Verohallinnon rajapinnan käyttöönotto eläkevakuutusohjelmistossa

Jaakko Aakula

Haaga-Helia ammattikorkeakoulu
Tradenomi
Opinnäytetyö
2023

Tekijä(t)

Jaakko Aakula

Tutkinto

Tradenomi

Raportin/Opinnäytetyön nimi

Verohallinnon rajapinnan käyttöönotto eläkevakuutusohjelmistossa

Sivu- ja liitesivumäärä

29

Tämä toiminnallinen opinnäytetyö käsittelee Verohallinnon rajapinnan kutsujen käyttöönoton suunnittelua ja implementaatiota. Työ tehtiin vakuutusjärjestelmään, ja siinä käsiteltiin eläkevakuutusasiakkaiden verotietoja. Työn tuotos tehtiin Java-ohjelmointikielellä. Suunnitteluosuuden ratkaisu ja työn tuotos oli onnistuttava pitämään vakuutusyhtiön asiakkaiden verotiedot mahdollisimman ajan tasalla. Työn tuotoksen tavoite oli todistaa suunnitteluosuuden ratkaisun toimivaksi, ja sitä piti pystyä esittelemään sisäisesti yrityksessä. Työn tuote oli pilottiversio rajapintojen käyttöönotosta. Opinnäytetyön tavoitteena oli saada onnistunut työn tuotos ja pysyä aikataulussa.

Projektin tietoperusta käsittelee termejä, Verohallinnon rajapinnan kutsuja, eläkeverotusta, varmenteita sekä ohjelmistoratkaisun teknologioita.

Projektin empiirisessä osassa käydään läpi suunnittelua, varmenteen hakemista ja testausta, sekä kutsujen käyttöönottoa.

Opinnäytetyön ja työn tuotoksen tavoitteet saavutettiin. Työn tuotos todisti suunnittelun ratkaisun toimivaksi, sekä siitä saatiin esiteltävä versio. Työ valmistui myös aikataulussa.

Asiasanat

Kutsu, testivarmenne, rajapinta, verokortti, verohallinto

Sisällys

1 Johdanto

Opinnäytetyön aiheena on eläkeverotietojen hakemisen ja ilmoittamisen suunnittelu, sekä implementointi Vero API -rajapinnan avulla Evitec Life -järjestelmään. Evitec Life on vakuutusjärjestelmä, jossa ylläpidetään esimerkiksi eläkevakuutuksia. Työn tuotoksen pitää pystyä pitämään eläkevakuutuksen asiakkaiden verotiedot mahdollisimman ajan tasalla käyttäen Vero API -rajapinnan kutsuja.

Työn tuotos on implementoinnin pilottiversio. Pilottiversiota tarvitaan suunnittelun tuloksen todistamiseen, sekä esittämiseen. Pilottiversio keskittyy uuden eläkkeen ilmoittamiseen ja verotietojen hakemiseen. Jos pilottiversio onnistuu todistamaan suunnittelussa tehdyn ratkaisun, suunnitteluosassa käytettyä ratkaisumallia tullaan käyttämään tulevassa projektissa. Silloin pilottiversiota tullaan myös käyttämään yrityksessä ratkaisun esittämiseen. Opinnäytetyön tavoitteena on saada työ valmiiksi ajoissa, sekä saada työn tuotos tehtyä mahdollisimman onnistuneesti. Työn onnistunut valmistumisajankohta on 16.5.2023.

Eläkeverokortin luonnissa ja haussa on kolme roolia. Työ koskee eläkevakuutusohjelmistoa, joten rooleja on vakuutusyhtiö, vakuutusyhtiön asiakas, sekä verohallinto (Verohallinto 2023c). Vakuutusyhtiö on taas työn toimeksiantajan Evitec Oy:n asiakas. Eläkevakuutus on vapaaehtoinen säästämistuote, joka ostetaan vakuutusyhtiöltä (Nordea 2023). Verokortin luonti ja hakeminen toimii seuraavanlaisesti (Kuva 1.).

1. Vakuutusyhtiön asiakas, eli eläkeläinen tai tuleva eläkeläinen haluaa aloittaa eläkkeensä. Hän kertoo asiasta vakuutusyhtiölle.

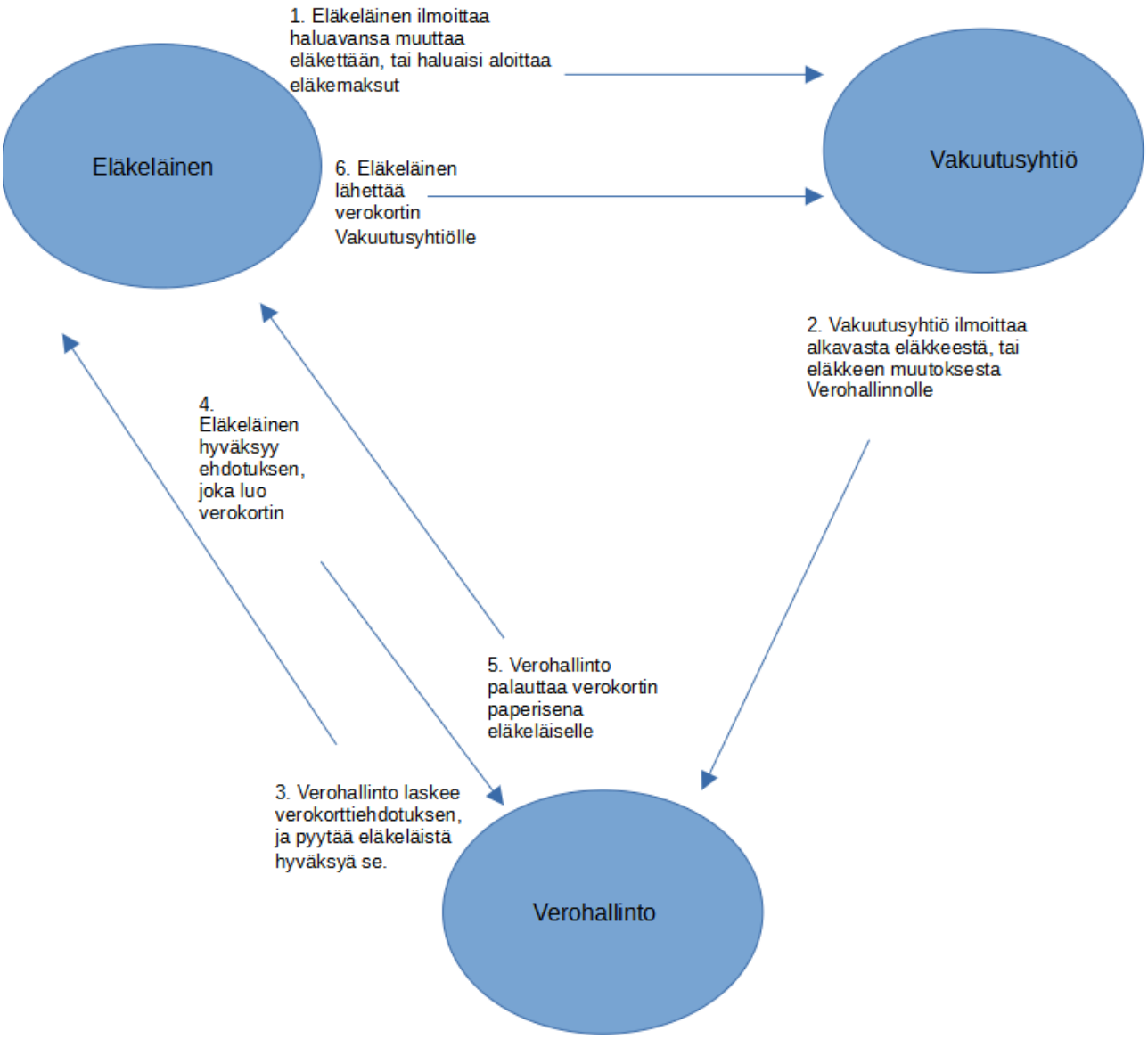
2. Vakuutusyhtiö ilmoittaa eläkkeestä ja sen määrästä Verohallinnolle.

3. Verohallinto luo verokorttiehdotuksen, sekä pyytää vakuutusyhtiön asiakasta eli eläkeläistä luomaan verokortin.

4. Eläkeläinen käy luomassa verokortin.

5. Verohallinto palauttaa verokortin paperisena eläkeläiselle.

6. Eläkeläinen palauttaa verokortin paperisena vakuutusyhtiölle.



Kuva 1: Eläkeverokortin luonti ja haku ennen Vero API:n kutsujen käyttöönottoa

Työ tulee digitalisoimaan verokortin luonnin ja haun vaiheet 2 sekä 6. Työn seurauksena eläkeläisen ei tarvitse toimittaa korttia vakuutusyhtiölle, vaan vakuutusyhtiön käyttämä Evitec Life -tuote hakee verokortin Verohallinnolta ilman eläkeläistä. Vaiheet tullaan digitalisoimaan Verohallinnon Vero API -rajapintaa käyttäen.

Työn suurimmat osat ovat API:n käytön suunnittelu tuotteessa, varmenteen haku ja testaus, sekä hakujen luonti tuotteeseen. Implementointiosa on tehty Java-kielellä.

Aihe on erittäin kiinnostava etenkin digitalisaation kannalta, sillä verokortit ovat muuttuneet digitaalisiksi OmaVeron ja Vero API:n myötä (Verohallinto 2020). Tämä ratkaisu tulee helpottamaan vakuutusyhtiöiden toimintaa merkittävästi. Työn toimeksiantaja on Evitec Oy.

2 Tietoperusta

Käyn työn tietoperustassa läpi ensimmäiseksi käsitteistöä. Tämän jälkeen kerron kolmesta eri API-kutsusta, joita tutkin työn suunnitteluosuudessa. Tietoperustassa kerron myös oleellisimpia osia eläkeverotuksesta. Etenkin siitä, kuinka verokortin hakeminen ja luonti onnistuu alkavassa eläkkeessä. Tietoperustaan kuuluu myös osa missä kerron asiakasvarmenteista.

