Aleksi Nivus

Salesforce-alustan hyödyntäminen finanssi-asiakkaan rekisteröitymis- ja luotonhakuproses-sissa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

28.11.2017

Tekijä Otsikko Sivumäärä Aika	Aleksi Nivus Salesforce-alustan hyödyntäminen finanssiasiakkaan rekisteröitymis- ja luotonhakuprosessissa. 35 sivua + 1 liite 28.11.2017
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikka
Ammatillinen pääaine	Ohjelmistotekniikka
Ohjaajat	Yliopettaja Auvo Häkkinen

Insinöörityössä toteutettiin nykyaikaiselle finanssialan yritykselle tarkoitettua järjestelmää, joka toimii Salesforce-pilvipalvelualustalla. Sen tehtävänä on kyetä vastaanottamaan ja pitämään yllä asiakkailta saatua dataa. Järjestelmä sisältää kaksi tärkeää toiminnallisuutta: käyttäjän rekisteröitymisen sekä luoton hakemisen REST-rajapintoja hyödyntäen.

Insinöörityön toteuttaminen aloitettiin datamallin suunnittelulla. Malli suunniteltiin siten, että se tulisi olemaan mahdollisimman yksinkertainen ja käyttäisi mahdollisuuksien mukaan Salesforcen standardiominaisuuksia.

Insinöörityön varsinaisessa kooditoteutuksessa tehtiin edellä mainitut kaksi REST-rajapintaa. Rajapintojen toteutuksessa hyödynnettiin Salesforcen tarjoamaa Apex REST-rajapinta mallia ja JSON-dataformaattia. Näistä ensimmäinen rajapinta pitää huolen käyttäjän rekisteröitymisestä. Kyseisen rajapinnan toteutus on hyvin yksinkertainen, sillä se pitää sisällään vain mahdollisen vanhan asiakkaan etsimisen tietokannasta ja tallentaa tai päivittää asiakkaan tiedot. Sen sijaan toinen rajapinta, joka on suunnattu luotonhakuun, on toteutukseltaan huomattavasti edeltävää monimutkaisempi. Se vaati paljon ideointia ja Salesforcen toiminnallisuuden tutkimista, jotta kaikki haluttu toiminnallisuus saataisiin toteutettua.

Lopputuloksena insinöörityöstä syntyi järjestelmä, joka tarjoaa finanssialan yritykselle nykyaikaiset työvälineet sekä mahdollisuuden kehittyä ja vastata nopeasti muuttuviin markkinoihin. Tämän insinöörityön tuloksena pystyy uuden ajan finanssialan yritys palvelemaan asiakkaitaan tehokkaasti ja luotettavasti.

Avainsanat	Salesforce, APEX, SOQL, REST, Rajapinta

Author Title Number of Pages Date	Aleksi Nivus Utilizing Salesforce to enroll a new customer and applying for credit 35 pages + 1 attachment 28 November 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Auvo Häkkinen, Principal Lecturer

The purpose of this bachelor's thesis was to create a system for a modern financial business by utilizing the Salesforce cloud platform. It is supposed to be able to receive and store data the customer has provided. The system's main functionalities are two REST interfaces from which the other is used for registering and the other for applying credit.

The first thing that was done for this bachelor's thesis was the design for the data model. The model was designed to be as simple as possible and to utilize Salesforce's standard functionalities as much as possible.

For the actual code implementation of this bachelor's thesis there were two previously mentioned REST interfaces to be implemented. The Salesforce Apex REST API functionality and JSON data format were used for the implementation of these interfaces. The first of the interfaces was the one that includes the registration functionality. The implementation of this interface turned out to be quite simple. It only has a query to search the existing customers and then a function to save or update customer data. Instead, the other REST interface meant for applying credit, was much more complicated to implement. It required a lot of brain storming and exploring to get all the necessary functionalities to work.

As a result, the bachelor's thesis created a system that provides the financial business with modern tools and the ability to develop and respond to rapidly changing markets. Thanks to this bachelor's thesis the company in question is now able to serve its customers efficiently and reliably.

Keywords	Salesforce, APEX, SOQL, REST, Rajapinta

Sisällys

Liite 1. Rekisteröitymis- REST-rajapinta

Lyhenteet

CRM	Customer Resource Management. Englanninkielinen termi asiakkuuden hallinnalle.
SF	Lyhenne sanasta Salesforce.
REST	Representational State Transfer. On verkkopalveluarkkitehtuuri-internet rajapintojen toteutukseen.
API	Application Programming Interface. Englanninkielinen sana ohjelmointirajapinnalle.
SOQL	Salesforce Object Query Language. Salesforcen käyttämä tietokantakyselykieli.
Apex	Salesforcen käyttämä Javan kaltainen ohjelmointikieli.
AML	Anti-Money Laundering. Englanninkielinen termi, joka tarkoittaa toimia jolla estetään rahanpesua.
PEP	Politically Exposed Person. Englanninkielinen termi poliittisesti vaikutusvaltaiselle henkilölle.
SaaS	Software as a Service. Englannin kielinen termi, joka tarkoittaa ohjelmiston kauppaamista palveluna eikä asennettavana ohjelmana.

1 Johdanto

Nykyaikainen finanssialan toiminta on hyvin erilaista kuin aikaisempina vuosikymmeninä. Niin kuin kaikessa muussakin nykymaailmassa on nykyaikaisen finanssialan toimijan kyettävä olemaan joustava ja nopea reagoimaan asiakkaiden kasvaviin vaatimuksiin ja muuttuviin markkinoihin. Koska Salesforce on pilvipohjainen palvelu, tarjoaa se tähän tarkoitukseen loistavan kehitysalustan. Pilvipohjainen palvelu mahdollistaa ketterän kehityksen. Tämä on avainasemassa, kun liikutaan maailmassa, jossa reagointi muutoksiin on elintärkeää.

Tämä insinöörityö käsittelee, kuinka nykyaikainen finanssijärjestelmä voidaan toteuttaa Salesforcella. Työssä perehdytään niin Salesforcen ominaisuuksiin kuin REST-arkkitehtuuriin ja niiden hyödyntämiseen tällaisessa käyttötarkoituksessa. Työssä tullaan toteuttamaan finanssiasiakkaan rekisteröityminen sekä luotonhakuprosessi Salesforcea hyödyntäen. Tämä tulee tarjoamaan finanssialan yritykselle modernit työkalut finanssitoimintaan.

Työn alussa tarkastellaan Salesforce-alustaa ja sen eri komponentteja. Tämän jälkeen perehdytään Salesforcen käyttämiin ohjelmointi- ja tietokantakyselykieliin. Seuraavaksi perehdytään verkkopalveluihin. Käydään läpi niiden historiaa ja tarkastellaan tarkemmin tässä työssä käytettävää REST (Representational State Transfer) -arkkitehtuuria ja Salesforcen soveltuvuutta REST-rajapintojen toteutukseen. Lopuksi työssä käydään läpi järjestelmän toteutus vaihe vaiheelta ja tehdään yhteenveto työstä.

2 Salesforce

Salesforce on alusta, jonka päälle tämä työ rakennetaan kokonaisuudessaan ja jonka kyvykkyyttä taipua finanssialan tarpeisiin on tarkoitus tutkia.

Salesforce on pilvipohjainen asiakkuudenhallintaohjelmisto (CRM, Customer Relationship Management), joka on aloittanut toimintansa vuonna 1999 (2. ja 4.). Se on tällä hetkellä maailman suosituin CRM-alusta hieman alle 20 prosentin markkinaosuudellaan (3). Jatkossa käytetään lyhennettä CRM, kun viitataan asiakkuudenhallintaohjelmistoon.

Salesforce aloitti alun perin yksinkertaisena CRM-ohjelmistona, josta syntyi sen ensimmäinen pilvipalvelu Sales Cloud. Sittemmin sen palvelut ovat laajentuneet erilaisiin pilvipalveluihin kuten Marketing Cloudiin, Community Cloudiin, IoT Cloudiin, Commerce Cloudiin ja Service Cloudiin. Tässä työssä hyödynnetään erityisesti Sales Cloud -palvelua (5.). Nykyisin Salesforce on siirtynyt pelkästä CRM-ohjelmistosta täysimittaiseksi SaaS (Software as a Service) -palveluksi (6).

Salesforcen halukkuus pyrkiä olemaan niin sanotusti ”No Software” -alusta tekee siitä tällä hetkellä halutuimman CRM-alustan. Tässä tapauksessa ”No Software” tarkoittaa sitä, että ei ole mahdollista syntyä sellaista ohjelmistoa, josta muodostuisi legacy-järjestelmä. Salesforce pyrkii jatkuvasti kehittämään uutta ja tarjoamaan välitöntä skaalautuvuutta asiakkailleen. Pilvipalveluidensa ansiosta Salesforce tarjoaa asiakkailleen ohjelmiston oikea-aikaisen päivitettävyyden, skaalattavuuden ja uusiutumisen. Tällä tavoin vältetään tilanne, jossa asiakkaalla olisi vanhentunutta ohjelmistoa. (7.)

Sales Cloud on yksi Salesforcen peruspilvipalveluista. Se on koko heidän CRMalustansa perusta. Tällä perustalla tapahtuu asiakkuuksien hallinta. Suuren kustomoitavuutensa ansioista Sales Cloud on mahdollista saada taipumaan moneen erilaiseen kaupalliseen prosessiin. Tämän työn aiheena onkin tarkastella sitä, kuinka Sales Cloud taipuu finanssitoiminnan tarpeisiin.

Sales Cloudin perimmäisenä tarkoituksena on hallita myyntiä ja asiakkuuksia. Ohjelmiston avulla käyttäjä pystyy pitämään yllä tietoja muun muassa asiakkaistaan, potentiaalisista asiakkaistaan sekä myyntimahdollisuuksista. Käyttäjä pystyy myös visualisoimaan, kuinka jokin asia esimerkiksi kehittyy. Tällaisia kehittymisen kohteita voivat olla muun

muassa asiakkaiden määrä. Seuranta on helppo toteuttaa käyttämällä Sales Cloudin raportointiominaisuutta. (8.)

3 Ohjelmointi – ja tietokantakyselykielet

Tämä luku käy läpi ja tarkastelee Salesforcessa käytettävää ohjelmointi- ja tietokantakyselykieltä. Perehdytään ensin siihen, mitä Salesforcen käyttämä Apex-kieli on ja miten se eroaa muun muassa Javasta. Tämän jälkeen tarkastellaan kielen erityispiirteitä ja käydään läpi sen syntaksia sekä asynkronista koodin ajoa. Luvun loppupuolella käsitellään käytettyä tietokantakyselykieltä SOQL (Salesforce Object Query Language). Tarkastellaan, mitä SOQL on yleisesti ja käydään sen käyttämää syntaksia läpi tähän työhön riittävällä tarkkuudella.

3.1 Apex

Apex on Salesforcen kehittämä yksityisomisteinen ohjelmointikieli, joka on hyvin samantyyppinen kuin Java tai C# (9). Kuten Java, on myös Apex vahvasti tyypitetty olio-pohjainen ohjelmointikieli. Javalla voidaan luoda ohjelmia, jotka pyörivät käytännössä kaikilla Javaa tukevilla alustoilla. Sen sijaan Apexilla voidaan tehdä ohjelmistoa, joka pyörii vain Force.com-alustan palvelimilla. Kielellä kyetään koodaamaan niin kustomoitua liiketoimintalogiikkaa, Apex-laukaisimia (engl. Triggers) kuin Visualforce-sivujen kontrollereita (10; 11.).

