaiset ohjelmointikielet tukevat niitä tavalla tai toisella.

Objekti on järjestelmätön kokoelma nimi-arvo-pareja, jonka määritelmä alkaa ”{”-merkillä ja loppuu ”}”-merkkiin nimi-arvo-parissa. Nimen jälkeen tulee ”:” kaksoispiste ja arvoparit on eroteltu ”,”-pilkulla. Taulukko on järjestelty kokoelma arvoista, jonka määritelmä alkaa ”[”-merkillä ja loppuu ”]”, arvot on eroteltu ”,” pilkulla (esimerkkikoodi 12). Arvo voi myös olla teksti, numero, boolean-arvo tosi tai epätosi sekä null. [17.]

```
{
  "EsimerkkiTeksti" : "testiArvo",
  "EsimerkkiObjekti" : {
    "Boolean" : tosi,
    "Nimi" : "Testi Henkilö"
  },
  "EsimerkkiTaulukko" : [
    1,2,4,5,6,7
  ]
  "EsimerkkiTaulukkoObjekteista" : [{
    "testiNimi" : "Nimi1",
    "testiNimi2" : "Nimi2"
  },
  {
    "testiIkä" : "Ikä",
    "testiIkä2" : "Ikä2"
  }
  ]
}
```

Esimerkkikoodi 12. JSON-määritelmiä teksti, objekti ja taulukko.

5 Projektin toteutus

5.1 Suunnittelu

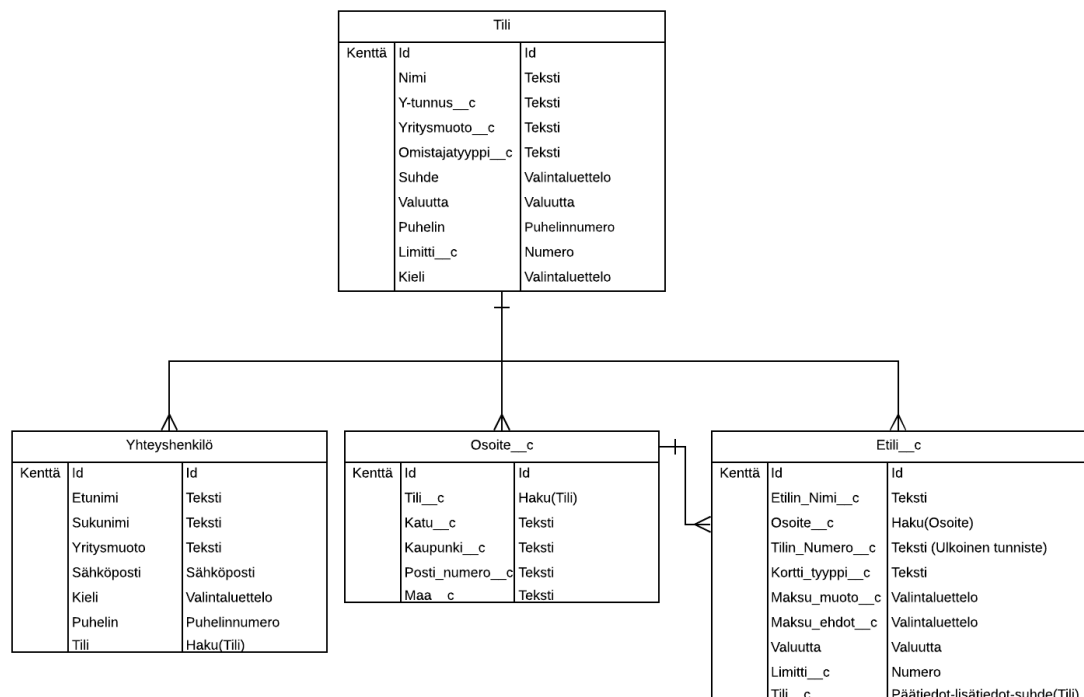
Projektin suunnitteluvaiheessa käytiin läpi, mitä projekti sisältää, mitä tarvitaan, kuinka toteutetaan, missä ajassa ja miten testataan. Koska työn alkuvaiheessa kävi jo selväksi, että projektin aikataulu on erittäin kiireellinen, suunnittelu ja testaus jäivät melko vähäiseksi.