2.1 Käsitteistö

Rajapinta termiä käytetään API:n synonyyminä raportissa. ”API eli ohjelmointirajapinta on joukko määriteltyjä sääntöjä, jotka mahdollistavat eri sovellusten välisen kommunikaation.” (IBM 2023) Työssä käytetään REST-rajapintaa.
REST-rajapinta on yleensä HTTP-protokollaa käyttävä rajapinta, mihin voi tehdä pyyntöjä tiedon muokkaamisesta, luomisesta, poistamisesta ja lukemisesta (Free Code Camp 2022).

HTTP-protokolla eli Hypertext transfer protocol on tapa, jolla dataa siirretään World Wide Webissä (JavaTpoint 2023).

Kutsu on raportissa sama asia kuin REST-pyyntö, joka on tapa hakea ja tuoda tietoa (IBM 2023).

Eläkevakuutus on vakuutustyyppi, jossa säästetään rahaa tulevaisuutta varten (Nordea 2023).

Korvauksella tarkoitetaan raportissani tilaa, jossa vakuutuksen asiakas täyttää kriteerit maksua varten ja sopimuksessa päätetään aloittaa asiakkaalle maksaminen.

Verokortti on verotietoja sisältävä asiapaperi, josta näkee veroprosentin (Verohallinto 2019).

Vero API on verohallinnon ylläpitämä palvelu, josta pystyy hakemaan ja ilmoittamaan verotietoja rajapinnan kautta (Verohallinto 2022).

Testivarmenne on käyttämäni asiakasvarmenne tyyppi, jota käytän Vero API:n kanssa. Vero API:n asiakasvarmenteet ovat Base64 muodossa (Verohallinto 2022).

”SSL (Secure Sockets Layer) ja sen seuraaja, TLS (Transport Layer Security), ovat protokollia todennettujen ja salattujen linkkien muodostamiseksi verkkoon kytkettyjen tietokoneiden välille. ” (SSL 2021)

Client Certificate Authentication on sertifikaatti, jota käytetään SSL-kättelyn aikana (Technopedia 2014). Vero API käyttää asiakasvarmenteena Client Authentication certificate -sertifikaattia (Verohallinto 2022).

PKCS12 on tiedostomuoto, jossa asiakasvarmenne pidetään työssä.

Body eli *runko* on osa HTTP-protokollan kutsua. Sitä ei käytetä esimerkiksi GET-kutsussa. Jokainen työssä käytetty kutsu on POST-kutsu, joten runko osuus on työn jokaisessa pyynnössä (JavaTpoint 2023).

Header eli *otsikkotiedot* on osa HTTP-protkollan kutsua. Otsikkotiedot ovat jokaisessa pyynnössä (JavaTpoint 2023).
Apache Maven on työssä käytetty rakentamisen hallintatyökalu (Apache 2023).

Docker on työkalu, mitä käytetään ohjelmiston kontittamiseen (Docker 2023).

Jenkins on työssä käytetty automaatiotyökalu (Jenkins 2023).
DAO eli Data Access Object on Java olio-ohjelmoinnissa käytetty suunnittelumalli, jota käytetään tiedonhallitsemissa (Baeldung 2022c).

Muuttuneet eläke ja etuustietojen ennakonpidätystiedot maksajille -kutsu on yksi Vero API:n rajapinnassa toimiva haku. Hakua käytetään yhtiön kaikkien asiakkaiden muuttuneiden verotietojen hakemiseen, joten käytän tekstissä tästä hausta nimitystä *massakutsu*. (Verohallinto 2023a)

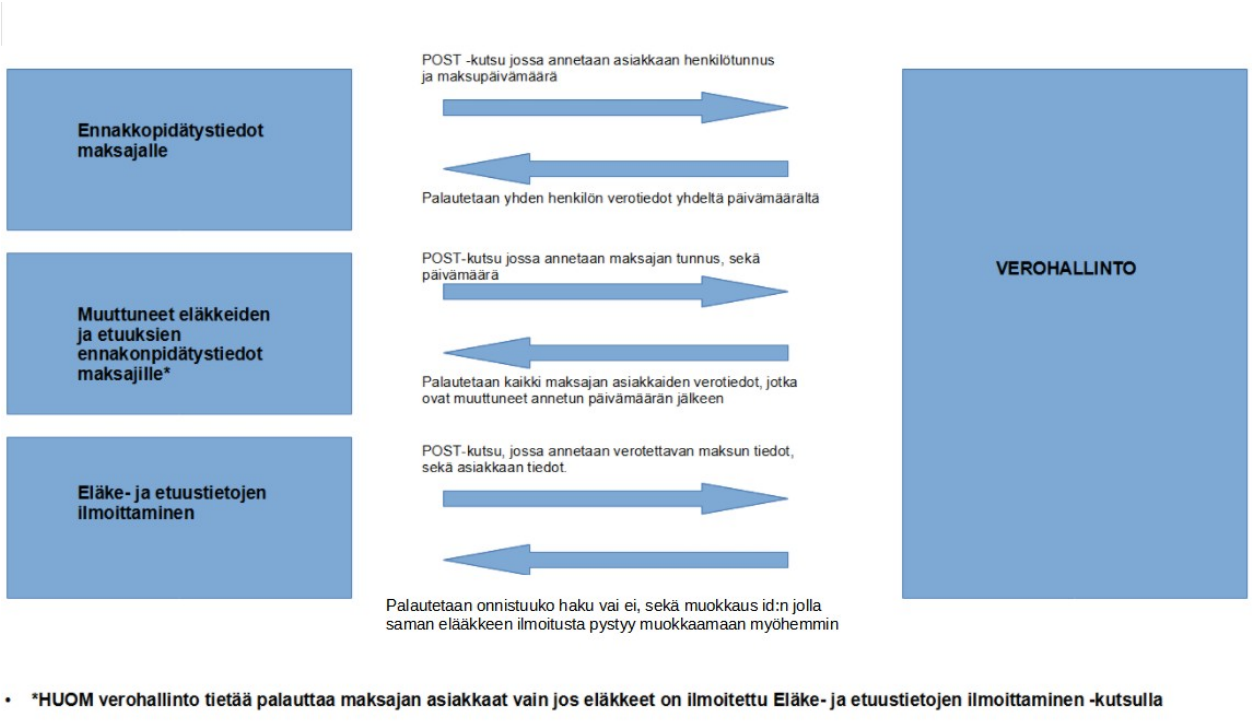
Eläke- ja etuustietojen ilmoittaminen -kutsulla ilmoitetaan eläkkeen tietoja (Verohallinto 2023a). Tästä kutsusta käytän tekstissä nimitystä *ilmoituskutsu*.

Ennakkopidätystiedot maksajille -kutsu on tarkoitettu yksittäisen asiakkaan verotietojen hakemiseen (Verohallinto 2023a). Siksi käytän tästä hausta tekstissä nimitystä *yksittäiskutsu*.

2.2 Vero API haut

Vero API rajapinnassa henkilökohtaisten verotietojen hakemiseen ja ilmoittamiseen liittyy kolme erilaista kutsua. Nämä kutsut ovat: *Eläke- ja etuustietojen ilmoittaminen*, *Muuttuneet eläkkeiden ja etuuksien ennakonpidätystiedot maksajille*, ja *Ennakonpidätysprosentin kysely*. (Kuva 2)

Käyn nämä kolme hakua yksitellen läpi. Näiden hakujen käytön suunnittelusta kerrotaan empiirisen osan suunnitteluvaiheessa. Kaikki listatut kutsut ovat POST-kutsuja. Jokaisessa kutsussa on samanlainen header eli otsikkotiedot. Kutsujen otsikkotiedoissa on kaksi arvoa: Vero-SoftwareId ja Vero-SoftwareKey. Vero-SoftwareId on hakua käyttävän organisaation itse keksimä, joka Verohallinnon dokumentaation mukaan yleisesti muodostuu yritystunnuksesta, alaviivasta ja kahdesta vapaavalintaisesta numerosta. Vero-SoftwareKey on taas Verohallinnon antama, joka on ohjelmistokohtainen. Kyseinen avain annetaan ohjelmiston rekisteröinnin yhteydessä. Rajapinnan käyttö onnistuu vain asiakasvarmenteen kanssa. (Verohallinto 2023a) Käyn läpi asiakasvarmenteen toimintaa myöhemmin. Hakujen käytöstä aion kertoa lisää suunnitteluosassa.



Kuva 2: Työssä ja työn suunnittelussa käytetyt Vero API:n kutsut

2.2.1 Ennakonpidätystiedot maksajalle

Ennakonpidätysprosentin kysely eli yksittäiskutsu on haku, jota käytetään erittäin monessa käyttötarkoituksessa. Haku palauttaa kysytyn henkilön verotiedot tietyltä päivämäärältä. Tälle voi siksi keksiä erittäin monta käyttötarkoitusta työnantajasta vakuutus- ja pankkisektorille. Haun runkoon kirjoitetaan maksutapahtuman päivämäärä, maksajan yritystunnus, sen asiakkaan henkilötunnus jonka verotietoja haetaan, ja lista siitä minkälaisen tulotyypin verotiedot halutaan.(Verohallinto 2023a) Hakua käytetään siis muissakin käyttötarkoituksissa kuin eläkeasioissa. Hakua ei lopulta käytetty työssä, koska oli helpompaa hakea verotiedot kerran massakutsulla, kuin kysyä jokaisen asiakkaan verotiedot yksitellen.

2.2.2 Muuttuneet eläkkeiden ja etuuksien ennakonpidätystiedot maksajille

Muuttuneet eläkkeiden ja etuuksien ennakonpidätystiedot maksajille -kysely eli massakutsu on taas täysin tarkoitettu etuuksien ja eläkkeiden verotietojen hakuun. Haku eroaa *Ennakonpidätysprosentin maksajille* -kyselystä myös sillä tavalla, että haku on maksajakohtainen. Tämä tarkoittaa sitä, että haussa ei kysytä tietyn asiakkaan henkilötunnuksella verotietoja, vaan kutsun runkoon kirjoitetaan vain yrityksen tunnus. Kutsun runkoon kirjoitetaan myös päivämäärä, minkä jälkeiset muutokset halutaan tietää, ja maksimi palautettava rivimäärä. Verohallinto tietää kyseisen yrityksen asiakkaat, ja antaa näiden asiakkaiden verotiedot, jos ne ovat muuttuneet annetun päivämäärän jälkeen. Verohallinto tietää ketkä ovat yrityksen asiakkaita, jos yritys käyttää kolmatta kutsua nimeltä Eläke- ja etuustietojen ilmoittaminen. (Verohallinto 2023a) Kyseistä hakua lopulta käytettiin työssä verotietojen hakemiseen.