Tietokantaoperaatiot sekä -kyselyt ovat vahvasti integroituna Apexiin (esimerkkikoodi 1). Apexilla pystytään suoraan tekemään tietokantaoperaatioita, kuten ”insert” tai ”update”. Lisäksi SOQL-kyselykieltä hyödyntämällä voidaan tehdä kyselyitä tietokantaan. (11.)

```
Account account = new Account();
account.Name = 'Demo Account';
insert account;

account = [SELECT Id, Name, Type
                  FROM Account
                  WHERE Id =: account.Id];
account.Name = 'Example';
update account;
```

Esimerkkikoodi 1. Account-objektin luonti ja haku tietokannasta hyödyntäen SOQL-kyselykieltä sekä kyseisen Account-objektin Name-kentän päivittäminen Apexilla.

Vahva integraatio tietokantaoperaatioiden ja -kyselyiden kanssa tekee Apexilla koodin suunnittelusta hyvin yksinkertaista. Kuten esimerkkikoodi 1:stä voidaan nähdä saadaan koodi, tietokantaoperaatiot ja -kyselyt sidottua siististi yhdeksi kokonaisuudeksi.

3.1.1 Syntaksi

Kuten aiemmin on jo todettu, Apex on Javan kanssa hyvin samantyyppinen ohjelmointikieli. Tämä näkyy myös kielen syntaksissa, joka mukailee Javan vastaavaa kieltä. Niin kuin Javassa on Apex-kielessä mahdollista käyttää primitiivisiä datatyyppejä kuten Integer, Boolean, String, Enum sekä kokoelmia (engl. Collections), listoja ja käyttäjän tai järjestelmän määrittelemiä luokkia. (11.)

Apex kuitenkin omaa myös tiettyjä piirteitä, jotka erottavat sen Javasta. Apex-kielessä on mukana sObject (Salesforce Object) -tyyppi (11.). SObject esittää Salesforcen käyttämiä standardiobjekteja tai mukautettuja tietokantaobjekteja, kuten Account-objekti. Standardi- ja mukautettuihin objekteihin viittaus eroaa Apexissa siten, että standardiobjektiin viitatessa, objektin API (Application Programming Interface) -nimi sisältää vain objektin nimen kuten Account, kun taas mukautettuun objektiin viitatessa on API-nimessä lisäosa ”__c”. Alla on esimerkki, kuinka Apexissa määritellään erityyppisiä muuttujia.

```
//sObject muuttujia
Account account = new Account();
CustomObject__c = new CustomObject__c();

//Primitiiviset muuttjat
Integer i = 5;
Boolean isActive = true;
String name = 'Demo Person';

//Kokoelmat ja enum arvot
List<String> names = new List<String>();
public enum EngineType{ELECTRIC, GAS, DIESEL}
//Käyttäjän ja järjestelmän määrittelemät Apex luokat
CustomClass cClass = new CustomClass();
Http http = new Http();
```

Esimerkkikoodi 2. Erityyppisten muuttujien initialisointi Apex-ohjelmointikielellä.

Tässä on hyvä kiinnittää huomiota siihen, että vaikka sObject- ja Apex-luokka saattavat näyttää hyvin samanlaisilta, käyttäytyvät ne kuitenkin hyvin eri tavoin. Apexin luokat ovat samankaltaisia kuin Javankin ja sinne voi tehdä omia metodeitaan, mukautettua toiminnallisuutta sekä muuttujia. SObject sen sijaan esittää Salesforcen tietokannassa olevaa tietokantaobjektia, johon voidaan kohdistaa tietokantaoperaatioita, kuten ”insert”, ”update” ja ”delete”.

SObjektilla on järjestelmän määrittelemiä standardimetodeja, eikä sille voi suoraan tehdä omia mukautettuja metodeita. Mukautetun metodin tekeminen onnistuu kuitenkin välillisesti myös sObjektille. Se onnistuu tekemällä esimerkiksi oman Apex-luokan, joka voi joko ottaa sObjektin parametrina tai hakea sen tietokannasta jollain identifikaatilla, kuten sen id:n perusteella. Kun sObjektia käyttää, tulee muistaa, että jos sObjekti on tietokannasta haettu, tulee SOQL-haussa olla mukana kaikki kentät, joita luokka käyttää. Tämä voidaan varmistaa esimerkiksi esimerkkikoodi 3:ssa olevalla tavalla.

```
public class CustomObjectUtils{
    private CustomObject__c customObject__c{get;set;}
    public Boolean isInitialized{get{
                                  if(customObject != null){
                                      return true;
                                  }else{return false;}
                                  }private set;}

    public CustomObjectUtils(CustomObject__c customObject){
        try{
            customObject.get('Name');
            customObject.get('Place__c');
        }catch(SObjectException e){
            throw new CustomObjectUtilsExcetion('Required field
missing');
        }
        this.customObject = customObject;
    }

    public CustomObject__c doChanges(String name, String place){
        this.customObject.Name = name;
        this.customObject.Place__c = place;
    }
    global class CustomObjectUtilsExcetion extends Exception{}
}
```

Esimerkkikoodi 3. Apex -luokan määrittely.

Tässä on muutama huomionarvoinen seikka yllä olevasta koodista: Apexissa getterin ja setterin voi määritellä vain kirjoittamalla {get; set;}. Tällöin muuttuja saa perus get- ja set

-metodit itselleen eikä niitä tarvitse erikseen kirjoittaa. Jos haluaa jotain erityistä tapahtuvan jommassakummassa metodissa, voi sen määritellä esimerkkikoodi 3:n mukaisesti.

3.1.2 Asynkroninen koodinajo

Yksi erityinen ominaisuus Apexissa, jolla Apexista saadaan tehtyä ”monisäikeinen”, on @future annotoidut -metodit. Tämä mahdollistaa koodin suorittamisen myöhempänä ajankohtana. Ajankohtaa ei kuitenkaan pysty itse hallitsemaan vaan Salesforcella on tietty ”future job” -jono, joka suorittaa ”@future”-annotoituja metodeita sinne tulleessa järjestyksessä aina, kun resursseja on saatavilla. Huomioitavaa on myös se, että future-metodille voi antaa parametrina vain primitiivisiä datatyyppejä. Näitä ovat esimerkiksi Id ja String. Future-metodi voi ottaa niitä vastaan joko yksittäisenä muuttujana, listana tai kokoelmana. Metodille ei voi antaa oliota tai sObjektia parametrina, koska ne saattavat muuttua ennen metodin ajoa ja näin ollen dataa saatettaisiin ylikirjoittaa. (12.)

Tällaisia metodeita pystytään hyödyntämään muun muassa pitkäkestoisissa verkkopalvelukutsuissa tai ison datamäärän käsittelyssä. Kutsuttaessa ulkoista verkkopalvelua future-metodissa, tulee metodin annotaation perään lisätä merkintä @future(callout = true). (12.) Se ilmaisee, että metodi suorittaa verkkopalvelukutsun . Esimerkkikoodi 4:ssä on määritelty tällaisia metodeita. Tällä tavoin vältetään muun koodin suorituksen estyminen, kun pitkäkestoinen suoritus saadaan taustalle.

```
@future(callout=true)
public static void doFutureCall(List<Id> sObjIds){
    try{
        for(CustomObject__c cObj: [SELECT Id, Name, Place__c,
                                   Country__c FROM CustomeObject__c
                                   WHERE Id IN: sObjIds]){
            try{
                //Kutsu kustomoituun luokkaan, joka suorittaa
                //itse kutsun ulkoiseen verkkopalveluun
                CustomObjectUtils.sendObject(sObj);
            }catch(CalloutException cex){
                System.debug(cex.getMessage());
            }
        }
    }catch(QueryException qex){
        System.debug(qex.getMessage());
    }
}
```

Esimerkkikoodi 4. @future annotoidun-metodin määrittely.

Toinen käyttömahdollisuus on sellainen, jossa tarvitsee tehdä ensin tietokantaoperaatio ja sitten kutsu ulkoiseen verkkopalveluun. Salesforce ei salli tehdä tietokantaoperaatiota ja sen jälkeen verkkopalvelukutsua saman transaktion aikana. (13.) Tätä käyttötarkoitusta varten tässäkin työssä on pitänyt ottaa @future-metodit käyttöön.

3.1.3 Rajoitteet ja niiden hallinta

Vaikka Salesforce on hyvin joustava alusta, joka kykenee suoriutumaan monista eri tilanteista, tuo se mukanaan myös paljon rajoitteita. Salesforcen kannalta rajoitteet ovat kuitenkin hyviä, koska tällä tavoin saadaan hallittua palvelimien kuormitusta paremmin eikä yksittäinen Apex-koodin suoritus vie resursseja kaikilta muilta, jotka palvelua käyttävät. (14.)

Merkittävimpiä rajoitteita, joita Salesforce on asettanut Apex-koodin suorittamiselle ovat, Apex-koodin käyttämä CPU (Central Processing Unit) -aika, SOQL-hakujen ja niiden palauttamien tietueiden määrä, tietokanta operaatioiden ja tietueiden määrää, joihin operaatio kohdistuu, sekä verkkopalvelukutsujen määrä. (14.)

Esimerkkinä rajoitteiden mahdollisesti tuomasta ongelmasta on SOQL-haun palauttamien tietueiden määrä. Tämä raja voi helposti päästä ylittymään etenkin isossa Salesforce-organisaatiossa. Isoissa organisaatioissa yhdellä tietokantaobjektilla saattaa olla jopa miljoonia tietueita, jolloin esimerkiksi jonkin kentän massapäivitys kyseiselle objektille aiheuttaa massiivisen määrän tietokantakyselyssä palautettuja tietueita.

Rajoitteiden tuomiin ongelmiin on olemassa Salesforcen puolesta ratkaisuja. Salesforce suosittaakin suunnittelemaan koodin aina niin, että tietokanta ja tietokannan hakuoperaatiot suoritettaisiin aina joukko-operaationa. Tämä tarkoittaa sitä, että jos on tarve suorittaa näitä operaatioita isolle joukolle tietueita, tulisi ne kasata joukkoihin, jolloin voidaan suorittaa vähemmän operaatioita yhtä transaktiota kohden.

Toinen mahdollinen tapa helpottaa rajoitteiden tuomia ongelmia on tehdä Apex-työ (engl. Apex Job). Apex-työssä voidaan käsitellä suuria määriä tietueita ylittämättä Salesforcen rajoitteita. Apex-työlle voidaan antaa tietokantahakukutsu, joka voi kokonaisuudessaan palauttaa jopa miljoona tietuetta. Tämä on mahdollista sen vuoksi, että Apex-työ osaa hakea näitä tietueita 200 kerrallaan, jolloin rajat koskevat aina tätä 200 tietueen joukkoa.

(12.) Näin pystytään käsittelemään esimerkiksi aikaisemmin mainittu suuren tietuemäärän päivitys.

3.2 SOQL

3.2.1 Ominaisuudet

SOQL on Salesforcen oma tietokannan kyselykieli, se tulee sanoista Salesforce Object Query Language (16). Sillä voidaan suorittaa tietokantakyselyjä Salesforcen tietokantaan Apex -luokista. SOQL on samantyyppinen kuin SQL (Structured Query Language) -kyselykieli, mutta sillä ei voi tehdä esimerkiksi JOIN-operaatiota tai käyttää haussa villiä korttia (16). Villin kortin käytön mahdollisuuden puuttuminen aiheuttaa sen, että SOQL-lause saattaa olla hyvin pitkä. Lause muodostuu pitkäksi, koska lauseessa joutuu määrittelemään jokaisen kentän erikseen, jonka haluaa tietyltä tietokantaobjektilta hakea.

