Kalle Rissanen

Todentamis- ja valtuuttamiskomponentti moderniin web-palveluun

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

26.5.2021

Tiivistelmä

Tekijä:	Kalle Rissanen
Otsikko:	Todentamis- ja valtuuttamiskomponentti moderniin web-palveluun
Sivumäärä:	41 sivua
Aika:	26.5.2021
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikan tutkinto-ohjelma
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Toimitusjohtaja Tero Pesonen Lehtori Vesa Ollikainen

---

Insinöörityössä tarkastellaan perinteisten ja modernien web-sovelluksien teknologioita ja menetelmiä todentamiseen ja valtuuttamiseen liittyen. Työssä suunnitellaan ja toteutetaan todentamis- ja valtuuttamiskomponentti moderniin web-palveluun, jota ollaan kehittämässä perinteisen web-sovelluksen korvaajaksi.

Työssä puntaroidaan kysymyksiä, kuten miten mahdollistaa yhteinen kirjautuminen uuteen sekä vanhaan sovellukseen ja kuinka laajentaa valtuuttamistoiminnallisuutta HTTP-pyyntö-tasolta kohdeoliotasolle.

Todentamis- ja valtuuttamiskomponentista saadaan toteutettua ensimmäinen versio, joka mahdollistaa laajan valtuuttamistoiminnallisuuden sekä yhteisen kirjautumisen. Toteutuksesta löydettiin kuitenkin ongelmia, jotka vaativat jatkokehitystä.

Avainsanat: Valtuutuslista, moderni web-sovellus, web-teknologiat, todentaminen, valtuuttaminen, kohdeolio, Spring

Abstract

Author:	Kalle Rissanen
Title:	Authentication and Authorization Component for a Modern Web Service
Number of Pages:	41 pages
Date:	26 May 2021
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology and Communications
Professional Major:	Software Engineering
Instructors:	Tero Pesonen, Chief Executive Officer Vesa Ollikainen, Senior Lecturer

---

In this thesis we examine traditional and modern web technologies for authentication and authorization. The thesis includes designing and developing an authentication and authorization component for a modern web service that is being developed to replace a traditional web application.

We consider different approaches for implementing a simultaneous login system between the two applications, and how to extend the current authorization HTTP request-based model to include authorization of domain objects.

First version of the authentication and authorization component was developed with the extended authorization capabilities along with simultaneous login system. However, problems were found in the component that require further development.

Keywords: Access Control List, ACL, modern web application, web technologies, authentication, authorization, access control, domain object, Spring

Sisällys

Lyhenteet

ACL:	Access Control List. Valtuutuslista.
CORS:	Cross-Origin Resource Sharing. Tekniikka, joka mahdollistaa resurssien hyödyntämisen eri osoitteista selainskripteillä.
CSRF:	Cross-Site Request Forgery. Hyökkäys, jossa hyökkääjä huijaa käyttäjän selaimen lähettämään HTTP-pyynnön hyökkääjän tiedoilla.
HTTP:	Hypertext Transfer Protocol.
HTTPS:	Hypertext Transfer Protocol Secure.
JPA:	Java Persistence API. Javan standardi ORM-toiminnallisuudelle.
JSON:	JavaScript Object Notation. Yleinen tietoformaatti.
JWT:	JSON Web Token. JWT-poletti.
MVP:	Minimum Viable Product. Pienin toimiva tuote.
ORM:	Object-relational mapping. Tekniikka relaatiotietokantamallin mukaisen datan muuttamiseksi oliopohjaiseksi.

RBAC:	Role-based access control. Roolipohjainen valtuutus.
REST:	Representational State Transfer. Arkkitehtuurityyli.
TP FONS:	TietoPiiri Oy:n toiminnanohjausjärjestelmä järjestöille.

1 Johdanto

Tarkastellaan tilannetta, jossa on aloitettu uuden selainpohjaisen järjestelmän kehitystyö. Uuden järjestelmän tarkoitus on uusia nykyinen vuosituhannen vaihteen teknologioilla toteutettu web-sovellus. Uuteen järjestelmään kuuluu web-palvelu, joka kommunikoi olemassaolevan tietokannan kanssa, sekä selaimella toimiva dynaaminen web-sovellus, joka käyttää hyväkseen web-palvelun tarjoamaa rajapintaa. Kyse on asiakasrekisterisovelluksesta, jolla asiakkaan työntekijät pystyvät tarkastelemaan, tallentamaan ja poistamaan asiakastietoja. Uuden järjestelmän kehitysversioon on lisätty asiakaskontakteihin liityvä perustoiminnallisuus, mutta sisäänkirjautumista, istunnonhallintaa ja käyttäjien valtuuttamista ei ole vielä suunniteltu.

Edellisessä kappaleessa esitelty tilanne on hyvin yleinen. Monet yritykset ovat siirtymässä käyttämään nykyaikaisia dynaamisia selainsovelluksia ja web-palveluita perinteisten web-sovelluksien sijaan. Modernit selainpuolen JavaScript-sovelluskehykset mahdollistavat perinteisiä verkkosivuja rikkaamman käyttökokemuksen selaimella. Palvelinpuolen ja asiakaspuolen eriyttäminen myös mahdollistaa niiden rinnakkaisen kehittämisen.

Tämä insinöörityö pureutuu alussa esitetyn vastaavanlaisen tilanteen haasteisiin. Työssä suunnitellaan ja toteutetaan todentamis- ja valtuuttamiskomponentti TietoPiiri Oy -nimisen yrityksen uuteen toiminnanohjausjärjestelmään. Todentamis- ja valtuuttamiskomponentin sunnittelussa ja toteuttamisessa keskitytään seuraaviin kysymyksiin.

- Miten vanhan ja uuden järjestelmän välille voi luoda yhteisen kirjautumisen?

- Miten todentaminen toteutetaan?

- Miten valtuuttaa rajapinnan tarjoamia HTTP-pyyntöjä?

- Miten valtuuttaa yksittäisiä resursseja järjestelmässä?

Todentaminen tarkoittaa käyttäjän tunnistamista ja identiteetin varmistamista. Esimerkiksi käyttäjätunnuksen ja salasanan lähettäminen sähköpostipalveluun on todentamista. Valtuuttamisella tarkoitetaan käyttäjän oikeuksia toimenpiteisiin ja resursseihin.

Luvussa 2 esitellään työn lähtökohdat, kuten toimintaympäristö ja tekniset reunaehdot. Luku 3 on työn teorialuku ja siinä esitellään teknologioita ja käsitteitä, joita hyödynnetään luvussa 4, joka on toteutusluku. Luvussa 5 arvioidaan työtä teknisestä näkökulmasta. Luku 6 on yhteenveto. Teorialuku on kirjoitettu mahdollisimman yleiseksi, jotta mahdollisimman moni lukija hyötysi siinä esitetyistä asioista. Toteutusluvussa esitellään Spring-sovelluskehystä ja todentamis- ja valtuuttamiskomponenttiin liittyviin teknisiin ratkaisuihin konkreettisen toteutuksen lisäksi. Seuraavassa luvussa esitellään tarkemmin yritystä sekä komponentin vaatimuksia.

2 Lähtökohdat

Tässä luvussa esitellään tarkemmin yritys, yrityksen nykyinen TP FONS -toiminnanohjausjärjestelmä sekä vaatimuksia uuden järjestelmän todentamis- ja valtuuttamiskomponentille. Luvun tarkoitus on antaa lukijalle pohjatiedot uuden järjestelmän tarpeesta.

2.1 TietoPiiri Oy

TietoPiiri Oy on vuonna 1974 perustettu IT-ratkaisuihin keskittynyt yritys. Yrityksen pääasiakkaita ovat erilaiset hyväntekeväisyysjärjestöt, jotka tarvitsevat työkaluja eritoten varainhankintaan sekä vapaaehtoisten kerryttämiseen. Yrityksen tärkein tuote on TP FONS -toiminnanohjausjärjestelmä.

2.2 TP FONS

TP FONS on web-pohjainen selaimella toimiva toiminnanohjausjärjestelmä. TP FONS -järjestelmä tarjoaa työkalut mm. kontaktien hallintaan, varainhankintaan sekä jäsenhallintaan. Nykyinen, tällä hetkellä käytössä oleva TP FONS on sukupolvea neljä. Tästä järjestelmästä ollaan kehittämässä viidennen sukupolven versiota. Tästä eteenpäin tässä työssä TP FONS v4 viittaa nykyiseen järjestelmään ja TP FONS v5 tulevaan.

TP FONS v4 on perinteinen web-sovellus, joka hyödyntää Java Servlet- ja Java Server Page (JSP) -teknologioita. Se tarjoaa palvelinpuolella generoituja staattisia verkkosivuja käyttäjälle. TP FONS v4 on saanut alkunsa 2000-luvun alkupuolella ja on ollut siitä lähtien käytössä. TP FONS v4 -järjestelmään on ajan saatossa lisätty monenmoista toiminnallisuutta. Toiminnallisuuden lisäämisen hankaluus ja käyttöliittymän vanhuus ovat pääsyyt viidennen version kehittämiselle moderneilla web-teknologioilla.

TP FONS v5 -järjestelmä kehitetään lähes puhtaalta pöydältä. Tarkoituksena, kuten johdannossa esitetyssä tilanteessa, on eriyttää asiakassovellus (front-end) ja web-palvelu (back-end) toisistaan pienempiin, helposti hallittavimpiin kokonaisuuksiin. Asiakassovellus ja web-palvelu kommunikoisivat rajapinnan avulla. Tietokantaa ei ole tällä hetkellä tarkoitus uusia, joten uusi järjestelmä tulee käyttämään TP FONS v4 -järjestelmän kanssa samaa tietokantaa. Asiakassovellus tulee ensiksi olemaan selainsovellus, mutta tämä asetelma mahdollistaisi myös esimerkiksi oman mobiilisovelluksen lisäämisen melko vaivattomasti. TP FONS v5 -järjestelmän käyttöönotto tulisi tapahtumaan osissa, sillä kokonaisen järjestelmän käyttöönotto kerralla ei olisi mielekästä pitkälti nykyisen TP FONS v4 -järjestelmän laajuuden takia. TP FONS v5 -järjestelmän liiketoimintatoiminnallisuutta on jo aloitettu kehittämään, mutta sen todentamis- ja valtuuttamistoiminnallisuutta ei ole vielä suunniteltu. Todentamis- ja valtuuttamiskomponentin lisääminen tarkoittaa todentamis- ja valtuuttamistoiminnallisuuden lisäämistä samaan koodikantaan liiketoimintatoiminnallisuuden rinnalle web-palveluun. Liiketoimintatoiminnallisuudella viitataan järjestelmän tietojen hallintatoimintoihin, kuten miten tietojen tarjoaminen asiakassovellukselle tai tietojen muuttaminen tai poisto tapahtuu.

2.3 Vaatimukset todentamis- ja valtuutuskomponentille

Ensimmäinen vaatimus TP FONS v5 -järjestelmän todentamis- ja valtuuttamiskomponentille on, että käyttäjien pitäisi sujuvasti pystyä käyttämään TP FONS v4 -järjestelmää TP FONS v5 -järjestelmän ohessa, koska se otetaan käyttöön osissa. Käytännössä tämä tarkoittaisi, että TP FONS v5 -selainsovelluksesta voisi esimerkiksi linkin kautta siirtyä vanhaan järjestelmään.

Toinen vaatimus on, että valtuuttamisesta tehtäisiin entistä kattavampaa. TP FONS v4 -järjestelmässä valtuutus ulottuu HTTP-pyyntöjen tasolle. Tulevaan järjestelmään halutaan, että HTTP-pyyntöjen lisäksi voisi valtuuttaa myös kohdeolioita. Kohdeolio tarkoittaa tässä yksittäistä tietokantatietuetta ja

mahdollisesti siihen suhteessa olevia tietokantatietueita. Esimerkkinä ovat yhden asiakkaan tiedot. Asiakkaalla on tietokannassa oma taulu, jossa on mm. asiakkaan nimi, sähköpostiosoite ja tunniste. Asiakkaalla voi myös olla nolla tai enemmän tilauksia. Tilauksille on myös oma taulunsa, jossa on tilauksen nimi, sarjanumero ja tilauksen tehneen asiakkaan tunniste. Yksittäinen kohdeolio voisi tässä olla asiakas ja sen tekemät tilaukset.