2.2.3 Eläke- ja etuustietojen ilmoittaminen

Eläke- ja etuustietojen ilmoittaminen -kutsulla ilmoitetaan Verohallinnolle alkavasta eläkkeestä. Kutsun runkoon annetaan seuraavanlaista tietoa: onko ilmoitus uusi, onko verotieto uusi, mikä maksuvuosi on kyseessä, maksajan avain, tietoa maksun vastaanottajasta ja tietoa maksusta. Maksun tiedoista pakollista on ilmoittaa määrä, maksun tyyppi, maksun alkupäivämäärä, päätöspäivämäärä, sekä maksun nimi ruotsiksi ja suomeksi. (Verohallinto 2023a) Hakua voidaan käyttää työssä alkavan eläkkeen ilmoittamisessa, ja sitä on pakko käyttää jos toteutuksessa tullaan käyttämään *Muuttuneet eläkkeiden ja etuuksien ennakonpidätystiedot maksajille* -kutsua. Käytin tätä kutsua työn implementointiosuudessa alkavien eläkkeiden ilmoittamiseen.

2.3 Eläkevakuutus

2.3.1 Eläkkeen veroluokat

Eläkkeen verotus on yleisimmin ansiotuloa. Kuitenkin eläke voi tietyissä tilanteissa olla myös pääomatuloa. Eläke on pääomatuloa yksilöllisissä eläkevakuutuksissa, jotka on aloitettu 5.4.2004 jälkeen. Jos eläkevakuutus on aloitettu tuota päivämäärää aikaisemmin ja arvoa on kertynyt vielä 1.1.2006 jälkeen, vuotta 2006 ennen kertynyttä eläkettä verotetaan pääomatulona ja muuta kertynyttä eläkettä verotetaan ansiotulona. Tämä tuottaa tilanteen, missä eläkettä verotetaan eri tavalla, riippuen siitä milloin se on aloitettu ja milloin siihen on kertynyt eläkettä. (Verohallinto 2023b) Tällaisessa tilanteessa on tehtävä kaksi ilmoituskutsua samalle eläkemaksulle, missä jakava tekijä on veroluokka (Verohallinto 2023a).

2.3.2 Peruslaskenta

Jos eläke kestää monta vuotta, on seuraavan vuoden eläkkeen määrä ilmoitettava vuoden lopulla. Tämä ei ole asiakkaan, vaan vakuutusyhtiön vastuulla. Tämä on siis ainoa tilanne, missä asiakkaan ei tarvitse käydä itse luomassa tai muokkaamassa verokorttia. Seuraavan vuoden ensimmäistä verokorttia sanotaan perusverokortiksi. Muutosverokortti on mikä vaan verokortti joka on luotu samana vuonna perusverokortin jälkeen. On myös tärkeää huomata, että vaikka verokortin luomisessa vakuutusyhtiö tekee ilmoituksen eläkkeen alkamisesta, on verokortin luonti silti asiakkaan vastuulla. Peruslaskentaan ilmoittaminen onnistuu *Eläke- ja etuustietojen ilmoittaminen* -kutsulla eli ilmoituskutsulla (Verohallinto 2023a).

2.4 Asiakasvarmenne

Vero API käyttää haun turvaamisessa asiakasvarmennetta. Työssä käytetään testivarmennetta, jonka hakemisesta ja testaamisesta kerron myös työn empiirisessä osassa. Tässä osassa kerron kuitenkin myös siitä, kuinka tulorekisterin rajapinta toimii testivarmenteen hakemisessa. Vero API:n autentikaatiotapa on Client Certificate Authentication. (Verohallinto 2022) Tämä tarkoittaa, että yhteyden muodostuessa tehdään SSL-protokollan mukainen kättely, jossa tarkastetaan asiakasvarmenne (Jscapem 2022) .

2.4.1 SSL-varmenne ja kättely

SSL-varmenne turvaa HTTPS-protokollan kutsuja. On olemassa palvelin- ja asiakasvarmenteita. Koska asiakkaan pitää varmistaa henkilöllisyys palvelimelle, työssä käytetään Client Authentication Certificate -sertifikaattia eli asiakasvarmennetta. Asiakasvarmenne on tiedosto, joka suojataan usein salasanalla. Varmenne on usein PKCS12 muodossa, niin kuin myös tässä työssä. Tässä työssä varmenne on kyseisessä muodossa oleva PFX-tiedosto. (Jscape 2022)

SSL-kättely on prosessi, joka tapahtuu ennen datan välittämistä asiakkaan ja palvelimen välillä. Varmenteiden tarkistaminen ei ole pakollinen osa kättelyä, joten kättely voi tapahtua myös ilman sitä. Kaikissa Vero API:n hauissa kuitenkin tarvitaan asiakasvarmennetta. Kättelyn tarkoituksena on se, että asiakas ja palvelin tunnistautuvat toisilleen. (Jscapem 2022)

2.4.2 Tulorekisterin rajapinta ja varmenteen haku

Testivarmenne pitää hakea Tulorekisterin rajapinnasta testauksen aloituksen ilmoittamisen jälkeen. Testivarmenteen hakemisessa pitää käyttää Tulorekisterin WebService-rajapintaa (Verohallinto 2022). Rajapintaa käytetään työssä vain varmenteen luomiseen ja hakemiseen. Tässä rajapinnassa on kolme erilaista kutsua: SignNewCertificate, RenewCertificate ja GetCertificate. SSL-testivarmenteen hakemisessa pitää käyttää hakuja SignNewCertificate ja GetCertificate. Kutsu tehdään XML-muodossa, ja siihen pitää tehdä XML Signature -allekirjoitus, jolla taataan viestin muuttumattomuus. (Tulorekisteri 2019)

SignNewCertificate on haku, joka tehdään ensimmäiseksi. Sillä pyydetään Tulorekisteriä luomaan varmenne. Pyyntö on XML-muodossa, johon lisätään verohallinnon antamat tiedot, sekä varmennepyyntö, joka on PKCS#10 muotoinen Base64-teksti. Haku palauttaa noutotunnuksen. Varmennepyynnön voi luoda esimerkiksi OpenSSL-ohjelmalla. (Tulorekisteri 2019)

GetCertificateRequest on haku, jolla haetaan lopullinen varmenne. Haussa käytetään SignNew-Certificate-haun palauttamaa noutotunnusta. Haku palauttaa asiakasvarmenteen. Asiakasvarmenne pitää lähettää sähköpostilla verohallinnolle. (Tulorekisteri 2019)

2.5 Ohjelmistoratkaisun teknologiat

2.5.1 Spring boot, Maven ja Jenkins

Tuotteen ohjelmistokehyksenä toimii Spring. Spring on Java-pohjainen ohjelmistokehys. (Spring 2023) Työssä käytetään päivittäisen eräajon tekemiseen Jenkins-automaatiotyökalua. Esimerkiksi työssä käytetty massakutsu ajetaan kyseisellä automaatiotyökalulla. Jenkins-automaatiotyökalulla voi myös automatisoida tuotteen rakennusta ja testausta. (Jenkins 2023)

Tuotteen kehitysympäristö pyörii Docker-kontissa, jossa tuotetta voi muokata ja testata. Docker on ohjelmistokehityksen kontittamiseen tarkoitettu työkalu. Kontittaminen tarkoittaa ohjelmiston pakkamista omaan ympäristöönsä tietokoneessa. Se tuottaa täysin puhtaan pöydän, jossa työskennellä. Sama kontitettu ohjelmisto pyörii samalla tavalla missä vaan ympäristössä. (Docker 2023)

Apache Maven on työssä käytetty rakentamisen hallintatyökalu (Apache 2023). Maven:ia käytetään työssä projektin tiedostojen rakentamiseen JAR-tiedostoksi. Rakennettu tiedosto siirretään päällä olevaan Docker-konttiin, jossa uusi tiedosto korvaa vanhan.

Ohjelmistomuutokset toimivat työssä siis seuraavanlaisesti. Ajetaan Maven:in rakentamiskomento komentokehotteessa. Näin ohjelmistoprojektista tehdään JAR-tiedosto. Seuraavaksi ajetaan komento, jolla siirretään tiedosto Docker-konttiin. Tämän jälkeen pakotetaan kontti uudelleenkäynnistymään. Kun kontti on käynnistynyt ja tuote käynnistyy uudestaan, muutokset näkyvät tuotteessa.

2.5.2 Data Access Object

Data Access Object eli DAO on Java olio-ohjelmoinnissa käytetty suunnittelumalli, missä erotetaan toimintalogiikka tiedonhallitsemislogiikasta. DAO eli tiedon hallinta olio on yleensä luokka, jonka metodeilla pystyy kommunikoimaan tietokannan kanssa. (Baeldung 2022c) Työn DAO-luokissa metodit sisältävät SQL-kielen injektioita Java-ohjelmistokieleen, koska tuotteen tietokanta on SQL-relaatiotietokanta.

2.5.3 JSON

JSON on formaatti, jota käytetään tiedon säilyttämiseen ja lähettämiseen (W3schools 2023). Vero API palauttaa tekstin JSON-muodossa, joten tieto pitää osata parsia ja muuttaa Java-olioiksi. Työssä se tehdään JSONObject-luokan avulla. Työssä kutsujen runko-osuus on myös JSON-muo-

dossa. JSON-muotoisen tiedon luonti onnistuu Java-kielellä esimerkiksi luomalla String-muuttujan, joka kirjoitetaan JSON-formaatin mukaisesti.

2.5.4 HTTPS-kutsu ja varmenteen käyttö Java-kielellä

Jotta SSL-varmennetta pystyy käyttämään kutsun luonnissa, on meidän käytettävä HTTPS-protokollaa. HTTPS-protokollaa käytetään Java-ohjelmistossa esimerkiksi HttpsURLConnection-luokan avulla.

HttpsURLConnection-luokka perii HttpURLConnection-luokan, joten ne toimivat yhteyden turvaamiseen lisättyjä metodeita lukuun ottamatta täysin samalla tavalla (Baeldung 2022a).