Salesforce mahdollistaa tietokannassaan objektien välisen viittauksen niin kuin yleensä tietokannoissa on mahdollista. Salesforcessa tämä tapahtuu määrittelemällä hakusuhdekenttä (Eng. Lookup Field) objektille, jolle halutaan määritellä suhde toiseen objektiin. Esimerkiksi Contact-objektilla, joka on standardiobjekti, on tällainen suhde Account -objektiin. Kun objektilla on edellä mainittu suhde toiseen objektiin, voidaan sitä hyödyntää SOQL-hakua tehdessä esimerkkikoodi 5:n osoittamalla tavalla eikä näin ollen tarvitse tehdä kahta erillistä tietokantakyselyä.

```
Contact contact = [SELECT Id, Firstname, CustomObject__c,
CustomObject2__r.Station__c Account.Type FROM Contact WHERE
Firstname = 'Demo' LIMIT 1];
System.debug(contact.Account.Type);
System.debug(contact.CustomObject2__r.Name);
```

Esimerkkikoodi 5. SOQL-tietokantakysely, joka sisältää haun sisällä viittauksen objektiin, johon on hakusuhde Contact-objektilla.

Kun hakusuhdetta käytetään SOQL-kyselyssä, tulee huomioida, että kyselyssä käytetty viittaus objektiin palauttaa vain Id:n kyseiselle objektille eikä koko objektia. Jos objektilta haluaa saada eri kenttien arvoja, tulee käyttää relaatioviittausta ja kentän nimeä.

Kun hakusuhteita käsitellään kyselyn sisällä, on huomionarvoista tarkastella sitä, kuinka viittaus tapahtuu riippuen siitä, onko kyseessä standardi, mukautettu kenttä vai

objekti. Esimerkkikoodi 5:ssä voi nähdä, kuinka objektilta haetaan sekä mukautettu objekti, sellaisen mukautettu kenttä sekä standardiobjektin standardikenttä.

Kuten jo luvussa 3.1.1 lyhyesti mainittiin, on mukautetun objektin API-nimessä aina ”__c” (kaksi alaviivaa ja c-kirjain) lisäliite. Esimerkkikoodi 5:ssä tästä mallina ”CustomObject__c”.

Kun SOQL -haussa halutaan viitata tällaisen objektin tiettyyn kenttään, muuttuu lisäliite muotoon ”__r” (kaksi alaviivaa ja r-kirjain). Tästä on hyvänä esimerkkinä esimerkkikoodi 5:ssä oleva viittaus ”CustomObject2__r.Street__c”. Tässä tapauksessa Contact-objektilla on olemassa hakuviittaus kyseiseen objektiin. Tämän hakuviittauskentän API-nimi Contact objektilla on ”CustomObject2__c”.

Standardiobjektiin viitatessa lisäliitteitä ei laiteta. Tästä esimerkkikoodi 5:ssä mallina ”Account.Type”. Tosin Salesforcessa on erikoispiirteitä joissain standardiobjekteissa historiallisista syistä, jolloin viittausten API-nimet saattavat poiketa siitä, mitä itse Salesforcen näkymästä voi nähdä. Toinen huomioitava seikka tässä on, jos haluaa viitata Contact objektilla siihen liitetyn Account-objektin id-kenttään, tulee hakuun kirjoittaa ”AccountId” sen sijaan, että kirjoittaisi vain ”Account”.

3.2.2 Syntaksi

SOQL-kielen perussyntaksi näyttää hyvin samalta kuin SQL-kyselykieli. SOQL-kysely alkaa aina sanalla ”SELECT”, jota seuraa lista kentän nimiä, jotka tulevat kyselyssä poimia. Tätä seuraa sana ”FROM”, jonka perään määritellään tietokantaobjekti, jolta ”SELECT”-kohdassa määritellyt kentät tulee hakea. Nämä osat riittävät kyselyn suorittamiseen ja rakentavat näin ollen perusrungon kyselylle.

Kuten esimerkkikoodi 6:ssa on nähtävissä, voidaan hakua rajata monin eri tavoin sekä käyttää muun muassa vamiiksi määriteltyjä arvoja kuten “TODAY”. Toinen huomionarvoinen asia on samassa esimerkissä viimeisellä rivillä olevan “Name”-kentän mukaan oleva suodaus. Tässä käytetyillä “%”-merkeillä ilmaistaan halu saada selville kaikki ne tiedot, joiden nimi sisältää sanan “Demo”. Samaa voidaan käyttää myös, jos halutaan saada tietoja, joiden kentän arvo alkaa sanoilla “Demo%” tai “%Demo”.

```
SELECT Id, Name FROM Account
SELECT Id, Name FROM Account WHERE Name = 'Demo Company'
SELECT count(Id) FROM Account WHERE CreatedDate = TODAY
SELECT Name FROM Account WHERE Type != null AND Name = '%Demo%'
```

Esimerkkikoodi 6. SOQL-kyselymalleja.

Tässä työssä on käytössä vain SOQL-kielen perusominaisuuksia eikä sen tarkoitus ole käsitellä kyseistä kieltä sen syvällisemmin, joten syvempi perehtyminen kielen saloihin ei ole työn kannalta oleellista.

4 Verkkopalvelut

Verkkopalvelu on palvelu, joka mahdollistaa laitteesta laitteeseen keskustelun ja datan välittämisen internetin yli. Verkkopalveluita varten on olemassa useita protokollia, kuten SOAP (Simple Object Access Protocol) ja REST. (18.) Näistä protokollista tässä työssä käytetään REST-protokollaa, sillä se on SOAP-protokollaan verrattuna huomattavasti modernimpi ja helpompikäyttöisempi.

Työn sisältämät REST-rajapinnat käyttävät tiedon siirtämiseen JSON (JavaScript Object Notation) muotoista dataa. Tämä on kevyt ja helppokäyttöinen tapa siirtää dataa moderneissa ohjelmistoissa, minkä vuoksi se sopii työn tarkoitukseen erittäin hyvin. JSON:in sekä REST-arkkitehtuurin käyttäminen on Salesforcessa suositeltu vaihtoehto. Tätä aihetta käsitellään tarkemmin luvuissa 4.2 ja 4.3.

4.1 REST

REST-rajapinnan kehitys alkoi tarpeesta saada yksinkertaisempi ja helppokäyttöisempi protokolla informaation lähettämistä varten internetpalveluille. Aikaisemmin vaihtoehtona oli käyttää SOAP-protokollaa, joka oli hyvin hankalakäyttöinen. Siinä käyttäjä joutuu luomaan XML (Extensible Markup Language) -tiedoston käsin, jonka sitten lähettää eteenpäin vastaanottavan palvelun osoitteeseen. (18.)

Aikaa ennen REST-protokollan määrittelyä ei osattu ymmärtää, kuinka suuri potentiaali internetpalveluilla voisi olla. Tämän tarpeen huomasi Roy Fielding tiimeineen vuosituhannen taitteessa ja alkoi kehittää uutta protokollaa. Heidän tavoitteenaan oli luoda uusi

standardi, jonka avulla mikä tahansa palvelin kykenisi keskustelemaan saumattomasti minkä tahansa muun palvelimen kanssa. Työnsä tuloksen Fielding esitteli väitöskirjassaan vuonna 2000. (19.)

4.1.1 Tilattomuus

REST-protokollan tarkoitus on olla tilaton. Tämä tarkoittaa sitä, että kaiken kutsun suorittamiseen liittyvän tiedon tulee sisältyä kutsuun, joka tulee asiakkaalta palvelimelle. Kutsun suorittaminen ei saa olla riippuvainen palvelimelle tallennetusta kontekstista. Tällä tavalla toimimalla saadaan session tila pidettyä asiakkaan hallussa koko ajan. (20.)

Tilattomuus parantaa merkittävästi kolmea seikkaa, jotka koskevat tämänkaltaisia protokollia. Nämä asiat ovat näkyvyys, luotettavuus ja skaalautuvuus. Näkyvyys paranee tilattomuuden myötä, sillä vastaanottava palvelu saa tietää koko kutsun tarkoituksen aina suoraan, kutsun itse toimittaman datan perusteella. Luotettavuus taas parantuu, koska osittaisesta virheestä on helpompi palautua. Tilattomuus mahdollistaa myös resurssien vapauttamisen palvelimella nopeammin kuin sellaisessa tilanteessa, jossa tilaa tulisi pitää yllä. Tämä auttaa palvelinta skaalautumaan paremmin, koska sen ei tarvitse jatkuvasti hallita resursseja peräkkäisten kutsujen tilojen ylläpitämiseksi. (20.)

4.1.2 Välimuistin käyttö

Yksi REST-protokollan tärkeistä ominaisuuksista on välimuistin rajoitteet, jotka vallitsevat asiakkaan ja palvelimen välillä. Jotta palvelin pysyy tilattomana, pidetään välimuistia yllä asiakkaan puolella. Tämän mahdollistamiseksi, tulee kyselyyn vastauksen olla merkitty välimuistiin tallennettavaksi tai ei tallennettavaksi. Kun vastaus on tallennettu välimuistiin, mahdollistaa se vastauksen uudelleen käytön, kun uusi vastaava kysely suoritetaan. (20.)

Vastauksen pitäminen välimuistissa ei tule ilman haittoja. Kun vastaus on välimuistissa, saattaa vastauksen data olla vanhentunutta siihen verrattuna, mitä se olisi ollut, jos kysely olisi tehty suoraan palvelimen kantaan. (20.)

Suurimmat hyödyt välimuistin käyttämisestä saadaan parantuneesta tehokkuudesta, skaalautuvuudesta sekä käyttäjän kokemasta pienemmästä viiveestä, joka on mahdollista saavuttaa vähempien kutsujen ansiosta. Kun vastaus on jo välimuistissa, vältytään turhilta kutsuilta palvelimelle. (20.)

4.1.3 Yhtenäinen rajapinta

Yhtenäinen rajapinta on yksi selkeimmistä eroista, joka erottaa REST-arkkitehtuurin muista internet-arkkitehtuureista. Kun rajapinnasta tehdään mahdollisimman yleiskäyttöisen, mahdollistaa se koko järjestelmän arkkitehtuurin yksinkertaistumisen sekä tekee integraatioista näkyvämpiä. (20.)

Tämäkin ominaisuus tuo mukanaan haittoja. Kun palvelu siirtää informaatiota standardisoisussa muodossa eikä tiettyä sovellusta varten räätälöidyssä muodossa, heikentää se palvelun tehokkuutta (20). REST on suunniteltu siirtämään tehokkaasti isoja määriä hypermediadataa. Tämä optimoi arkkitehtuurin tavalliseen selainkäyttöön, mutta tekee siitä vähemmän optimoidun muunlaisiin integraatioihin.

4.1.4 Operaatiometodit

REST-protokollan kanssa käytettään normaaleita HTTP (Hypertext Transfer Protocol) -metodeja määrittämään operaation laatu (Taulukko 1). (21.)

Taulukko 1. Yleisimmät HTTP -metodit ja -tehtävät (21.)

