jamk.fi

OpenID Connect ja sen hyödyntäminen mikropalveluissa

Mikko Tiitinen

Opinnäytetyö
Joulukuu 2020
Tekniikan ja liikenteen ala
Insinööri (AMK), Ohjelmistotekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Tiitinen, Mikko	Opinnäytetyö, AMK	joulukuu 2020
	Sivumäärä 33	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
OpenID Connect ja sen hyödyntäminen mikropalveluissa

Tutkinto-ohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
Ari Rantala, Esa Salminkangas

Toimeksiantaja(t)
Combitech Oy

Tiivistelmä

Opinnäytetyössä tehtiin selvitys OpenID Connect -protokollan hyödyntämisestä ja implementoinnista mikropalveluympäristössä. Tavoitteena oli saada aikaan esimerkkitoteutus aiheesta, sekä dokumentaatiota toimeksiantajalle, jota mahdollisesti voidaan hyödyntää tulevissa projekteissa.

Toteutuksessa rakennettiin mikropalveluympäristö, johon kuului mikropalveluiden lisäksi yhdyskäytäväpalvelu, joka toimi yksittäisenä tulopisteenä sen takana oleville mikropalveluille sekä rekisteripalvelu, joka pystyi jakamaan tietoja palveluista muille. Ympäristö toteutettiin Java-ohjelmointikielellä, sekä sen ympärille luodulla Spring Framework -sovelluskehyksellä. OpenID Connect -protokolla implementoitiin Keycloak OpenID Connect -tarjoajalla, sekä Spring Security -sovelluskehyksellä. Keycloak tarjosi valmiin implementoinnin käyttäjien autentikoimiselle OpenID Connectin autentikointiprosesseilla, sekä OpenID Connectin identiteettipoletin ja valtuutuspoletin jakamisen asiakassovellukselle. Spring Securityn avulla yhdyskäytäväpalvelussa pystyttiin aloittamaan autentikointiprosessi protokollan mukaan, sekä vastaanottamaan ja jakamaan OpenID Connectin valtuutuspolettia eteenpäin sen takana oleville mikropalveluille. Mikropalveluissa Spring Securityn avulla pystyttiin vastaanottamaan ja validoimaan yhdyskäytäväpalvelun toimittama valtuutuspoletti, sekä purkamaan siitä käyttäjälle määritellyt roolit.

Lopputuloksena saatiin aikaan mikropalveluympäristö, jossa OpenID Connectia hyödynnettiin onnistuneesti käyttäjän autentikoimisessa yhdyskäytäväpalvelussa, sekä mikropalveluiden päätepisteiden pääsynvalvonnassa. Toimeksiantajalle saatiin aikaan esimerkkitoteutus, sekä dokumentaatio aiheesta, jota se voi mahdollisesti hyödyntää tulevissa projekteissa.

Avainsanat (asiasanat)
OpenID Connect, OAuth 2.0, Autentikointi, Valtuutus, Mikropalvelut

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Tiitinen, Mikko	Bachelor's thesis	December 2020
		Language of publication: Finnish
	Number of pages: 33	Permission for web publication: x

Title of publication

OpenID Connect and utilizing it in microservices

Degree programme

Software Engineering

Supervisor(s)

Ari Rantala, Esa Salminkangas

Assigned by

Combitech Oy

Abstract

In the thesis, a study was made about the utilization and implementation of OpenID Connect protocol in a microservice environment. The aim was to provide an example implementation about the topic, and to provide documentation for the client that could possibly be used in future projects.

In the implementation, a microservice environment was built, which included, in addition to microservices, an edge service that acted as a gateway to the microservices behind it, as well as a registry service that could share information about the services to the other services. The environment was implemented in the Java programming language using Spring Framework application framework that was created around it. OpenID Connect protocol was implemented with Keycloak OpenID Connect provider and in the environment with the Spring Security application framework. Keycloak provided a readymade implementation for user authentication with OpenID Connect authentication processes, as well as sharing an OpenID Connect identity token and authorization token with the client. Using Spring Security in the edge service it was possible to start the authentication process according to the protocol, as well as to receive and distribute the OpenID Connect authorization token forward to the microservices behind it. In the microservices it was possible to receive and validate the authorization token provided by the edge as well as to decode the roles defined for the user using Spring Security.

The result of the implementation was a microservice environment in which OpenID connect was successfully utilized in user authentication at the edge service as well as in access control of microservice endpoints. An example implementation was provided to the client as well as documentation about the topic which can be utilized in future projects.

Keywords/tags (subjects)

OpenID Connect, OAuth 2.0, Authentication, Authorization, Microservices

Miscellaneous

Sisältö

Kuviot

Sanasto

Annotaatio

Annotaatio on merkintätapa, jolla pystytään Java koodissa kertomaan esim. luokalle metatietoa.

Autentikointi

Autentikointi tarkoittaa käyttäjän identiteetin varmentamista.

CSRF

CSRF eli Cross-Site Request Forgery. Hyökkäys, jossa hyökkääjä saa uhrin selaimella lähetettyä HTTP-pyynnön hyökkääjän haluamaan kohdepalveluun.

Mikropalveluympäristö

Sovellusympäristö, joka koostuu itsenäisistä palveluista, jotka ovat ajettavissa muista palveluista riippumatta.

Päätepiste

Osoite, johon asiakassovellus ottaa yhteyttä ja joka palauttaa takaisin pyydettyjä resursseja.

Sovelluskehys

Kokoelma valmiiksi luotuja koodikirjastoja, joiden tarkoitus on helpottaa ja nopeuttaa tietyn sovellusosan implementointia.

1 Johdanto

1.1 Toimeksiantaja

Combitech Oy on Saab-konserniin kuuluva pohjoismainen yritys, joka on erikoistunut tekniseen konsultointiin. Combitechillä on yli 1900 työntekijää Ruotsissa, Suomessa, Norjassa ja Tanskassa. Suomessa se työllistää yli 80 työntekijää neljässä eri toimipisteessä Espoossa, Tampereella, Jyväskylässä ja Säkylässä. Suomessa Combitech on erikoistunut kyberturvallisuuden ja puolustuksen kehittyneisiin teknisiin ratkaisuihin ja palveluihin. Asiakkaita pohjoismaissa ovat pääasiassa puolustusvoimat, teollisuus yritykset ja julkinen sektori. (Tietoja meistä N.d.)

1.2 Taustat

Huhtikuussa 2018 opinnäytetyöntekijä rekrytointiin Combitech Oy:lle puoleksi vuodeksi harjoittelijaksi, ja samalla osapuolet myös sopivat, että opinnäytetyö tultaisiin myös tekemään jossain vaiheessa. Marraskuussa 2018 kirjoitettiin uusi puolen vuoden sopimus, jossa sovittiin, että opinnäytetyö tultaisiin tekemään sen sopimuksen aikana.

Opinnäytetyön aiheita mietittiin molempien sopimuksien aikana. Toimeksiantajalla oli halu tehdä selvitys OpenID Connect -teknologian implementoinnista ja hyödyntämisestä erityisesti mikropalveluarkkitehtuurissa, joten se valikoitui lopulliseksi opinnäytetyön aiheeksi.

Opinnäytetyön tekijällä oli jo hieman aikaisempaa kokemusta OpenID Connectista. Opinnäytetyöntekijä pääsi Jyväskylän Ammattikorkeakoulun järjestämällä projektikurssilla mukaan asiakasprojektiin, jossa tätä kyseistä teknologiaa hyödynnettiin osana mobiililaitteelle tehtyä sovellusta. Tämä tulisi hyödyksi toteutusvaiheessa. Mikropalveluarkkitehtuurista opinnäytetyön tekijällä ei ollut aikaisempaa kokemusta.

Opinnäytetyön tekijälle henkilökohtaisena tavoitteena oli ottaa hyöty irti aiheesta ja laajentaa aikaisempaa osaamista OpenID Connectista samalla tuottaen yritykselle mahdollisimman hyvä toteutus ja dokumentaatio aiheesta, että työstä voisi olla todellista hyötyä toimeksiantajalle tulevaisuudessa.

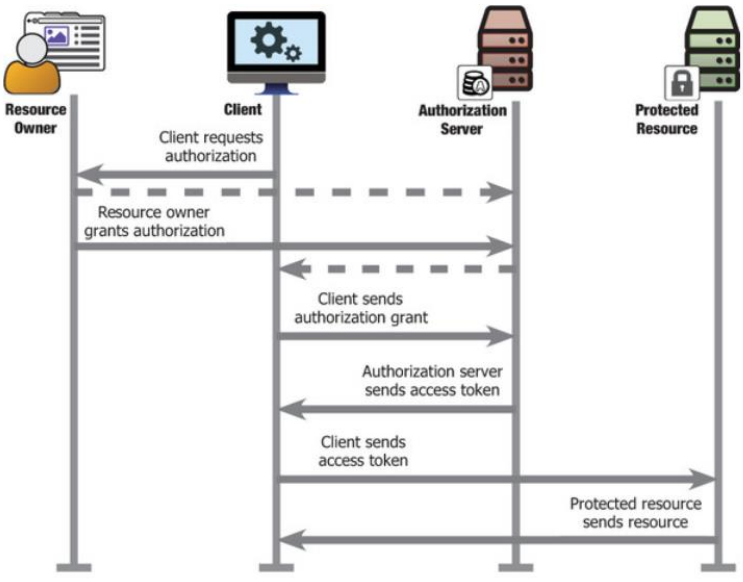
2 Teoriaa

2.1 OAuth 2.0

OAuth 2.0 on valtuutusprotokolla, jonka avulla joku henkilö, joka hallitsee jotain tiettyä resurssia, pystyy sallimaan asiakassovelluksen käyttää kyseistä resurssia heidän puolestaan, ilman että se esittää olevansa kyseisen resurssin omistaja. Sovellus pyytää resurssin omistajalta lupaa ja se vastaanottaa poletteja, joita se voi käyttää resurssin käyttämiseen. Kaikki tämä tapahtuu ilman, että sovelluksen täytyy esittää olevansa resurssin omistaja, koska poletit edustavat nimenomaan käyttäjän myöntämää valtuutettua käyttöoikeutta. (Bihis 2015.)