HttpsURLConnection-luokka tarvitsee SSLContext-luokan. Tämän luokan kanssa käytetään Keystore API:a, joka on Javan oma rajapinta. Keystore rajapintaa käytetään sertifikaattien tallentamiseen. (Baeldung 2022b) Varmenne on tässä työssä PKCS12 muodossa. SSLSocketFactory-luokkaa käytetään työssä yhteyden luomiseen.

Varmenteiden käsittelyn jälkeen HTTPS-kutsu tehdään täysin samalla tavalla kuin HttpURLConnection-luokan kanssa. Kyseinen luokka luodaan URL-luokan avulla, jolle annetaan kutsun URL-osoite. URL-luokka aloittaa yhteyden ja luo HttpURLConnection-luokan openConnection-metodilla. Luokalle annetaan kutsun tyyppi, joka voi olla esimerkiksi GET tai POST. Luokalle annetaan otsikkotiedot setRequestProperty-metodilla. Luokalle annetaan myös parametrejä, ja sille voi antaa myös rungon. Työn molemmissa kutsuissa annetaan kutsulle myös runko. Runko on JSON-muodossa, ja se luodaan Java-kielessä String-tyyppiseksi muuttujaksi. Kutsut luetaan StringBuffer-luokalla, ja lopuksi yhteys lopetetaan metodilla disconnect. (Baeldung 2023)

3 Empiirinen osa

Työn empiirisessä osassa käyn läpi työn suunnitteluprosessia, varmenteen hakua ja testaamista, sekä implementointia.

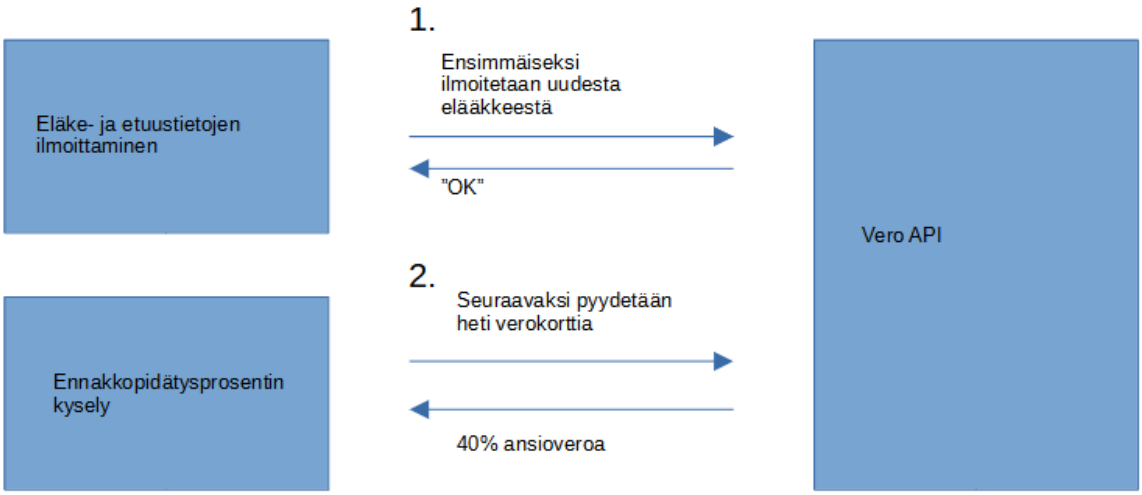
3.1 Hakujen käytön suunnittelu

Työn päämäärä oli saada eläkevakuutuksen korvauksen maksuihin eläkeverokortit automaattisesti käyttäen Verohallinon Vero API -rajapintaa. Tämän aikaansaamiseksi piti ymmärtää mitä hakuja kannattaisi käyttää. Suunnittelu alkoi siis eläkevakuutusten verotuksen ja Vero API:n hakujen tutkimisella. Tärkeää oli ymmärtää, miten eläkeverotietojen haku onnistuu, kuinka se tapahtuu juuri eläkevakuutuksessa ja kuinka järjestelmän verotiedot saadaan pidettyä mahdollisimman ajan tasalla.

Suunnittelussa tuli selväksi, että verotietojen hakemisessa tullaan käyttämään 1-3 erilaista hakua. Nämä kolme hakua on listattu myös opinnäytetyön teoriaosuudessa. Listaan ne kuitenkin kertauksena: *Ennakonpidätysprosentin kysely*, *Muuttuneet eläkkeiden ja etuuksien ennakonpidätystiedot maksajille*, ja *Eläke- ja etuustietojen ilmoittaminen*.

3.1.1 Uuden eläkkeen ilmoittaminen ja hakeminen

Ensimmäinen ratkaisumalli verotietojen hakuun liittyen oli seuraavanlainen. Asiakas ilmoittaa vakuutusyhtiölle haluavansa aloittaa korvauksen ja eläkkeen seuraavana vuonna. Tämän jälkeen yhtiö tekee korvaustapahtuman, missä järjestelmä kertoo verolle alkavasta eläkkeestä *Eläke- ja etuustietojen ilmoittaminen* -kutsulla. Tämän jälkeen järjestelmä hakee heti asiakkaan verotiedot hänen henkilötunnuksellaan *Ennakonpidätysprosentin kysely* -kutsulla. (Kuva 3)



Kuva 3: Kuva ensimmäisestä ja lopulta väärästä ratkaisumallista verotietojen hakemisen toiminnasta

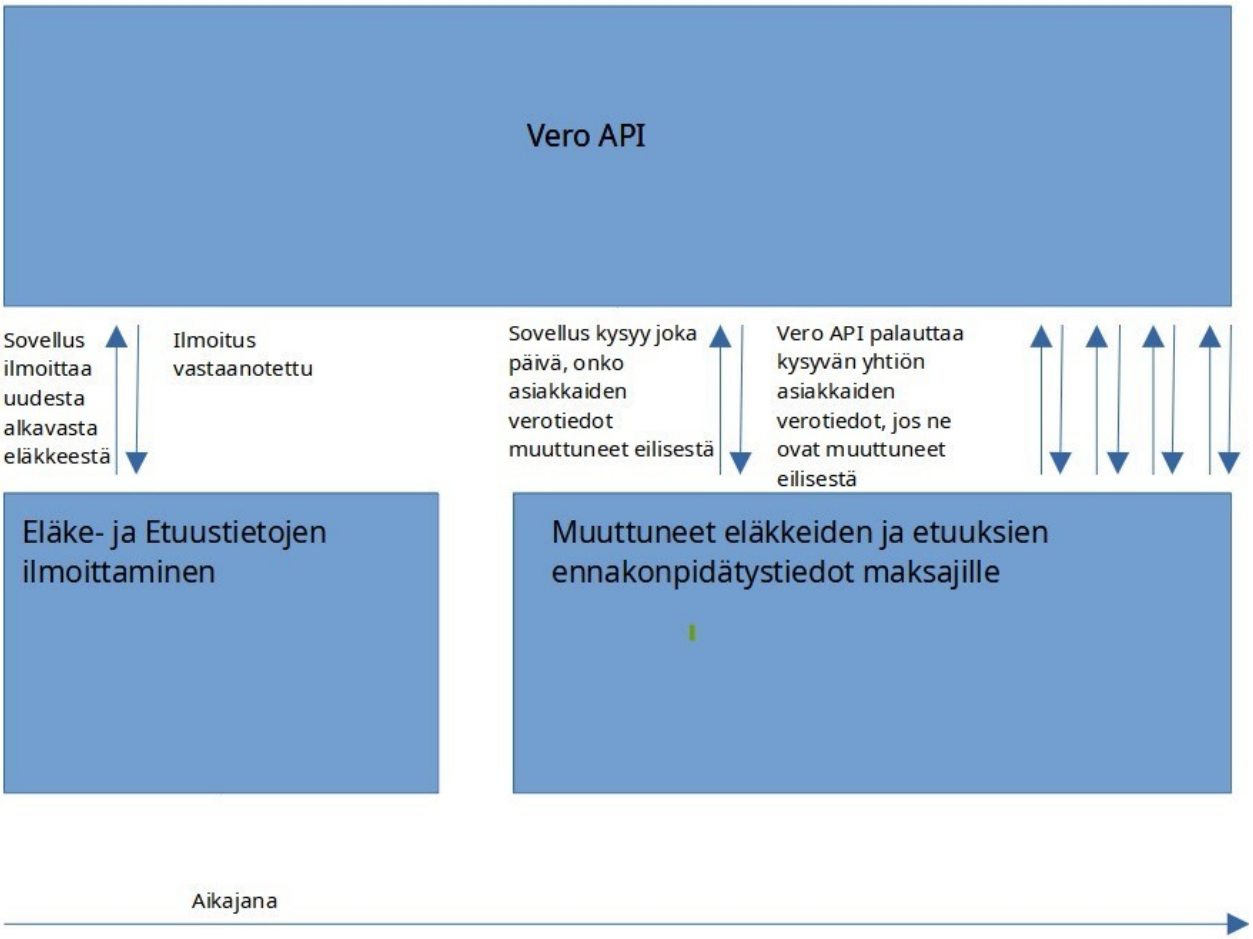
Tällä ratkaisumallilla oli seuraavanlaisia ongelmia. En vielä täysin tiennyt kuinka eläkeverotuksen verokortin luonti onnistuu. Pystyykö vakuutus- tai eläkeyhtiö tilaamaan verokortin ilman asiakasta, vai pitääkö asiakkaan mennä itse hakemaan verokorttia? Tämä oli erittäin relevantti kysymys, koska se kertoi kuinka kauan verokortin laskemisessa kestää. Kyseinen ratkaisumalli on käypä vain silloin, jos verotiedot luodaan suoraan alkavan eläkkeen ilmoittamisen jälkeen. Ongelmia loi myös dokumentaatiossa monta kertaa toistettu peruslaskenta. Johtopäätöksiin oli mahdotonta päästä ennen kuin olin varma mitä peruslaskenta tarkoittaa. *Eläke- ja etuustietojen ilmoittaminen* -kyselyn dokumentaatiossa luki seuraavaa: “Jos tiedot ilmoitetaan tällä rajapinnalla peruslaskentaa varten, ennakonpidätystiedot voidaan hakea vain henkilöverotuksen *Muuttuneet eläkkeiden ja etuuksien ennakonpidätystiedot maksajille* -rajapinnalla.” (Verohallinto 2023a)

Lähetin Verohallinnolle kysymyksiä sähköpostilla. Olennaisimmat kysymykset suunnittelun prosessissa olivat: Mikä on peruslaskenta? Onko verotiedot heti haettavissa eläkkeen ilmoittamisen jälkeen, vai tarvitseeko asiakkaan käydä hyväksymässä hakemus?
Verohallinnolla kesti noin puolitoista viikkoa vastata kysymyksiin. Verohallinto vastasi, että peruslaskenta tarkoittaa seuraavan vuoden verokorttien laskentaa. Eli perusverokortti on vuoden ensimmäinen verokortti, joka lasketaan. Muutosverokortti on taas verokortti, joka tehdään vuoden aikana perusverokortin tilalle. Verohallinto myös kertoi, että asiakkaan on itse haettava verokorttia OmaVerosta, joten verokortti ei ole heti ilmoittamisen jälkeen haettavissa rajapinnasta. Vastaukset olivat erittäin tärkeitä rajapintojen käyttöönoton suunnittelussa.