Järjestelmä sisältää kolme APEX-luokkaa, joista yksi on REST-API-luokka, joka vastaanottaa kutsuja. Toinen on REST-luokan hallinnointiluokka, jonka avulla toteutetaan validointi, DML-operaatiot sekä hallinnoidaan virhetiloja. Kolmas luokka on testiluokka, jossa testataan molempien luokkien toteutus Salesforce-organisaatiota vasten. Työssä

hyödynnetään neljää eri objektia, joista kaksi on standardiobjekteja ja kaksi kustomoituja. Muokattuja kenttiä on tehty sekä standardiobjekteihin kuten myös kustomoituihin objekteihin.

5.2 Tietokanta ja datamalli

Työn tietokanta on toteutettu Salesforcen standardien mukaisesti ja mukautettuja objekteja hyödyntäen. Tarkoituksena oli käyttää mahdollisimman paljon Salesforcen standardiminaisuuksia, jotta välttyttäisiin turhalta mukauttamiselta.



Kuva 3. Datamallin objektit

Salesforcen Datamalli sisältää standardiobjekteja kuten Tili ja Yhteyshenkilö. Mukautettuja objekteja ovat Osoite__c ja Etili__c. Kuten kuvasta 3 nähdään, kaikki objektit ovat yhteydessä Tili-standardiobjektiin.

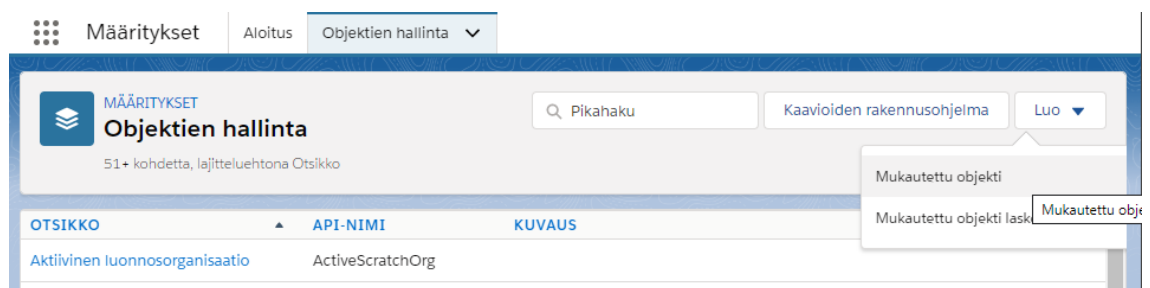
Tili-objekti sisältää kaiken tarvittavan tiedon asiakkaasta.

Tilille liitetään osoiteobjekti, joka sisältää asiakkaan laskutusosoitetiedot kuten maa, ka-tuosoite, postinumero ja postitoimipaikka.

Yhteyshenkilöobjekti liitetään Tiliin, jotta tiedetään, kehen olla yhteydessä esimerkiksi sähköpostin avulla.


Etilin avulla tiedetään, että kenelle lähetetään korttilasku. Etili sisältää nimen, linkin osoi-teobjektiin, tilin numeron, korttityypin, maksun muodon, maksuehdot, valuutan, limitin ja linkin Tili-objektiin.

Uuden objektin luominen Salesforcessa on tehty erittäin helpoksi ja yksinkertaiseksi. Mu-kautettujen objektien luonti tapahtuu ”Määrittelyt”-välilehdeltä Salesforce-organisaa-tiosta valitaan Objektien hallinta, Luo ja Mukautettu objekti (kuva 4).



Kuva 4. Määrittelyt Objektien hallinta

Uuden mukautetun objektin luontiprosessi on ohjeistettu selkeästi ja tehty yksinker-taiseksi. Objektille annetaan tarvittavat tiedot ja tallennetaan uusi mukautettu objekti (kuva 5).


MÄÄRITYKSET
Uusi mukautettu objekti

Uusi mukautettu objekti
Tämän sivun ohje

1 Kaikki tämän objektin käyttöoikeudet ovat oletuksena poissa käytöstä kaikissa profiileissa. Voit ottaa objektikäyttöoikeudet käyttöön käyttöoikeusjoukoissa tai muokkaamalla mukautettuja profileja. [Lue lisää!](#) [Älä näytä tätä viestiä uudelleen](#)

Mukautetun objektin määrittäminen: Tallenna Tallenna ja uusi Peruuta
Muokkaus

Mukautetun objektin tiedot = Pakolliset tiedot
Yksikkö- ja monikkumuotoisia otsikoita käytetään välilehdissä, sivunasetteluissa ja raporteissa.
Otsikko Esimerkki: Tili
Monikkumuotoinen otsikko Esimerkki: Tilit
Objektin nimeä käytetään, kun objektiin viitataan API:n kautta.
Objektin nimi Esimerkki: Account
Kuvaus
Tilannekohtaisen ohjeen asetukset ☒ Avaa Salesforce.com:in Ohje ja koulutus -vakioikkuna ☐ Avaa ikkuna Visualforce-sivun avulla
Sisällön nimi