OAuth 2.0 -ympäristö sisältää neljä eri komponenttia. Resurssin omistajan, joka on henkilö, yleensä käyttäjä, joka omistaa jotain tietoa, jollakin palvelimella, ja joka valtuuttaa pääsyn niihin tietoihin. Asiakassovelluksen, joka pyytää pääsyä resurssin omistajan omistamiin tietoihin. Valtuutuspalvelimen, komponentin, jonka kautta käyttäjä valtuuttaa asiakassovellukselle pääsyn sen omistamiin tietoihin sen puolesta. Resurssipalvelimen, komponentin, joka sisältää resurssin omistajan hallitsemaa tietoa. (Bihis 2015.)

Valtuutusprosessi kulkee korkealla tasolla näiden komponenttien läpi Kuvion 1 mukaisesti. (Bihis 2015.)



Kuvio 1. Valtuutusprosessi komponenttien välillä.

Valtuutuspoletin hankkimiseksi asiakassovellus lähettää ensin sitä käyttävän resurssin omistajan valtuutuspalvelimelle pyytääkseen, että tämä valtuuttaa sen. Resurssin omistaja autentikoituu valtuutuspalvelimelle, ja sille esitetään yleensä mihin resurssin omistajan tietoihin sovellus haluaa pääsyn. Resurssin omistaja pystyy tässä vaiheessa päättämään, valtuuttaako se asiakassovelluksen. Jos resurssin omistaja valtuuttaa asiakassovelluksen, asiakassovellus voi pyytää valtuutusta kuvaavaa valtuutuspolettia valtuutuspalvelimelta. Tätä valtuutuspolettia asiakassovellus voi käyttää resurssipalvelimella, päästääkseen resurssin omistajan omistamiin tietoihin. (Bihis 2015.)

OAuth 2.0 -protokollan avulla resurssin omistajat pystyvät valtuuttamaan asiakassovelluksille pääsyn tämän omistamiin tietoihin resurssipalvelimella, ilman, että ne paljastavat niille heidän kirjautumistietojaan. Autentikointi tapahtuu valtuutuspalvelimella, jolloin asiakassovellus ei koskaan pääse näkemään resurssin omistajan käyttäjätietoja.

2.2 OpenID Connect

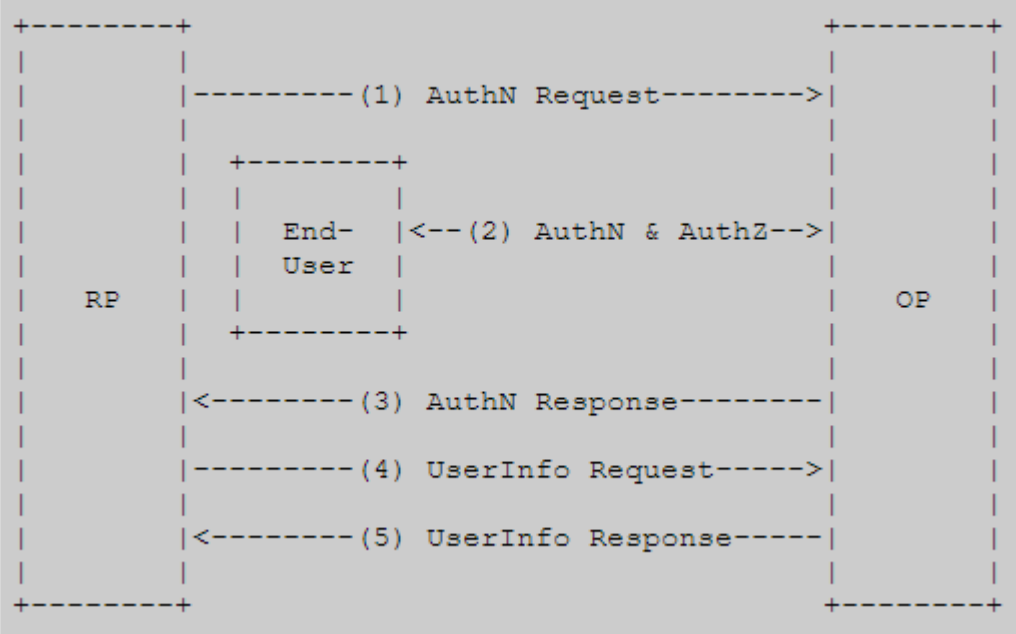
OAuth 2.0 -protokolla määrittää protokollan valtuutusten hallintaan, mutta se ei määritä käyttäjän autentikointia (Sanso 2017). Asiakaskassovellukset eivät voi sen avulla todentaa sitä käyttävän loppukäyttäjän identiteettiä. Valtuutuspoletit kuvaavat pelkästään valtuutusta, eivätkä pysty esittämään käyttäjän identiteettiä.

OpenID Connect (OIDC) on vuonna 2014 julkaistu OAuth 2.0 -protokollan päälle rakennettu protokolla, joka tarjoaa ratkaisun sekä todennusta, että valtuutusta varten. OpenID Connect tarjoaa lyhyesti sanottua identiteettikerroksen OAuth 2.0 -valtuutusprotokollan päälle. Tämä sallii asiakassovellusten varmistaa sitä käyttävän käyttäjän identiteetin, pohjautuen valtuutuspalvelimen suorittamaan käyttäjän todennukseen käsittelemättä kuitenkaan näiden kirjautumistietoja. (Sanso 2017.)

OpenID Connectin avulla kehittäjät voivat sivustoillaan ja sovelluksissaan todentaa käyttäjän. Se mahdollistaa kaiken tyyppisten asiakassovellusten, kuten selain pohjaisten ja mobiilisovelluksien, käynnistää todennusprosessi ja vastaanottaa varmistettavissa olevaa tietoa käyttäjästä ja valtuutuspalvelimen suorittamasta todennuksesta.

OpenID Connect -ympäristö sisältää kolme eri komponenttia. Loppukäyttäjän, jonka identiteetin asiakassovellus haluaa varmistaa. Asiakassovelluksen, joka haluaa varmistaa sitä käyttävän loppukäyttäjän identiteetin. OpenID Connect -tarjoajan, valtuutuspalvelimen, joka kykenee autentikoimaan loppukäyttäjän, sekä toimittamaan identiteettipoletteja ja valtuutuspoletteja.

Todennusprosessi korkealla tasolla kulkee näiden komponenttien läpi Kuvion 2 mukaisesti. (Bradley, J., De Medeiros, B., Google., Jones, M., Microsoft., Mortimore, C., NRI., Ping Identity., Sakimura, N., & Salesforce. 2014.)



Kuvio 2. Todennusprosessi komponenttien välillä

Asiakassovellus (RP) lähettää todennus-/valtuutuspyynnön valtuutuspalvelimelle (OP). Valtuutuspalvelin todentaa loppukäyttäjän, ja saa tältä valtuutuksen. Valtuutuspalvelin vastaa asiakassovelluksen pyyntöön lähettämällä vastauksen, joka sisältää OpenID Connectissa määritellyn identiteettipoletin, sekä OAuth 2.0 -protokollassa määritellyn valtuutuspoletin. Tässä vaiheessa identiteettipoletti sisältää vain pakolliseksi määritellyt väitteet. Jos asiakassovellus tarvitsee lisää väitteitä käyttäjästä. esim. käyttäjäprofiili tietoja, se voi valtuutuspoletin avulla hakea uuden identiteettipoletin valtuutuspalvelimelta, joka sisältää nämä väitteet. (Bradley ym. 2014.)

2.3 Poletit

2.3.1 JSON Web Token

JSON Web Token (JWT) on avoin standardi, joka määrittelee kompaktin, URL-turvallisen tavan lähettää ja esittää tietoja osapuolten välillä JSON-objekti muodossa (RFC 7519:2015, 1). JWT on merkkijono, jossa tieto on koodattu itseensä ja digitaalisesti allekirjoitettu, sekä mahdollisesti vielä salattu. Allekirjoituksen avulla tietoihin pystytään luottamaan, sekä niiden eheys pystytään varmistamaan. Salauksen avulla tiedot pystytään piilottamaan muilta osapuolilta.

Allekirjoitettu JWT käyttää JSON Web Signature (JWS) rakennetta, joka koostuu kolmesta osasta: Otsikosta, tietosisällöstä, sekä allekirjoituksesta, jotka on erotettu toisistaan pisteillä Base64-muodossa (kts. Kuvio 3).