Vastaukset kertoivat, että edelliset ratkaisumallit kutsujen käytöstä eivät ole tarpeeksi toimivia. Jos verotiedot eivät ole haettavissa heti verotietojen ilmoittamisen jälkeen, niin verotietoja on pidettävä ajankohtaisina hakemalla tietoja tietyin väliajoin.

Viimeinen ratkaisumalli, jota loppujen lopuksi käytin, oli jo mietteissä ennen Verohallinnon vastauksia, mutta nyt vastauksien tultua oli ilmeistä, että tulin tekemään implementoinnin seuraavalla tavalla: Asiakas ilmoittaa vakuutusyhtiölle haluavansa aloittaa eläkkeen korvaukset. Vakuutusyhtiö ilmoittaa Verohallinnolle alkavasta eläkkeestä *Eläke- ja etuustietojen ilmoittaminen* -kutsulla. Vakuutusyhtiö hakee myös päivittäin muuttuneita verotietoja kutsulla *Muuttuneet eläkkeiden ja etuuksien ennakonpidätystiedot maksajille*. (Kuva 4)

Päätin käyttää massakutsua *Ennakonpidätysprosentin kysely* -kutsun eli yksittäiskutsun sijasta, koska se on yhtiökohtainen. Tällöin päivässä tehdään vain yksi haku, missä kysytään, ovatko minkään kyseisen yhtiö asiakkaiden verotiedot muuttuneet eilisen jälkeen. Toisessa haussa olisi pitänyt hakea tiedot asiakaskohtaisesti, jolloin hakuja olisi joutunut tekemään hyvin monta.



Kuva 4. Havainnollistava kuva lopullisesta hakujen käytön suunnitelmasta

3.1.2 Käynnissä olevien eläkkeiden ilmoitukset

Käynnissä olevista eläkkeistä pitää ilmoittaa Verohallinnolle kahdessa eri tapauksessa: Ensimmäinen tilanne on se, että eläkkeessä on joku muutos, jolloin halutaan luoda uusi verokortti. Toinen tapaus on se, että eläke jatkuu seuraavaan vuoteen, ja eläkkeestä on ilmoitettava perusverokortin laskentaan. (Verohallinto 2023a)

Ilmoituskutsu palauttaa uuden eläkkeen alkaessa uniqueIdentifier-arvon. Tätä arvoa käyttämällä on mahdollista tehdä muutoksia vanhaan eläkkeeseen. Perusverokortin laskentaan tehtävän ilmoituksen suunnittelin tapahtuvan vuosittaisessa eräajossa. Muutosverokortin ilmoituksen suunnittelin tapahtuvan automaattisesti, kun korvausta muokataan. Muuten se ei eroaisi ollenkaan alkavan eläkkeen ilmoittamisesta.

3.2 Vero API:n testausympäristöön pääsy

Kun työn suunnittelu oli valmis, vuorossa oli päästä käsiksi rajapinnan testausympäristöön. Tämä alkoi testauksen aloittamisen ilmoittamisella. Ilmoitus piti tehdä paperisesti ja kyseinen asiakirja piti lähettää Verohallinnon sähköpostiin. Onneksi tällä kertaa Verohallinnon kanssa kommunikointi oli erittäin nopeaa. Sain tarvittavat tiedot testivarmenteen hakemiseen jo seuraavana päivänä.

3.2.1 Testivarmenteen hakeminen Tulorekisterin rajapinnasta

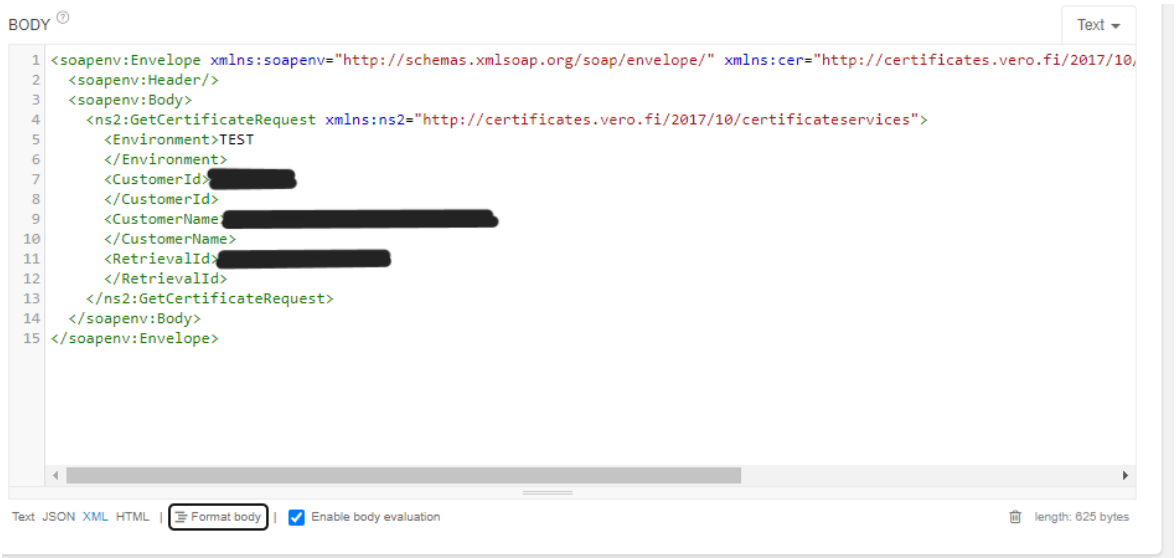
Testivarmenne piti hakea Tulorekisterin rajapinnasta. Tulorekisterin rajapinta toimi XML-kutsuilla, joita en ollut ikinä ennen tehnyt. Lisäksi Verohallinnon palauttamista tiedoista, jotka sisälsi teko yritystunnuksen sekä yritysnimen, piti luoda Client Certificate Request. (Tulorekisteri 2019) Kyseisen pyynnön loin OpenSSL ohjelmalla. XML tiedostoon piti tehdä myös XML-allekirjoitus, josta minulla ei ollut myöskään minkäänlaista aikaisempaa kokemusta. Onneksi työpaikalla oltiin työskennelty muiden julkisten palveluiden rajapintojen kanssa, jolloin työpaikalla oli tehty sovellus, joka luo ja allekirjoittaa haun.

Haut toteutettiin Talent API Tester -selainlaajennuksella. Ensimmäinen haku oli pyyntö luoda testivarmenne. Tämä tehtiin SignNewCertificate-kutsulla (Kuva 5).

Kuva 5. Esimerkki SignNewCertificate-kutsun rungosta

Kuva 6. Esimerkki SignNewCertificate-kutsun vastaussanoman rungosta.

Seuraavaksi oli vuorossa luodun sertifikaatin hakeminen GetCertificate-kutsulla. Kyseisessä haussa luodaan myös XML-runko, johon pitää laittaa SignNewCertificate-kutsun vastaussanomassa annettu RetrievalId. (Kuva 7)



Kuva 7. Esimerkki GetCertificate-kutsusta.

GetCertificate-haun vastaussanoma palautti testivarmenteen (Kuva 8). Tämän testivarmenteen käyttöön ottamiseksi piti varmenne vielä lähettää sähköpostilla Verohallinnolle.

Kuva 8. Esimerkki GetCertificate-haun vastaussanomasta.

3.2.2 Varmenteen testaaminen

Sain ilmoituksen siitä, että testivarmenne on käytettävissä, samana päivänä varmenteen lähettämisestä Verohallinnolle. Seuraavaksi oli vuorossa kutsujen testaaminen Postman-sovelluksessa. Tämän jälkeen suunnittelin testata hakuja tuotteessa kovakoodatulla datalla.

Ensimmäiseksi oli ymmärrettävä, kuinka juuri saatu testivarmenne toimii Postmanin kanssa. En ollut ennen käyttänyt minkäänlaisia SSL-varmenteita, joten tämän ymmärtämiseen meni aikaa. Päätin käyttää Postman-sovellusta Talent API Tester -selain laajennuksen sijasta, koska varmenteen käyttö Postman-sovelluksen kanssa oli paljon helpompaa.

Varmenteen käyttö Postman-sovelluksessa oli yksinkertaista. Loin varmenteesta CRT-tiedoston. Tiedoston polku piti laittaa sovelluksen asetusten Certificates-osioon. Sinne myös piti laittaa Client Certificate Request:in luonnin yhteydessä tehdyn avaimen polku. Luomani tiedosto ei toiminut heti. Tämä johtui sen muodosta. Tiedosto oli Base64-muodossa, ja olin laittanut tekstiin kaksi rivinvaihtoa liikaa. Tämän huomatessani pääsin käsiksi testiympäristöön.

Viimeiseksi päätin testata hakuja ja varmennetta vielä sovelluksessa. Loin yksinkertaisen Java-luokan joka ajaa metodin joka päivä. Tämä metodi teki *Muuttuneet eläkkeiden ja etuuksien ennakonpidätystiedot maksajille* -kutsun. Kutsulle ei annettu ChangedFrom-arvoa, jotta se palauttaisi kaiken datan. Haku onnistui ja kutsujen implementointi pystyi alkamaan.

3.3 Kutsujen käyttöönotto ohjelmistossa

Päätin luoda kutsuille oman luokan. Laitoin tälle luokalle kaksi julkista metodia. Toista metodia käytetään päivittäisissä eräajoissa, jossa haetaan muuttuneita verotietoja, ja toista metodia käytetään alkavan eläkkeen ilmoittamiseen, kun eläkekorvaus on hyväksytty järjestelmässä. Projekti toimii Spring-ohjelmistokehyksessä.