Syötä tietuenimen otsikko ja muoto
Tietuenimi näkyy sivunasetteluissa, avainluetteloissa, tietueeseen liittyvissä luetteloissa, hakuja suoritettaessa ja hakutuloksissa. Esimerkiksi tilin tietuenimi on "Tilin nimi" ja tapauksen tietuenimi puolestaan "Tapauksen numero". Huomaa, että Tietuenimi-kenttää kutsutaan aina nimellä Nimi, kun siihen viitataan API:n kautta.
Tietuenimi Esimerkki: Tilin nimi
Tietotyyppi

Kuva 5. Esimerkki uuden mukautetun objektin luomisesta.

Kun uusi objekti on luotu, sille voidaan lisätä tarvittavia uusia kustomoituja kenttiä. Mukautettujen kenttien luomisprosessi on samankaltainen kuin objektien luomisprosessi.

5.3 Uuden asiakkaan luominen

Asiakastilin luominen Salesforceen tapahtuu REST-rajapinnan avulla, joka vastaanottaa asiakkaan tiedot kuten nimen, y-tunnuksen, tilinnumeron, osoitetiedot, maksutiedot sekä muut tarvittavat tiedot. Rajapinta vastaanottaa JSON-objektin, jonka parametreina ovat tarvittavat tiedot. Seuraavaksi esimerkki JSON-objektin parametreista (esimerkkikoodi 13).

```
{
  "operationType":1,
```

```

"customerNumber":"FI12345671",
"invoiceReference": "10728129",
"customerCurrencyCode":"EUR",
"tradingName":"Testi-Yritys",
"addressLines":"Testi Katu 3 C",
"city":"ESPOO",
"postalCode":"02600",
"country":"FI",
"CountryCode":"FI",
"keyAccount":2,
"custGroup":2,
"name":"Testi-Yritys",
"languageId":"fin",
"blocked":"1",
"paymentTermsId":2,
"paymentMethodId":4,
"vatNumber":"01234567",
"contactPhoneNumber":"09 1234567",
"riskFactor":4,
"primaryLob":2,
"companyId":12
}

```

Esimerkkikoodi 13. JSON:n parametrien kuvaus, jonka rajapinta vastaanottaa.

Rajapintaa varten loin uuden Apex-luokan. Apex-luokan luonti onnistuu Salesforcen Apex-luokat sivulta painamalla ”uusi”-nappia. Salesforce tarvitsee luokan alkuun tunnisteen ”@RestResource(urlMapping='/asiakasURL/v1/*')”, jotta se tunnistaa luokan olevan REST-rajapintaluokka. Jokaisen REST-rajapintaluokan urlMapping-arvon tulee olla uniikki (esimerkkikoodi 14).

```

@RestResource(urlMapping='/asiakasURL/v1/*')
global class IntegrationRestApi {
    //REST-Rajapinta luokan koodi
}

```

Esimerkkikoodi 14. REST-rajapintaa toteuttavan Apex-luokan määrittely.

REST-rajapintaluokka sisältää JSON-objektisisäluokan, palautusluokan, POST-metodin, virheluokan sekä validointimetodin.

Rajapintaa varten luodaan sisäluokka, joka vastaa JSON-objektin sisältöä. Kuten luvussa 5.4 tullaan kertomaan, myös rajapinnan palauttamaan JSON-payloadia varten tulee olla sisäluokka (esimerkkikoodi 15).

```

global class CustomPayload{
    public String recordType{get;set;}
    public Integer operationType{get;set;}
    public String companyId{get;set;}
    public String customerNumber{get;set;}
}

```

```

    public String payerCustomerNumber{get;set;}
    public String oldSystemNumber{get;set;}
    public String invoiceReference{get;set;}
    public String customerCurrencyCode{get;set;}
    public String name{get;set;}
    public String customerShortName{get;set;}
    public String tradingName{get;set;}
    public Boolean isSubAccount{get;set;}
}

```

Esimerkkikoodi 15. Vastaanotetun JSON-objektin Apex-sisäluokka.