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWI
iOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiw
iaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fw
pMeJf36POk6yJV_adQssw5c
```

Kuvio 3. JWT koodatussa muodossa.

Otsikko sisältää käytetyn allekirjoitusalgoritmin, sekä poletin tyypin. Tietosisältö sisältää siirrettävän tiedon. Allekirjoitus osio saadaan, kun käännetään edelliset osiot Base64-muotoon, sekä otetaan jokin sovittu merkkijono ja käytetään allekirjoitus algoritmia näihin.

2.3.2 Valtuutuspoletti

OAuth 2.0 ja OpenID Connect -protokollat käyttävät valtuutuspolettia kuvaamaan käyttäjän valtuuttamaa käyttöoikeutta. Se on merkkijono, joka voi olla satunnainen, jolloin valtuutuspoletti on yleensä tallennettuna tietokantaan yhdistettynä käyttäjään. Yleinen tapa on kuitenkin käyttää JWT merkkijonoa. OAuth 2.0 -spesifikaatio ei ota kantaa, minkälainen valtuutuspoletin pitäisi olla.

2.3.3 Identiteettipoletti

Identiteettipoletti on OpenID Connectin tuoma lisäys OAuth 2.0 -protokollan päälle. Se on määritetty olemaan muodoltaan JWT ja se sisältää väitteitä valtuutuspalvelimen suorittamasta käyttäjän todennuksesta, sekä mahdollisesti muita pyydettyjä väitteitä esim. käyttäjäprofiili tietoa (kts. Kuvio 4.). Valtuutuspalvelimen suorittamasta autentikoinnista OpenID Connect palauttaa oletuksena vähintään seuraavat väitteet: (Bradley ym. 2014.)

- iss (Issuer) - HTTPS URL, joka toimii identiteettipoletin myöntäjän tunnisteena.
- sub (Subject) - Uniikki merkkijono, joka toimii käyttäjän tunnisteena.
- aud (Audience) - Merkkijono, joka kertoo, kenelle identiteettipoletti on tarkoitettu. Yleensä sovellukselle määritelty uniikki merkkijono.
- exp (Expiration) - Aika, jolloin identiteettipoletti vanhenee, eikä sitä voi käyttää enää.
- iat (Issued at) - Aika, jolloin identiteettipoletti on myönnetty.

```
{
    "iss": "https://issuer.com",
    "iat": 1552417424,
    "exp": 1583953424,
    "aud": "audience",
    "sub": "user"
}
```

Kuvio 4. Autentikointiväitteet puretussa muodossa.

OpenID Connect määrittelee, että identiteettipoletin täytyy olla allekirjoitettuja käyttäen JWS rakennetta. Identiteettipoletin salaus on valinnaista ja se toteutetaan käyttäen JSON Web Encryption (JWE) standardia. (Bradley ym. 2014.)

2.4 Väitteet

Väitteet ovat avainarvo pareja, joiden avulla OpenID Connect esittää tietoa käyttäjästä, sekä valtuutuspalvelimen suorittamasta todennuksesta. Väitteet voivat olla rekisteröityjä, julkisia tai yksityisiä.

Rekisteröidyt väitteet on määritelty Internet Assigned Numbers Authorityn (IANA) ylläpitämässä JWT Web Token Claims rekisterissä, joka on määritelty JWT standardissa.

Niitä ei ole pakko käyttää, mutta ovat suositeltavia. Julkiset väitteet on määritelty samassa rekisterissä. Yksityiset väitteet ovat mukautettuja väitteitä, joiden käytöstä on sovittu niitä käyttävien osapuolten välillä. Ne eivät ole rekisteröityjä eivätkä julkisesti määriteltyjä. (RFC 7519:2015, 8.)

OpenID Connect määrittelee palautettavia käyttäjätieto väitteitä, jotka ovat myös julkisiksi määriteltyjä. Käyttäjätieto väitteitä voi pyytää yksitellen tai kokonaisen ryhmän. Väitteet, joita asiakassovellus haluaa, määritellään valtuutuspalvelimelle lähetetyssä kutsussa. Väiteryhmille käytetään OAuth 2.0 scope parametriä, jolloin parametriksi annetaan väiteryhmän avain. Yksittäisille väitteille käytetään claims parametriä, jolle arvoksi annetaan yksittäisen väitteen avain. OpenID Connectin käyttämät käyttäjätieto väitteet ja väiteryhmät ovat:

- profile - Palauttaa käyttäjän profiilitietoa
    - name, family_name, given_name, middle_name, nickname, preferred_username, profile, picture, website, gender, birthdate, zoneinfo, locale, updated_at.
- email - Palauttaa käyttäjän sähköpostiosoitteen, sekä tiedon onko se vahvistettu.
    - email, email_verified
- address - Palauttaa käyttäjän osoitetiedon, tai osoitetiedot eriteltynä lisäkenttiin
        - formatted, street_address, locality, region, postal_code, country
- phone - Palauttaa käyttäjän puhelinnumeron, sekä tiedon onko puhelinnumero varmistettu
    - phone_number, phone_number_verified

2.5 Autentikointi ja autentikointiprosessit

Ennen polettien luovuttamista käyttäjän täytyy autentikoitua jonkun kolmen OpenID Connectin määrittämän autentikointiprosessin avulla. Autentikointiprosessit määrittelevät miten poletit palautetaan loppukäyttäjälle (Bradley ym. 2014). OpenID Connectissa määritellyt prosessit ovat käytännössä samoja kuin OAuth 2.0 -protokollassa määritellyt, mutta ne palauttavat valtuutuspoletin lisäksi identiteettipoletin.

Valtuutuskoodi prosessia käytettiin toteutuksessa, joten se käydään tarkemmin läpi. Muut prosessit käydään läpi prosessin kulku tasolla, mutta ei tarkemmin, koska niitä ei toteutuksessa tarvittu.

2.5.1 Valtuutuskoodi prosessi

Valtuutuskoodi prosessi on kaksivaiheinen, jossa ensimmäisessä vaiheessa valtuutuspalvelin palauttaa sovellukselle valtuutuskoodin, jonka se voi vaihtaa identiteettipolettiin, sekä valtuutuspolettiin toisessa vaiheessa polettipäätepisteestä. Valtuutuskoodi prosessissa poletteja ei paljasteta missään vaiheessa autentikointia suoritettavalle selaimelle tai mahdollisesti muille haitallisille sovelluksille, joilla on pääsy selaimeen. Valtuutuspalvelin voi myös todentaa asiakassovelluksen ennen valtuutuskoodin vaihtamista poletteihin. (Bradley ym. 2014.)

Prosessin kulku menee seuraavasti (Bradley ym. 2014):

1. Sovellus valmistelee autentikaatiopyynnön.
2. Sovellus lähettää pyynnön valtuutuspalvelimen valtuutuspäätepisteeseen.
3. Valtuutuspalvelin autentikoi käyttäjän ja saa tämän valtuutuksen sovelluksen pyytämiin väitteisiin / oikeuksiin.
4. Valtuutuspalvelin lähettää käyttäjän takaisin sovellukseen valtuutuskoodin kanssa.
5. Sovellus tekee polettipyynnön polettipäätepisteeseen valtuutuskoodin kanssa.
6. Sovellus vastaanottaa vastauksen, joka sisältää identiteettipoletin, sekä valtuutuspoletin
7. Sovellus validoi identiteettipoletin

Ensimmäisessä vaiheessa menevä autentikaatiopyyntö (kts. Kuvio 5) on GET- tai POST-pyyntö (Bradley ym. 2014).

```
GET /authorize?
  response_type=code
  &scope=openid%20profile%20email
  &client_id=s6BhdRkqt3
  &state=af0ifjsldkj
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb HTTP/1.1
Host: server.example.com
```

Kuvio 5. Esimerkki valtuutuspalvelimelle menevästä autentikaatiopyynnöstä.

Pyynnön mukana menevät vähintään seuraavat parametrit:

- scope – Parametri määrittelee, mitä väitteitä tai oikeuksia sovellus haluaa.
- response_type – Parametri määrittelee, mitä vastaus palauttaa.
- client_id – Sovelluksen yksilöllinen tunniste.
- redirection_uri – Osoite johon vastaus lähetetään.

Parametrin scope täytyy sisältää openid arvo. Parametri response_type täytyy valtuutuskoodi prosessin mukaan olla arvoltaan code. Parametri state ei ole pakollinen, mutta suositeltava. Sen avulla pyritään estämään CSRF hyökkäyksiä.

Valtuutuspalvelimen täytyy varmentaa pyyntö. Valtuutuspalvelin varmistaa, että kaikki tarvittavat parametrit ovat mukana pyynnössä, sekä validoi ne spesifikaation mukaan. Varmentamisen jälkeen, palvelin suorittaa autentikoinnin. Autentikoinnin jälkeen, käyttäjän täytyy vielä myöntää lupa sovellukselle pääsyn tämän tietoihin. Miten autentikointi, ja luvan myöntäminen tapahtuu, ei kuulu OpenID Connect -spesifikaatioon (Bradley ym. 2014).

Onnistuneen autentikointipyynnön jälkeen valtuutuspalvelin lähettää sovellukselle vastauksen (kts. Kuvio 6), joka sisältää valtuutuskoodin. (Bradley ym. 2014.)

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb?
  code=SplxlOBeZQQYbYS6WxSbIA
  &state=af0ifjsldkj
```

Kuvio 6. Esimerkki valtuutuspalvelimen lähettämästä vastauksesta onnistuneen autentikointipyynnön jälkeen.