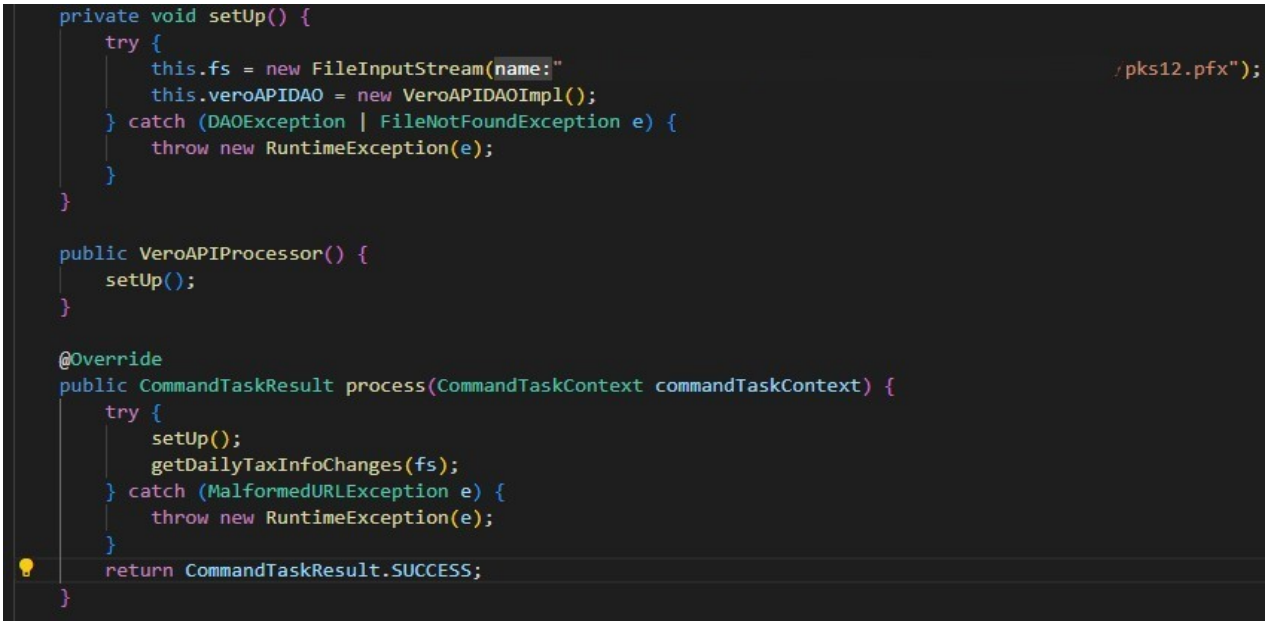
Päivittäisen eräajon getDailyTaxInfoChanges-metodi ajetaan Jenkins automaatio-ohjelmalla. Tuotteessa on jo päivittäinen eräajo. Tämän käyttäminen helpotti työtä, koska minun ei tarvinnut tehdä erillistä eräajoa. Luokan nimi oli VeroAPIProcessor, koska tuotteessa eräajon luokat on tapana nimetä prosessoriluokiksi.

Jokainen kuvankaappaus on kyseisestä luokasta. Tein työssä myös DAO-luokkia millä tuotteen tietokannassa olevaa dataa pystyy muokkaamaan. En näytä kyseisiä luokkia, koska tuotteen tietokanta ei ole julkista tietoa.

3.3.1 Varmenteen käyttöönotto ja yhteyden luominen

HTTP-kutsut Java-ohjelmistossa ovat aika yksinkertaisia, joten kutsujen luonnissa suurin työ oli saada varmenne toimimaan. Aloitin tekemällä metodin, jolla luodaan yhteys rajapintaan. Muutin testivarmenteen PKCS12 muotoiseen tiedostoon ja laitoin sen tuotteen Docker-konttiin.

Kun luokka ajetaan, prosessor-metodi käynnistää setUp-metodin, jolla haetaan testivarmenne ja alustetaan DAO-luokka (Kuva 9). Metodin nimi on process, koska eräajo ajaa luokista aina process-metodin.



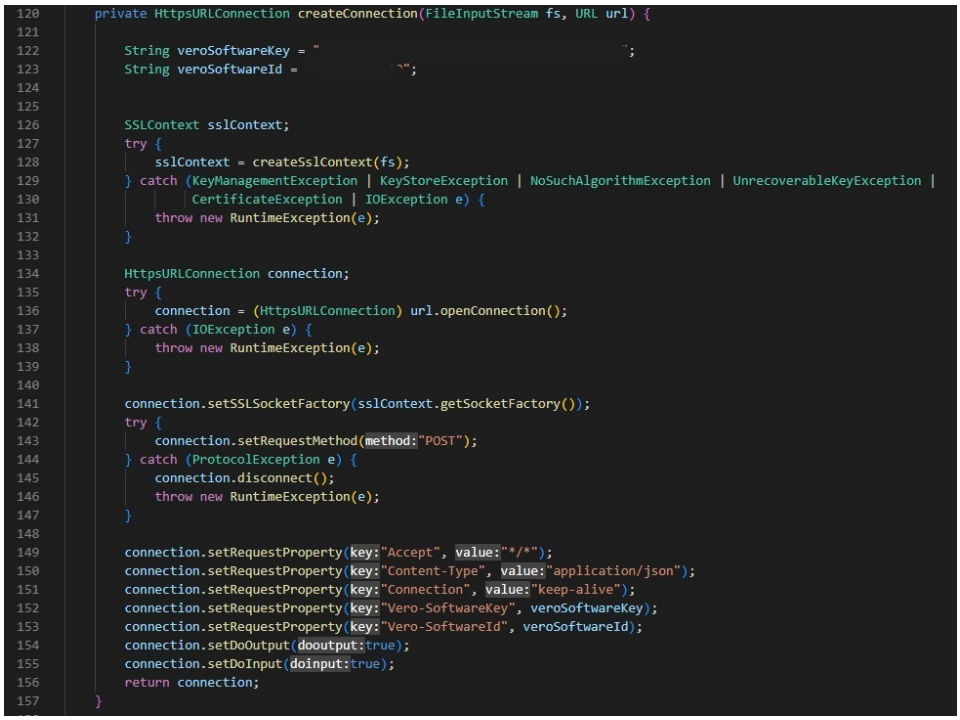
```
private void setUp() {
    try {
        this.fs = new FileInputStream(name:"                                    /pks12.pfx");
        this.veroAPIDAO = new VeroAPIDAOImpl();
    } catch (DAOException | FileNotFoundException e) {
        throw new RuntimeException(e);
    }
}

public VeroAPIProcessor() {
    setUp();
}

@Override
public CommandTaskResult process(CommandTaskContext commandTaskContext) {
    try {
        setUp();
        getDailyTaxInfoChanges(fs);
    } catch (MalformedURLException e) {
        throw new RuntimeException(e);
    }
    return CommandTaskResult.SUCCESS;
}
```

Kuva 9. Kuva process- ja setUp-metodeista.

GetDailyTaxInfoChanges-metodi käynnistää vuorostaan tuottamani createConnection-metodin. Tämä createConnection-metodi luo yhteyden rajapinnan kanssa. Päätin käyttää samaa createConnection-metodia molempien kutsujen tekemisessä, koska molemmissa kutsuissa on sama varmenne ja otsikkotiedot. Molemmat kutsut ovat myös POST-kutsuja. Käytin createConnection-metodissa HttpsURLConnection-luokkaa, koska yhteys pitää turvata testivarmenteella.



```
120     private HttpsURLConnection createConnection(FileInputStream fs, URL url) {
121
122         String veroSoftwareKey = "                                          ";
123         String veroSoftwareId =              ";
124
125
126         SSLContext sslContext;
127         try {
128             sslContext = createSslContext(fs);
129         } catch (KeyManagementException | KeyStoreException | NoSuchAlgorithmException | UnrecoverableKeyException |
130                  CertificateException | IOException e) {
131             throw new RuntimeException(e);
132         }
133
134         HttpsURLConnection connection;
135         try {
136             connection = (HttpsURLConnection) url.openConnection();
137         } catch (IOException e) {
138             throw new RuntimeException(e);
139         }
140
141         connection.setSSLSocketFactory(sslContext.getSocketFactory());
142         try {
143             connection.setRequestMethod(method:"POST");
144         } catch (ProtocolException e) {
145             connection.disconnect();
146             throw new RuntimeException(e);
147         }
148
149         connection.setRequestProperty(key:"Accept", value:"*/*");
150         connection.setRequestProperty(key:"Content-Type", value:"application/json");
151         connection.setRequestProperty(key:"Connection", value:"keep-alive");
152         connection.setRequestProperty(key:"Vero-SoftwareKey", veroSoftwareKey);
153         connection.setRequestProperty(key:"Vero-SoftwareId", veroSoftwareId);
154         connection.setDoOutput(dooutput:true);
155         connection.setDoInput(doinput:true);
156         return connection;
157     }
```

Kuva 10. kuva createConnection-metodista.

SSLContext-luokan luomiseen tein oman createSSLContext nimisen metodin. Metodissa luodaan KeyStore-luokka, jolle ladataan testivarmenne. Salasanan jätin tyhjäksi. KeyManagerFactory-luokka on tarkoitettu annetun avaimen ylläpitämän KeyManager-luokan luontiin. Luokalle annetaan varmenteen pitämä keystore. SSLContext-luokka, jonka createSSLContext-metodi palauttaa, luodaan TLS-tyyppiseksi. TLS on SSL-protokollan seuraaja (SSL 2021). Lopuksi SSLContext:ille annetaan KeyManagerFactory:n luoma KeyManager-luokka.

```
159     private SSLContext createSslContext(FileInputStream fs)
160             throws KeyManagementException, KeyStoreException, NoSuchAlgorithmException, UnrecoverableKeyException,
161             CertificateException, IOException {
162
163         KeyStore keystore = KeyStore.getInstance(type:"PKCS12");
164         keystore.load(fs, "".toCharArray());
165
166         KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
167         kmf.init(keystore, null);
168
169         SSLContext sslContext = SSLContext.getInstance("TLS");
170         sslContext.init(kmf.getKeyManagers(), null, null);
171         return sslContext;
172     }
173
```

Kuva 11. createSSLContext-metodi.