Kun Apex payload-luokka on luotu, voidaan käyttää hyväksi Salesforcen JSONParser-luokkaa ja parsia JSON Apex sisäluokaksi. Tämän jälkeen voidaan vastaanotettuihin arvoihin viitata kuten normaaleihin Apex-luokan muuttujiin (esimerkkikoodi 16).

```

CustomPayload payload = new CustomPayload();
CustomResponse response = new CustomResponse();
try{
    JSONParser parser = JSON.createParser(req.requestBody.toString());
    payload = (CustomPayload)parser.readValueAs(CustomPayload.class);
}catch (JSONException jex) {
    System.debug(jex.getMessage());
}

response.success = true;
response.message = 'Custom success message';

```

Esimerkkikoodi 16. Payload- ja Response-sisäluokkien esittely sekä JSON-parserin käyttö.

Seuraavaksi tulee luoda HTTP POST -tapahtumaa varten metodi, joka tulee annotoida @HttpPost-kommentilla. Salesforce tällöin tietää, mikä metodi ajetaan, kun HTTP POST -kutsu tulee luokassa määriteltyn osoitteeseen. Tässä työssä käytetään vain POST-kutsua (esimerkkikoodi 17).

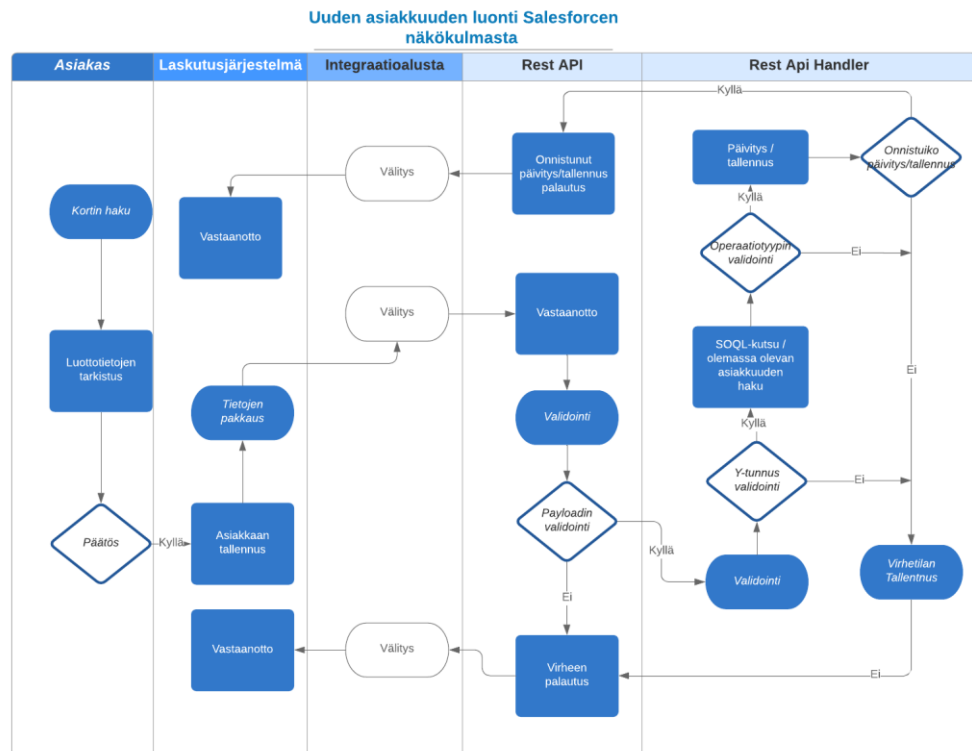
```

@HttpPost
global static CustomResponse doPost(){
    // Kutsun ja vastauksen luonti
    // vastaanotetun tiedon läpikäynti ja palautus
}

```

Esimerkkikoodi 17. HTTP-tapahtumaa varten luotu POST-metodi.

Varsinainen asiakkuuden luonti tai päivitys tapahtuu esimerkkikoodin 17 sisältämässä metodissa. Metodi sisältää kutsun ja palautuksen, palautus- ja kutsuluokat, JSON-datan validoinnin, asiakkaan tietojen tallennuksen ja päivityksen sekä virheen käsittelyn. Kuva 6 esitetään prosessikaaviolla yleisnäkymänä uuden asiakkuuden luonti Salesforceen



Kuva 6. Prosessikaavio uuden asiakkuuden luomisesta.