METODI	TEHTÄVÄ
GET	Noutaa dataa tietokannasta.
POST	Käytetään tiedon tallentamiseen/päivittämiseen tietokannassa.
PUT	Käytetään tiedon tallentamiseen/päivittämiseen tietokannassa.
PATCH	Määriteltyjen kenttien päivittämiseen tietokannassa.

DELETE	Käytetään datan poistamiseen tietokannasta.

Näitä metodeja hyödyntämällä kyetään tekemään erilaisia toimintoja REST-rajapintoihin. Esimerkiksi GET-metodilla voidaan toteuttaa tietojenhaku tietokannasta ja POST-metodilla tiedonlisäys tai muokkaus toiminnallisuutta.

4.2 JSON

JSON on kevyt datan siirtoformaatti. Se perustuu JavaScript-kielen osajoukkoon, joka on määritelty (Standard ECMA-262 3rd Edition – December 1999) standardissa. Ihmisen on helppo lukea JSON-muotoista dataa. Se on myös laitteille helppoa jäsentää ja generoida.

Tekstiformaatti JSON:issa on täysin ohjelmointikieliriippumatonta. Se noudattaa tuttua kaavaa, mikäli tuntee C-ohjelmointikieliperheeseen kuuluvia kieliä, kuten C:n, C++:n, C#:n, Javan sekä JavaScriptin. Nämä tekijät tekevät JSON-formaatista ideaalin datansiirtoon. (23.)

JSON koostuu kahdesta erilaisesta rakenteesta. Ensimmäinen näistä on rakenne, joka koostuu nimi-arvo-pareista. Monissa ohjelmointikielissä nämä realisoituvat olioina, tietueina ja muina vastaavina. Toinen rakenne on järjestetty lista. Tämä vastaa ohjelmointikielissä muun muassa taulukkoa, vektoria tai listaa. (23.)

Kaikki JSON:in käyttämät datamallit ovat universaaleita. Tämän vuoksi käytännössä kaikki modernit ohjelmointikielet pystyvät muuttamaan JSON-datan kielen käyttämäksi datamalliksi. Tämä tekee JSON:ista loistavan tavan välittää dataa. (23.)

JSON:issa kentän, olion tai taulukon nimi määritellään aina lainausmerkkien väliin. Tätä seuraa kaksoispiste, jonka jälkeen määritellään haluttu datatyyppi. Erilliset kentät, objektit ja taulukot erotellaan pilkulla toisistaan (esimerkkikoodi 7.).

Pelkän kentän määrittely tapahtuu esimerkkikoodi 7:n ensimmäisellä rivillä esitetyllä tavalla. Tässä kaksoispisteen jälkeen tulee haluttu kentänarvo, joka voi olla joko numero-, totuusarvo- tai merkkijonomuuttuja. Arvo voi myös olla ”null”. Näistä merkkijonomuuttuja

tulee laittaa heittomerkkien sisälle ja muut arvot sellaisenaan. Suomalaiseen käytäntöön tottuneena on numeroarvoissa hyvä huomioida desimaaleja sisältävät arvot. Desimaalit tulee erotella pisteellä pilkkujen sijaan. Tämän pitäisi toki olla tuttua kaikille, jotka tuntevat ohjelmointikieliä.

```
{
"atributeName" : "atributeValue",
"objectName" : {
                "fieldName" : true,
                "numberFieldName" : 10243
               }
"arrayName" : [{
                "fieldName1" : "value1",
                "fieldName2" : "value2"
               },
               {
                "fieldName1" : "differentValue1",
                "fieldName2" : "differentValue2"
               }
              ]
}
```

Esimerkkikoodi 7. JSON-formaatin rakenne. Esiteltynä yksittäinen kenttä, objekti ja taulukko.

Objektia määriteltäessä tulee objekti aina avata oikealle olevalla ({) ja sulkea vasemmalle olevalla (}) aaltosululla (esimerkkikoodi 7). Objektin sisällä arvot määritellään samoin kuin erillisiä kenttiä tehdessä. Objekti voi myös sisältää toisia objekteja.

Kun halutaan määritellä taulukko, tulee sille ensin määrittää nimi, niin kuin muillekin arvoille. Tämän jälkeen taulukon määrittely alkaa aina oikealle olevalla ([) hakasululla ja päättyy vasemmalle olevaan (]) hakasulkuun (esimerkkikoodi 7). Niin kuin moderneissa ohjelmointikielissä, tulee taulukon sisältää vain samantyyppistä dataa. Tällöin taulukoon ei voi laittaa sekaisin esimerkiksi objekteja ja erillisiä kenttiä.

4.3 Salesforce ja REST

Salesforce-standardin REST-rajapinnat mahdollistavat integroinnin ulkoisten järjestelmien kanssa, erityisesti ne mahdollistavat integraatiot mobiilijärjestelmiin. Kutsut käyttävät simppeleitä HTTP-metodeita ja niiden kanssa voidaan käyttää joko XML- tai JSON-formaattia. Salesforce tukee myös Apexin REST-rajapintoja, joka mahdollistaa mukautettujen rajapintojen koodauksen Apex-ohjelmointikieltä käyttäen. (24.) Tässä työssä

käytetään mukautettuja Apexin REST-rajapintoja, joten siitä seuraavaksi hieman tarkemmin.

Niin kuin kaikessa muussakin koodissa, jota Salesforce-alustalle tekee, tulee muistaa ottaa huomioon Salesforcen asettamat rajoitukset, kuten koodin suoritusaikaraja ja SOQL-hakujen määrät. Apexin REST-luokille erityisiä rajoja ovat kyselyn ja vastauksen maksimikoko, joka on 6 MB synkroniselle ja 12 MB asynkroniselle kutsulle. (26.)

Mukautetun Apexin RES-rajapinnan määrittely aloitetaan luomalla uusi Apex-luokka ja määrittelemällä se REST-resurssiksi. Tämä tapahtuu lisäämällä @RestResource-annotaatio luokan alkuun (esimerkkikoodi 8). Tämä mahdollistaa kutsut ulkoisista palveluista, jotka käyttävät REST-arkkitehtuuria. Samaan tapaan kuin luokan määrittäminen REST-resurssiksi voidaan metodi tämän Apex-luokan sisällä määrittää vastaamaan HTTP-metodia kuten GET asettamalla @HttpGet-annotaatio kyseiselle metodille. (26.)

```
@RestResource('customURL/*')
global class ThisIsAnRestAPI{
    @HttpGet
    global static CustomResponseClass doGet(){
        ...
    }
}
```

Esimerkkikoodi 8. Apex-luokan määritys REST-rajapinnaksi sekä metodin määritys HTTP-metodia vastaavaksi.

Apexin REST-rajapintaa määriteltäessä tulee huomioida, että rajapinta voi sisältää vain yhden kutakin HTTP-metodia vastaavan Apex-metodin. Rajapinnassa ei siis voi määritellä esimerkiksi kahta metodia, jotka ovat @HttpGet-annotoituja. (27.)

Apexin REST-rajapinnan metodi voi palauttaa erilaisia datatyyppejä, jotka Salesforce automaattisesti kääntää määriteltyyn palautusmuotoon. Kuten aiemmin on jo mainittu, voi rajapinta palauttaa joko XML- tai JSON-muotoista dataa. Tämän työn rajapinnat käyttävät vain JSON-muotoista palautusta. Datatyypit, joita rajapinta voi palauttaa, ovat void, sObject, primitiiviset muuttujat sekä käyttäjän määrittelemä mukautettu Apex-luokka. (27.)

Apex REST -rajapinnassa on mahdollista päästä käsiksi REST-kysely ja -vastaus kontekstiin (27). Tämä tapahtuu käyttämällä hyväksi Apexin tarjoamaa staattista luokkaa

nimeltä RestContext. Tällä tavoin pääsee käsittelemään kyselyssä olevaa sanomaa, jonka voi lopulta parsia esimerkiksi käyttäjän määrittelemäksi Apex-luokaksi. Samoin pääsee vastaukseen asettamaan sanoman tyypin sekä HTTP-statuskoodin (esimerkkikoodi 9).

```
RestRequest req = RestContext.request;
RestResponse res = RestContext.response;
CustomResponseClass response;
CustomPayloadClass payload;
try{
    JSONParser parser =
                JSON.createParser(req.requestBody.toString());
    payload =(CustomPayloadClass)parser
                .readValueAs(CustomPayloadClass.class)
}catch(JsonException jex){
    System.debug(jex.getMessage());
}
//Rajapinnan
...
res.statusCode = 200;
return CustomResponseClass;
```

Esimerkkikoodi 9. REST-kysely ja -vastaus sanomaan viittaus ja niiden käyttö.

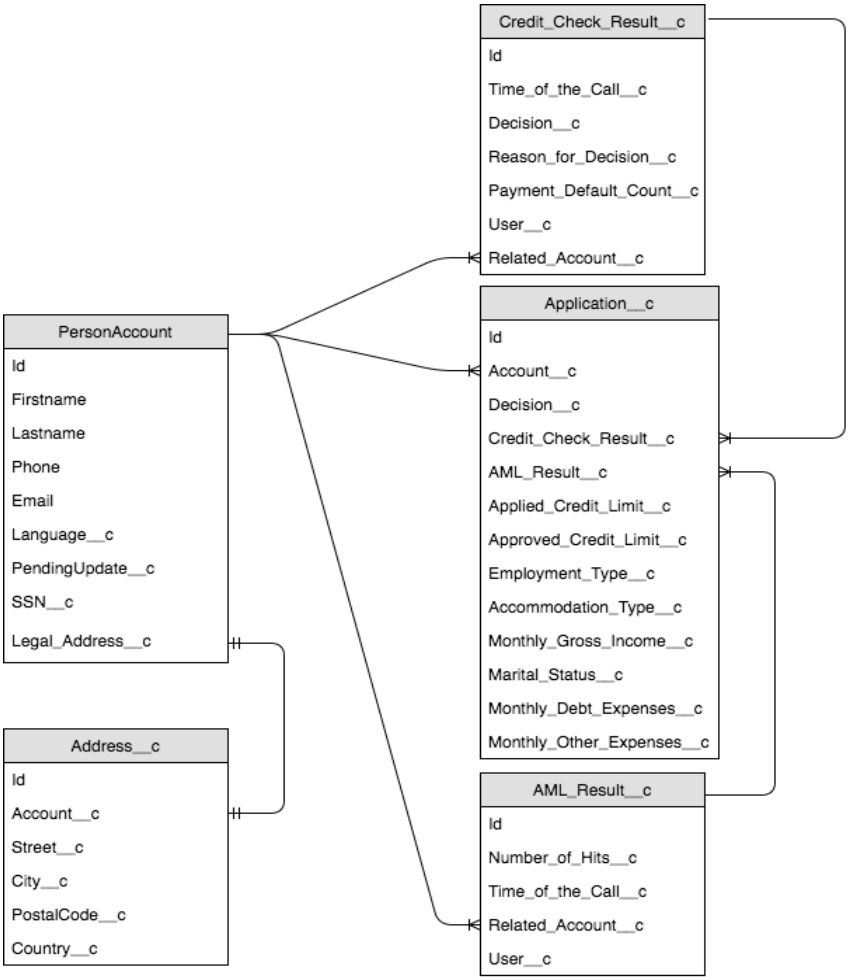
Nämä yksinkertaiset askeleet riittävät toimivan Apexin REST-rajapinnan määrittelyyn. Apexin REST-rajapinnat ovat joustava ja tehokas tapa altistaa Salesforce-data ulkoiselle järjestelmälle ja mahdollistaa näin saumaton integraatio sen ja Salesforcen välillä.