3.3.2 Verotietojen hakeminen

Tuotteeseen päätin siis implementoida kaksi kutsua. Ensimmäinen kutsu on *Muuttuneet eläkkeiden ja etuuksien ennakkopidätystiedot maksajille* eli massakutsu, ja toinen oli *Eläke- ja etuustietojen ilmoittaminen* eli ilmoituskutsu. Aloitin massakutsulla, koska haun tekemiseen ei tarvinnut muuta kuin yhtiön avaimen.

Massakutsu piti tehdä joka päivä, joten eräajo piti olla päivittäinen. Päätin käyttää jo olemassa olevaa päivittäistä eräajoa, koska se vähensi omaa työtä merkittävästi.

Verotietojen hakeminen massakutsulla oli erittäin yksinkertaista luoda, koska ainoat tarvittavat arvot kutsun runkoon olivat asiakkaan tunnus, sekä päivämäärä jonka jälkeisiä muutoksia kysyttiin.

```java
private void getDailyTaxInfoChanges(FileInputStream fs) throws MalformedURLException {

    URL url = null;
    try {
        url = new URL(spec:"https://pintatesti.vero.fi/FIS/Return/IIT/Test/GetWithholdingData/v1");
    } catch (MalformedURLException e) {
        throw new RuntimeException(e);
    };

    HttpsURLConnection connection = createConnection(fs, url);
    String jsonBody = "{\n"
            + "  \"PayerId\": \"" + payerId + "\",\n"
            + "  \"RowLimit\": 1000\n"
            + "}";

    try(OutputStream os = connection.getOutputStream()) {
        byte[] input = jsonBody.getBytes(charsetName:"utf-8");
        os.write(input, off:0, input.length);
    } catch (IOException e) {
        connection.disconnect();
        throw new RuntimeException(e);
    }

    connection.setConnectTimeout(timeout:5000);
    connection.setReadTimeout(timeout:5000);

    BufferedReader in;
    try {
        in = new BufferedReader(
                new InputStreamReader(connection.getInputStream()));
    } catch (IOException e) {
        connection.disconnect();
        throw new RuntimeException(e);
    }
    String inputLine;
    StringBuffer content = new StringBuffer();
    while (true) {
        try {
            if (!((inputLine = in.readLine()) != null)) break;
        } catch (IOException e) {
            connection.disconnect();
            throw new RuntimeException(e);
        }
        content.append(inputLine);
    }

    try {
        in.close();
    } catch (IOException e) {
        connection.disconnect();
        throw new RuntimeException(e);
    }

    readContent(content);
    connection.disconnect();
};
```

Kuva 12. getDailyTaxInfoChanges-metodi

Massakutsun metodi eli getDailyTaxInfoChanges luo yhteyden aloittamalla createConnection-metodin, jonka jälkeen se luo yksinkertaisen rungon haulle JSON-muodossa (Kuva 12). Runkoon ei annettu changedFrom-päivämäärän arvoa, koska verohallinnon palauttamassa testidatassa kaikki verotiedot olivat samalta päivämäärältä. Tämä oli harmittavaa, koska etenkin pilotin esittämisessä olisi ollut hienoa näyttää muuttuneita verotietoja tuotteen päivämäärää vaihtamalla. Kun rajapinta palauttaa datan, ajetaan readContent-metodi. (Kuva 13)

```
174     private void readContent(StringBuffer content) {
175         JSONArray jsonArray = new JSONObject(content.toString()).getJSONArray("WithholdingPercentages");
176         for (int i = 0; i < jsonArray.length(); i++) {
177             JSONObject taxInfo = jsonArray.getJSONObject(i);
178             String id = String.valueOf(taxInfo.get("RecipientId"));
179             int beneficiaryTK = veroAPIDAO.findClaimBeneficiaryTkWithClientPin(id);
180             int beneficiaryTKContainer = veroAPIDAO.findClaimBeneficiaryTkContainerWithClientPin(id);
181             if (beneficiaryTK != 0) {
182                 TaxCard taxCard = jsonTaxInfoToObject(taxInfo, beneficiaryTK, beneficiaryTKContainer);
183                 veroAPIDAO.saveTaxCard(taxCard);
184             }
185         }
186     }
187 
188     private TaxCard jsonTaxInfoToObject(JSONObject taxInfo, int beneficiaryTk, int beneficiaryTkContainer) {
189 
190         Date date = null;
191         TaxCard taxCard = new TaxCard();
192         taxCard.setParentTK(beneficiaryTk);
193         taxCard.setTkContainer(beneficiaryTkContainer);
194         taxCard.setWithholdingPercentage(taxInfo.getDouble("WithholdingPercentage"));
195         taxCard.setValidFrom(date.valueOf(taxInfo.getString("ValidFrom")));
196 
197         if(taxInfo.toString().contains("AdditionalPercentage")){
198             taxCard.setAdditionalPercentage(taxInfo.getDouble("AdditionalPercentage"));
199         }
200 
201         if(taxInfo.toString().contains("IncomeLimit")){
202             taxCard.setIncomeLimit(taxInfo.getDouble("IncomeLimit"));
203         }
204 
205         //mahdollisesti myös pensionCode ja pensionType
206         return taxCard;
207     }
```

Kuva 13. metodit readContent ja jsonTaxInfoToObject.

Palautetut verotiedot palautettiin rajapinnasta listana. Metodi readContent tarkastaa, onko järjestelmässä yhtäkään palautetun verotiedon henkilötunnuksen omaavaa eläkkeen maksunsaajaa. Jos näin on, luodaan JSON-tiedosta TaxCard-olio. Objekti annetaan luomalleni saveTaxCard-metodille, joka tallentaa verokortin tietokantaan, ja lopettaa maksunsaajan nykyisen verokortin. Tallentamisessa kesti kauemmin, koska päätin aluksi tehdä sen SQL-injektiolla. Tietokannan muokkaamisessa pitää osata lisätä muokkaustapahtuman avain sekä verotiedon tekninen avain, mikä ei ollut mahdollista tehdä pelkällä injektiolla. Tiedon tallennus toimi loppujen lopuksi vain tuotteen tietokannan muokkaamiseen tarkoitettujen jo luotujen luokkien avulla, missä luodaan muokkaustapahtuma ja tekninen avain.

3.3.3 Verotietojen ilmoittaminen

Kuten massakutsussa, ilmoittamisen kutsun luonti oli yksinkertaista, kun varmenne asiat oli saatu hoidettua. Kuitenkin tietokannan tiedon saaminen oikeaan muotoon alkavan eläkkeen ilmoittamista varten tuotti ongelmia. Ongelmia tuotti myös oikeanlaisten eläkesopimusten luonti, koska verotietojen ilmoittamisessa on monta sääntöä. Esimerkiksi alkava eläke on alettava ilmoituksen tekemisen kuluvana vuonna, tai seuraavana vuonna, jos ilmoitus tehdään 1. lokakuuta ja 9. marraskuuta välisenä aikana (Verohallinto 2023a).

```
209     public void filePensionData(                                    {
210         URL url = null;
211         try {
212             url = new URL(spec:"https://pintatesti.vero.fi/FIS/Return/IIT/Test/FilePensionAndBenefitData/v1");
213         } catch (MalformedURLException e) {
214             throw new RuntimeException(e);
215         }
216 
217         VeroPaymentInfo veroPaymentInfo = veroAPIDAO.getVeroPaymentInfo(                ;
218 
219         HttpsURLConnection connection = createConnection(fs , url);
220         String jsonBody = "{\n"
221                 + "    \"PayerId\": \""+ payerId +"\",\n"
222                 + "    \"NewPensionOrBenefit\": true,\n"
223                 + "    \"WithholdingYear\": " + veroPaymentInfo.getWithholdingYear() + ",\n"
224                 + "    \"Recipient\": {\n"
225                 + "        \"Type\": 1,\n"
226                 + "        \"PersonalId\": \""+ veroPaymentInfo.getPersonalId() +"\"\n"
227                 + "    },\n"
228                 + "    \"Income\": {\n"
229                 + "        \"Amount\": "+ veroPaymentInfo.getAmount() +",\n"
230                 + "        \"TypeNew\": \"1181\",\n"
231                 + "        \"IncomeNameFI\": \"JaskanVakuutukset-EläkeSäästö\",\n"
232                 + "        \"IncomeNameSE\": \"JaskaFörsäkring-PensionBesparingar\",\n"
233                 + "        \"StartDate\": \"" + veroPaymentInfo.getStartDate() + "\",\n"
234                 + "        \"DecisionDate\": \"" + veroPaymentInfo.getDecisionDate() + "\",\n"
235                 + "        \"DecisionFinal\": true\n"
236                 + "    }\n"
237                 + "}";
238 
239         try(OutputStream os = connection.getOutputStream()) {
240             byte[] input = jsonBody.getBytes(charsetName:"utf-8");
241             os.write(input, off:0, input.length);
242         } catch (IOException e) {
243             connection.disconnect();
244             throw new RuntimeException(e);
245         }
246 
247         connection.setConnectTimeout(timeout:5000);
248         connection.setReadTimeout(timeout:5000);
249 
250         try {
251             connection.getResponseCode();
252         } catch (IOException e) {
253             throw new RuntimeException(e);
254         }
255 
256         connection.disconnect();
257     }
258 }
```

Kuva 14. filePensionData-metodi

Alkavan eläkkeen ilmoittamiseen tehty filePensionData-metodi ajetaan työssä uuden korvauksen hyväksyttäessä (Kuva 14). Loin VeroPaymentInfo-objektin, joka palautetaan luomastani metodista

getVeroPaymentInfo. Tiedon haussa päätin käyttää SQL-injektiota, koska se oli minulle yksinkertaisin tapa saada tietokannasta tietoa mahdollisimman vähillä koodimuutoksilla. Metodin getVeroPaymentInfo luonnissa oli pieniä ongelmia, koska järjestelmässä tietoja pidetään eri paikassa riippuen korvauksen ja sopimuksen tyypistä.

Tässä versiossa vastaussanomasta ei tallenneta mitään tietokantaan. Kutsu kuitenkin palauttaa uniqueIdentifier-avaimen, mutta koska en tee työssä suunniteltuja muutosverokortin tai vuosittaisen peruslaskennan ilmoittamisen kutsuja, avainta ei käytetä.