Prosessikaaviossa asiakas hakee luottokorttia. Kun luottotiedot on tarkastettu hyväksytyllä päätöksellä, tallennetaan asiakas laskutusjärjestelmään. Ulkoinen järjestelmä lähettää laskutusjärjestelmän pakkaaman JSON-tiedoston Salesforceen Rest API:iin. Payloadin arvoista riippuen luomis-/päivitysprosessi käynnistyy. Mikäli validaatiot sekä luomisoperaatiot onnistuvat, lähetetään takaisin Response-luokka onnistuneilla arvoilla sekä Salesforce Id:llä.

Virheen käsittelyä hallinnoidaan kahdella try- ja catch -koodilohkolla. Ensimmäisessä lohossa parsitaan vastaanotettu JSON ja tehdään tästä luokka, mikäli ei ole virheitä.

Virhetilanteessa käytetään Exception-luokkaa, jonka avulla lähetetään virhetilanteen viesti ja tilakoodi 400 (esimerkkikoodi 18).

```
try{
    //kutsun JSON:n parse
    try{
        JSONParser parser = JSON.createParser(req.requestBody.toString());
        payload = (CustomPayload)parser.readValueAs(CustomPayload.class);
    }catch (JSONException jex) {
        throw new CustomIntegrationException('INVALID PAYLOAD');
    }
} catch (CustomIntegrationException 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;

global class CustomIntegrationException extends Exception{}
```

Esimerkkikoodi 18. Virheenkäsitteleyesimerkki.

Toisessa catch-lohkossa otetaan kiinni sellaiset virhetilanteet, jotka ovat systeemivirheitä, eli sellaisia tilanteita, mihin ei olla valmiiksi varauduttu, kuten vastaanotetun JSON:n datan null-virhetilanteet. Tällöin otetaan geneerinen Exception-virhetila kiinni, jonka tilakoodi on 500 (esimerkkikoodi 18).

Koska lähetetty data voi olla tyhjä tai null, tulee pakollisten kenttien validaatio yhdeksi kohdaksi koodia. Koodin validaatio on hyvin yksinkertainen, se tehdään ennen kuin aletaan hallinnoimaan itse asiakkuuden luontia tai päivitystä (esimerkkikoodi 19). Mikäli kenttä ei sisällä mitään tietoa, lähetetään tilannekoodi 400 ja virheviesti.

```
private static void validateMandatory(CustomPayload payload){

    if(String.isEmpty(''+payload.operationType)
        || payload.operationType == null){
        throw new CustomIntegrationException('Operation Type is missing');
    }
    if(String.isEmpty(payload.customerNumber)
        || payload.customerNumber == null){
        throw new CustomIntegrationException('Customer Number is missing');
    }
    if(String.isEmpty(payload.country) || payload.country == null){
        throw new CustomIntegrationException('Country Code is missing');
    }
}
```

Esimerkkikoodi 19. Vastaanotetun JSON-objektin validointi.

Kun validaatio on saatu tehtyä, päästään itse asiakkuuden luontiin. Koska luonti ja päivitys tarvitsee myös validaatioita ja metodeita, luodaan tätä varten hallinnointiluokka. Tässä hallinnointiluokassa tapahtuu koko koodin tallennus sekä päivitys DML-operaatioille. Hallinnointiluokka palauttaa onnistuessaan erillisen vastausluokan, totuusarvomuuttujan "success" arvolla tosi, tekstiarvomuuttujan viestillä, joka sisältää Etilin tietueen Id:n ja RestContext response -objektin tilakoodilla 200. Virhetilanteessa luokka palauttaa Exception-luokan (esimerkkikoodi 20).

```
try{
    returnId = CustomIntegrationHandler.handlePayload(payload);
}catch (CustomIntegrationHandler.CustomIntegrationHandlerException ghex){
    throw new CustomIntegrationException(ghex.getMessage());
}
response.success = true;
response.message = 'Etilin tallennus/päivitys onnistui, Id: ' + returnId;
res.statusCode = 200;
```

Esimerkkikoodi 20. CustomIntegrationHandlerin kutsuminen ja virhetilanteen hallinta.

CustomIntegrationHandler parametrina on vastaanotetun JSON-objektin JSON-luokka. Koska luokka sisältää kaikki objektin arvot, on niitä helppo hyödyntää koodin sisällä. Koska REST-Apex-luokan validoitujen arvojen virhetilanteet eivät luo Salesforceen tapausobjektia, niin hallinnointiluokassa tarkastellaan heti ensimmäiseksi yritystunnuksen löytyminen. Mikäli tätä ei ole täytetty, luodaan tapausobjekti ja siihen liitetään koko vastaanotettu JSON-objekti arvoineen. Yritystunnus myös validoidaan maakohtaisten formaattien perusteella.