Kohdeolioiden valtuutus tekee valtuuttamisesta hankalampaa superkäyttäjän näkökulmasta. Kohdeolioiden valtuuttaminen yksitellen on todella työlästä niiden määrän takia. Siksi valtuutuksessa pitää myös keskittyä siihen, miten tietylle roolille voisi antaa valtuudet tiettyyn joukkoon kohdeolioita. TP FONS v4 -järjestelmässä esiintyy kohdeolioiden välillä luonnollisia suhteita. Esimerkiksi yhteen ryhmään voi kuulua monta kontaktia, moni kontakti voi kuulua moneen ryhmään, sekä yhteen lahjoituskohteeseen on voitu tehdä monta lahjoitusta. Kolmas vaatimus todentamis- ja valtuuttamiskomponentille on, että kohdeolioiden valtuuttamisessa on voitava hyödyntää kohdeolioiden välisiä suhteita esimerkiksi luomalla uuden roolin, jolla on oikeudet vain ryhmään liitettyihin kontakteihin.

Neljäs vaatimus todentamis- ja valtuuttamiskomponentille on, että sen toteutuksen tulisi aiheuttaa mahdollisimman vähän muutoksia nykyiseen TP FONS v4 -järjestelmään.

3 Web-sovellus

Tämä luku tarjoaa todentamis- ja valtuuttamiskomponentin ymmärtämiseksi tarvittavat pohjatiedot web-tekniikoista. Luvun alussa tutustutaan HTTP-protokollaan, joka mahdollistaa web-sovellukset tarjoamalla työkalut tiedon lähettämiseen. Luku myös avaa todentamisen ja valtuuttamisen merkitystä ja perinteisten ja modernien web-sovellusten eroja.

3.1 Hypertext Transfer -protokolla

Hypertext Transfer -protokolla, tuttavallisemmin HTTP, on perusta web-sovelluksille ja verkkosivuille. Se mahdollistaa resurssien, kuten HTML-dokumenttien siirtämisen. HTTP on asiakas-palvelin-protokolla. HTTP on luonteeltaan tilaton. [1.] HTTP:n tietoliikenne on salaamatonta, mutta on olemassa HTTPS (Hypertext Transfer Protocol Secure), missä tietoliikenne on salattua.

HTTP-pyyntö (engl. HTTP request) koostuu HTTP-metodista, osoitteesta, johon pyyntö lähetetään, HTTP-versiotiedosta, otsakkeista (engl. header) ja mahdollisesta sisällöstä (engl. body). Metodi on yleensä toiminnallisuutta kuvastava verbi, kuten GET tai POST. Otsakkeet voivat sisältää ylimääräistä tietoa palvelimelle, kuten tietoa evästeistä (engl. cookie) tai sisällön tyypistä. [1.] Moderneissa web-sovelluksissa pyynnön sisältö on yleensä JSON-formaatissa (JavaScript Object Notation). Esimerkkikoodissa 1 on kuvattu yksinkertainen POST-pyyntö.

```
POST /test HTTP/1.1
Host: esimerkki.fi
Content-Type: application/x-www-form-urlencoded
Content-Length: 25

etunimi=john&sukunimi=doe
```

Esimerkkikoodi 1. HTTP-pyyntö POST-metodilla

Koodiesimerkissä lähetetään osoitteeseen esimerkki.fi/test kaksi parametriä, etunimi ja sukunimi. Ne on kuvattu esimerkin alimmalla rivillä.

HTTP-vastaus koostuu HTTP-versiotiedosta, tilakoodista (engl. status code), tilaviestistä, otsakkeista ja mahdollisesta sisällöstä. Tilakoodi on numero, joka kertoo HTTP-pyynnön tilan. [1.] Esimerkiksi koodi 200 tyypillisesti tarkoittaa HTTP-pyynnön onnistuneen ja koodi 404, ettei pyydettyä resurssia löytynyt. Koodeille on olemassa omat standardinsa, mutta on palvelimen päätös, minkä koodin palvelin palauttaa.

HTTP-evästeet ovat olennainen osa web-sovelluksia. Evästeet sisältävät pieniä määriä tietoa, jonka selain tai muu asiakassovellus voi tallentaa paikallisesti myöhempää käyttöä varten. Todentamisessa evästeitä usein hyödynnetään käyttäjän istunnon ylläpitämiseen. Palvelin voi pyytää asiakassovellusta tallentamaan evästeen HTTP-vastaukseen lisätyn Set-Cookie-otsakkeen avulla. Kun eväste on tallennettu, asiakassovellus lisää sen automaattisesti pyyntöjen, jotka lähetetään evästeessä määritettyyn verkkotunnukseen, Cookie-otsakkeeseen. [2.]

Käytettäessä evästeitä tietoturvan kannalta merkittävien tietojen tallentamiseen, on huomioitava evästeen vanhenemisaika, joka kannattaa laittaa mahdollisimman lyhyeksi, käytännössä istuntoevästeeksi (engl. session cookie), jolloin tietoturvallinen selain poistaa sen selaimen sulkemisen yhteydessä. Idea on, että aikaikkuna tietoturvallisten evästeiden väärinkäyttämiselle olisi mahdollisimman lyhyt. Tietoturvan kannalta merkittävä tieto voisi olla esimerkiksi istuntotunniste.

Evästeelle voi palvelin määrittää myös lisäominaisuuksia. Näistä keskeisimmät ovat Secure-, httpOnly- ja SameSite-ominaisuudet. Secure-ominaisuus kertoo selaimelle, että kyseisen evästeen saa lähettää ainoastaan suojatun HTTPS-protokollan avulla. httpOnly-ominaisuus kertoo, että selaimessa ajettava JavaScript-koodi ei pysty lukemaan httpOnly-evästeen tietoja [2]. SameSite-ominaisuus, joka on esitetty RFC6265bis-dokumentissä, suojaa käyttäjää

hyökkäyksiltä, joissa hyökkääjä käyttää käyttäjän evästeitä väärentääksen HTTP-pyynnön, kuten esimerkiksi Cross-Site Request Forgery -hyökkäyksissä tehdään [3, luku 1; 4, luku 8.2].

Kaiken kaikkiaan HTTP-protokollaan liittyy hurja määrä erilaisia sopimuksia ja mekanismeja, joiden varassa web-sovellusten tietoturva riippuu.

3.2 Cross-Site Request Forgery

Cross-Site Request Forgery -hyökkäys (CSRF) on pitkään ollut etenkin perinteisissä web-sovelluksissa varsinainen murheenkryyni, mutta nykyään se on pienempi ongelma. Tämän huomaa, jos vertaa vuoden 2010 OWASP Top 10 -listaa vuoden 2017 listaan [5; 6]. Vuonna 2010 CSRF-hyökkäys oli sijalla viisi, kun taas vuonna 2017 se on tippunut listalta kokonaan.

CSRF-hyökkäyksessä hyökkääjä hämää käyttäjän selainta lähettämään pahantahtoisen pyynnön siten, että palvelin luulee sen tulevan käyttäjältä. Tavallisesti tässä hyödynnetään evästeitä, jotka lähetetään jokaisen pyynnön yhteydessä evästeessä määritettyyn osoitteeseen, vaikka pyyntö lähetettäisiinkin hyökkääjän verkkosivulta (ellei SameSite-ominaisuutta ole määritelty evästeeseen). Perinteisesti CSRF-hyökkäyksiltä on suojauduttu Synchronizer Token -suunnittelumallilla. Suunnittelumallissa käyttäjän istuntoon lisätään satunnainen merkkijono, joka toimii CSRF-polettina. HTTP-pyynnön yhteydessä palvelin palauttaa käyttäjälle poletin, jonka käyttäjä sitten lisää tuleviin palvelimen resurssien tilaa muuttaviin pyyntöihin. CSRF-polettia ei kannata lähettää palvelimelle ainoastaan evästeenä, koska silloin se lähtee myös hyökkääjän toimesta, eikä siten suojaa hyökkäykseltä. [7, s. 5-11.]

Myös palvelimen kannalta tilattomia ratkaisuja CSRF-suojaukseen on olemassa. Yksi näistä on nk. kaksoislähetys, jossa CSRF-poletti palautetaan selaimelle evästeenä ja käyttäjän tehdessä palvelimen resurssien tilaa muuttavan toiminnon poletti luetaan evästeestä ja lisätään manuaalisesti HTTP-

pyyntöön, esimerkiksi otsakkeeseen. Pyynnön saapuessa palvelimelle palvelin vertaa evästeenä saapunutta polettia otsakkeen polettiin. Jos poletit ovat samat, palvelin hyväksyy pyynnön. Kaksoislähetyksen tietoturva pohjautuu siihen, ettei toisen verkkotunnuksen evästeitä pysty lukemaan JavaScriptillä, eikä omia otsakkeita pysty HTTP-pyyntöön lisäämään ilman JavaScriptiä.

Moderneissa web-sovelluksissa CSRF-suojauksen välttämättömyys ei ole täysin suoraviivaista. Oletetaan, että johdannosta tutun web-palvelun HTTP-pyyntöjen sisällöt ovat JSON-tyyppisiä ja web-palvelu validoi sisältötyypin. Nyt tyypilliset CSRF-hyökkäykset, jotka tehdään HTML-lomakkeiden avulla, eivät ole enää mahdollisia, koska (tällä hetkellä) HTML-lomakkeella ei pysty lähettämään dataa JSON-sisältötyypillä. Ajaxilla tehdyt pyynnöt taas kaatuvat Cross-Origin Resource Sharing -asetuksiin (CORS), jotka on (toivottavasti) konfiguroitu oikein palvelinpuolella. [8.] Tällaiset sopimukset saattavat kuitenkin ajan saatossa muuttua, ja niitä voikin kutsua vain viritelmiksi oikean CSRF-suojauksen sijaan.

3.3 Web-sovelluksen määritelmä

Jim Conallen määrittelee web-sovelluksen järjestelmänä, jossa web-palvelin tarjoaa tietoa selaimelle HTTP:n välityksellä tietoverkossa ja jossa käyttäjä pystyy muokkaamaan palvelimen tarjoaman tiedon tilaa. Täten esimerkiksi staattiset verkkosivut eivät ole web-sovelluksia, koska käyttäjä ei pysty muokkaamaan palvelimen tarjoavan tiedon tilaa. Web-sovellukset ovat asiakas-palvelin-järjestelmiä. Web-sovellusten käyttöliittymä toimii selaimella. [9, s. 63-64.] Edellinen määritelmä on vuodelta 1999, mutta se pätee edelleen. Web-sovelluksien totetuksissa on kuitenkin tapahtunut suurta muutosta, ja tämän vuoksi tässä työssä erotellaan perinteiset ja modernit web-sovellukset toisistaan.

Perinteinen web-sovellus (engl. traditional web application) hoitaa sivujen luomisen palvelinpuolella. Luotu sivu lähetetään selaimelle HTTP-pyynnön

yhteydessä, ja selain lopulta näyttää sen käyttäjälle. Sovelluksen esityslogiikka sijaitsee palvelinpuolella. [10, s. 9.]

Modernin web-sovelluksen (engl. modern web application) esityslogiikka sijaitsee asiakaspuolella, eli selaimella. Palvelinpuolen tehtäväksi jää alustavan sivun lataamisen jälkeen tarjota selainsovellukselle tietoa ja toimintoja web-rajapinnan kautta [10, s. 12-13]. Web-rajapinnan (engl. web application programming interface, API) takana on web-palvelu (engl. web service) [11, luku 1], joka pitää huolen palvelun tarjoaman tiedon tilasta.