Sovelluksen täytyy varmistaa, että vastaus sisältää valtuutuskoodin. Jos state parametriä käytettiin, sovelluksen täytyy varmistaa, että sen sisältämä merkkijono vastaa alustavassa autentikointipyynnössä lähetettyä merkkijonoa.

Toisessa vaiheessa menevä polettipyyntö on POST-pyyntö, jonka mukana menee seuraavat parametrit:

- grant_type – Parametri määrittelee, mitä valtuutusprosessia käytetään.
- code – Sisältää valtuutuskoodin.
- client_id – Sovelluksen yksilöllinen tunniste.
- redirect_uri – Osoite, johon vastaus lähetetään.

Valtuutuspalvelimen täytyy tässäkin vaiheessa varmentaa pyyntö. Valtuutuspalvelin varmistaa, että valtuutuskoodi, jonka se lähetti alustavassa autentikaatiopyynnössä on sama kuin pyynnössä ja että se on validi. Valtuutuspalvelin myös varmistaa, että pyynnössä tuleva osoite redirect_uri, johon vastaus lähetetään, on sama mikä lähetettiin alustavassa autentikaatiopyynnössä.

Varmistettuaan polettipyynnön valtuutuspalvelin lähettää sovellukselle takaisin vastauksen, joka vähintään seuraavat tiedot:

- access_token – Valtuutuspoletti
- token_type – Poletin tyyppi
- id_token – Identiteettipoletti

Sovelluksen täytyy varmistaa, että vastaus sisältää vähintään nämä tiedot, ja että identiteettipoletti on spesifikaation mukaan validi.

2.5.2 Epäsuora prosessi

Epäsuora prosessi on yksivaiheinen, joka valtuutuskoodi prosessista poiketen palauttaa sovellukselle heti ensimmäisen pyynnön jälkeen identiteettipoletin, sekä valtuutuspoletin valtuutuspäätepisteestä. Epäsuoraa prosessia käyttävät pääasiassa asiakassovellukset, jotka on toteutettu selaimessa. Koska poletit palautetaan suoraan asiakkaalle, prosessi saattaa paljastaa ne loppukäyttäjälle ja haitallisille sovelluksille, joilla on pääsy selaimeen. Prosessissa valtuutuspalvelin ei suorita asiakassovelluksen autentikointia. (Bradley ym. 2014.)

Prosessin kulku menee seuraavasti (Bradley ym. 2014):

1. Sovellus valmistelee autentikaatiopyynnön.
2. Sovellus lähettää pyynnön valtuutuspalvelimen valtuutuspäätepisteeseen.
3. Valtuutuspalvelin autentikoi käyttäjän ja saa tämän valtuutuksen sovelluksen pyytämiin väitteisiin / oikeuksiin.
4. Valtuutuspalvelin lähettää käyttäjän takaisin sovellukseen identiteettipoletin, sekä valtuutuspoletin kanssa
5. Sovellus validoi identiteettipoletin

2.5.3 Hybridi prosessi

Hybridi prosessi on sekoitus valtuutuskoodi prosessia ja epäsuora prosessia. Hybridi prosessissa osa poleteista palautetaan heti ensimmäisessä vaiheen jälkeen valtuutuspäätepisteestä ja osa toisen vaiheen jälkeen polettipäätepisteestä.

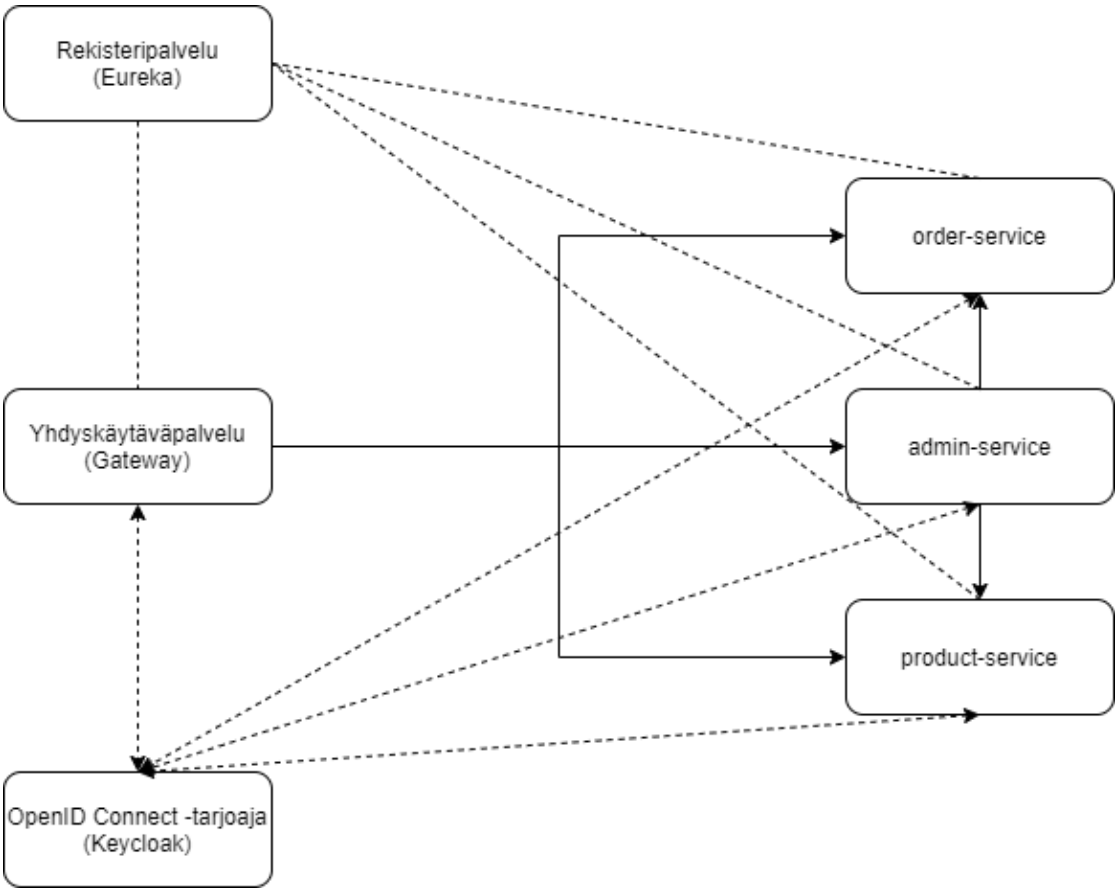
Prosessin kulku menee seuraavasti (Bradley ym. 2014):

1. Sovellus valmistelee autentikaatiopyynnön.
2. Sovellus lähettää pyynnön valtuutuspalvelimen valtuutuspäätepisteeseen.
3. Valtuutuspalvelin autentikoi käyttäjän ja saa tämän valtuutuksen sovelluksen pyytämiin väitteisiin / oikeuksiin.
4. Valtuutuspalvelin lähettää käyttäjän takaisin sovellukseen valtuutuskoodin ja pyynnössä määriteltyjen polettien kanssa
5. Sovellus tekee polettipyynnön polettipäätepisteeseen valtuutuskoodin kanssa.
6. Sovellus vastaanottaa vastauksen, joka sisältää identiteettipoletin, sekä valtuutuspoletin
7. Sovellus validoi identiteettipoletin

3 Toteutus

3.1 Arkkitehtuurikuvaus

Toteutusta varten luotiin Kuvion 7 mukainen arkkitehtuuri. Arkkitehtuuri sisälsi kolme mikropalvelua, rekisteripalvelun, yhdyskäytäväpalvelun, sekä OpenID Connect -tarjoajan. Rekisteripalvelu tarjosi palvelurekisterin, jonne palvelut lukuun ottamatta OpenID Connect -tarjoajaa rekisteröityivät. OpenID Connect -tarjoaja tarjosi OpenID Connect -protokollan implementoivan palvelun. Yhdyskäytäväpalvelu tarjosi yksittäisen tulopisteen mikropalveluille. Se autentikoi kutsuja tekevät käyttäjät OpenID Connect -tarjoajan avulla, ja välitti valtuutuspoletteja mikropalveluille. Mikropalvelut validoivat saatuja poletteja tarjoajan avulla.



Kuvio 7. Mikropalveluarkkitehtuuri

3.2 Valitut teknologiat

Toteutukseen valittiin Java-ohjelmointikieli, koska se oli opinnäytetyötekijälle kaikista tutuin, sekä myös toimeksiantajalla laajasti käytetty. Sen takia valitut teknologiat keskittyivät Java-ohjelmointikielen ympärille. Projektissa käytettiin palveluissa pohjana Java-ohjelmointikielen ympärille rakennettua Spring Framework -sovelluskehystä. Spring Framework tarjoaa laajan ekosysteemin, joka sisältää useita eri projekteja ja se mahdollisti helpon ja nopean tavan rakentaa mikropalveluympäristö, jossa palvelut ovat ajettavissa muutamalla rivillä koodia.