Seuraavaksi tämän tarkistetun ja validoidun y-tunnuksen avulla haetaan olemassa olevia tilejä, joita käytetään tallennusoperaatioissa. Mikäli Tiliä ei löydy ja operaatiotyyppi on luonti, tehdään myös uusi Tili. Uuden tilin luonnin yhteydessä tarkistetaan myös osoitetiedot ja luodaan Tilille uusi osoiteobjekti. Etilin, osoitteen ja yhteyshenkilön luonnissa hakukenttää hyödynnetään, jotta kaikki objektit ovat keskenään yhteydessä. Koska osa näistä DML-operaatioista halutaan palauttaa virhetilanteissa, käytetään koodissa hyödyksi Salesforcen tarjoamaa setSavepoint-metodia (esimerkkikoodi 21).

```
global class CustomIntegrationHandler {
```

```

    global static Id handlePayload(CustomIntegrationRestApi.CustomPayload payload) {
        checkVAT(payload);
        List<Tili> accounts = [SELECT id,RecordTypeId, (SELECT Id FROM Etili)
FROM Tili WHERE National_Organization_Number__c = :formatVatNumber(payload)];

        Id retId = null;
        if(accounts.isEmpty() && payload.operationType == 1){
            Tili eAccount = createAccount(payload);
            checkAddress(payload,eAccount);
            Savepoint sp = Database.setSavepoint();
            try{
                Address__c eAddress = createAddress(payload,eAccount);
                insert eAddress;
                eAccount.Type = 'Customer';
                update eAccount;
                Etili etili = createEtili(payload,eAccount);
                insert etili;
                Contact eContact = createContact(payload,eAccount);
                insert eContact;
                retId = euroShellAccount.Id;
            }catch(DmlException dmlE) {
                Database.rollback(sp);
                Case eCase = createCase(payload, accounts, dmlE.getMessage(),payload.Country);
                insert eCase;
                createAttachment(eCase.Id, payload);
                throw new CustomIntegrationRestApi.CustomIntegrationException(dmlE.getMessage());
            }catch(Exception e){
                Database.rollback(sp);
                throw new CustomIntegrationRestApi.CustomIntegrationException(e.getMessage());
            }
        }
    }
}

```

Esimerkkikoodi 21. Hallinointiluokan luontiprosessin läpikäynti.

Database.setSavepoint-metodin avulla määritellään kohta, mistä lähtien kaikki DML-operaatiot palautetaan. Koska Tili halutaan aina luoda, sitä ei oteta savepointiin mukaan, mutta kaikki tämän jälkeen tulevat DML-operaatiot kuuluvat tähän. Virhetilanteessa Database.rollback(sp) palauttaa tuohon määritettyyn savepointiin asti tehdyt DML-operaatiot, jotta koodi toimii osittain. Eli jos esimerkiksi osoiteobjektin luonnissa tapahtuu virhe, Tilin luonti tapahtuu silti metodissa createAccount(payload).

Päivitysosiossa päivitetään olemassa olevaa Etiliä. Tämä riippuu siitä, löytyykö Etiliä samalla tilin numerolla, mikä tulee vastaanotetussa datassa. Mikäli päivitysoperaatio onnistuu, palautetaan Etilin tietueen Id. Koska operaatiotyyppi voi olla luonti ja Tili löytyy Salesforcesta, on vielä kolmas vaihtoehto, missä luodaan Etili. Tässä katsotaan, onko

olemassa olevan Etilin tyyppi sama kuin vastaanotetussa datassa. Mikäli ei ole, luodaan uusi Etili ja palautetaan Id.

5.4 Testaaminen

Salesforcen Apex-yksikkötestit ovat tärkeitä niin laadunvarmistuksellisesti kuin käyttöön-oton kannalta. Yksikkötestien on katettava vähintään 75 % Apex-koodista ja kaikkien testimetodien on suoritettava se onnistuneesti.