REST (representational state transfer) on kova sana moderneissa web-rajapinnoissa. REST on arkkitehtuurityyli, joka esiteltiin vuonna 2000 Roy Fieldingin väitöskirjassa. REST-arkkitehtuurityylin tarkoitus on parantaa mm. skaalautuvuutta, rajapintojen yhtenäisyyttä sekä komponenttien erillistä käyttöönottoa [13, s. 105].

Nykyään REST-rajapinnalla tarkoitetaan usein, että rajapinnan tarjoama tieto ja toiminnallisuus on selkeiden ja yhtenäisten osoitteiden takana ja että rajapinnassa käytetään osuvia pyyntömetodeja sekä vastausten tilakoodeja (engl. status code). Tässä työssä TP FONS v5 -web-palvelun rajapintaa ei kuvailla REST-rajapintana, vaikka siinä onkin tiettyjä REST-piirteitä.

3.4 Käyttäjien todentaminen

Todentaminen (engl. authentication) tarkoittaa käyttäjän identiteetin varmistamista [14]. Tässä työssä käyttäjällä tarkoitetaan asiakassovellusta käyttävää henkilöä ellei toisin mainita. Web-sovelluksissa käyttäjä tyypillisesti lähettää pääsytietonsa palvelimelle, joka sitten varmistaa ne palvelimella sijaitsevien tietojen pohjalta. Pääsytiedot ovat yleensä käyttäjätunnus ja salasana -yhdistelmä, ja varmistus yleensä lähetetyn salasanan vertaamista tietokannassa sijaitsevaan salasanaan. Mikäli salasanat vastaavat toisiaan, käyttäjän identiteetti on varmistettu, eli käyttäjä on todennettu. Huomautuksena

mainittakoon, että salasanat eivät yleensä ole tietokannassa selkotekstinä, vaan ne on tiivistetty (engl. hashed).

Kahdenkertainen todennus on nimensä mukaisesti kahden eri todentamismenetelmän vaatimista käyttäjältä [15, s. 318-319]. Pääsytietojen lisäksi toinen todentamismenetelmä voi olla esim. käyttäjän sähköpostiin lähetetty koodi, jonka käyttäjä lähettää pääsytietojen mukana palvelimelle, joka todentaa käyttäjän web-sovelluksen käyttöä varten.

Käyttäjän käyttäjätunnuksen ja salasanan lähettäminen joka HTTP-pyynnön yhteydessä on täysin mahdollista, ja sille onkin kirjoitettu oma menetelmä nimeltään Basic Authentication. Menetelmässä käyttäjätunnus ja salasana laitetaan HTTP-otsikon sisään ja koodataan Base64-menetelmällä. Basic Authentication -menetelmä on sellaisenaan altis man-in-the-middle-hyökkäyksille, ja menetelmää käyttäessä tulee käyttää suojattua HTTPS-protokollaa [16, s. 3-8.].

Käytettävyyden kannalta Basic Authentication -menetelmä ei web-sovelluksissa ole erityisen mielekäs, sillä käyttäjän tulisi jokaisen HTTP-pyynnön yhteydessä syöttää käyttäjätunnus ja salasana, tai asiakassovelluksen tulisi tallentaa käyttäjätunnus ja salasana asiakassovellukseen ja sitten lisätä ne pyyntöihin. Käyttäjätunnuksen ja salasanan tallentaminen asiakassovellukselle, tässä tapauksessa selaimelle, ei ole erityisen hyvä idea, koska se tekisi käyttäjätunnuksesta ja salasanasta alttiin mm. Cross-Site Scripting (XSS) -hyökkäyksille, joissa hyökkääjä sisällyttää haitallista koodia selainsovellukseen, joka näkyy muille käyttäjille. Monet käyttäjät käyttävät samoja salasanoja eri sovelluksissa, ja tällöin käyttäjätunnuksen ja salasanan joutumisella väärin käsiin voi olla vakavat seuraukset. [12, luku 12.3.]

Perinteisissä web-sovelluksissa käytetään usein istunnon (engl. session) ylläpitämiseen istuntotunnisteita. Kun käyttäjä ensimmäisen kerran lataa web-sovelluksen, palvelin luo sille istunnon sekä istuntotunnisteen, joita se säilyttää yleisesti muistissa tai tietokannassa. Käyttäjän kirjautuessa sisään

onnistuneesti palvelin merkitsee istunnon todennetuksi. Istuntotunnisteet lähetetään asiakassovellukselle usein evästeinä (engl. cookie). Selain lähettää istuntotunniste-evästeen automaattisesti HTTP-pyynnön yhteydessä palvelimelle. Palvelin sitten tarkistaa lähetetyn istuntotunnisteen avulla onko käyttäjän istunto todennetty ja sen perusteella keskeyttää pyynnön käsittelyn tai jatkaa sitä. [17.]

Moderneissa web-sovelluksissa käytetään usein istuntotunnisteiden sijasta tilattomia poletteja (engl. token) istunnon ylläpitämiseen. JSON Web Token (JWT) on paljon käytetty teknologia polettipohjaiseen todentamiseen. Tilattomassa polettipohjaisessa todentamisessa palvelin luo käyttäjälle poletin sisäänkirjautumisen yhteydessä. Poletissa on tyypillisesti tietoja käyttäjästä, kuten käyttäjätunnus ja käyttäjällä olevat valtuudet, sekä metadataa, kuten poletin vanhentumisaika. Poletti allekirjoitetaan (engl. sign) palvelimen toimesta, joka takaa sen, ettei poletin tietoa voi muuttaa ilman, että palvelin sen huomaisi, ellei tiedä palvelimen käyttämää allekirjoitussalasanaa. [18, s. 9.] Asiakassovellus lähettää poletin HTTP-pyyntöjen yhteydessä. Olennainen ero istuntotunnisteisiin on se, että nyt palvelimen ei tarvitse pitää kirjaa käyttäjistä, vaan sen täytyy ainoastaan allekirjoittaa pyynnössa tullut poletti ja verrata sitä poletin mukana tulleeseen allekirjoitukseen. Näin saadaan selville, onko polettia muokattu luvatta matkan varrella.

Koska tilattomista poleteista ei web-palvelun tarvitse pitää kirjaa, on tilattomilla poleteilla web-sovelluksen skaalaus helpompaa. Tässä kuitenkin tehdään kompromissi, sillä nyt jokaisen HTTP-pyynnön yhteydessä on lähetettävä kaikki todentamiseen ja valtuuttamiseen liittyvät tiedot allekirjoituksineen ja metadatoineen. Tilattomuudestaan johtuen polettien mitätöinti web-palvelusta käsin ei ole mahdollista. Tästä johtuen poleteilla on yleensä umpeutumisaika, joka on asetettu mahdollisimman lyhyeksi. Tämä kuitenkin haittaa käyttäjäkokemusta, ja välillä polettien, etenkin JWT-polettien kanssa, käytetään virkistyspolettia (engl. refresh token), joka hoitaa poletin uusimisen [19].

Poletteja käyttää myös OAuth 2.0 -protokolla. OAuth 2.0 -protokolla mahdollistaa ulkopuolisten järjestelmien pääsyn web-sovelluksen tietoihin rajoitetusti [20].

OWASP Top 10 ja OWASP API Security TOP 10 -listoissa rikkinäinen todentaminen on sijalla kaksi [5; 21]. Valtuutukseen, jota seuraava luku käsittelee, liittyvät tietoturvauhat esiintyvät listoilla kolmesti.

3.5 Valtuutus

Valtuutus (engl. authorization, access control), jota kutsutaan myös pääsynvalvonnaksi, tarkoittaa todennetun käyttäjän oikeuksia tehdä tietty toiminto. Toiminto voi olla mm. uuden resurssin luominen tai tietyn resurssin lukeminen. Moderneissa web-sovelluksissa tyypillinen tapa jaotella toimintoja on web-rajapinnan tarjoamien HTTP-pyyntöjen mukaan. Tarkastellaan seuraavaksi roolipohjaista valtuuttamista ja valtuutuslistaa.

Roolipohjainen valtuutus (engl. RBAC, role-based access control) tarkoittaa käyttäjien jakamista rooleihin. Käyttäjällä voi olla yksi tai monta roolia. Rooleille sitten sallitaan joukko oikeuksia, jotka valtuuttavat tiettyjen toimintojen tekemisen. Roolipohjaisen valtuutuksen suurin etu on, että se ei suoraan kytke käyttäjiä ja oikeuksia toisiinsa. Roolipohjaista valtuuttamista kannattaa harkita, kun

- oikeudet eivät muutu tiuhaan
- valtuutusroolit vastaavat web-sovellusta käyttävän organisaation sisäisiä rooleja, eivätkä ole vain joukko abstrakteja oikeusryhmiä
- oikeuksien ja roolien lukumäärän suhde pysyisi mahdollisimman suurena, jotta roolien hallitsemista ei tulisi liian työlästä superkäyttäjille. [14.]

Valtuutuslista (engl. ACL, access control list) on lista oikeuksia, jotka on liitetty tiettyyn kohdeolioon. Valtuutuslista pitää kirjaa mitkä kohdeoliot ovat keiden käyttäjien tai roolien käytettävissä. [22.] Valtuutuslistaa käytetään tyypillisesti kohdeolioiden valtuuttamiseen. Valtuutuslistan ongelmia ovat valtuuksien hallinnan monimutkaistuminen ja suorituskyky. Valtuutettavien kohdeolioiden kasvaessa valtuutuslistan ylläpito alkaa viemään enemmän ja enemmän tilaa puhumattakaan siitä, että käyttäjän oikeudet jokaiseen kohdeolioon pitää tarkastaa erikseen. Tämä saattaa hidastaa käyttäjäkokemusta huomattavasti haettaessa isoja tietomääriä.

Valtuutuslistan ja roolipohjaisen valtuutuksen yhteiskäyttö on täysin mahdollista. Toteutusluvussa roolipohjaista valtuuttamista ja valtuutuslistaa käytetään yhdessä yksityiskohtaisen valtuutustoiminnalisuuden saavuttamiseksi.

Yksittäisten tietokantatietueiden valtuuttamiseen tietyille käyttäjille on myös jossain tietokannoissa valmiit työkalut. Esimerkiksi PostgreSQL-tietokantajärjestelmä tarjoaa Row Level Security -toiminnon, jolla pystyy rajoittamaan tiettyjen käyttäjien pääsyä tiettyihin tietokantatietueisiin [23]. Tätä ei kuitenkaan tässä työssä tutkittu pidemmälle, koska työssä käytetty MariaDB-tietokantajärjestelmä ei tue kyseistä toiminnallisuutta.

4 Todentamis- ja valtuuttamiskomponentti

Tämä luku kuvaa todentamis- ja valtuuttamiskomponentin toteutusvaiheen. Luvussa esitellään tärkeimmät työkalut, teknologiset valinnat sekä kuvaus toteutuksesta.

4.1 Sovellusympäristö

Tässä aliluvussa esitellään työkaluja ja teknologioita, joiden avulla todentamis- ja valtuuttamiskomponentti toteutetaan.

TP FONS v5 -web-palvelu on kirjoitettu Javalla hyödyntäen Spring-sovelluskehystä. Spring [24] on sovelluskehys pääasiassa Java-ohjelmointikielelle. Spring-sovelluskehykseen on tarjolla paljon virallisia lisäosia, jotka mahdollistavat erilaisten ohjelmien ja palveluiden luonnin. Spring-sovelluskehyksen käytön päätös todentamis- ja valtuuttamiskomponentissa oli selvä, koska TP FONS v5 web-palvelu käyttää Spring-sovelluskehystä ja sen lisäosia, jolloin sen käyttö myös todentamis- ja valtuuttamiskomponentissa on luonnollista.