3.2.1 Spring Cloud Netflix Eureka

Spring Cloud Netflix Eureka on alun perin Netflixin kehittämä avoimen lähdekoodin projekti, joka on integroitu osaksi Spring Cloud -sovelluskehystä. Se tarjoaa työkalut rekisteripalvelun rakentamiseen Spring-applikaationa. Sen tarkoituksena on mahdollistaa muiden ympäristössä sijaitsevien palveluiden rekisteröidä itsensä sen palvelurekisteriin. Palvelurekisterin kautta se voi jakaa muista ympäristössä olevista palveluista tietoa esim. niiden osoitteen sitä käyttävälle palvelulle.

3.2.2 Spring Cloud Gateway

Spring Cloud Gateway on Spring Cloud -sovelluskehykseen kuuluva projekti, joka tarjoaa työkalut API-yhdyskäytävän rakentamiseen Spring-applikaationa. API-yhdyskäytävän tarkoituksena on toimia yksittäisenä tulopisteenä kaikille pyynnöille, jotka kohdistuvat sen takana oleviin palveluihin.

3.2.3 Spring security

Spring Security on sovelluskehys, joka tarjoaa työkalut autentikointi- ja valtuutusratkaisujen implementointiin Spring-applikaatioihin. Se sisältää toteutuksessa tarvittavat implementaatiot OpenID Connect -protokollan mukaiselle autentikoinnille, sekä valtuutukselle.

3.2.4 Keycloak

Keycloak on JBoss yhtiön kehittämä avoimen lähdekoodin ohjelmisto, joka tarjoaa identiteetin- ja pääsynhallinnan ratkaisut sitä käyttäville sovelluksille. Toteutuksessa sitä käytettiin OpenID Connect -tarjoajana, joka autentikoi käyttäjät, sekä myönsi ja validoi poletit.

3.2.5 Postman

Postman on API-testaukseen tehty työkalu, jonka kautta voi lähettää HTTP-pyyntöjä valittuun päätepisteeseen. Postmanilla voi muun muassa lähettää POST-pyyntöjä määritellyllä sisällöllä. Se mahdollistaa myös otsakkeiden ja evästeiden lisäämisen pyyntöön.

3.3 Palveluiden luonti

Rekisteripalvelulle määritettiin konfiguraatio application.properties-tiedostoon Kuvion 8 mukaisesti.

```
#General
spring.application.name: service-registry
server.port: 8761

#Eureka
eureka.client.register-with-eureka: false
```

Kuvio 8. Rekisteripalvelun konfiguraatio

Rekisteripalvelulle tarvitsi määritellä nimi spring.application.name parametrilla, portti server.port parametrillä, sekä määritellä ettei se rekisteröi itseään palvelurekisteriin eureka.client-register-with-eureka parametrilla. Tämän jälkeen palvelun pääluokkaan täytyi vielä lisätä @EnableEurekaServer annotaatio Kuvion 9 mukaisesti. Annotaation avulla palvelu ilmaisi olevansa rekisteripalvelu ja muut palvelut pystyivät rekisteröitymään sen palvelurekisteriin. Rekisteripalvelun käynnistyttyä siihen sai yhteyden konfiguroidussa portissa, joka aukaisi palvelun tilasivun.

```
@EnableEurekaServer
@SpringBootApplication
public class ServiceRegistryApplication {
    public static void main(String[] args) {
        SpringApplication.run(ServiceRegistryApplication.class, args);
    }
}
```

Kuvio 9. Rekisteripalvelun pääluokka

Jokaiselle mikropalvelulle määritettiin oma konfiguraatio niiden application.properties-tiedostoon Kuvion 10 mukaisesti.

```
#General
spring.application.name=product-service
server.port=8081

#Eureka
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
eureka.instance.preferIpAddress=true
```

Kuvio 10. Mikropalvelun konfiguraatio

Konfiguraatioon määritettiin rekisteripalvelun tapaan nimi ja portti, jotka olivat jokaiselle mikropalvelulla erilaiset. Palvelut rekisteröityivät rekisteripalvelun palvelurekisteriin konfiguraatiossa annetulla nimellä. Palveluille määritettiin missä osoitteessa rekisteripalvelu sijaitsee eureka.client.serviceUrl.defaultZone parametrilla. Lisäksi määritettiin, että Eureka kertoo muille palveluille palvelun IP-osoitteen isäntänimen sijaan eureka.instance.preferIpAddress parametrillä.

Mikropalveluiden pääluokkiin täytyi vielä lisätä annotaatio @EnableDiscoveryClient annotaatio Kuvion 11 mukaisesti. Annotaation avulla palvelut rekisteröityivät rekisteripalvelun palvelurekisteriin käynnistyksen yhteydessä.

```
@EnableDiscoveryClient
@SpringBootApplication
public class ProductServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(ProductServiceApplication.class, args);
    }
}
```

Kuvio 11. Mikropalvelun pääluokka

Yhdyskäytäväpalvelulle määritettiin konfiguraatio Kuvion 12 mukaisesti.

```
#Gateway
spring.cloud.gateway.default-filters[0]=TokenRelay
spring.cloud.gateway.routes[0].id=product-service
spring.cloud.gateway.routes[0].uri=lb://product-service
spring.cloud.gateway.routes[0].predicates[0]=Path=/product-service/**
spring.cloud.gateway.routes[0].filters[0]=StripPrefix=1
spring.cloud.gateway.routes[1].id=order-service
spring.cloud.gateway.routes[1].uri=lb://order-service
spring.cloud.gateway.routes[1].predicates[0]=Path=/order-service/**
spring.cloud.gateway.routes[1].filters[0]=StripPrefix=1
spring.cloud.gateway.routes[2].id=admin-service
spring.cloud.gateway.routes[2].uri=lb://admin-service
spring.cloud.gateway.routes[2].predicates[0]=Path=/admin-service/**
spring.cloud.gateway.routes[2].filters[0]=StripPrefix=1
```

Kuvio 12. Yhdyskäytäväpalvelun konfiguraatio

Konfiguraatioon määritettiin rekisteripalvelun ja mikropalveluiden tapaan nimi ja portti. Palvelu rekisteröityi rekisteripalvelun palvelurekisteriin konfiguraatiossa annetulla nimellä. Parametrin spring.cloud.gateway.default-filters avulla palvelulle määritettiin jokaiseen reittiin TokenRelay-suodatin. TokenRelay-suodatin lisäsi jokaiseen kutsuun, joka meni mikropalvelulle Authorization-otsakkeen, joka sisälsi valtuutuspoletin. Parametrin spring.cloud.gateway.routes avulla määritettiin muiden mikropalveluiden sijainnit, sekä polut minkä kautta niihin otettiin yhteyttä. Koska käytössä oli rekisteripalvelu, parametreihin täytyi määrittää vain palveluiden nimet, joilla ne olivat rekisteröityneet rekisteripalvelun palvelurekisteriin. Lopuksi määritettiin missä osoitteessa rekisteripalvelu sijaitsi mikropalveluiden tapaan. Yhdyskäytäväpalvelun pääluokkaan täytyi vielä lisätä annotaatio @EnableDiscoveryClient mikropalvelujen tapaan, jotta se osasi rekisteröityä rekisteripalvelun palvelurekisteriin.

3.4 Päätepisteet ja niiden suojaaminen

Product mikropalvelulle määritettiin kaksi päätepistettä Kuvion 13 mukaisesti.

```
@PostMapping("/add")
public Product createProduct(@Valid @RequestBody Product product) {...}

@GetMapping("/get")
public List<Product> getAllProducts() { return products; }
```

Kuvio 13. Product mikropalvelun päätepisteet

Päätepisteellä /add pystyi lisäämään palvelun kautta uuden tuotteen tekemällä siihen POST-pyynnön, joka sisälsi uuden tuotteen JSON-muodossa. Palvelu vastasi palauttamalla lisätyn tuotteen JSON-muodossa. Tekemällä GET-pyynnön /get päätepisteeseen palvelu palautti vastauksessa kaikki olemassa olevat tuotteet. Mikropalvelu ei tässä toteutuksessa ollut yhteydessä tietokantaan, vaan se piti kirjaa olemassa olevista tuotteista products nimisessä List-tyyppisessä muuttujassa.

Product mikropalvelulle määritettiin SecurityConfig perimällä WebSecurityConfigurerAdapter-luokka ja yliajamalla sen configure-metodi Kuvion 14 mukaisesti.

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers( ...antPatterns: "/get")
                    .permitAll()
                .antMatchers( ...antPatterns: "/add")
                    .hasRole("SHOP-ADMIN");
    }
}
```

Kuvio 14. Product mikropalvelun SecurityConfig

@EnableWebSecurity annotaation avulla Spring osasi palvelun käynnistyessä konfiguroida päätepisteiden suojauksen luokassa luodun konfiguraation perusteella. Konfiguraatioketjuun määritettiin, että /get päätepisteeseen on pääsy kaikilla, kun taas /add päätepiste vaatii SHOP-ADMIN roolin.

Order mikropalvelulle määritettiin kolme päätepistettä Kuvion 15 mukaisesti.