5 Toteutus

Tässä luvussa käsitellään itse työn toteutusta vaihe vaiheelta. Lisäksi luvussa tarkastellaan niitä havaintoja, joita tehtiin työn aikana. Toteutuksen käsittely koostuu Salesforce -tietokantarakenteen käsittelystä sekä itse REST-rajapintatoteutusten läpikäynnistä ja tarkastelusta. Tarkoituksena on esittää yleisellä tasolla, kuinka tällainen järjestelmä on mahdollista toteuttaa hyödyntämällä Salesforcea ja sen toiminnallisuutta. Toteutuksessa ei käsitellä testausta, koska tätä työtä varten on toteutettu ainoastaan Salesforcen minimi vaatimus koodin kattamiseksi.

5.1 Salesforce-tietokanta ja sen datamalli

Tietokanta suunnitellaan Salesforcen standardeja ja mukautettuja objekteja hyödyntäen. Tietokannassa pyritään hyödyntämään mahdollisimman paljon standardiominaisuuksia, jotta turhalta kustomoinnilta vältyttäisiin.



Kuva 1. Kuvaus käytössä olevasta datamallista.

Datamalli koostuu seuraavista tietokantaobjekteista (Salesforcessa tietokanta tauluja kutsutaan objekteiksi); PersonAccount, Address__c, Application__c, Credit_Check_Result__c, AML_Result__c.

Tässä datamallissa kaikki muut objektit ovat yhteydessä PersonAccount-objektin kanssa. Tälle objektille tallennetaan kaikki tieto asiakkaasta, jota hänestä halutaan ottaa

talteen. Osa tallennettavista tiedoista saadaan rekisteröitymisrajapinnan kautta ja loput tiedot saadaan, kun tehdään luottoluokituksentarkastus. Person Account on yksi Salesforcen standardiobjekteista, joka on niin sanottu hybridi kahdesta muusta standardiobjektista (Account ja Contact).

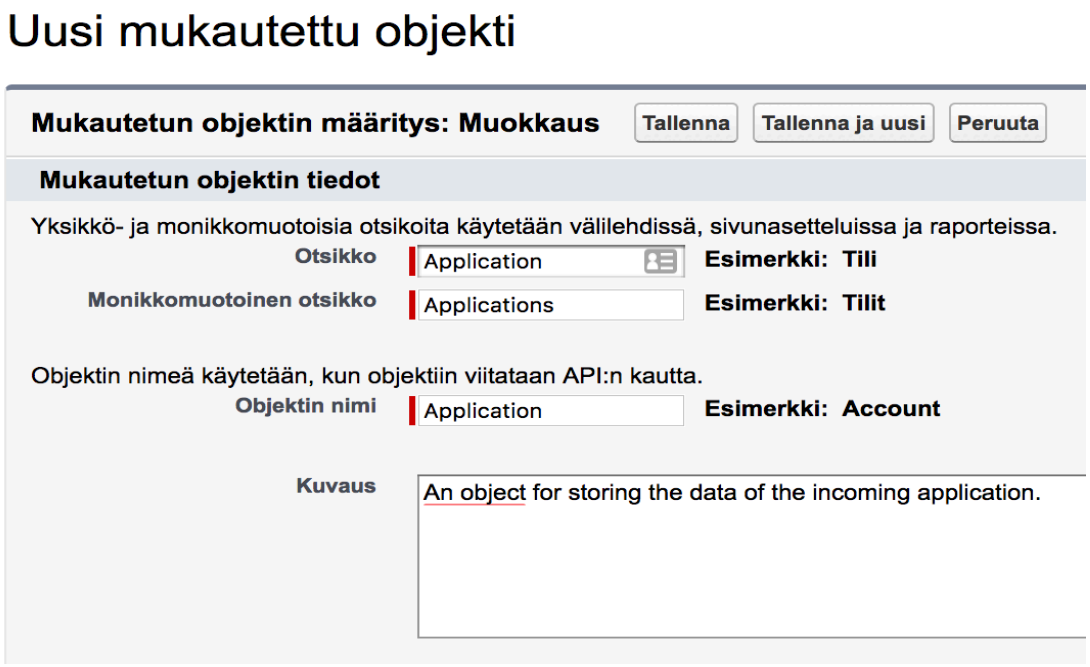
Jokaiselle Person Accountille on mahdollista liittää Address__c-objekti. Objektille tallennetaan perustiedot asiakkaan osoitteesta, kuten maa, katuosoite, postinumero ja postitoimipaikka. Tässä työssä tätä objektia hyödynnetään tallentamaan asiakkaan mahdollinen lain mukainen osoite, se saadaan väestörekisterikeskukselta luottotietotarkistuksen yhteydessä.

Application__c-objekti on täysin kustomoitu objekti. Sitä hyödynnetään asiakkaan luotonhakulomakkeen tietojen tallennukseen. Objektille tallennetaan luotonhaku REST-rajapinnan kautta saatavat tiedot sekä tarvittavat ulkoisen järjestelmän identifikaationumerot, jotka saadaan palautuksena ulkoista järjestelmää kutsuttaessa. Luotonhakuun liittyvää REST-rajapintaa käsitellään myöhemmässä luvussa.

Kun luottoa haettaessa suoritetaan luottoluokituksen tai AML (Anti-money laundering) -tietojen tarkastus, tallennetaan niistä tiedot siitä kuka, milloin ja ketä asiakasta varten kutsu suoritettiin. Lain mukaan tiedot henkilöluottotietojen käsittelystä tulee kirjata talteen (1.). Tätä varten datamallissa on objektit Credit Check Result ja AML Result.

Kuten kuvasta 1 voidaan nähdä, tallennetaan näille objekteille myös tietoa kyselyn tuloksesta. Credit Check Result -objektille tallennetaan tieto siitä, onko luoton myöntäminen asiakkaalle kannattavaa sekä mahdollinen syy päätökselle. AML Result -objektille taas tallennetaan tieto siitä, kuinka monta osumaa henkilö sai kyselyssä. Kuvasta 1 voidaan myös nähdä, että nämä kaksi objektia ovat yhteydessä Application-objektiin. Tämä on toteutettu hakusuhdekenttiä käyttämällä. Kuten nimistäkin jo voi päätellä, viittaa Credit_Check_Result__c kenttä Credit Check Result -objektiin ja AML_Result__c kenttä AML Result -objektiin.

Tietokantaobjektien luonti Salesforcessa on hyvin helppoa ja yksinkertaista. Mukautettujen objektien luonti tapahtuu kätevästi käytössä olevan Salesforce-organisaation ”mukautetut objektit” -sivulta klikkaamalla ”Uusi mukautettu objekti” -painiketta. Tämä vie käyttäjän selkeän luomisprosessin läpi (kuva 4).



Kuva 2. Esimerkki mukautetun objektin luomisesta. Objektille annetaan vain tarvittavat tiedot luomista varten.

Kun objekti on saatu luotua, pystyy sille lisäämään kenttiä klikkaamalla ”lisää uusi kenttä” -objektia pääsivulla. Mukautetun kentän luominen on samankaltainen prosessi kuin objektinkin luonti. Kenttien luomisstandardi mukautetulle objektille on hyvin samanlainen prosessi. Salesforcen sisäisten perusasioiden toiminta ei ole tämän työn pääosassa, joten aiheeseen ei tämän tarkemmin perehdytä.

5.2 Käyttäjän rekisteröityminen

Salesforcen kannalta käyttäjän rekisteröiminen käsittää REST-rajapinnan, jonka avulla vastaanotetaan uuden asiakkaan perustiedot kuten nimi, sosiaaliturvatunnus, sähköpostiosoite, puhelinnumero, maa- ja kielikoodi. Rajapinta on suunniteltu vastaanottamaan JSON-objektin, joka sisältää kaikki tarvittavat tiedot käyttäjästä. Tästä on malli esimerkkikoodi 10:ssä.

```
{
“ssn” : “111111-9999”,
“mobilePhone” : “+358501234567”,
“email” : “demo@email.com”,
“country” : “FI”,
“lastName” : “User”,
“firstName” : “Demo”,
“language” : “FIN”
}
```

Esimerkkikoodi 10. Kuvaus JSON:ista, jonka rekisteröitymisrajapinta ottaa vastaa.

Tätä rajapintaa varten luotiin Apex-luokka. Uuden Apex-luokan luominen onnistuu käytössä olevan Salesforce-organisaation Apex-luokat sivulta klikkaamalla ”uusi”-nappia. Kun luokka on luotu, voidaan varsinainen koodaaminen aloittaa. Jotta Salesforce tunnistaa luokan olevan REST-rajapintakuvaus, tulee luokan määrityksen yläpuolelle kirjoittaa seuraavanlainen annotaatio: @RestResource(urlMapping=’/Register/v1/*’). Tässä urlMapping-arvon tulee olla uniikki. Ensimmäisen / -merkin jälkeen oleva teksti ilmaisee rajapinnan nimen ja seuraan jälkeen oleva ”v1” on versioidentifikaatio rajapinnalle.

```
@RestResource(urlMapping=’/Register/v1/*’)
global class RegistrationRestAPI{
    //Luokan sisältö
}
```

Esimerkkikoodi 11. Apex-luokan määrittely REST-rajapintaa varten.

Kuten luvussa 4.4 kerrotaan, voi rajapintaa varten toteutuksessa olla sisäluokka, joka vastaa JSON-objektin sisältöä. Myös rajapinnan palauttamaa JSON-objektia varten tulee olla sisäluokka (esimerkkikoodi 8). Rajapinnan vastaanottama JSON tulee parsia määritellyksi sisäluokaksi Apexin JSON-parseria hyväksikäyttämällä. Esimerkki tästä löytyy luvussa 4.4, jossa perehdytään REST-rajapintojen käyttöön ja luontiin Salesforcessa. Tämän jälkeen pystytään vastaanotettuihin arvoihin viittaamaan samaan tapaan kuin normaaliin Apex-luokan muuttujiin. Koko Apex-luokan esimerkkitoteutus löytyy liitteestä 1.

```
public class Payload{
    public String ssn         {get;set;}
    public String mobilePhone {get;set;}
    public String email       {get;set;}
    public String country     {get;set;}
    public String lastName    {get;set;}
    public String firstName   {get;set;}
    public String language    {get;set;}
}
```

Esimerkkikoodi 12. Kuvaus sisäluokasta, joka vastaa vastaanotettua JSON-objektia.

Seuraava tarvittava palanen REST-rajapintaa varten on metodin luonti HTTP POST -tapahtumaa varten. Tämä tapahtuu esimerkkikoodi 13:ta mukaisesti tekemällä metodi, joka annotoidaan @HttpPost. Tämä kertoo Salesforcelle, mikä metodi tulee ajaa, kun HTTP POST -kutsu tulee luokassa määriteltyyn osoitteeseen. Luokalle voisi määritellä myös muita HTTP-tapahtumia vastaavia metodeita, kuten GET, mutta niitä ei tässä työssä tarvita.

```
@HttpPost
global static Response doPost(){
    //Metodin sisältö
}
```

Esimerkkikoodi 13. Apex-metodin määrittely vastaamaan HTTP-tapahtumaa. Palautusarvona metodilla on sisäluokkana määritelty Apex-luokka, josta Salesforce tekee JSON-objektin automaattisesti paluuviestiä varten.