Spring-sovelluskehyksen ytimessä on Inversion of Control -periaate (IoC) ja Dependecy Injection -suunnittelumalli (DI). IoC-periaatteen idea on siirtää ohjelman hallinta käyttäjältä sovelluskehykselle. Yksinkertaistettuna tämä tarkoittaa, että käyttäjä ei enää kirjoita pääsilmukkaa ja sen kutsumia metodeita, vaan pääsilmukka jätetään sovelluskehykselle, joka sitten kutsuu ohjelmoijan kirjoittamia metodeita. DI-suunnittelumallin idea on luoda olio, joka hoitaa luokkien välisten riippuvuuksien alustamisen. [25.] Tällöin ohjelmoijan ei tarvitse itse luoda monimutkaisia toteutuksia olioiden ilmentämiselle ja alustamiselle.

Spring Boot -lisäosa tarjoaa työkalut itsenäisten Spring-sovellusten luontiin minimaalisella konfiguraatiolla. Spring-sovelluskehystä on usein kritisoitu Spring-sovellusten vaatimasta monimutkaisesta XML-konfiguraatiosta. Spring

Boot pyrkii korjaaman tämän ongelman tarjoamalla valmiin konfiguraation useimpiin käyttökohteisiin. Tämä vähentää tarvittavan konfiguroinnin määrää. Spring Boot myös yksinkertaistaa riippuvuuksien hallintaa tarjoamalla starter-riipuvuuksia, joihin on koottu oleellisimmat riippuvuudet kategorioiden mukaan.

Spring Data -lisäosa tarjoaa tietojen käsittelyyn tarvittavaa toiminnallisuutta, kuten Object-relational mapping (ORM) -työkalut sekä REST-henkisten web-palveluiden toteuttamiseen tarvittavaa toiminnallisuutta.

Spring Security -lisäosa tarjoaa todentamis- ja valtuuttamistoiminnallisuutta ja dokumentoinnin mukaan se on de-facto-standardi Spring-pohjaisten sovellusten turvaamiselle [26]. Spring Security tarjoaa myös valmiit suojaukset useimpiin verkkohyökkäyksiin.

Tietokantana web-palvelulle käytetään pääasiassa MariaDB-tietokantaa [27]. MariaDB on relaatiotietokanta, joka pohjautuu MySQL-tietokantaan.

4.2 Arkkitehtuuri ja tekniset ratkaisut

Tässä alaluvussa käsitellään todentamis- ja valtuuttamiskomponentin arkkitehtuuria sekä teknisia ratkaisuja. Luvussa pohditaan tapoja täyttää komponentin vaatimukset ja vertaillaan eri teknologioiden valintojen seuraksia.

4.2.1 Spring Security

Ensimmäinen kysymys oli päättää, miten todentamis- ja valtuuttamistoiminnallisuus toteutetaan. Päävaihtoehtoina olivat Spring Security -lisäosan hyödyntäminen tai toiminnallisuuden kirjoittaminen itse käyttäen matalan tason toiminnallisuutta. Toiminnallisuuden kirjoittaminen itse ei tuntunut järkevältä vaihtoehdolta, sillä pienikin virhe koodissa saattaisi aiheuttaa suuria tietoturva-aukkoja, puhumattakaan siitä, että toiminnallisuuden kirjoittamiseen

olisi kulunut huomattava määrä aikaa. Täten valinta näiden välillä oli helppo, ja Spring Security liitettiin web-palveluun.

4.2.2 Käyttäjänhallinta

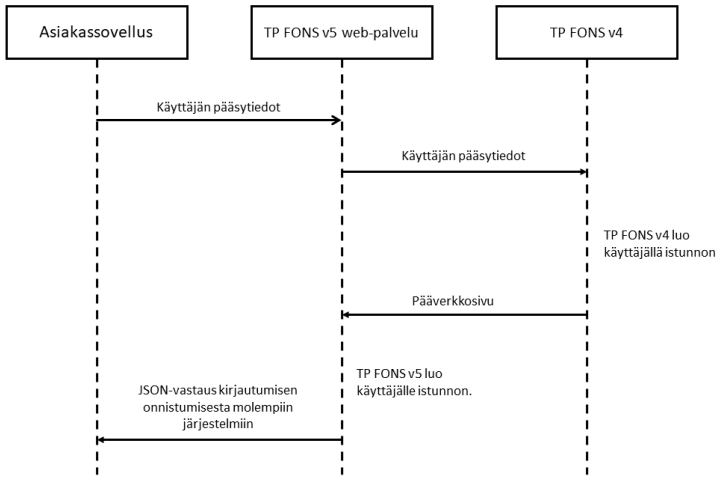
Käyttäjän pääsytietojen hallintaan oli kaksi selvää tapaa. TP FONS v5 ja TP FONS v4 -järjestelmät voisivat joko jakaa pääsytiedot tai pitää ne erillisinä. Jos järjestelmät jakaisivat pääsytietonsa, mahdollistaisi se melko helpon tavan toteuttaa yhteinen kirjautuminen. Se olisi myös kehityksen kannalta vikkelämpää, mutta toisaalta se rajoittaisi uusien toiminnallisuuksien lisäämistä uuteen TP FONS v5 -järjestelmään, koska uudet toiminnallisuudet vaatisivat muutoksia myös TP FONS v4 -järjestelmän puolella. Esimerkkinä tällaisestä toiminnallisuudesta olisi salasanojen tiivistefunktion algoritmin vaihto. Pääsytietojen pitäminen erillisenä ratkaisisi tämän.

Lopulta päätimme jakaa käyttäjätiedot yhteisen kirjautumisen vuoksi ja koska ennakoidut muutokset TP FONS v4 -järjestelmään pystyttäisiin toteuttamaan nopeasti pienellä työmäärällä.

4.2.3 Yhteinen kirjautuminen

Koska käyttäjänhallinnassa päädyttiin jakamaan tunnukset TP FONS v4- ja TP FONS v5 -järjestelmien kesken, on yhteisen kirjautumisen toteuttamiseen helppo keino. TP FONS v5 -asiakassovellus lähettäisi normaaliin tapaan käyttäjätunnuksen ja salasanan TP FONS v5 -web-palvelulle. Pyynnön saapuessa web-palvelu lähettäisi uudelleen käyttäjätunnuksen ja salasanan TP FONS v4 -järjestelmään. TP FONS v4 -järjestelmässä ei kuitenkaan ole modernia web-rajapintaa, joten järjestelmä reagoi palauttamalla sovelluksen pääverkkosivun. Itse sivulla web-palvelu ei tekisi mitään, vaan se poimisi verkkosivun mukana tulleesta HTTP-vastauksen evästeotsakkeesta istuntotunnisteen ja lisäisi sen asiakassovellukselle lähetettävään HTTP-vastaukseen evästeenä TP FONS v5 -järjestelmän todentamistiedon ohelle.

Näin asiakassovellukselle tallentuisi TP FONS v4 -istuntotunniste sekä TP FONS v5 -järjestelmän todentamistieto. Kuva 1 esittää onnistunutta kirjautumisyritystä.



Kuva 1. Sekvenssikaavio onnistuneesta kirjautumisyrityksestä.

Samojen pääsytietojen lisäksi tämä vaatii toimiakseen, että TP FONS v4 ja TP FONS v5 ovat saman verkkotunnuksen alla, sillä useimmat selaimet eivät muuten suostu tallentamaan evästettä tietoturvasyistä johtuen. TP FONS v5 -web-palvelun kannalta tämä ei ollut ongelma.

4.2.4 Todentaminen

Todentamisessa liikuttiin paljon evästepohjaisten istuntotunnisteiden ja JWT-polettien välillä. Myös OAuth 2.0 -protokollaa harkittiin sen vuoksi, että ratkaisu mahdollistaisi TP FONS v5 -web-palveluiden eriyttämisen myöhemmin omiin palveluihinsa, mutta tästä luovuttiin sen monimutkaisuuden vuoksi.

Lopulta päädyimme käyttämään todentamisessa evästepohjaisia istuntotunnisteita. JWT-polettien suurinta etua, skaalautuvuutta, ei tämänkokoisessa projektissa tarvittu ja palvelinpuolella haluttiin säilyttää mahdollisuus mitätöidä käyttäjän istunto. Vaakaa painoi myös Spring Security -

lisäosan valmis istunnonhallintajärjestelmä, joka pohjautui evästepohjaisiin istuntotunnisteisiin.

Koska todentamiseen käytetään evästepohjaisia istuntotunnisteita, tultiin tulokseen, että lisäämme CSRF-suojauksen tilattomalla kaksoislähetystekniikalla. Tekniikka toimii siten, että resurssien tilaa muuttavat pyynnöt (POST, PUT, DELETE yms.) vaativat oman otsakkeensa, joka sisältää CSRF-poletin. Uusi poletti palautetaan jokaisen pyynnön yhteydessä evästeeseen, jos polettievästettä ei lähetetty palvelimelle pyynnön mukana. Tilaa muuttavien pyyntöjen yhteydessä palvelin vertaa evästepolettia sekä otsakepolettia. Jos ne ovat samat, pyyntö hyväksytään. Polettievästeeseen ei lisätä httpOnly-ominaisuutta, koska evästeen on oltava luettavissa selaimelta, jotta se pystyy lisäämään ne tuleviin pyyntöihin. Tekniikan tietoturva on selitetty teorialuvussa.

4.2.5 Valtuuttaminen

Valtuuttaminen oli hankalin osa todentamis- ja valtuuttamiskomponenttia. Miten valtuuttaa yksittäisiä kohdeolioita? Ennen sitä paneudutaan aluksi helpompaan HTTP-pyyntöjen valtuuttamiseen. Spring Security tarjoaa tähänkin työkalut auktoriteettien (engl. authority) avulla. Kaikessa yksinkertaisuudessaan auktoriteetti on Spring Security -kontekstissa merkkijono. (Se voi olla myös muu olio [26, luku 11.1.1], mutta yksinkertaisuuden vuoksi pitäydymme merkkijonoissa.) Nämä auktoriteetit voidaan sitten liittää käyttäjien istuntoihin sekä eri metodeihin ja toiminnallisuuteen, esimerkiksi tässä tapauksessa HTTP-pyyntöihin. Tällä tavalla voimme valtuuttaa eri käyttäjille eri HTTP-pyyntöjen käytön.

Spring Security ei kuitenkaan sen kummemmin tarjoa valmista ratkaisua auktoriteettien hallintaan, joten tämä täytyy itse toteuttaa. Toteutusidea on, että roolipohjaisen RBAC-mallin mukaisesti jaottelemme käyttäjät rooleihin. Jokaiselle roolille annetaan yksi tai useampi auktoriteetti. Todentamisen

yhteydessä käyttäjän istuntoon lisätään käyttäjän roolin auktoriteetit. Auktoriteetit on jaoteltu kohdeluokkien ja toiminnallisuuden mukaan. Esimerkiksi PERSON-READ-auktoriteetti valtuuttaisi järjestelmässä olevien henkilökontaktien hakemisen, PERSON-CREATE-auktoriteetti taas henkilökontaktien luomisen. Tämä helpottaa roolien ylläpitämistä, koska auktoriteetit on jaoteltu yleisen toiminnallisuuden mukaan pieniin osiin.

Kohdeolioiden valtuuttamiseksi ajateltiin aluksi tietokantaan lisättävää saraketta, joka lisättäisiin jokaiseen tauluun, josta oli määrä tehdä valtuutettava. Ensimmäinen idea sarakkeelle oli valtuutustasoarvo. Rooleille lisättäisiin myös valtuutustaso, ja kun käyttäjä hakisi kohdeolioita, käyttäjän roolin valtuutustaso lisättäisiin SQL-kyselyyn siten, että tietokanta palauttaisi kaikki rivit, joiden valtuutustaso on yhtä suuri tai pienempi kuin käyttäjän roolin valtuutustaso.