```
@PostMapping("/add")
public Order createOrder(@Valid @RequestBody Order order) {...}

@GetMapping("/get")
public List<Order> getAllOrders() { return orders; }
```

Kuvio 15. Order mikropalvelun päätepisteet

Päätepisteellä /add pystyi lisäämään palvelun kautta uuden tilauksen tekemällä siihen POST-pyynnön, joka sisälsi uuden tilauksen JSON-muodossa. Palvelu vastasi palauttamalla lisätyn tilauksen. Tekemällä GET-pyynnön /get päätepisteeseen palvelu palautti vastauksessa kaikki palvelussa olevat tilaukset. Mikropalvelu ei tässä toteutuksessa ollut yhteydessä tietokantaan, vaan se piti kirjaa olemassa olevista tuotteista orders nimisessä List-tyyppisessä muuttujassa.

Order mikropalvelulle määritettiin SecurityConfig samalla periaatteella, kuin product mikropalvelulle Kuvion 16 mukaisesti.

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers( ...antPatterns: "/get")
                    .hasRole("SHOP-ADMIN")
                .antMatchers( ...antPatterns: "/add")
                    .hasRole("SHOP-USER");
    }
}
```

Kuvio 16. Order mikropalvelun SecurityConfig

@EnableWebSecurity annotaation avulla Spring osasi palvelun käynnistyessä konfiguroida päätepisteiden suojauksen luokassa luodun konfiguraation perusteella. Konfigurointiketjuun määritettiin /get päätepisteelle roolivaatimus SHOP-ADMIN ja /get/customer, sekä /add päätepisteille roolivaatimus SHOP-USER.

Admin mikropalvelulle määritettiin kaksi päätepistettä Kuvion 17 mukaisesti.

```
@PostMapping("/add-product")
public Product addProduct(@Valid @RequestBody Product product) {...}

@GetMapping("/get-orders")
public List<Order> getAllOrders() {...}
```

Kuvio 17. Admin mikropalvelun päätepisteet

Päätepisteellä /add-product pystyi lisäämään admin mikropalvelun kautta uuden tuotteen product mikropalveluun tekemällä siihen POST-pyynnön, joka sisälsi uuden tuotteen JSON-muodossa. Admin mikropalvelu teki POST-pyynnön product mikropalvelimen /add päätepisteeseen, jonka kautta se lisäsi uuden tuotteen sen products muuttujaan. Päätepiste vastasi palauttamalla lisätyn tuotteen.

Tekemällä GET-pyynnön /get-orders päätepisteeseen palvelu palautti kaikki Order mikropalvelussa olemassa olevat tilaukset. Admin mikropalvelu teki GET-pyynnön Order mikropalvelun /get päätepisteeseen, jonka kautta se palautti vastauksessaan kaikki olemassa olevat tilaukset.

Molemmat päätepisteet ottivat suoraan yhteyttä mikropalveluun, joten mikropalveluille menevään kutsuun täytyy lisätä Authorization-otsakkeeseen yhdyskäytäväpalvelulta saatu valtuutuspoletti. Valtuutuspoletti haettiin lisättäväksi Spring Securityn SecurityContext-rajapinnasta Kuvion 18 mukaisesti. Tässä yhteydessä hyödynnettiin myös rekisteripalvelun palvelurekisteriä, jotta saatiin osoitteet mikropalveluille.

```
private void setAuthorizationHeader(HttpHeaders httpHeaders) {
    JwtAuthenticationToken token = (JwtAuthenticationToken) SecurityContextHolder
        .getContext().getAuthentication();
    httpHeaders.set("Authorization", "Bearer " + token.getToken().getTokenValue());
}
```

Kuvio 18. Valtuutuspoletti SecurityContext-rajapinnasta

Admin mikropalvelulle määritettiin SecurityConfig samalla periaatteella, kuin muillekin mikropalveluille Kuvion 19 mukaisesti.

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers( ...antPatterns: "/add-product")
                    .hasRole("SHOP-ADMIN")
                .antMatchers( ...antPatterns: "/get-orders")
                    .hasRole("SHOP-ADMIN");
    }
}
```

Kuvio 19. Admin mikropalvelun SecurityConfig

@EnableWebSecurity annotaation avulla Spring osasi palvelun käynnistyessä konfiguroida päätepisteiden suojauksen luokassa luodun konfiguraation perusteella. Konfiguraatioketjuun määritettiin, että molemmat päätepisteet vaativat SHOP-ADMIN-roolin.

Yhdyskäytäväpalvelulle määritettiin demonstroinnin ja testauksen takia yksi päätepiste Kuvion 20 mukaisesti, joka palautti Spring Securityn SecurityContext-rajapinnasta tietoja autentikoituneesta käyttäjästä. Päätepisteen avulla pystyttiin varmistamaan, että Keycloakin kautta tehty autentikaatioprosessi on onnistunut, sekä että Spring Security on tallentanut autentikoinnin tiedot SecurityContext-rajapintaan.

```
@GetMapping(value = "/get-authentication")
public Mono<String> getAuthentication() {...}
```

Kuvio 20. Yhdyskäytäväpalvelun päätepiste

Yhdyskäytäväpalvelulle määritettiin SecurityConfig Kuvion 21 mukaisesti.

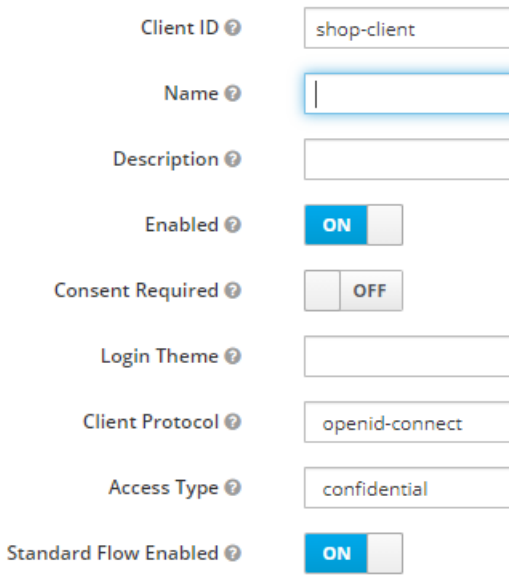
```
@EnableWebFluxSecurity
public class SecurityConfig {
    @Bean
    public SecurityWebFilterChain securityWebFilterChain(ServerHttpSecurity http) {
        return http
            .oauth2Login()
            .and()
            .authorizeExchange(exchange ->
                exchange
                    .pathMatchers( ...antPatterns: "/get-authentication")
                        .authenticated()
                    .pathMatchers( ...antPatterns: "/product-service/get")
                        .permitAll()
                    .pathMatchers( ...antPatterns: "/product-service/add")
                        .authenticated()
                    .pathMatchers( ...antPatterns: "/order-service/**")
                        .authenticated()
                    .pathMatchers( ...antPatterns: "/admin-service/**")
                        .authenticated())
            .csrf()
                .disable()
            .build();
    }
```

Kuvio 21. Yhdyskäytäväpalvelun SecurityConfig

Yhdyskäytäväpalvelun SecurityConfig määritti käyttäjän autentikointi tarpeen. Konfiguraatioketjuun on määritetty, että lukuun ottamatta product mikropalvelun /get päätepistettä, käyttäjän täytyy olla autentikoituna tehtäessä pyyntöjä muihin päätepisteisiin.

3.5 Keycloak konfigurointi

Keycloak palvelimelle lisättiin uusi asiakassovelluskonfiguraatio Kuvion 22 mukaisesti.



Kuvio 22. Keycloak asiakassovelluskonfiguraatio

Asiakassovellukselle määritettiin tunniste, jolla se tunnistettiin (Client ID), sekä protokolla, jota se käytti käyttäjän autentikoimiseen (Client-Protocol). Access type confidental arvolla tarkoitti, että asiakassovelluksen pyytäessä käyttäjän autentikointia, sen täytyy toimittaa ennalta sovittu salainen merkkijono (secret). Salaisen merkkijonon avulla Keycloak varmisti asiakassovelluksen, kun se tekee autentikaatiopyynnön. Lisäksi otettiin käyttöön valtuutuskoodi autentikointiprosessi Standard Flow valitsimella.

Palvelimelle lisättiin kaksi käyttäjää shop-admin ja shop-user, sekä kaksi roolia shop-admin ja shop-user. Käyttäjille täytettiin perustiedot, lisättiin id attribuutti, sekä liitettiin ne omiin rooleihinsa. Valtuutuspoletin sisällöksi konfiguroitiin id, sekä roles väitteet. Identiteettipoletin sisällöksi konfiguroitiin myös id ja roles, sekä email, profile ja roles väitteet.

3.6 OpenID Connectin käyttöönotto

Toteutuksessa käyttäjien autentikaatio keskitettiin yhdyskäytäväpalveluun, koska kaikki pyynnöt kulkivat sen kautta. Jotta yhdyskäytäväpalvelu pystyi autentikoimaan käyttäjän, sen konfiguraatioon tarvitsi määrittää OpenID Connect -tarjoajan tiedot, sekä asiakassovelluksen varmistamiseen tarvittavat tiedot Kuvion 23 mukaisesti.