Rekisteröintipyynnön varsinainen käsittely on toteutettu esimerkkikoodi 13:ssa esitellyn metodin sisälle. Uuden asiakkaan rekisteröityminen oli loppuen lopuksi hyvin suoraviivainen tehtävä. Rekisteröinnin tuli sisältää seuraavat asiat: virheen käsittelyn, kutsussa tulevan JSON-datan validoinnin, duplikaattien etsimisen sosiaaliturvatunnuksen perusteella, täysi-ikäisyyden tarkistuksen sekä asiakkaan tietojen tallennuksen tai päivityksen.

Metodin virheenhallintaa lähdettiin toteuttamaan try- ja kahta catch -koodilohkoa hyödyntäen. Onnistunut try-lohkon suoritus johtaa onnistuneeseen kutsuun ja HTTP 200 -statukseen. Jos taas tapahtuu ennalta odotettava virhe, kuten sellainen, jossa vastaanotettu JSON sisältää virheellistä dataa, heitetään Apex-luokassa määritelty oma virhe (Liite 1 (3/3)). Tämä otetaan ensimmäisessä catch-lohkossa kiinni, se johtaa virhevastaukseen ja HTTP 400 -statukseen. Toisessa catch-lohkossa on tarkoitus ottaa kiinni mahdolliset

systeemivirheet, joihin ei ole varauduttu. Tällainen systeemivirhe voi olla erimerkiksi tilanne, jossa tapahtuu odottamaton null pointer exception. Tällöin kyseinen lohko ottaa virheen kiinni. Tällaisessa tapauksessa on kyse palvelinpuolen virheestä, jolloin palautetaan HTTP 500 -statuskoodi.

```
try{
     //Lohkon sisältö-->
     ...
     //<--
     //Esimerkki oman virheen heittämisestä
     throw new RegistrationException('INVALID PAYLOAD');
}catch(RegistrationException rex){
     res.statusCode = 400;
     response.message = rex.getMessage();
}catch(Exception e){
     res.statusCode = 500;
     response.message = e.getMessage();

}
```

Esimerkkikoodi 14. Virheen käsittelyn toteutus rekisteröintiluokassa.

Kutsun mukana tulevan JSON-datan validointiin päädyttiin tekemään hyvin simppeli metodin, joka tarkastaa jokaisen vaaditun kentän sen varalta, että ne ovat olemassa ja niihin on annettu jokin arvo (esimerkkikoodi 15). Jos saatu kenttä ei vastaa kriteereitä, heitetään itse määritelty poikkeus ja palautetaan virhettä kuvaava teksti. Kuten edellisessä kappaleessa mainittiin, on tällaisessa tapauksessa virheen koodina HTTP status 400.

```
private static void mandatoryFilled(Payload payload){
     if(String.isEmpty(payload.firstName)){
          throw new
               RegisterAccountException('MISSING FIRST NAME');
     }
     if(String.isEmpty(payload.ssn)){
          throw new RegisterAccountException('MISSING SSN');
     }
     ...
}
```

Esimerkkikoodi 15. Metodille annetaan Apex-luokaksi parsittu JSON-parametrina. Tämän jälkeen kaikki pakolliset kentät tarkastetaan.

Duplikaattien etsintä toteutettiin SOQL-kyselyä hyödyntämällä. Luotiin kysely, joka etsii olemassa olevia PersonAccount-objekteja JSON-datassa saadun sosiaaliturvatunnuksen perusteella. Jos kyselyllä saadaan osuma, päivitetään kyseisen PersonAccountin

tiedot kutsussa saaduilla tiedoilla. Kyselyn palauttaessa nolla osumaa luodaan uusi PersonAccount, jolle annetaan kyselyssä saadut tiedot.

```
List<Account> oldAccounts = new List<Account>();
oldAccounts = [SELECT Id, Lastname, Firstname, Phone,
                  Language__c,Email
                  FROM Account WHERE SSN__c =: payload.ssn];

if(oldAccounts.isEmpty()){
    //Luo uusi PersonAccount
}else{
    //Päivitä vanhaa PersonAccounttia
}
```

Esimerkkikoodi 16. Duplikaattien etsintä rekisteröityessä.

Täysi-ikäisyys tarkastusta varten luotiin hyvin yksinkertainen metodi, joka tekee saadusta sosiaaliturvatunnuksesta Apex-Date-tyypin muuttujan ja sekä toisen Date-tyypin muuttujan, joka on koodin ajon aikana vallitseva päivä, josta on vähennetty 18 vuotta. Näitä kahta vertaamalla saadaan selville, onko henkilö täysi-ikäinen (esimerkkikoodi 17).

```
private static Boolean isAdult(String ssn){
    String yearStart = '';
    if(ssn.contains('-')){
        yearStart = '19';
    }else if(ssn.contains('A')){
        yearStart = '20';
    }else if(ssn.contains('+')){
        yearStart = '18';
    }
    Date birthDay = Date.newInstance(
                //Year
                Integer.valueOf(yearStart+ssn.substring(4,6)),
                //Month
                Integer.valueOf(ssn.substring(2,4)),
                //Day
                Integer.valueOf(ssn.substring(0,2)));
    Date today = Date.today();
    return today.addYears(-18) >= birthday;
}
```

Esimerkkikoodi 17. Henkilön täysi-ikäisyyden tarkastaminen.

Viimeiseksi asiaksi rajapinnassa toteutettiin PersonAccountin luonti/päivitysoperaatio. Se luotiinko vai päivitettiinkö PersonAccounttia, haluttiin palauttaa vastauksessa. Tätä varten tämä operaatio toteutettiin käyttämällä Apexin Database-luokkaa hyväkseni. Tätä

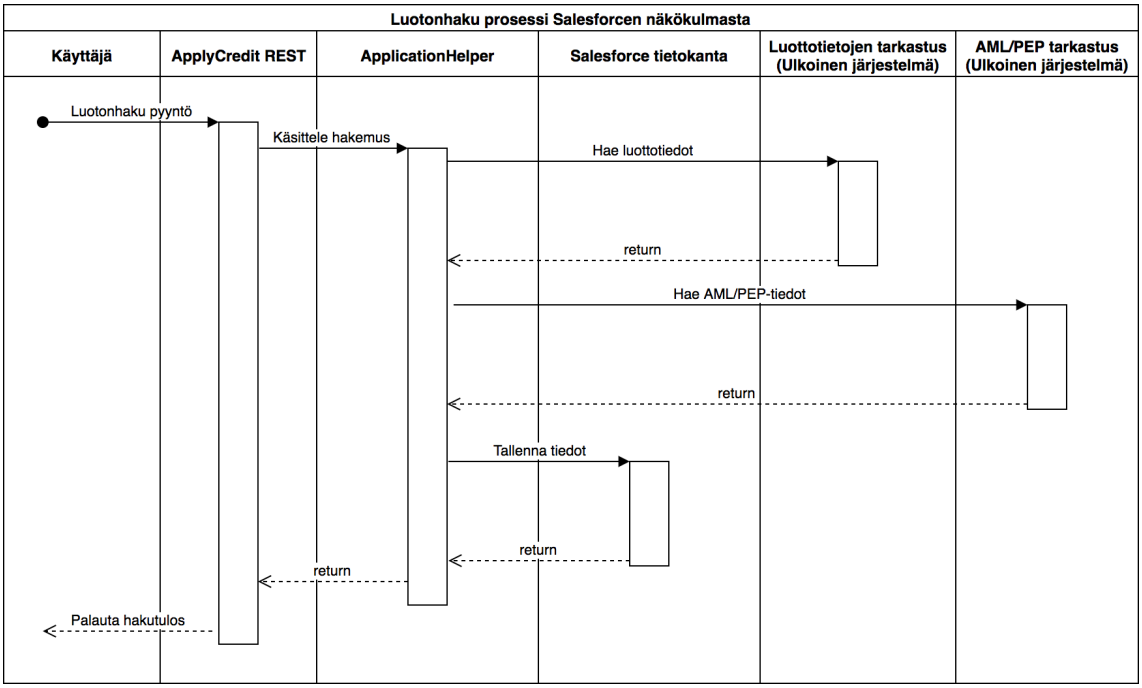
luokkaa käyttämällä kyetään ottamaan ”upsert”-tietokantaoperaation tulos talteen. Siltä nähdään, oliko kyseessä ”insert” vai ”update” (Esimerkkikoodi 18.). Tietokantaoperaation jälkeen asetetaan rajapinnan paluu JSON:ia vastaavan Response -luokan arvot ja laitetaan HTTP-statuskoodin arvoksi 200 ja palautetaan vastaus.

```
Database.UpsertResult upsertResult;
try{
    upsertResult = Database.upsert(account);
}catch(DmlException dex){
    throw new RegistrationException('UPSERT FAILED. ' +
                                    dex.getMessage());
}
if(upsertResult.isCreated()){
    //Uuden accountin haku tietokannasta
    ...
    response.message = 'New account create';
    response.created = true;
}else{
    response.message = 'Old account updated';
    response.created = false;
}
```

Esimerkkikoodi 18. Upsert-tietokantaoperaatio Apex Database-luokkaa hyödyntäen tuloksen saamiseksi.

5.3 Luotonhaku

Luotonhakuprosessi käsittää Salesforcen osalta seuraavanlaiset komponentit. REST-rajapinta vastaanottaa kaikki tarvittava informaatio luottohakemusta varten. Tämä rajapinta sisältää datan validoinnin, virheiden hallinnan ja tekee pyynnön suorittaa luotonhakua koskevat kutsut. Toinen osa toteutusta on erillinen luokka, joka suorittaa luottotietojen-tarkastuksen sekä AML- ja PEP (Politically exposed person) -tietojen tarkastamisen. AML ja PEP ovat saman kutsun sisällä. Tämän lisäksi on toteutettu erilliset luokat molempia kutsuja varten, jotta eri osia koodista olisi tarpeen tullen mahdollisimman helppo vaihtaa (kuva 3).



Kuva 3. Yleisnäkymä luotonhakuprosessin kulusta Salesforcen näkökulmasta.

Koko luotonhakuprosessi on kuvattuna kuvassa 3. Tästä nähdään, kuinka Apex-luokat toimivat keskenään REST-kutsun tullessa järjestelmään. Kuvasta voidaan myös nähdä, kuinka tarvittavat verkkopalvelukutsut suoritetaan koodia ajettaessa.

5.3.1 REST-rajapinta

Luotonhakuprosessin implementointi aloitettiin luomalla uusi Apex-luokka REST-rajapintaa varten samaan tapaan kuin rekisteröintivaiheessakin. Rajapinta suunniteltiin vastaanottamaan esimerkkikoodi 19:sta esiteltävä JSON -objekti. Tämä objekti sisältää tarvittavat tiedot luotonhakua varten. Luotonhaussa käytetään myös informaatiota, joka saatiin jo rekisteröintivaiheessa, kuten sosiaaliturvatunnusta ja nimeä.

```
{
“customerID” : “C999999”,
“appliedLimit” : 1000,
”employmentType” : “PERMANENT”,
“accommondationType” : “RENTAL”,
“monthlyGrossIncome” : 4000,
“maritalStatus” : “SINGLE”,
“monthlyDebtExpenses” : 400,
“monthlyOtherExpenses” : 500
}
```

Esimerkkikoodi 19. Luotonhaku REST-rajapinnan vastaanottama JSON.