3.3.4 Vuosittainen peruslaskentaan ilmoitus, muutosverokortti, sekä veroluokat

Työssä suunnitteluosuudessa jäi selväksi, että jotkut osat suunnitellusta implementaatiosta jää opinnäytetyössä tekemättä. Mielestäni työ täyttää tarpeensa eli suunnitellun ratkaisumallin onnistumisen todistamisen myös ilman näitä lisäyksiä. Lisäyksien luomiseen ei olisi myöskään jäänyt aikaa.

Ensimmäinen pois jätetty osa oli vuosittainen peruslaskennan ilmoitus. Eläkeverokortti on aina maksimissaan vuoden ajan voimassa. Jos eläke jatkuu vielä vuoden jälkeen, on tehtävä uusi ilmoituskutsu. Tämän olisin implementoinut vuosittaiseen eräajoon, jossa oltaisiin tarkastettu jokainen voimassa oleva eläke, ja katsottu jatkuuko se toiselle vuodelle. Jos eläke olisi jatkunut toiselle vuodelle olisi tehty ilmoituskutsu, jossa käytettäisiin uniqueIdentifier-avainta.

Toinen pois jätetty osa oli muutosverokortin ilmoittaminen. Eli jos tuotteessa olisi tehty joku muutos eläkkeeseen, olisi siitä automaattisesti tehty ilmoituskutsu. Tässäkin kutsussa olisi käytetty uniqueIdentifier-avainta.

Koska samalla eläkkeellä on mahdollisesti monta erilaista veroluokkaa, viimeinen pois jäänyt implementaatio oli näiden laskeminen. Tämä olisi tehty katsomalla kuinka vanha eläkesopimus on, ja mitkä maksut oli tehty mihinkin aikaan. Näin olisi voinut jakaa eläkkeen määrän eri veroluokkiin. Jokaiselle omille eläkkeen veroluokille olisi tehty oma ilmoituskutsu.

3.4 Lopputulos

Testasin työn tuotosta seuraavanlaisesti. Loin eläkevakuutussopimuksen, jonka laitoin korvaukseen. Luomaani eläkevakuutussopimukseen laitoin maksunsaajaksi testihenkilön, jolle annoin saman henkilötunnuksen mitä massakutsu oli palauttanut Postman-testauksen aikana. Korvauksen luonnissa testasin ilmoitusta IntelliJ-ohjelmistomuokkaussovelluksen debugger-toiminnolla. Ilmoituskutsu toimi.

Massakutsua testasin myös debugger-toiminnolla, mutta sen lisäksi se palautti järjestelmän käyttöliittymässä näkyvän veroprosentin. Päivittäisen eräajon ajamisen jälkeen muuttunut veroprosentti näkyi käyttöliittymällä, joka tarkoitti, että pilottiversio toimi tarkoitetulla tavalla (Kuva 15). Veroluokkaa ei tallennettu pilottiversiossa tietokantaan.

End date	Tax class	Basic %
06/01/2022	Capital tax	30.00
12/31/9999		27.00

Kuva 15: Massakutsun onnistunut verokortin palautus käyttöliittymässä. Ylempi veroprosentti on korvauksen luonnissa tehty oletusverokortti. Alempi on rajapinnan palauttama.

Toimiva pilottiversio todisti suunnittelun ratkaisumallin toimivaksi. Molemmat kutsut toimivat toivotulla tavalla, sekä suunnitteluosuuden ratkaisua on käytetty tulevan projektin suunnittelussa. Olen myös esittänyt työn pilottiversiota yrityksen sisällä ja minut on kutsuttu keskustelemaan tulevasta projektista yrityksen asiakkaiden kanssa. Pidän työn tuotosta siis erittäin onnistuneena.

4 Pohdinta

Työn tavoite onnistui erittäin hyvin. Suunnittelun ratkaisu todistettiin toimivalla pilottiversiolla. Opinnäytetyön tavoitteena oli onnistunut työn tuotos, sekä onnistunut aikatauluttaminen. Työ on valmistunut aikataulussa, sekä työn tuotos oli onnistunut. Työ valmistui 15.5.2023. Onnistunut aikatauluttaminen oli mielestäni hyvän työn rajaamisen ansiota. Työssä jäi implementoimatta peruslaskentaan lähettäminen, muokatun eläkkeen ilmoittaminen ja eri veroluokkien laskeminen. Tämä olin päättänyt jo opinnäytetyön suunnittelussa.

Vaikeuksia työssä oli etenkin ohjelmistoratkaisusta kirjoittaminen. Ratkaisun tekeminen ei ollut vaikeaa, mutta siitä kertominen oli. On vaikea ymmärtää, mitkä teknologiat ovat oleellisia työn tuloksessa, kun niiden kanssa työskentelee joka päivä.

Työtä tullaan jatkokehittämään tulevassa projektissa, missä etenkin suunnittelun ratkaisua käytetään hyväksi. Jos jatkaisin työtä, lisäisin siihen ilmoituskutsut muokatuille eläkkeille sekä peruslaskentaan lähettämiselle. Tekisin myös lisäyksen missä laskettaisiin eläkemaksun määrät eri veroluokille, jos eläkemaksuissa on monta veroluokkaa.

Opin työn teossa erittäin paljon julkisista rajapinnoista ja varmenteista. En ollut esimerkiksi ennen työn alkamista kuullut asiakasvarmenteista. Palvelinvarmenteet ovat erittäin yleisiä, joten uskon saaneeni paljon arvoa varmenteen kanssa työskentelystä. Opin myös paljon itsenäisen työn aikatauluttamisesta, Evitec Life -tuotteesta ja Spring-ohjelmistokehyksen kanssa työskentelystä.

Lähteet

Apache 2023. Welcome to Apache Maven. Luettavissa: https://maven.apache.org/. Luettu: 12.5.2023

Baeldung 2023. Do a simple HTTP Request in Java. Luettavissa: https://www.baeldung.com/java-http-request. Luettu: 4.5.2023

Baeldung 2022a. Java HTTPS Client Certificate Authentication. Luettavissa: https://www.baeldung.com/java-https-client-certificate-authentication. Luettu: 15.4.2023

Baeldung 2022b. Java Keystore API. Luettavissa: https://www.baeldung.com/java-keystore. Luettu: 15.4.2023

Baeldung 2022c. The DAO Pattern in Java. Luettavissa: https://www.baeldung.com/java-dao-pattern. Luettu: 12.5.2023

Docker 2023. Documentation. Luettavissa: https://docs.docker.com/get-started/. Luettu: 12.5.2023

Free Code Camp 2022. What is REST? Rest API definition for beginners. Luettavissa: https://www.freecodecamp.org/news/what-is-rest-rest-api-definition-for-beginners/. Luettu: 26.3.2023

IBM 2023. What is an API (application programming interface). Luettavissa: https://www.ibm.com/topics/api. Luettu: 15.3.2023

JavaTpoint 2023. HTTP. Luettavissa: https://www.javatpoint.com/computer-network-http. Luettu: 26.3.2023

Jenkins 2023. Jenkins User Documentation. Luettavissa: https://www.jenkins.io/doc/. Luettu: 12.5.2023

Jscape 2022. What Is Client Certificate Authentication. Luettavissa: https://www.jscape.com/blog/client-certificate-authentication. Luettu: 13.4.2023

Nordea 2023. Hoida eläkevakuutustasi. Luettavissa: https://www.nordea.fi/henkiloasiakkaat/palvelumme/saastaminen-sijoittaminen/sijoittaminen/hoida-elakevakuutustasi.html. Luettu 15.5.2023

Rouse Margaret 28.3.2014. Client Authentication Certificate. Technopedia. Luettavissa: https://www.techopedia.com/definition/29760/client-authentication-certificate. Luettu 13.4.2023

Spring 2023. Spring Framework. Luettavissa: https://spring.io/projects/spring-framework. Luettu: 12.5.2023

SSL 2021. mikä on SSL. Luettavissa: https://www.ssl.com/fi/FAQ/faq-mik%C3%A4-on-SSL/. Luettu: 13.4.2023

Tulorekisteri 2019. Varmennepalvelu – Rajapintakuvaus. Luettavissa: https://www.vero.fi/globalassets/tulorekisteri/dokumentaatio-2019/varmennepalvelu---rajapintakuvaus.pdf. Luettu: 13.4.2023

Verohallinto 2023a. API-Portaali. Luettavissa: https://avoinomavero.vero.fi/APIPortal/_/#1. Luettu: 1.3.2023

Verohallinto 2023b. Eläketulon verotus. Luettavissa: https://www.vero.fi/syventavat-vero-ohjeet/ohje-hakusivu/48865/elaketulon-verotus7/#2-el%C3%A4kkeiden-veronalaisuus. Luettu: 6.4.2023

Verohallinto 2023c. Tilaa verokortti eläkettä varten. Luettavissa: vero.fi/henkiloasiakkaat/verokortti-ja-veroilmoitus/verokortti/elaketulon_verokortt/. Luettu: 6.4.2023

Verohallinto 2022. Vero API yleiskuvaus. Luettavissa: https://www.vero.fi/tietoa-verohallinnosta/kehittaja/veron-rajapintapalvelut/vero-api/vero-api-yleiskuvaus/. Luettu: 13.4.2023

Verohallinto 2020. Paperinen verokortti on valtaosalle turhake – Verohallinto alkaa julkaista ensi vuoden verokortteja OmaVerossa 2. joulukuuta alkaen. Luettavissa: https://www.vero.fi/tietoa-verohallinnosta/uutishuone/lehdist%C3%B6tiedotteet/2020/paperinen-verokortti-on-valtaosalle-turhake–verohallinto-alkaa-julkaista-ensi-vuoden-verokortteja-omaverossa-2.-joulukuuta-alkaen/. Luettu: 1.3.2023

Verohallinto 2019. Mikä verokortti on? Luettavissa: https://www.vero.fi/henkiloasiakkaat/verokortti-ja-veroilmoitus/verokortti/mik%C3%A4-verokortti-on/. Luettu 15.3.2023

W3schools 2023. what is JSON?. Luettavissa: https://www.w3schools.com/whatis/whatis_json.asp. Luettu: 4.5.2023