```
#OAuth2
spring.security.oauth2.client.provider.keycloak.issuer-uri=http://localhost:8080/auth/realms/shop
spring.security.oauth2.client.provider.keycloak.user-name-attribute=preferred_username
spring.security.oauth2.client.registration.keycloak.client-id=shop-client
spring.security.oauth2.client.registration.keycloak.client-secret=a26e4895-cfea-49c1-9f73-05049cc7d6b4
spring.security.oauth2.client.registration.keycloak.scope=openid
```

Kuvio 23. OpenID Connect -konfiguraatio yhdyskäytäväpalvelussa

Konfiguraatiossa spring.security.oauth2.client.registration alkuisilla parametreillä määritettiin yhdyskäytäväpalvelun varmistamiseen tarvittavat arvot, sekä halutut väitteet. Parametrin client-id täytyi olla sama, mikä oli Keycloakin asiakassovelluksen konfiguraatiossa. Parametri client-secret täytyi olla myös sama, kuin Keycloakin asiakassovelluksen konfiguraatiossa. Arvo haettiin konfiguraatiosta, jossa se oli automaattisesti generoitu. Parametri scope määritteli yhdyskäytäväpalvelun haluamat väitteet. Arvoksi asetettiin vain openid, koska se on pakollinen. Keycloakin konfiguraatioon määritettiin, että se palauttaa oletuksena tarvittavat väitteet pyytämättä, joten muita arvoja ei tarvittu.

Parametri spring.security.oauth2.client.provider.keycloak.issuer-uri määritti Keycloakin osoitteen. OpenID Connect -tarjoajat tarjoavat metadataa, kuten niiden päätepisteiden osoitteet .well-known/openid-configuration päätepisteestä. Parametri on määritelty Spring Securitya varten, joka ottaa parametrin arvon ja liittää edellä mainitun päätepisteen sen perään, ja hakee autentikoimiseen ja valtuutukseen tarvittavat päätepisteet ja tiedot sieltä. Parametri spring.security.oauth2.client.provider.keycloak.user-name-attribute määritti nimen, jolla käyttäjä haluaa tulla viitatuksi asiakassovellukselle. Arvo preferred_username tarkoitti käyttäjän käyttäjätunnusta, jolla se kirjautuu Keycloakkiin.

Yhdyskäytäväpalvelussa OpenID Connect otettiin käyttöön lisäämällä sen SecurityConfig luokan konfiguraatioketjuun oAuth2login()-metodi. Metodi määritti Spring Securitylle, että jos käyttäjä ei ole autentikoituna, se autentikoi käyttäjän jollakin OpenID Connect -protokollassa määritellyllä autentikointiprosessilla. Koska Keycloak konfiguraatioon määritettiin yhdyskäytäväpalvelun autentikointiprosessiksi valtuutuskoodi prosessi, se käytti sitä.

Konfiguroinnin jälkeen otettaessa yhteyttä mihin tahansa yhdyskäytäväpalvelussa suojatuksi määritettyyn päätepisteeseen, yhdyskäytäväpalvelu aloitti autentikoinnin, jos käyttäjä ei ollut autentikoituna. Spring Security määritteli, onko käyttäjä autentikoituna tallentamalla käyttäjän autentikoinnin tiedot sen SecurityContext-rajapintaan, mukaan lukien valtuutuspoletin. Autentikoinnin yhteydessä Spring Security asetti käyttäjän selaimeen evästeen JSESSIONID. Jos pyynnön mukana tuli JSESSIONID-eväste, Spring Security tarkisti sen avulla SecurityContext-rajapinnasta, onko käyttäjä autentikoituna. Autentikoitumisen voimassaoloon vaikutti myös valtuutuspoletin vanhenemisaika.

Mikropalveluiden konfiguraation tarvitsi määrittää OpenID Connect -tarjoajan osoite Kuvion 24 mukaisesti.

```
#OAuth2
spring.security.oauth2.resourceserver.jwt.issuer-uri=http://localhost:8080/auth/realms/shop
```

Kuvio 24. OpenID Connect -konfiguraatio mikropalvelussa

Parametrin issuer-uri avulla Spring Security haki tarjoajan .well-known/openid-configuration päätepisteestä osoitteen, josta se voi hakea Keycloakin tarjoaman julkisen avaimen valtuutuspoletin validoimiselle.

Mikropalveluissa OpenID Connect otettiin käyttöön lisäämällä mikropalveluiden konfiguraatioketjuun oauth2ResourceServer()-metodi. Metodi määrittelee Spring Securitylle, että sen täytyy mikropalveluille tulevista kutsuista etsiä Authorization-otsake ja poimia sieltä valtuutuspoletti. Jos otsake ja poletti löytyivät, Spring Security validoi valtuutuspoletin Keycloakilta haetulta julkisella avaimella, sekä asettaa käyttäjän roolit sen SecurityContext-rajapintaan.

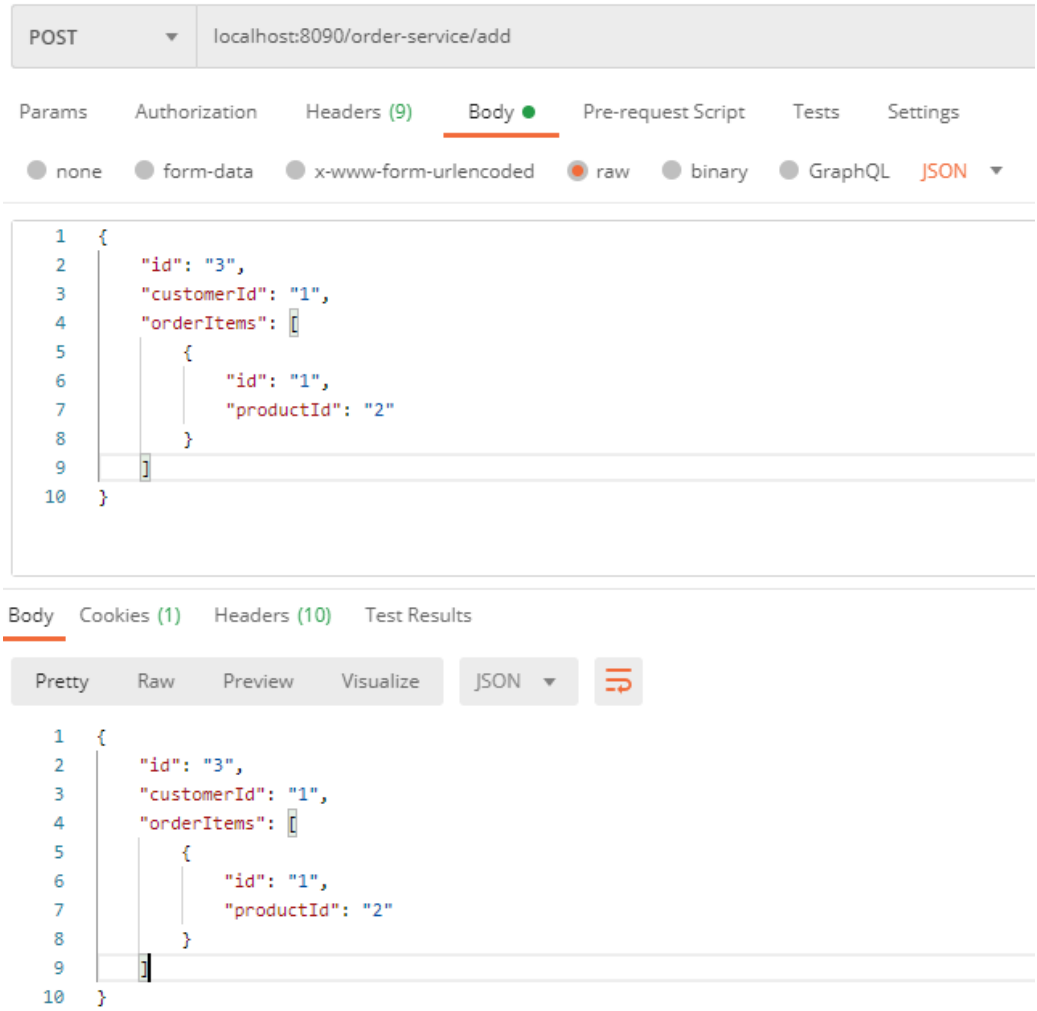
4 Tulokset

Opinnäytetyön lopputuloksena saatiin rakennettua mikropalveluympäristö, sekä onnistuneesti implementoitua OpenID Connect siihen. Kun yhdyskäytäväpalvelun kautta ensimmäisen kerran tehtiin pyyntö product mikropalvelun /get päätepisteeseen, autentikointia ei tapahtunut, koska /get päätepiste oli määritelty kaikille saatavaksi. Tekemällä pyyntö yhdyskäytäväpalvelun /get-authentication päätepisteeseen Spring Security tunnisti, että käyttäjä ei ollut autentikoituna JSESSIONID-evästeen puututtua ja aloitti autentikoitumisen Keycloakin asiakassovelluksen konfiguraatiossa määritellyllä valtuutuskoodi prosessilla. Yhdyskäytäväpalvelu ohjasi Keycloakin kirjautumissivulle, jonne syötettiin shop-user käyttäjätunnus ja salasana. Onnistuneen autentikoinnin jälkeen, ja poletit saatuaan, Spring Security tallensi käyttäjän autentikoinnin tiedot SecurityContext-rajapintaan, ja asetti selaimeen evästeen JSESSIONID. Sen jälkeen yhdyskäytäväpalvelu palautti määritellyt tiedot käyttäjän autentikoinnista Kuvion 25 mukaisesti. Autentikointi OpenID Connect -protokollan avulla todettiin toimivaksi.