Samoin kuin rekisteröintirajapinta, sisältää tämäkin vastaanotettavan ja takaisin lähetettävän JSON-objektin mallin Apex-sisäluokkana. Rajapinta sisältää myös virheiden käsittelyn, joka noudattaa samaa tapaa kuin rekisteröintirajapintakin (Esimerkkikoodi 14.). Luokalle määritettiin esimerkkikoodi 11:n mukaisesti uniikki osoite, jotta sitä voidaan kutsua; @RestResource(urlMapping='/ApplyCredit/v1/*'). Rajapinnan datan validointi on myös toteutettu samaan tyyliin kuin rekisteröitymisrajapinnassa. Tämä on esitelty esimerkkikoodi 15:sta.

Seuraavana rajapinnassa tarvitsee hakea rekisteröintivaiheessa luotu PersonAccount sekä mahdollisesti olemassaoleva osoite. Tämä on toteutettu SOQL-kyselyllä, jossa uniikkina avaimena toimii kutsussa tuleva ”customerID”, joka vastaa PersonAccountin Id:tä. Sen avulla haetaan tarvittavat kentät, kuten FirstName, LastName, SSN__c, Legal_Address__c. Osoite haetaan käyttämällä PersonAccountilla olevaa Legal_Address__c hakusuhdekenttää hyväksi (esimerkkikoodi 20).

```
Account account = [SELECT FirstName, LastName,
                          SSN__c, Legal_Address__c
                   FROM Account WHERE Id =: payload.customerID];
Address__c legalAddress = [SELECT Account__c, Street__c,
                                  City__c, PostalCode__c,
                                  Country__c
FROM Address__c WHERE Id =: account.Legal_Address__c];
```

Esimerkkikoodi 20. PersonAccountin ja Address__c-objektien haku tietokannasta luotonhakuprosessin esitiedoiksi.

Kun asiakkaan jo olemassa olevat perustiedot on haettu, voidaan luoda itse Application__c-objekti. Tälle objektille annetaan REST-kutsun mukana saadut tiedot sekä annetaan viite PersonAccount-objektiin (esimerkkikoodi 21).

```
private static Application__c createApplication(Account account,
Payload payload){
    Application__c application = new Application();
    application.Account__c = account.Id;
    application.Applied_Credit_Limit__c = payload.appliedLimit;
    application.Employment_Type__c = payload.employmentType;
    application.Accommodation_Type__c =
                                  payload.accommodationType;
    application.Monthly_Gross_Income__c =
                                  paylaod.monthlyGrossIncome;
```

```
    application.Marital_Status__c = payload.maritalStatus;
    application.Monthly_Debt_Expenses__c =
                                    paylaod.monthlyDebtExpenses;
    application.Monthly_Other_Expenses__c =
                                    payload.monthlyOtherExpenses;
    return application;
}
```

Esimerkkikoodi 21. Application__c-objektin luonti REST-luokassa.

Tässä vaiheessa rajapinnan suoritusta on saatu kasaan kaikki tarvittava data itse luotonhakuprosessia varten. Tätä prosessia varten luotiin toinen Apex-luokka, jotta mahdollisten muutosten vaikutus itse rajapintaan olisi mahdollisimman vähäinen. Tämä luokka hoitaa molemmat tarvittavat kutsut sekä kaikkien tarpeellisten objektien tallennuksen. Kyseinen luokka käydään tarkemmin läpi seuraavassa luvussa.

```
ApplicationHelper.Creditworthiness creditworthiness =
ApplicationHelper.getCreditworthiness(account, application,
legaladdress);
```

Esimerkkikoodi 22. Kutsu käsittelevään luokkaan REST-rajapinnasta.

Kun käsittelevä luokka palauttaa tuloksen, voidaan kaikki tarvittavat objektit hakea tietokannasta käyttämällä id:itä, jotka löytyvät Creditworthiness-olion sisältä. Tämä haku suoritetaan, jotta varsinainen päätös luoton myöntämisestä saadaan, sillä tämä logiikka on sijoitettu Apex-laukaisimeen. Tästä tarkemmin seuraavassa luvussa.

```
Application__c application =
                [SELECT Decision__c
                FROM Application__c
                WHERE Id =: creditworthiness.application.Id];
```

Esimerkkikoodi 23. Päätöksen haku tietokannasta Creditworthiness-oliota hyödyntäen.

Kun päätös on saatu haettua tietokannasta, voidaan paluusanomaobjekti täyttää tiedoilla, jotka hakijalle halutaan palauttaa ja lopuksi tehdä itse palautus.

```
//http status koodi
res.statusCode = 200;
//itse määritellyn paluusanoman täyttö
response.decision = application.decision;
response.message = 'application succesfully handled';
response.success = true;
```

Esimerkkikoodi 24. Luotonhaku rajapinnan paluusanoma

5.3.2 Luotonhakupyynnön käsittely

Kuten esimerkkikoodi 22:sta voidaan nähdä, palauttaa luotonhakuprosessi ”Creditworthiness” nimisen Apex-luokan, joka on ApplicationHelper-luokan sisäluokka. Tämä sisäluokka pitää sisällään kaikki tarpeelliset objektit, jotka tulee tallentaa kyselyn lopuksi. Tämä on niin sanottu ”wrapper”-luokka. Tämä wrapper-luokka annetaan aina eteenpäin metodilta metodille haun suorituksen aikana, matkalla metodit täyttävät tätä wrapperia uusilla objekteilla aina tarpeen tullen.

```
public class Creditworthiness{
    public Application__c          application{get;set;}
    public Account                 account{get;set;}
    public Address__c              legalAddress{get;set;}
    public AML_Result__c           amlResult{get;set;}
    public Credit_Check_Result__c ccResult{get;set;}
}
```

Esimerkkikoodi 25. Creditworthiness wrapper -luokan määrittely

Kyseinen wrapper otettiin käyttöön, koska Salesforce ei pysty tekemään saman transaktion aikana tietokantaoperaation jälkeen verkkopalvelukutsua. Tätä wrapper-luokkaa käyttäen voidaan aina verkkopalvelukyselyn seurauksena luotu objekti ottaa talteen. Koko prosessin lopuksi suoritetaan tietokantaoperaatio, joka tallentaa kaikki objektit objekti kerrallaan.

ApplicationHelper sisältää kolme metodia, jotka yhdessä huolehtivat luottoluokitus- ja AML/PEP-kutsuista. Luokka sisältää myös muutaman pienemmän metodin hoitamaan muun muassa tallennuksen ja käsittelyn asynkronisesti. Kolme päämetodia ovat itse getCreditworthiness-metodi sekä checkCredit- ja checkAML -metodit.

```
public static Creditworthiness getCreditworthiness(
                              Application__c application,
                              Account account,
                              Address__c legalAddress){
    Creditworthiness cw = new Creditworthiness();
    //Application ja Account ovat pakollisia
    if(application == null || account == null){
        //Itse määritelty virhe luokka
        throw new CreditworthinessException(
                              'Required parameters missing');
    }
```

```
        //Lisätään creditworthiness objektille saatavilla olevat
        //objektit. Tähän lisätään objekteja sitä mukaan kun niitä
        //luodaan kutsujen aikana.
        cw.account = account;
        cw.application = application;
        cw.legalAddress = legalAddress;
        //Luottotietojen tarkastus. Tämä metodi suorittaa
        //luottoluokitus kutsun sekä lisää Credirworthiness
        //objektille Credit_Check_Result__c objektin, joka sisältää
        //kutsun tiedot sekä luottoluokitus päätöksen. Saman kutsun
        //mukana saadaan lainmukainen osoite, joka myös otetaan
        //talteen.
        cw = checkCredit(cw);
        //AML ja PEP tietoten tarkastus. Suorittaa itse
        //verkkopalvelu kutsun sekä lisää AML_Result__c objektin
        //Creditworthiness objektille. AML_Result__c pitää sisällään
        //tiedon osumien määrästä
        cw = checkAML(cw);
        //Lopuksi suoritetaan tietokanta objektien tallennus. Metodi
        //tallentaa/päivittää pakolliset objektit eli Application__c
        //- ja Account objektin sekä kaikki objektit jotka
        //matkanvarrella luotiin.
        cw = saveRecords(cw);
        return cw;
}
```

Esimerkkikoodi 26. Luotonhaun prosessointi ApplicationHelper-luokassa.

Pienemmistä metodeista asynkronisesti luotonhakukäsittelyn hoitava metodi on @Future annotoitu getCreditworthinessAsync, joka mahdollistaa sen ajamisen myöhempänä ajankohtana. Koodin asynkronisesta ajosta tarkemmin luvussa 3.1.3, jossa on selitettynä, kuinka asynkroninen koodinajo Salesforcessa tapahtuu. Asynkroninen luokka on toteutettu, jotta Application-objektin luonti onnistuu myös käsin Salesforcen käyttöliittymän kautta niin, että luottoluokitus ja AML/PEP-tiedot ja -päätös saadaan automaattisesti hakemukselle.

Tätä asynkronista metodia kutsutaan Application-objektin Apex-laukaisimesta. Kuten jo aiemmin olen tässä työssä maininnut, ei Salesforce kykene saman transaktion aikana tekemään verkkopalvelukutsuja enää tietokantaoperaation jälkeen. Näin ollen verkkopalvelukutsuja ei voi tehdä suoraan Apex-laukaisimesta vaan tulee käyttää @Future-annotoitua metodia.

Kolmesta päämetodista ensimmäinen käsitellään tarkemmin esimerkkikoodi 26:ssa. Tämä metodi pitää sisällään kutsut kahteen muuhun tärkeään metodiin; checkCreditiin

ja checkAML:iin. Molempien metodien suorittamat verkkopalvelukutsut on vielä eriytetty erillisiin luokkiin, jotta osia pystytään lisäämään ja poistamaan tarpeen tullen.

Näistä checkCredit pitää sisällään luottoluokituksen hakukutsun, jonka perusteella se luo Credit_Check_Result__c-objektin.

```
private Creditworthiness checkCredit(Creditworthiness cw){
    //Erillisessä luokassa oleva HTTP kutsu
    ResponseCredit responseCredit = CreditCheck.doCallout(cw);
    Credit_Check_Result__c ccResult =
                                new Credit_Check_Result__c();
    //Credit_Check_Result__c objektin muuttujien asetus
    ccResult.Decision__c = responseCredit.decision;
    ...
    //Osoitteen asetus
    Address__c legalAddress = new Address__c();
    legalAddress.Street__c = responseCredit.address.street;
    ...
    cw.ccResult = ccResult;
    cw.legalAddress = legalAddress;
    return cw;
}
```

Esimerkkikoodi 27. Metodi, joka suorittaa luottoluokitustarkastuksen.

Toinen osa tietojen tarkastusta on checkAML-metodi, joka suorittaa verkkopalvelukutsun saadakseen tiedot mahdollisista AML/PEP-merkinnöistä, joita luotonhakijalla saattaa olla. Metodi on hyvin samankaltainen checkCredit-metodin kanssa. Tämäkin metodi ottaa talteen oman historiaobjektin, AML_Result__c, jolle tallennetaan mahdolliset osumat.

Kuten aiemmin jo mainittiin, tallennetaan kaikki kutsussa muokatut tai luodut objektit. Tallennettavista objekteista Application-objektin Apex-laukaisimeen on sijoitettu itse päätöksenteko. Päätös siitä myönnetäänkö luottoa vai ei, on sijoitettu metodiin, joka käy läpi kaikki halutut parametrit, joiden perusteella automaattinen päätös halutaan tehdä. Päätöksen tekoon liittyvät parametrit ja tekijät ovat hyvin yksilöllisiä ja riippuvat siitä, kuinka yritys haluaa tehdä päätöksen luoton myöntämisestä. Tämän vuoksi tässä ei perehdytä itse päätöksen syntymiseen sen tarkemmin, sillä työn tarkoituksena on perehtyä vain yleisesti siihen, kuinka Salesforce soveltuu tähän tarkoitukseen. Metodin lopuksi asetetaan syntynyt päätös Application__c-objektille.