Tämä olisi perin näppärä ja tehokas tapa kohdeolioiden valtuuttamiselle. Se ei kuitenkaan riitä, koska se rajoittaa valtuutustasojen kohdeolioiden joukot olemaan osajoukkoja ylemmän valtuutustason kohdeolioiden joukosta. Kyse olisi siis eräänlaisesta tasohierarkiasta. Esimerkkinä tällaisestä on tilanne, jossa käyttäjä haluaa luoda kaksi erillistä vapaaehtoinen-roolia, jotka näkevät eri kohdeolioita. Valtuutustasoilla tätä ei voi toteuttaa.

Seuraavaksi vaihtoehto oli edellä mainitun valtuutustason laajentamista. Nyt valtuutustasosarake ei olisi enää pelkkä arvo, vaan viite toiseen tauluun, joka loisi yksi-moneen-suhteen valtuutusolioiden ja roolien välille. Roolille voisi siis valtuuttaa useita kohdeolioita. Tämä on kuitenkin vain hivenen parempi kuin edellinen järjestelmä, koska se ei ratkaise pääongelmaa: järjestelmä on yhä liian rajoittava. Tarvittiin monta-moneen-suhde kohdeolioiden ja roolien välille. Tässä kohtaa esiin astui luvussa kolme esitelty valtuutuslista (ACL).

Yksinkertaisimmillaan valtuutuslistan voisi toteuttaa kahdella tietokantataululla. Toinen esittäisi kohdeolion, jossa olisi tietokantataulun nimi (esim. henkilö) ja tietueen tunniste. Näillä tiedoilla kohdeolion löytäisi tietokannasta. Toisessa

taulussa olisi roolien valtuudet kohdeolioihin. Tästä taulusta valtuutusjärjestelmä voisi tarkistaa, onko kyseisellä käyttäjällä valtuudet tiettyihin kohdeolioihin.

Valtuutuslistan toteuttamista itse harkittiin, mutta Spring Security tarjoaa tähän valmiin työkalun: Spring Security ACL -komponentin. Siinä on kahden sijaan neljä taulua, ja enemmän toiminnallisuutta kuin edellä mainitussa yksinkertaisessa valtuutuslistassa, kuten kohdeolion omistajuus sekä tuloksien varastointi välimuistiin, joka parantaa tehokkuutta. Mutta Spring Security ACL -komponentin opettelu vaatisi aikaa, ja osa siitä menisi hukkaan, koska kaikkea toiminnallisuutta ei tultaisi tarvitsemaan. Toisaalta pyörän kehittäminen uudestaankaan ei tuntunut järkevältä vaihtoehdolta, ja lisätoiminnallisuudet saattaisivat koitua hyödyksi tulevaisuudessa. Näistä syistä päätimme käyttöönottaa Spring Security ACL -komponentin kohdeolioiden valtuuttamista varten. Todentamis- ja valtuuttamiskomponentti tulisi siis käyttämään RBAC- ja ACL-hybridimallia.

Spring Security ACL -komponentissa oli kuitenkin ongelma, joka ilmenee, kun TP FONS v4 -järjestelmän käyttäjä lisäisi kohdeolion, esimerkiksi henkilökontaktin, tietokantaan. Tällöin valtuutuslista, joka on TP FONS v5 -toiminto, jäisi päivittämättä. Tämä vaatisi muutoksia TP FONS v4 -järjestelmään, tai olisi tarpeen tehdä jonkinlainen ajastettu toiminto, joka tarkastaa TP FONS v4 -järjestelmän tekemät muutokset, ja päivittäisi valtuutuslistaa asianmukaisesti. Ensimmäisessä ratkaisumallissa täytyisi kehittää TP FONS v4 -järjestelmää, josta on tarkoitus päästä eroon. Toinen ratkaisumalli tuntui hankalalta toteutettavalta. Kuinka usein toiminto tarkastaisi kohdeoliot, että synkronointi ei pettäisi? Niiden sijaan päätettiin muokata valtuutuslistan valmista toiminnallisuutta tarpeitamme vastaavaksi.

Muutos, johon päädyttiin, oli luoda jokerikohdeolio. Tämä kohdeolio esittäisi mitä tahansa saman tyypin kohdeoliota. Mikäli TP FONS v5 -käyttäjällä olisi valtuudet jokeriolioon, niin hänellä olisivat oikeudet kaikkiin kohdeolioihin, joita ei erikseen ole kielletty. Tällöin jos TP FONS v4 -käyttäjä lisäisi kohdeolion ja

valtuutuslista ei päivittyisi, TP FONS v5 -käyttäjä pystyisi yhä pääsemään käsiksi kaikkiin kohdeolioihin.

Spring Security ACL -komponentti on tehty muokattavaksi, joten yllä mainitut muutokset olivat mahdollisia.

4.3 Toteutus

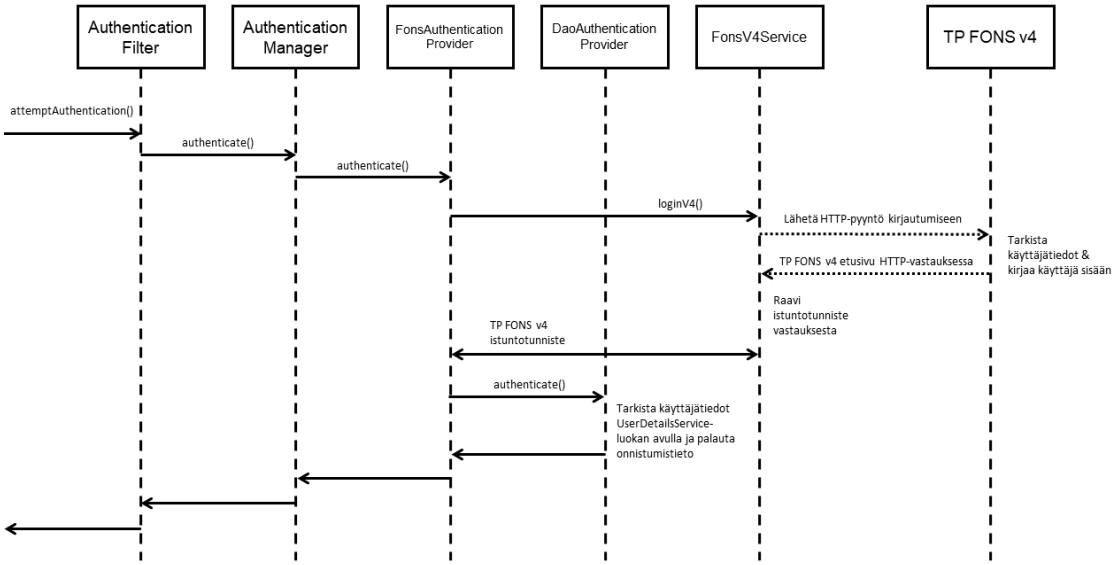
Tässä aliluvussa käydään läpi toteutus- ja valtuuttamiskomponentin toteutusta. Toteutus hyödyntää Spring Security -lisäosaa, ja tästä aliluvusta onkin eniten hyötyä lukijalle, jolla on peruskokemusta Spring Security -lisäosasta. Aliluvun tarkoitus ei ole opettaa Spring Security -lisäosaa vaan toimia nk. reseptikirjana tietyille ongelmille. Koodiesimerkit tässä aliluvussa ovat pelkistettyjä ja näyttävät vain olennaisen osan koodista.

4.3.1 Todentaminen

Todentamis- ja valtuuttamiskomponentin toteutus lähti liikkeelle tietoturvakonfiguraatiosta. Spring Security -lisäosassa se tapahtuu perimällä `WebSecurityConfigurerAdapter`-luokkaa ja ylikirjoittamalla `configure`-metodin. Konfigurointi on Java-pohjaista, ja se sisältää mm. CORS-asetukset, valtuutusmäärittelyt ja todentamisessa käytetyt toteutukset.

Kuvassa 2 näkyy todentamisen sekvenssikaavio. Kaaviossa näkyy todentamiseen liittyvät luokat ja onnistuneesta sisäänkirjautumisyrityksestä aiheutuva toiminta. Oikealla puolella katkoviivalla merkityt nuolet merkitsevät

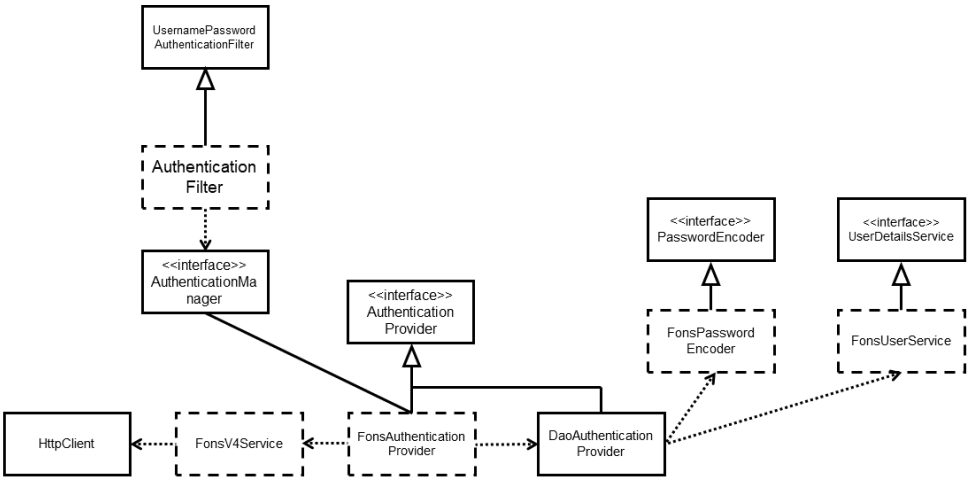
kommunikointia HTTP-protokollan yli TP FONS v4 -järjestelmään. Tavalliset nuolet merkitsevät kommunikointia Javan sisällä.



Kuva 2. Sekvenssikaavio todentamislogiikasta. Kaaviossa on kuvattu onnistunut sisäänkirjautumisyritys. Kaaviota on yksinkertaistettu.

Kuva 3 esittää UML-kaaviota todentamiseen liittyvistä luokista ja rajapinnoista. Tulevat kappaleet pureutuvat todentamisen toiminnallisuuteen yksityiskohtaisemmin. Kuvat eivät sisällä kaikki luokkia, vaan ne antavat

peruskäsityksen toiminnallisuudesta.



Kuva 3. UML-kaavio todentamiseen liittyvistä luokista ja rajapinnoista sekä niiden välisistä suhteista. Kaaviota on yksinkertaistettu. Katkoviivalla ympäröidyt luokat ovat itse toteutettuja.

Spring Security -lisäosan oletusasetukset ovat pitkälti tarkoitettu perinteiselle web-sovellukselle. Modernia web-palvelua varten täytyi tehdä muutoksia. Ensimmäinen asia oli muuttaa kirjautuminen tapahtumaan pyynnön JSON-sisällön avulla. Sitä varten luotiin uusi `AuthenticationFilter`-luokka, joka periytyy Spring Security -lisäosan `UsernamePasswordAuthenticationFilter`-luokasta. Uudessa luokassa ylikirjoitettiin `attemptAuthentication`-metodi. Nimensä mukaisesti sen tarkoitus on käynnistää todentamisprosessi. Se saa parametrikseen matalan tason `HttpServletRequest`-olion, joka kuvaa HTTP-pyyntöä. `attemptAuthentication`-metodi lukee pyynnön JSON-sisällön ja muuttaa sen Java-olioksi Jackson-kirjastolla. Esimerkkikoodi 2 esittää tämän.

```
public Authentication attemptAuthentication(
     HttpServletRequest request, HttpServletResponse response)
     LoginDTO creds;
     try {
          creds = new ObjectMapper()
               .readValue(request.getInputStream(), LoginDTO.class);
     } catch (IOException e) {
          //poikkeuden käsittely
     }

     return authenticationManager.authenticate(
          new UsernamePasswordAuthenticationToken(
               creds.getUsername(),
               creds.getPassword(),
               new ArrayList<>()));
}
```

Esimerkkikoodi 2. HttpServletRequest-olion JSON-sisällön muuttaminen Java-olioksi. LoginDTO-luokan tarkoitus on olla JSON-oliota vastaava Java-luokka. Se sisältää ainoastaan käyttäjätunnus- ja salasana-attribuutit.