Salesforce Apex -luokkien testaaminen tapahtuu omalla Test-luokalla. Testi-luokan määrittäminen tapahtuu @IsTest-annotaatiolla. Testidata tulee luoda joko testi-setupilla tai jokaisessa testimetodissa erikseen. Työssä luodaan testi-setup-metodi, joka annotoidaan @TestSetup:lla ja tyyppi on staattinen tyhjiö eli metodilla ei ole palautusarvoa. REST-rajapintaa testatessa luodaan RestResponse-metodi, joka luo kutsun ja käyttää hyödyksi RestContextia. Vastaanotetun JSON-luokan luonti tapahtuu omana metodinaan ja näitä kutsutaan testimetodista, jonka voi annotoida @IsTest tai sitten käyttää static testMethod void -merkintätapaa. Lopuksi tulee vielä kutsua itse REST-rajapinta luokan doPost()-metodia (esimerkkikoodi 22).

```
@TestSetup
static void initData() {
    List<Etili> etilit = new List<Etili>();
    List<Account> accounts = new List<Account>();
    List<Address__c> addresses = new List<Address__c>();
    Cost_Center__c cs = new Cost_Center__c(
        Name = '1000629'
    );
    insert cs;
    Cost_Center__c cs2 = new Cost_Center__c(
        Name = '101101'
    );
    insert cs2;
    Account ac = new Account(
        Name = 'Test Account FI',
        Type = 'Prospect',
        CurrencyIsoCode = 'EUR',
        Phone = '+3581111111',
        Limitti__c = 2000,
        Cost_Center__c = cs.Id,
        National_Organization_Number__c = '0128113-2'
    );
    accounts.add(ac);

    insert accounts;
    Address__c address = new Address__c(
        AccountID__c = accounts[0].Id,
```

```

        Street__c = 'Test Street 123',
        City__c = 'Helsinki',
        Country__c = 'Finland',
        PostalCode__c = '00100'
    );
    addresses.add(address);

    insert addresses;
    accounts[0].Type = 'Customer';

    update accounts;
    Etili etili = new Etili(
        Account__c = ac.Id,
        Name = 'Test',
        CurrencyIsoCode = 'EUR',
        Tilin_Numero__c = 'FI21001499',
        Blocked__c = false,
        Limitti__c = 100,
    );
    etilit.add(etili);

    insert euroshells;
}

static RestResponse initRest(String jsonBody) {
    RestRequest req = new RestRequest();
    RestResponse res = new RestResponse();

    req.addHeader('Content-Type', 'application/json');
    req.requestURI = '/services/apexrest/asiakasURL/v1*';
    req.httpMethod = 'POST';
    req.requestBody = Blob.valueOf(jsonBody);

    RestContext.request = req;
    RestContext.response = res;

    return res;
}

static CustomIntegrationRestApi.CustomPayload makeBody(Integer opType, String
flow) {
    CustomIntegrationRestApi.CustomPayload payload = new CustomIntegrationRe-
stApi.CustomPayload();
    payload.recordType = null;
    payload.operationType = opType;
    payload.companyId = '12';
    payload.primaryLob = 2;
    payload.country = 'FI';
    if (flow.equals('empty') && opType == 1) {
        payload.customerNumber = 'FI21001499';
        payload.vatNumber = '01281132';
        payload.primaryLob = 1;
    }
    return payload;
}

@IsTest
static void insertEtiliTest() {

    String jsonBody = JSON.serialize(makeBody(1, 'empty'));

    initRest(jsonBody);
}

```

```
CustomIntegrationRestApi.CustomResponse response = CustomIntegration-  
RestApi.doPost();  
}
```

Esimerkkikoodi 22. Testiluokan testisetupin initData()-luonti, testimetodin insertEtili-Test() ja staattisten metodien käyttö.

Esimerkkikoodissa 22 testi-setupissa luodaan tarvittavat testidatat itse testiluokkia varten, koska koodissa pitää tarkistaa, onko jo olemassa olevia tilejä. Luodaan testi-setupissa Tili, osoite, cost center, Etili ja nämä linkitetään Tilille. Staattisessa metodissa makeBody() tehdään JSON-body kutsua varten. Metodi palauttaa JSON-objektin, jota käytetään InitRest() staattisessa metodissa, jossa luodaan testikutsu määriteltyyn osoitteeseen JSON-bodylla .

6 Yhteenveto

Insinööriyön tarkoituksena oli kehittää asiakkuuksien luontijärjestelmä REST-rajapintaa hyödyntäen asiakkaalle. Työn tekeminen alkoi tutustamalla Salesforce-alustaan ja sen erilaisiin palveluihin. Tutustumisen perusteella todettiin, että haluttu logiikka voidaan rakentaa Salesforce REST-arkkitehtuuria hyödyntäen.