6 Yhteenveto

Työ alkoi tutustumisella Salesforce-alustaan ja sen erilaisiin palveluihin. Käytiin läpi mitä ja mikä Salesforce on, sekä tarkastelin, mikä työssä käytettävä Sales Cloud on. Salesforce-alustan läpikäynnissä todettiin sen olevan nykyajan liiketoimintaan hyvin sopiva työkalu, jolle on helppo rakentaa yritystoimintalogiikkaa ilman pelkoa niin sanotun legacy-ohjelmiston syntymisestä.

Tämän jälkeen työssä perehdyttiin syvemmin Salesforcen käyttämiin ohjelmointi- ja tietokantakyselykieliin, sekä perehdyin syvemmin verkkopalveluihin. Verkkopalveluista tarkasteltiin erityisesti REST-rajapintoja ja sen käyttöä Salesforcessa.

Tämän insinöörityön tavoitteena oli rakentaa Salesforce-alustalle kaksi verkkopalvelurajapintaa REST-tekniikkaa käyttäen. Toinen näistä rajapinnoista huolehtii finanssikäyttäjän rekisteröinnistä ja toinen huolehtii luottohakemuksen käsittelystä. Päämääräisenä tavoitteena oli ensin perehtyä siihen, kuinka REST-rajapinta toteutetaan Salesforce-alustalle. Tämän jälkeen tarkoituksena oli perehtyä siihen, kuinka REST-rajapintaa saadaan hyödynnettyä rekisteröitymisen ja luotonhaun käsittelyssä. Molemmat rajapinnat ja niihin liittyvät luokat toteutettiin Salesforcen Apex-ohjelmointikieltä sekä SOQL-tietokantakyselykieltä hyödyntäen.

Eniten päänvaivaa tuotti Salesforcen rajoittuvuus silloin, kun on tarve tehdä useita tietokantaoperaatioita sekä verkkopalvelukutsuja saman transaktion aikana. Tämä ongelma ratkaistiin käyttämällä asynkronisia metodeja verkkopalvelukyselyiden tekemiseen. Toinen tapa, jolla tätä rajoitetta kierrettiin, oli käyttää ”wrapper”-luokkia. Nämä luokat pitävät sisällään tarvittavat tiedot sen aikaa, kun verkkopalvelukutsut suoritetaan, jonka jälkeen kaikki objektit voidaan tallentaa yhtä aikaa.

Tätä työtä tehdessä oppi monia uusia asioita siitä, miten Salesforcella voidaan luoda rajapintoja ulkoisille palveluille sekä millaisia rajoitteita Salesforcessa on. Opittiin, että REST-rajapintojen toteutus Salesforcella on hyvin yksinkertaista ja totesin niiden olevan hyvin käyttökelpoisia moniin eri tarkoituksiin. Opittiin myös REST-rajapintojen olevan paljon SOAP-rajapintoja helpompikäyttöisiä, mikä muun muassa nopeuttaa uusien asioiden kehitystä alustalle.

Jatkokehityksenä työhön voisi lisätä uuden rajapinnan, jonka kautta olisi mahdollista tehdä muutoksia jo tehtyyn hakemukseen. Tämä mahdollistaisi esimerkiksi luoton määrän muutoksen tai muiden hakemuksessa annettujen tietojen muutokset.

Loppupäätelmänä voidaan todeta Salesforcen soveltuvan skaalautuvuutensa ja ketteryytensä ansioista loistavasti finanssialan tarpeisiin. Omista rajoitteistaan huolimatta se selviytyy vaikeimmistakin tehtävistä vaivatta ja tarjoaa loppukäyttäjälle hyvän ja saumattoman käyttökokemuksen.

Lähteet

1 Luottotietolaki. Verkkodokumentti. http://www.finlex.fi/fi/laki/ajantasa/2007/20070527. Luettu 11.10.2017.

2 What is Salesforce. Verkkodokumentti. https://www.salesforce.com/products/what-is-salesforce/. Luettu 25.10.17.

3 Top CRM Software Vendors by Market Share. Verkkodokumentti. http://www.crmsearch.com/crm-software-market-share.php. Luettu 25.10.17.

4 About us. Verkkodokumentti. https://www.salesforce.com/company/about-us/. Luettu 25.10.17.

5 Products. Verkkodokumentti. https://www.salesforce.com/products/. Luettu 25.10.17.

6 Salesforce: Beyond CRM to Cloud Enterprise SaaS. Verkkodokumentti. https://www.accountexnetwork.com/blog/2016/03/salesforce-beyond-crm-to-cloud-enterprise-saas/. Luettu 25.10.17.

7 Salesforce No Software, We Mean No 'Legacy' Software. Verkkodokumentti. https://www.forbes.com/sites/adrianbridgwater/2015/05/21/salesforce-no-software-we-mean-no-legacy-software/#1423e68526ec. Luettu 31.10.17.

8 Introduction to Salesforce: How to Use Salesforce to Manage Sales. Verkkodokumentti. http://blog.nuvemconsulting.com/introduction-to-salesforce-how-to-use-salesforce-to-manage-sales/. 31.10.17.

9 Introduction to Apex for Programmers. Verkkodokumentti. https://developer.salesforce.com/events/webinars/intro-to-apex-code. Luettu 2.11.17.

10 What is Apex. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_intro_what_is_apex.htm. Luettu 2.11.17.

11 An Introduction to Force.com Apex Code. Verkkodokumentti. https://developer.salesforce.com/page/An_Introduction_to_Apex. Luettu 2.11.17.

12 Future Methods. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_invoking_future_methods.htm. Luettu 7.11.17.

13 Callout Limits and Limitations. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_callouts_timeouts.htm. Luettu 7.11.17.

14 Execution Governors and Limits. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_gov_limits.htm. Luettu 7.11.17.

15 Batch Apex. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_batch.htm. Luettu 7.11.17.

16 Salesforce Object Query Language(SOQL). https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql.htm. Luettu 7.11.17.

17 SOQL SELECT Syntax. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql_select.htm. Luettu 16.11.17.

18 The History of REST APIs. Verkkodokumentti. https://blog.readme.io/the-history-of-rest-apis/. Luettu 16.11.17.

19 Experience and Evaluation. Verkkodokumentti. http://www.ics.uci.edu/~fielding/pubs/dissertation/evaluation.htm. Luettu 16.11.17.

20 Representational State Transfer (REST). Verkkodokumentti. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Luettu 16.11.17.

21 Figure 5-3. Verkkodokumentti. https://www.ics.uci.edu/~fielding/pubs/dissertation/stateless_cs.gif. Luettu 16.11.17.

22 REST and HTTP Methods. Verkkodokumentti. https://spring.io/understanding/REST. Luettu 19.11.17.

23 Introducing JSON. Verkkodokumentti. http://www.json.org/. Luettu 19.11.17.

24 REST API. Verkkodokumentti. https://developer.salesforce.com/page/REST_API. Luettu 19.11.17.

25 Salesforce REST. Verkkodokumentti. https://s3.amazonaws.com/dfc-wiki/en/images/d/d9/REST-API-01.png. Luettu 19.11.17.

26 Introduction to Apex REST. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_rest_intro.htm. Luettu. 19.11.17.

27 Apex REST Methods. Verkkodokumentti. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_rest_methods.htm. Luettu 19.11.17.

Rekisteröitymis REST-rajapinta

```
@RestResource(urlMapping='/Register/v1/*')
global class RegistrationRestAPI{
    public class Payload{
        public String ssn         {get;set;}
        public String mobilePhone {get;set;}
        public String email       {get;set;}
        public String lastName    {get;set;}
        public String firstName   {get;set;}
        public String language    {get;set;}
    }
    public class Response{
        public String  message        {get;set;}
        public Boolean created        {get;set;}
        public String  ssn            {get;set;}
        public String  mobilePhone    {get;set;}
        public String  email          {get;set;}
        public String  country        {get;set;}
        public String  lastName       {get;set;}
        public String  firstName      {get;set;}
        public String  language       {get;set;}
    }

    @HttpPost
    global static Response doPost(){
        //REST muuttujien määrittely
        RestRequest req = RestContext.request;
        RestResponse res = RestContxt.response;

        //Paluuviestin sisältötyypin määritys
        res.addHeader('Content-Type', 'application/json');

        //Sisäluokkien sekä Account sObjektin initialisointi
        Payload payload = new Payload();
        Response response = new Response();
        Account account = new Account();

        try{
            try{
                JSONParser parser =
                            JSON.createParser(
                                req.requestBody.toString()
                            );
               payload = (RegisterAccountPayload)parser.
                     readValueAs(RegisterAccountPayload.class);
            }catch(JSONException jex){
                throw new
                    RegistrationException('INVALID_PAYLOAD');
            }
            //Tarkasta, että tarvittavat kentät on täytetty
            mandatoryFilled(payload);
```

```
            List<Account> oldAccounts = new List<Account>();
            oldAccounts = [SELECT
                                Id,
                                Lastname,
                                Firstname,
                                Phone,
                                Language__c,
                                Email
                           FROM Account
                           WHERE SSN__c =: payload.ssn]
            if(oldAccounts.isEmpty){
                //Lisää uudelle accountille
                //payloadissa olevat arvot
                account.LastName = payload.lastName;
                account.FirstName = payload.firstName;
                account.Phone = payload.mobilePhone;
                account.Email = payload.Email;
                account.Language__c = payload.language;
            }else{
                //Sosiaaliturvatunnus on uniikki joten haku
                //ei koskaan voi palauttaa enempää kuin
                //yhden tuloksen
                account = oldAccounts.get(0);
                //Päivitä vanhaa accounttia uusilla tiedoilla
                account.LastName = payload.lastName;
                account.FirstName = payload.firstName;
                account.Phone = payload.mobilePhone;
                account.Email = payload.Email;
                account.Language__c = payload.language;
            }
            upsert account;
        }catch(RegistrationException rex){
            //Kun itse määritetty virhe heitetään tarkoittaa se
            //http 400 virhettä
            res.statusCode = 400;
            response.message = rex.getMessage();
        }catch(Exception e){
            //Kun yleinen virhe on se http 500 virhe
            res.statusCode = 500;
            response.message = e.getMessage();
        }
    }

    private static void mandatoryFilled(Payload payload){
        if(String.isEmpty(payload.firstName)){
            throw new
            RegisterAccountException('MISSING FIRST NAME');
        }
        if(String.isEmpty(payload.ssn)){
            throw new RegisterAccountException('MISSING SSN');
        }
        if(String.isEmpty(payload.country)){
            throw new
```

```
                RegisterAccountException('MISSING COUNTRY');
            }
            if(String.isEmpty(payload.email)){
                throw new RegisterAccountException('MISSING EMAIL');
            }
            if(String.isEmpty(payload.mobilePhone)){
                throw new
                 RegisterAccountException('MISSING PHONE NUMBER');
            }
            if(String.isEmpty(payload.language)){
                throw new
                 RegisterAccountException('MISSING LANGUAGE');
            }
        }

        //Oman virheen määrittely
        global class RegistrationException extends Exception{}
    }
```