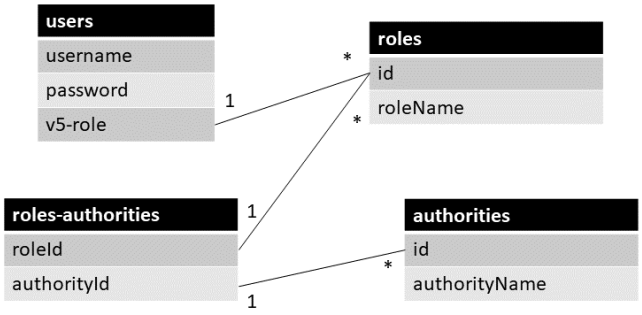
Seuraavaksi metodi kutsuu `authenticationManager`-olion `authenticate`-metodia. `authenticationManager` kutsuu sitten `AuthenticationProvider`-rajapintaa toteuttavaa luokkaa. Tätä varten tehtiin uusi `FonsAuthenticationProvider`-luokka, joka toteuttaa em. rajapintaa. Luokka sisältää `FonsV4Service`-attribuutin, joka hoitaa HTTP-pyynnön tekemisen TP FONS v4 -järjestelmään sekä toisen `AuthenticationProvider`-rajapintaa toteuttavan attribuutin, toteutus toisessa `AuthenticationProvider`-attribuutissa on Spring Security -lisäosan valmis `DaoAuthenticationProvider`-luokka. `FonsAuthenticationProvider`-luokan idea on, että todentamisvaiheessa se pyyttää `FonsV4Service`-attribuuttia kirjautumaan sisään TP FONS v4 -järjestelmään. `FonsV4Service` tekee tämän `HttpClient`-oliota käyttäen, joka lähettää käyttäjätunnuksen ja salasanan TP FONS v4 -järjestelmään, poimii vastauksena tulleista otsakkeista JSESSIONID-istuntotunnisteen ja palauttaa sen. `FonsAuthenticationProvider` tallentaa saadun istuntotunnistemerkkijonon Spring Security -lisäosan `Authentication`-olioon ja pyytää `DaoAuthenticationProvider`-attribuuttia jatkamaan todentamisprosessia. `Authentication`-olio jatkaa mukana. `DaoAuthenticationProvider` kutsuu `UserDetailsService`- ja

`PasswordEncoder`-rajapintojen toteutuksia, jotka hoitavat käyttäjän lähettämän käyttäjätunnuksen ja salasanan vertaamista tietokannassa oleviin tietoihin. `UserDetailsService`-toteutus todentamis- ja valtuuttamiskomponentin tapauksessa hoitaa myös auktoriteettien lisäämisen `Authentication`-olioon.

Todentamisen onnistuessa Spring Security päätyy kutsumaan itse toteutetun `FonsAuthenticationSuccessHandler`-luokan `onAuthenticationSuccess`-metodia. Tämä luokka periytyy `SimpleUrlAuthenticationSuccessHandler`-luokasta. Metodissa lisätään HTTP-vastaukseen asianmukainen JSON-sisältö sekä Set-Cookie-otsake TP FONS v4 -istuntotunnisteelle. TP FONS v5 -istuntotunniste lisätään automaattisesti Spring Security -lisäosan toimesta.

4.3.2 HTTP-pyyntöjen valtuutus

Seuraavaksi läpikäydään auktoriteettien ja roolien hallinta. Tätä varten tehtiin uusi tietokantataulu `authorities`, jonka attribuutteina ovat tunniste ja auktoriteetin nimi. Auktoriteetin nimi olisi luvun 4.2 mukaisesti esimerkiksi PERSON-READ tai PERSON-CREATE. Toinen luotava taulu on nimeltään `roles`. Taulussa on tunnisteen lisäksi roolin nimi. TP FONS v4 -järjestelmän `users`-tauluun luotiin uusi attribuutti, nimeltä `v5-role`, joka viittaa `roles`-tauluun. Näin saatiin yksi-moneen-suhteen käyttäjien ja roolien välille. Enää tarvittiin taulu, jolla yhdistää auktoriteetit ja roolit. Siksi luotiin uusi taulu, `roles-authorities`, jossa on kaksi tunnistetta: toinen viittaa roolitunnisteeseen ja toinen auktoriteettitunnisteeseen. Nyt roolien ja auktoriteettien välillä vallitsee monta-moneen-suhde. Kuva 4 esittää tietokantatauluja ja niiden välisiä suhteita.



Kuva 4. HTTP-pyyntöjen valtuuttamiseen liittyvät tietokantataulujen väliset relaatiot esitettynä. Tietokantataulut ovat yksinkertaistettuja.

Nyt TP FONS v5 -järjestelmässä käyttäjällä on mahdollista olla vain yksi rooli yksi-moneen-suhteen takia. Tämän suhteen pystyisi tulevaisuudessa muuttamaan monta-moneen-suhteeksi lisäämällä uuden taulun roolien ja käyttäjien välille.

Uudet tietokantataulut mahdollistavat HTTP-pyyntöjen valtuuttamisen tietyille auktoriteeteille. Tämä tapahtui luvun alussa esitellyn `WebSecurityConfigurerAdapter`-luokan `configure`-metodissa. Auktoriteetit kovakoodattiin `authorities`-taulun auktoriteettinimien mukaisesti asianmukaisiin HTTP-pyyntöihin. Esimerkkikoodissa 3 on ote `configure`-metodista. Siinä valtuutetaan superkäyttäjille ja PERSON-READ-auktoriteetin omaaville käyttäjille lukuoikeudet henkilöihin. Esimerkkikoodi 3 esittää /persons-reitin GET-pyyntöjen valtuuttamista superkäyttäjille sekä PERSON-READ-auktoriteetin omaaville käyttäjille.

```
.antMatchers(HttpMethod.GET, "/persons/**")
     .hasAnyAuthority(SecurityUtils.SUPERUSER, "PERSON-READ")
```

Esimerkkikoodi 3. `WebSecurityConfigurerAdapter`-luokan `configure`-metodi. Valtuutamme superkäyttäjille ja käyttäjille, joilla on PERSON-READ-auktoriteetti, henkilöiden lukuoikeudet.

Edellisessä aliluvussa kuvattu CSRF-suojauksen lisääminen oli yksinkertaista. Se tapahtui lisäämällä `WebSecurityConfigurerAdapter`-luokan `configure`-metodiin koodiesimerkissä 4 esitetyn koodipalasen. `CookieCsrfTokenRepository`-luokka kuvaa tilattoman kaksoislähetystekniikan CSRF-suojaukselle. Luokka on Spring Security -lisäosan mukana tuleva.

```
http.csrf().csrfTokenRepository(
     CookieCsrfTokenRepository.withHttpOnlyFalse());
```

Esimerkkikoodi 4. `WebSecurityConfigurerAdapter`-luokan `configure`-metodiin voi lisätä koodiesimerkin mukaisen koodipalasen CSRF-suojauksen lisäämiseksi. `csrfTokenRepository`-metodin argumentti määrittelee suojaustekniikan. Tässä se on `CookieCsrfTokenRepository`, eli tilaton kaksoislähetys.

4.3.3 ACL-valtuutuslista

ACL-valtuutuslistaa varten täytyi ensin lisätä kolme riippuvuutta `pom.xml`-tiedostoon. Riippuvuudet ovat muodossa groupId, artifactId.

- org.springframework.security, spring-security-acl
- net.sf.ehcache, ehcache-core
- org.springframework, spring-context-support.

Ensimmäinen on ACL-komponentti. Toinen on Ehcache-välimuisti, jota ACL-valtuutuslista tarvitsee toimiakseen. Spring Security -lisäosan ACL-valtuutuslista

hyödyntää välimuistia parantaakseen tehokkuutta. Viimeinen listalla on tukikomponentti Ehcachea käyttäville Spring-sovelluksille.

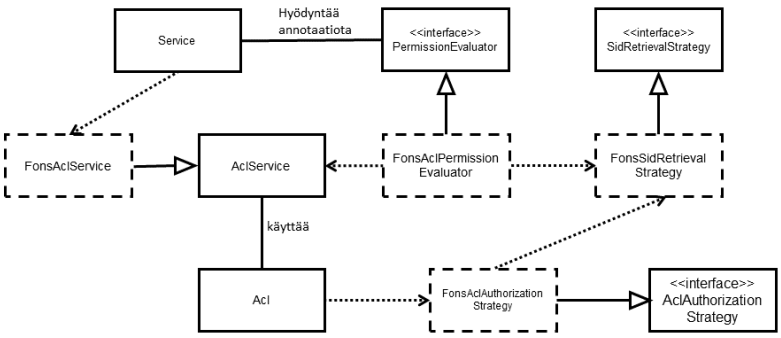
Seuraavaksi ACL-valtuutuslistaan täytyi lisätä tarvittavat tietokantataulut. Spring Security Reference -dokumentistä löytyy luontikoodit useimmille tietokantajärjestelmille [26]. ACL-valtuutuslistaan kuuluu neljä taulua:

- `acl_class`-taulu joka kuvaa kohdeolion luokan Java-JPA-entiteetin.
- `acl_sid`-taulu joka kuvaa valtuutettavan käyttäjän tai roolin.
- `acl_object_identity`-taulu joka kuvaa kohdeolion. Taulussa on viite `acl_class`-tauluun sekä tunnisteattribuutti.
- `acl_entry`-taulu joka kuvaa käyttäjän tai roolin valtuuksia tiettyyn kohdeolioon. Siinä on viite `acl_sid`-tauluun sekä `acl_object_identity`-tauluun.

Jokerikohdeoliolle varattiin oma tunniste, joka ei ole eikä tule olemaan minkään muun tietokantatietueen tunniste. Tätä tunnistetta kutsutaan jokeritunnisteeksi.

Tietokantataulujen luonnin jälkeen ACL-valtuutuslistan toteutus alkoi konfiguroinnista. Sitä varten luotiin uusi `MethodSecurityConfiguration`-luokka, joka periytyy `GlobalMethodSecurityConfiguration`-luokasta. Luokkaan lisättiin `@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)` annotaatio, joka mahdollistaa hyödyllisten annotaatioiden käytön kohdeolioiden valtuuttamisessa. Tässä luokassa konfiguroidaan ACL-valtuutuslistan keskeiset luokat ja Ehcache-välimuisti. Suurin osa on oletustoteutuksia, mutta muutama toteutus tehtiin itse. Kuva 5 esittää yksinkertaistettua UML-luokkakaaviota, josta näkyvät oleellisimmat luokat ja rajapinnat. Vasemman yläkulman Service-luokka sisältää liiketoimintagiikkaa liittyen kohdeolioihin. Niitä on järjestelmässä useita, kuten

`PersonService`, joka hoitaa henkilöiden liiketoimintatoiminnallisuuden, sekä `OrganizationService`, joka hoitaa organisaatioiden liiketoimintatoiminnallisuuden.



Kuva 5. UML-kaavio ACL-valtuutuslistan keskeisistä luokista ja rajapinnoista, sekä niiden välisistä suhteista. Kaaviota on yksinkertaistettu. Katkoviivalla ympäröidyt luokat ovat itse toteutettuja.

Aluksi luotiin `FonsAclService`-luokka, joka periytyy `JdbcMutableAclService`-luokasta. `JdbcMutableAclService`-luokka tarjoaa toiminnallisuutta käydä käsiksi ACL-tietokantatauluihin. `FonsAclService`-luokan tarkoitus on abstraktoida tätä toiminnallisuutta, jotta liiketoimintalogiikka voisi helposti hyödyntää sitä. Luokan toinen tarkoitus on ottaa huomioon jokerikohdeoliot, jotka esiteltiin edellisessä aliluvussa.