Tavoitteena oli rakentaa REST-rajapinta Salesforce-alustalle hyödyntäen Salesforce REST -arkkitehtuuria. Uusien asiakkuuksien luonti sekä jo valmiiksi olevien päivitys, olivat työn keskeisimmät prosessit. Näitä prosesseja varten luotiin oma hallinnointiluokka, joka teki kaikki prosessin DML-operaatiokutsut.

Uusien Etilien luonti rajoittui vain kahteen eri tyyppiin. Koska tallennus tapahtui ensin olemassa olevan Tilin hakukutsulla, tuli tähän logiikkaan lisätä myös Etilin haku, jotta voitiin tarkastaa, oliko asiakastilillä jo olemassa oleva saman tyyppinen Etili. Jatkokehityksen kannalta tulisi koko prosessi automatisoida hakemuksesta lähtien, jotta pystyttäisiin pitämään data korruptoitumattomana sekä pitämään manuaalisen työn määrä mahdollisimman pienenä.

Projektin toteutuksen kiireellisyyden takia jäi järjestelmä mielestäni hieman vajavaiseksi. Järjestelmän toteutus onnistui tekohetkellä niin kuin oli suunniteltukin. Jatkokehityssuunnitelman avulla tätä projektia on muokattu ja päivitetty

Lähteet

- 1 McCarthy, Ben 2016. A Brief History of Salesforce.com. Verkkodokumentti. <<https://www.salesforceben.com/brief-history-salesforce-com/>> Luettu 19.9.2018.
- 2 Techopedia. Legacy System. Verkkodokumentti. <<https://www.techopedia.com/definition/635/legacy-system>> Luettu 19.9.2018.
- 3 Shivvers, Liz 2017. Dreamforce '17 by the Numbers. Verkkodokumentti. <<https://www.salesforce.com/blog/2017/11/dreamforce-17-by-the-numbers.html>> Luettu 19.9.2018.
- 4 TechTarget. Inside the Salesforce platform: The Salesforce clouds. Verkkodokumentti. <<https://searchsalesforce.techtarget.com/essentialguide/Inside-the-Salesforce-platform-The-Salesforce-clouds>> Luettu 22.9.2018.
- 5 Markovski, Milo 2017. Top 10 CRM Software Vendors and Market Forecast 2016-2021. Verkkodokumentti. <<https://www.appsruntheworld.com/top-10-crm-software-vendors-and-market-forecast/>> Luettu 22.9.2018.
- 6 Rouse, Margaret 2016. Salesforce Sales Cloud. Verkkodokumentti. <<https://searchsalesforce.techtarget.com/definition/Salesforce-Sales-Cloud>> Luettu 22.9.2018.
- 7 Salesforce. Concepts, Products and Services. Verkkodokumentti. <https://help.salesforce.com/articleView?id=basics_sf_concepts_terms.htm&type=5> Luettu 27.9.2018.
- 8 Developer Documentation. Data Model. Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.api.meta/API/data_model.htm#!> Luettu 27.9.2018.
- 9 Developer Documentation. Custom Objects. Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.object_reference.meta/object_reference/sforce_api_objects_custom_objects.htm> Luettu 27.9.2018.
- 10 Developer Documentation. Validation Rules. Verkkodokumentti. <https://help.salesforce.com/articleView?id=fields_about_field_validation.htm&type=5> Luettu 13.02.2019.
- 11 Developer Documentation. Create Custom Fields. Verkkodokumentti. <https://help.salesforce.com/articleView?id=adding_fields.htm&type=5> Luettu 13.02.2019.

- 12 Developer Documentation. Custom Metadata Types. Verkkodokumentti. < https://help.salesforce.com/articleView?id=custommetadatatypes_overview.htm&type=5> Luettu 13.02.2019.
- 13 Developer Documentation. What is Apex? Verkkodokumentti. < https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_intro_what_is_apex.htm> Luettu 25.02.2019.
- 14 Developer Documentation. Primitive Data Types Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_primitives.htm> Luettu 25.02.2019.
- 15 Developer Documentation. sObject Types Verkkodokumentti. < https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_SObject_types.htm> Luettu 02.04.2019.
- 16 What is REST? Verkkodokumentti. < <https://www.codecademy.com/articles/what-is-rest>> Luettu 09.08.2019.
- 17 Introducing JSON Verkkodokumentti. < <https://www.json.org>> Luettu 28.10.2019.
- 18 Introduction to Apex REST Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_rest_intro.htm> Luettu 31.10.2019.
- 19 Apex REST Methods Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_rest_methods.htm> Luettu 31.10.2019.