Toinen luokka, joka toteutettiin itse, oli `FonsAclPermissionEvaluator`, joka toteuttaa `PermissionEvaluator`-rajapinnan. Tämä on pitkälti kopioitu valmiista `AclPermissionEvaluator`-luokasta, mutta mukaan lisättiin logiikka jokerikohdeolioita varten. `FonsAclPermissionEvaluator`-luokan

`checkPermission`-metodi tarkistaa valtuudet tiettyyn käsiteltävään kohdeolioon. Metodin algoritmi on esitetty esimerkkikoodissa 5.

```
Tarkista acl_entry-taulu ja etsi tietue käyttäjän roolin tunnisteella
sekä kohdeolion tunnisteella.

Jos tietue löytyi:
     Valtuuta / Evää pääsy kohdeolioon granting-attribuutin mukaan.

Jos tietuetta ei löytynyt:
     Tarkista acl_entry-taulu ja etsi tietue käyttäjän roolin
     tunnisteella sekä jokeritunnisteella.

     Jos tietue löytyi:
          Valtuuta / Evää pääsy kohdeolioon granting-attribuutin
          mukaan.

     Jos tietuetta ei löytynyt:
          Evää pääsy kohdeolioon.
```

Esimerkkikoodi 5. FonsAclPermissionEvaluator-luokan checkPermission-metodin algoritmin logiikka selitettynä.

Viimeiseksi lisättiin luokka `FonsSidRetrievalStrategyImpl`, joka toteuttaa `SidRetrievalStrategy`-rajapintaa. Luokan tarkoitus on tarjota `FonsAclPermissionEvaluator`-luokalle auktoriteetit ja käyttäjät, joiden valtuutukset tulisi tarkistaa. `FonsSidRetrievalStrategyImpl`-luokka eroaa oletustoteutuksesta siten, että se tarjoaa ainoastaan käyttäjien roolin. Tämä johtaa siihen, että käyttäjän auktoriteeteistä tarkastetaan ainoastaan rooli.

Enää toteutettavaksi jäi `FonsAclAuthorizationStrategy`-luokka, joka toteuttaa `AclAuthorizationStrategy`-rajapinnan. `FonsAclAuthorizationStrategy` sisältää valtuuttamistoiminnallisuuden liittyen ACL-valtuutuslistan hallinnallisiin toimintoihin kuten kohdeolioiden muokkaamiseen ja lisäämiseen. Oletustoteutus tarkistaa

1. onko käyttäjä superkäyttäjä

2. omistaako käyttäjä kyseisen kohdeolion

3. onko käyttäjällä ACL-valtuutuslistassa valtuutus kyseiselle kohdeoliolle.

`FonsAclAuthorizationStrategy`-luokka muuttaa kohdan kaksi edellisestä listasta. Käyttäjän sijaan luokan toteutus tarkistaa, omistaako käyttäjän rooli kyseisen kohdeolion. `FonsAclService`-luokassa toteutetun toiminnallisuuden rinnalla tämä takaa, että Spring Security ACL-valtuutuslistan kohdeolioiden omistaja voi ainoastaan olla rooli, ei käyttäjä. Tämä yksinkertaistaa toiminnallisuutta, koska käyttäjien omistajuustoiminnallisuutta ei todentamis- ja valtuuttamiskomponentissa tarvita.

Seuraavaksi tarkastellaan, miten liiketoimintalogiikan yhteyteen lisättiin ACL-valtuutuksen hallintaan liittyvä toiminnallisuus.

Käytetään esimerkkinä web-palvelun toiminnallisuutta kontaktien hallintaan. Kontakti on tässä kohdeolio. Kontakteja voi lukea, luoda ja päivittää eri HTTP-pyyntöjen avulla. Spring-sovelluksilla on tyypillisesti kontrollereita (engl. controller), jotka hoitaa HTTP-pyyntöjen vastaanottamisen. Kontrollerit kutsuvat Service-luokkia, jotka keskustelevat repository-luokkien kanssa, jotka hoitavat tietokantayhteydet ja -haut. ACL-valtuutuksen lisääminen kontaktien lukumetodiin tapahtui lisäämällä metodin ylle `@PostFilter("hasPermission(filterObject, 'ADMINISTRATION')")`-annotaatio. `PostFilter`-annotaatio ottaa kohdeolion tai joukon kohdeolioita ja tarkistaa käyttäjän roolin valtuudet niihin. Annotaatiossa esiintyvä `hasPermission`-metodi viittaa toteutetun `FonsAclPermissionEvaluator`-luokan toiminnallisuuteen. Argumentteinä ovat `filterObject`, joka on kyseessä oleva kohdeolio sekä merkkijono `'ADMINISTRATION'`. Spring Security ACL-valtuutuslistassa olisi mahdollisuus valtuuttaa tietylle käyttäjälle tiettyyn kohdeolioon tiettyjä toimintoja. Tämä mahdollistaisi vielä yksityiskohtaisemman valtuutustoiminnallisuuden, mutta todentamis- ja valtuuttamiskomponentissa sitä ei tarvittu, minkä vuoksi argumentiksi asetettiin aina `'ADMINISTRATION'`-merkkijono. Nyt

valtuuttamattomat kohdeoliot karsiutuvat pois ennen kuin ne palautuvat asiakassovellukselle.

Myös Service-luokan muut toiminnallisuudet toimivat samalla tavalla. Spring Security tarjoaa `PostFilter`-annotaation lisäksi myös `PreFilter`, `PostAuthorize` ja `PreAuthorize`-annotaatiot, joista viimeisellä voidaan evätä pääsy ennen kuin metodi toteutetaan, mikä on hyödyllistä mm. kontaktin päivityksessä.

```
Tarkista, onko käyttäjän roolilla valtuudet kontaktien
jokerikohdeolioon.

Jos valtuudet löytyvät:
     Jätä ACL-valtuutuslista ennalleen. (= Käyttäjän roolilla on jo
     valtuudet luotuun kohdeolioon jokerikohdeolion takia)

Jos valtuuksia ei löydy:
     Luo uusi tietue acl_object_identity-tauluun.
     Luo uusi tietue acl_entry tauluun, jossa käyttäjän roolille
     valtuutetaan pääsy luotuun kontaktiin.
```

Esimerkkikoodi 6. Uuden kohdeolion päivitysalgoritmi selitettynä.

Uuden kontaktin luonti eroaa hieman edellä mainituista tekniikoista. Koska kyseessä on uusi kontakti, ei siihen voida soveltaa annotaatioita. Ensiksi tarkistetaan, onko käyttäjän roolilla auktoriteetti kontaktien luomiseen. Jos on, niin kontakti luodaan ja samalla päivitetään ACL-valtuutuslistaa. Päivitysalgoritmi on esitetty esimerkkikoodissa 6.

4.3.4 Valtuutuksien hallinta

Kokonaan kovakoodatut valtuutusmäärittelyt eivät olisi käyttäjille kovinkaan mielekkäitä, joten todentamis- ja valtuuttamiskomponentti tarvitsi työkalut valtuutuksien hallintaan. Tässä aliluvussa käydään läpi valtuutuksien hallintaan liittyvä toiminnallisuus kevyesti. Valtuutuksien hallintaan sisältyy TP FONS v4 -käyttäjien aktivointi TP FONS v5 -järjestelmään, roolien määrittely ja käyttäjien roolittaminen sekä ACL-valtuutuslistan hallinta.

Todentamis- ja valtuuttamiskomponentin toteutuksen aikaan ei ollut toivottavaa, että käyttäjiä pystyttäisiin luomaan TP FONS v5 -järjestelmästä. Tarkoitus oli, että käyttäjien luominen tapahtuisi ainoastaan TP FONS v4 -järjestelmästä käsin. Siksi oli aiheellista luoda HTTP-pyyntö käyttäjien aktivoimiselle TP FONS v5 -järjestelmään. Tällöin käyttäjät, joita ei ole aktivoitu superkäyttäjän toimesta, eivät pysty käyttämään TP FONS v5 -järjestelmää.

HTTP-pyyntö aktivoimiselle on yksinkertainen PATCH-pyyntö, jonka sisältönä on käyttäjän ja roolin tunniste. HTTP-pyyntö aktivoi käyttäjän ja roolittaa hänet roolin tunnisteen mukaan. Pyyntö hylätään mikäli kyseessä olevalla käyttäjällä on tietokannassa jo TP FONS v5 -rooli.

Roolien ja auktoriteettien lukemiselle on omat HTTP-pyyntönsä. Pyyntöön on sisällyttävä luotavan roolin nimi, auktoriteetit sekä alustava boolean-tyypinen tieto siitä, valtuutetaanko roolille jokerikohdeolio ACL-valtuutuslistassa. Käyttäjien roolien vaihtamiselle on myös oma pyyntönsa.

ACL-valtuutuslistan hallitsemiseen toteutettiin jokaista kohdeolion tyyppiä varten kaksi pyyntöä. Ensimmäinen pyyntö palauttaa ACL-valtuutusmäärittelyt kohdeolioiden tunnisteita vastaan. ACL-valtuutusmäärittelyt kertovat, mitkä roolit ovat valtuutettuja pyydettyihin kohdeolioihin. Toinen pyyntö määrittää ACL-valtuudet pyynnön mukana tulleen sisällön mukaisesti. Pyynnön sisältö koostuu roolin tunnisteesta, kohdeolion tunnisteesta sekä boolean-tyyppisestä arvosta, joka kertoo, valtuutetaanko kohdeolio kyseiselle roolille vai evätäänkö se.

5 Työn arviointi

Tässä luvussa arvioidaan todentamis- ja valtuutuskomponentin totetutusta teknisestä näkökulmasta.

Todentamis- ja valtuuttamiskomponentti toteutettiin onnistuneesti. Komponenttia testattiin ja toiminnallisuus todettiin toimivaksi. Yhteisen kirjautumisen toimimiseen selaimella liittyi epävarmuutta, mutta testaamalla tätä se todettiin toimivaksi. Kirjautumispyynnön lähettäminen TP FONS v4 -järjestelmään TP FONS v5 -kirjautumisen aikana ei kuitenkaan aina toiminut, vaan joskus pyyntö epäonnistui heti kättelyssä. Tämä liittyy todennäköisesti jollain tavalla `java.net.http`-paketin `HttpClient`-luokan toteutukseen, jota käytettiin HTTP-pyynnön lähettämisessä. Tähän on syytä paneutua tulevaisuudessa. Todentamisessa ja pyyntöjen valtuuttamisessa ei esiintynyt huomattavia ongelmia. Yksityiskohtainen valtuutus saatiin toteutettua Spring Security -lisäosan ACL-valtuutuslistan avulla. ACL-valtuutuslista onnistuttiin myös muokkaamaan paremmin yrityksen tarpeita vastaavaksi, mutta ongelmattomaksi toteutus ei jäänyt.

Suurimmat toteutukseen liittyvät haasteet liittyivät Spring-ympäristöön ja sen oikkuihin pitkälti sen massiivisuuden takia. Suurin murheenkryyni oli Spring Security -lisäosan ACL-valtuutuslista.

Kuten aikaisemmin mainittin, Spring Security -lisäosan ACL-valtuutuslista sisältää hurjan määrän turhaa toiminnallisuutta, eikä dokumentaatiota ja esimerkkejä löydy erityisen paljon. ACL-valtuutuslistan toteuttaminen oli pitkälti kokeilemista, epäonnistumista, lähdekoodin ja dokumentaation lukua sekä kokeilemista uudelleen. ACL-valtuutuslistassa suurimmat ongelmat, joita ei saatu tämän työn aikana ratkaistua, ovat valtuutuslistan tehokkuus sekä liiketoimintalogiikkapuolen sivutuksen (engl. pagination) yhteensopimattomuus. Molemmat asioista liittyvät `PostFilter`-annotaation käyttöön kohdeolioiden lukemisessa. Käydään aluksi läpi sivutusongelma.

Kohdeolioiden luku rajapinnan kautta tapahtuu HTTP-pyynnölla. Normaalisti pyynnön mukana lähetetään sivunumero sekä sivussa olevien kohdeolioiden määrä. Näin web-palvelu pystyy rajoittamaan palautettavien kohdeolioiden määrää, koska kaikkien kohdeolioiden palautus yhdellä kertaa heikentäisi käyttökokemusta mittavasti, sekä se veisi todella paljon muistia ja tehoa niin palvelimelta kuin asiakkaaltakin. Spring Data -lisäosalla on oma luokkansa ja toiminnallisuus sivuihin ja sivuttamiseen. `PostFilter`-annotaatio ei kuitenkaan ole yhteensopiva kyseisen toiminnallisuuden kanssa, ja sille on hyvä syy. `PostFilter` nimittäin ajetaan vasta sen jälkeen, kun kohdeoliot on haettu tietokannasta. Jos sivutus olisi käytössä, tietokannasta haetut kohdeoliot olisivat hetken aikaa sivutettuina kunnolla, mutta kun `PostFilter` suodattaa valtuuttamattomat kohdeoliot pois, koko sivutus rikkoutuu, koska yhtäkkiä sivussa ei olekaan pyydettyä määrä kohdeolioita. Nopea yritys korjata tämä on muuttaa pyynnön parametrejä siten, että sivunumeron ja kohdeolioiden määrän sijaan parametreinä olisivat kohdeolioiden indeksiluku sekä yläraja kohdeolioiden määrälle. Tämä tosin haittaa käyttökokemusta asiakaspuolella, koska enää ei ole takeita, montako kohdeoliota palautuu pyynnön mukana vai palautuuko yhtäkään. Tämä tuo meidät toiseen isoon ongelmaan: tehokkuusongelmaan.

TietoPiirin suurimmilla asiakkailla voi olla yli 100 000 kohdeoliota tietokannassa. Oletetaan, että on olemassa rooli. Kutsutaan sitä VAPAAEHTOINEN-nimellä. Roolilla on pääsy kymmeneen kohdeolioon. VAPAAEHTOINEN-roolin käyttäjä haluaa lukea kaikki kymmenen kohdeoliota, joihin hänen rooli on valtuutettu. Palvelimen täytyy nyt pahimmassa tapauksessa lukea tietokannasta kaikki 100 000 kohdeoliota ja `PostFilter`-annotaation avulla karsia pois kaikki paitsi kymmenen kohdeoliota. Toki edellä mainittu prosessi olisi syytä tehdä osissa, mutta lienee sanomattakin selvää, että kovin toimiva tekniikka tämä ei ole.

Työn aikana tätä kuitenkin testattiin luomalla kanta, jossa on 100 000 kohdeoliota. Postman-sovelluksella luotiin HTTP-pyyntö, jossa palvelinta pyydettiin palauttamaan kaikki 10 valtuutettua kohdeoliota. Ylärajaparametriksi

asetettiin 10 000, eli palvelin käy 10 000 kohdeoliota läpi kerralla. Pyyntö pitäisi siis lähettää 10 kertaa, jotta kaikki valtuutetut kohdeoliot saataisiin varmuudella palautettua. Pyynnössa meni omalla kannettavallani noin 30 sekuntia. 30 sekuntia kertaa 10 tekee 300 sekuntia, eli kymmenen kohdeolion hakemiseen 100 000 kohdeolion tietokannasta menisi tässä tapauksessa noin 5 minuuttia. `PostFilter`-annotaation käyttö kohdeolioiden hakemisessa ei siis ole järkevä vaihtoehto.

`PostFilter`-annotaation sijaan kohdeolioiden haku on suoritettava suoraan tietokannassa oikeanlaisen SQL-kyselyn avulla, tai käyttäen muuta Java-pohjaista työkalua, kuten JPA Criteria API -kirjastoa kyselyiden toteuttamiseen. Näin saadaan korjattua sivutus- sekä tehokkuusongelma yhdellä iskulla. Sivutusongelma ratkeaa, koska nyt palautettua kohdeoliojoukkoa ei enää tarvitse suodattaa Java-puolella. Tehokkuusongelma ratkeaa, koska tällaisten hakujen tekeminen on tietokannalle hyvin helppoa verrattuna kohdeolioiden suodattamiseen Java-puolella.

6 Yhteenveto

Insinöörityön tavoitteena oli suunnitella ja toteuttaa todentamis- ja valtuuttamiskomponentti uuteen TP FONS v5 -toiminnanohjausjärjestelmään ja samalla esitellä modernien web-sovellusten todentamiseen ja valtuuttamiseen liittyviä haasteita ja ratkaisuja.

Todentamis- ja valtuuttamiskomponentista saatiin toteutettua ensimmäinen versio. Kun komponentti yhdistetään muuhun liiketoiminnallisuuteen sekä selaimella toimivaan asiakassovellukseen, yritys pystyy rakentamaan pienimmän toimivan tuotteen (engl. minimum viable product, MVP) järjestelmästä.

Todentamis- ja valtuuttamiskomponentti mahdollistaa alussa esitetyt vaatimukset. Yhteinen kirjautuminen mahdollistaa TP FONS v5 -järjestelmästä siirtymisen TP FONS v4 -järjestelmään vaivattomasti esimerkiksi selainsovellukseen lisätyllä linkillä, kunhan järjestelmät sijaitsevat saman verkkotunnuksen alla. Valtuuttamismahdollisuuksia laajennettiin kohdeoliotasolle HTTP-pyyntötasolta. Tällä hetkellä ainoastaan kontaktikohdeoliot ovat valtuutettavissa, mutta ACL-valtuutuslista mahdollistaa minkä tahansa tietokannassa sijaitsevan kohdeolion valtuuttamisen pienillä lisäyksillä koodiin. Valtuutuksien hallintaan liittyvien pyyntöjen avulla, TP FONS v5 -web-palvelun liiketoimintatoiminnallisuudella sekä selainsovelluksen yhteispelillä on käyttäjän mahdollista muuttaa ACL-valtuutuksia helposti hyödyntäen kohdeolioiden välisiä suhteita. Todentamis- ja valtuuttamiskomponentti onnistuttiin toteuttamaan täysin ilman muutoksia TP FONS v4 -järjestelmään. Todentamis- ja valtuuttamiskomponentilla on vielä kuitenkin paljon muutoksia sekä uutta toiminnallisuutta ja refaktorointia edessä, sillä muutenhan ei voitaisi puhua MVP-henkisyydestä. Uudet toiminnallisuudet saattavat aiheuttaa muutoksia TP FONS v4 -järjestelmään tulevaisuudessa. Esimerkki uudesta toiminnallisuudesta on kahdenkertainen todentaminen, johon tullaan paneutumaan lähitulevaisuudessa.

Modernien web-sovellusten todentamiseen ja valtuuttamiseen on tarjolla paljon uusia teknologioita ja perinteisiä teknologioita, ja valinta niiden välillä saattaa tuntua hyvinkin tuskastuttavalta. Uutukainen teknologia ei välttämättä ole ideaali vaihtoehto omaan käyttötarkoitukseen, vaikka se olisikin uusin trendivillitys ohjelmistokehitysmaailmassa. Tuskastuneisuuden tunnetta saattaa helpottaa, että tällaisissä tilanteissa kaiken kattavaa hopealuotiratkaisua tuskin on olemassa.

Lähteet

1 An overview of HTTP. 2021. Verkkoaineisto. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. Luettu 22.4.2021.

2 Using HTTP cookies. 2021. Verkkoaineisto. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. Luettu 24.4.2021.

3 M. West, M. Goodwin. 2016. Same-Site Cookies. Verkkoaineisto. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site-00.

4 A. Barth. 2011. HTTP State Management Mechanism. Verkkoaineisto. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc6265#section-8.2.

5 OWASP Top 10. 2017. Dokumentti. The OWASP® Foundation.

6 OWASP Top 10. 2010. Dokumentti. The OWASP® Foundation.

7 Johannes Ylipiha. 2015. Request Forgery -haavoittuvuuden paikkaus pitkälle kehitetyssä järjestelmässä. Theseus-tietokanta. http://urn.fi/URN:NBN:fi:amk-201505138007.

8 Matt Evert. 2020. CSRF in the Age of JSON. Verkkoaineisto. https://www.directdefense.com/csrf-in-the-age-of-json/#:~:text=The%20application%2Fjson%20MIME%20type,means%20of%20submitting%20the%20request. Luettu 1.5.2021.

9 Jim Conallen. 1999. Modeling Web Application Architectures with UML. Communications of the ACM.

10 Joonas Teurokoski. 2013. MODERNIT WEB-SOVELLUKSET. Theseus-tietokanta. http://urn.fi/URN:NBN:fi:amk-2013120920435.

11 Mark Masse. 2011. REST API Design Rulebook. O'Reilly.

12 Sailu Reddy, Marjorie Sayer, Anshu Aggarwal, David Gourley, Brian Totty. 2002. HTTP: The Definitive Guide. O'Reilly.

13 Roy Fielding. 2000. Architectural Styles and the Design of Network-based Software Architectures. Väitöskirja. UNIVERSITY OF CALIFORNIA.

14 Cade Cairns, Daniel Somerfield. 2017. The Basics of Web Application Security. Verkkoaineisto. https://martinfowler.com/articles/web-security-basics.html. Luettu 1.4.2021.

15 Asoke Nath, Tanushree Mondal. 2016. Issues and Challenges in Two Factor Authentication Algorithms. St. Xavier’s College. International Journal of Latest Trends in Engineering and Technology.

16 J. Reschke. 2015. The 'Basic' HTTP Authentication Scheme. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc7617#section-2.

17 Sherry Hsu. 2018. Session vs Token Based Authentication. Verkkoaineisto. https://medium.com/@sherryhsu/session-vs-token-based-authentication-11a6c5ac45e4. Luettu 7.4.2021.

18 Mikko Tiitinen. 2020. OpenID Connect ja sen hyödyntämi-nen mikropalveluissa. Theseus-tietokanta. http://urn.fi/URN:NBN:fi:amk-2020121427961.

19 Sebastian Peyrott. 2020. Refresh Tokens: When to Use Them and How They Interact with JWTs. Verkkoaineisto. https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/. Luettu 19.4.2021.

20 D. Hardt. 2012. The OAuth 2.0 Authorization Framework. Internet Engineering Task Force (IETF). Verkkoaineisto. https://tools.ietf.org/html/rfc6749.

21 OWASP API Security Top 10. 2019. Dokumentti. The OWASP® Foundation.

22 Use Access Control List (ACL) to set up permission-based access to data. 2020. Verkkoaineisto. OutSystems. https://success.outsystems.com/Documentation/Best_Practices/Architecture/Use_Access_Control_List_(ACL)_to_set_up_permission-based_access_to_data. Luettu 10.4.2021.

23 Tomas Vondra. 2016. Application users vs. Row Level Security. Verkkoaineisto. https://www.2ndquadrant.com/en/blog/application-users-vs-row-level-security/. Luettu 5.5.2021.

24 Etusivu. Verkkoaineisto. Spring. https://spring.io/. Luettu 9.5.2021.

25 Martin Fowler. 2004. Inversion of Control Containers and the Dependency Injection pattern. Verkkoaineisto. https://martinfowler.com/articles/injection.html. Luettu 5.5.2021.

26 Spring Security Reference. Versio 5.4.6. Spring Security -dokumentti. Spring. https://docs.spring.io/spring-security/site/docs/5.4.6/reference/html5/.

27 Etusivu. Verkkoaineisto. MariaDB. https://mariadb.com/. Luettu 9.5.2021.