```
ID: 1
Name: Shop User
Email: shop-user@shop.com
Roles: ROLE_USER
ID Token:
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI0YVQ1cHl0Sm5WOElNaUtDSWZjY3F1NUxkZ2ZWVXZveHhybkh2QWJWVWV3In0.
eyJleHAiOjE2MDYyNTUyNzQsImlhdCI6MTYwNjI1NDk3NCwiYXV0aF90aW1lIjoxNjA2MjU0NTEwLCJqdGkiOiI3ZjhiM2Q2MS1hZDUzLTQxOTIt
OGJmYy04Yjg0OTg1ZDkwYjgiLCJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjgwODAvYXV0aC9yZWFsbXMvc2hvcCIsImF1ZCI6InNob3AtY2xpZW50
Iiwic3ViIjoiMTNjMWUyMTMtMmMxYi00MTZjLThlMTItNDUxNDZlMjMwNTJmIiwidHlwIjoiSUQiLCJhenAiOiJzaG9wLWNsaWVudCIsIm5vbmNl
IjoiNG9qYjMweFh0WG9pcDFwYmF4cnZDQ2tlVmdLd1k3c1VKcDZ2QXZjVkx2USIsInNlc3Npb25fc3RhdGUiOiIwODRiZDk1OC0wZjVlLTRkMTUt
YTE0MS1jYmZhMWEzYWY4NWQiLCJhdF9oYXNoIjoiaTNJRlBEcERSblBWS2ZCS2tGMTNsQSIsImFjciI6IjAiLCJuYW1lIjoiU2hvcCBVc2VyIiwi
cHJlZmVycmVkX3VzZXJuYW1lIjoic2hvcC11c2VyIiwiaWQiOiIxIiwiZ2l2ZW5fbmFtZSI6IlNob3AiLCJmYW1pbHlfbmFtZSI6IlVzZXIiLCJl
bWFpbCI6InNob3AtdXNlckBzaG9wLmNvbSJ9.co4HzIG6Cl3QnvBgCq01d-_4j7mD7hKnmQr2rV_14uTX1PRZkpHsdcRvfe-trQFVxNi3hKJNRw-
RUiph5pynpdVpT3tZGBdWZpSM0iQsHt1PMsNkI9lUo8wES7M9FTOB-33pnTUv0AH_-
hUSVib6yS8DSngMip56Izq3nbjZxz7AzG544SQvwzzMmJJz2AHzkKzvRj8sFJtMSO5KOwCAFXI6uD2nQFM0x0-NEh1_sWmVsPlUHKET_zZSVz-
7sIxZZWoAj0pVEg9GlxRgeXUe9R-PjNeAHNXjmOQCOiYF7_eb8Icl4Or2RUgCoNjib7_Z9cUTAci_Z-mUyb91yVoUkg
Access token:
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI0YVQ1cHl0Sm5WOElNaUtDSWZjY3F1NUxkZ2ZWVXZveHhybkh2QWJWVWV3In0.
eyJleHAiOjE2MDYyNTUyNzQsImlhdCI6MTYwNjI1NDk3NCwiYXV0aF90aW1lIjoxNjA2MjU0NTEwLCJqdGkiOiJiZTFlNGZmYy00OWZlLTQ0ZDYt
YTM3OC00MDMzYTNiMmU5ZDUiLCJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjgwODAvYXV0aC9yZWFsbXMvc2hvcCIsImF1ZCI6ImFjY291bnQiLCJz
dWIiOiIxM2MxZTIxMy0yYzFiLTQxNmMtOGUxMi00NTE0NmUyMzA1MmYiLCJ0eXAiOiJCZWFyZXIiLCJhenAiOiJzaG9wLWNsaWVudCIsIm5vbmNl
IjoiNG9qYjMweFh0WG9pcDFwYmF4cnZDQ2tlVmdLd1k3c1VKcDZ2QXZjVkx2USIsInNlc3Npb25fc3RhdGUiOiIwODRiZDk1OC0wZjVlLTRkMTUt
YTE0MS1jYmZhMWEzYWY4NWQiLCJhY3IiOiIwIiwicmVhbG1fYWNjZXNzIjp7InJvbGVzIjpbInNob3AtdXNlciJdfSwic2NvcGUiOiJvcGVuaWQg
cHJvZmlsZSBlbWFpbCIsImlkIjoiMSJ9.TV6tf0YH0UhUMF9S75nexRdS-
dzp1RB3ZhN8he7mpDyNyZKP4jI3MH0JvZneKrR90skn8unaUoc9JVgI3uB4OxQtGCvHZ1yAfvKRE-
TXo2sddsV98nN5KkEakx5UA8hCADFRkGKNZ3Uu51J6BSDgwLqhKW405R8BWCnA3qmgK6njl5_mSPUx_TmHF47qdsN-
wMsOIKvxzM83PtiGWSgeHNeIEssNf68ojkqcBPbJjpLKyxPsU0LQfXugIk-HiapUU8A90plisvAujWEbm-fEA04CHWmMe7MXuGDuVRwI1nq8-
b2cBjrCHm9ZLJtwp--2FScgx1WuCTUYo95K7ykZ_A
```


Kuvio 25. Käyttäjän autentikointitiedot yhdyskäytäväpalvelussa

Tekemällä pyyntö Admin mikropalvelun /get-orders päätepisteeseen yhdyskäytävä-palvelu ei ohjannut uudelleen autentikoitumaan, koska aiemmin kirjauduttiin shop-user käyttäjällä, ja valtuutuspoletti oli vielä voimassa. Spring Security lisäsi onnistuneesti mikropalveluille menevien kutsujen mukaan Authorization-otsakkeen, joka sisälsi Keycloakilta saadun valtuutuspoletin. Valtuutuspoletti sisälsi autentikoituneen käyttäjän roolit, sekä id attribuutin. Koska shop-user käyttäjällä ei ollut tarvittavia rooleja, admin mikropalvelu vastasi tilakoodilla 403. Tekemällä POST-pyyntö Postman-sovelluksella order mikropalvelun /add päätepisteeseen, joka vaatii shop-user roolin, se vastasi samalla viestillä takaisin Kuvion 26 mukaisesti roolivaatimuksien täyttyessä.



Kuvio 26. Order mikropalvelun vastaus

Jotta käyttäjää saatiin vaihdettua, tyhjennettiin selaimen evästeet. Sen jälkeen tehtäessä pyyntö admin mikropalvelun /get-orders päätepisteeseen, yhdyskäytäväpalvelu ohjasi uudelleen autentikoitumaan JSESSIONID-evästeen puututtua. Kirjautumissivulle syötettiin shop-admin kirjautumistiedot, jolloin päätepiste palautti kaikki order mikropalvelussa olevat tilaukset admin mikropalvelun kautta onnistuneesti roolivaatimuksien täyttyessä. Tehtäessä POST-pyyntö Postman-sovelluksella admin mikropalvelun /add-products päätepisteeseen, admin mikropalvelu vastasi samalla viestillä takaisin roolivaatimuksien täyttyessä. Todettiin, että päätepisteiden valvonta, sekä valtuutuspolettien kuljettaminen mikropalveluiden välillä toimii.

Yhteenvetona voitiin todeta, että OpenID Connectin ja sen valtuutuskoodi autentikointiprosessin avulla pystyttiin autentikoimaan käyttäjä turvallisesti yhdyskäytäväpalvelussa niin, että ympäristössä ei tarvinnut käsitellä sen kirjautumistietoja ollenkaan. Valtuutuspoletin avulla pystyttiin hallinnoimaan mikropalveluiden päätepisteiden pääsynvalvontaa. Toimeksiantajalle saatiin luotua esimerkkitoteutus, sekä dokumentaatio aiheesta, jota se voi halutessaan käyttää tulevissa projekteissa.

5 Pohdinta

OpenID Connect -protokolla toimi hyvin mikropalveluympäristössä. Sen tarjoama valtuutuskoodi autentikointiprosessi takasi turvallisen käyttäjän autentikoimisen yhdyskäytäväpalvelussa niin, että ympäristössä ei tarvinnut käsitellä käyttäjän tunnuksia ollenkaan. Missään vaiheessa poletit eivät myöskään paljastuneet käyttäjälle. Valtuutuspoletti JWT muodossa sopi oikein hyvin mikropalveluiden päätepisteiden pääsynvalvontaan. OpenID Connectin tarjoama identiteettipoletti jäi tässä toteutuksessa hyödyntämättä, sillä sen hyöty olisi ollut asiakassovelluksessa joka mikropalveluympäristöä käyttää. Myös polettien uusiminen jäi työssä ja toteutuksessa huomioimatta, vaikka OpenID Connect sellaisen mahdollisuuden tarjoaa.

OpenID Connect -protokollan implementointi mikropalveluympäristöön oli valmiilla työkaluilla helppoa.

Spring Securityn valmiiksi tarjoamat implementaatiot, sekä valmis OpenID Connect -tarjoaja Keycloak mahdollistivat helpon ja nopean tavan ottaa OpenID Connect käyttöön mikropalveluympäristössä. Oman OpenID Connect -tarjoajan rakentaminen, sekä ympäristössä autentikointiprosessien rakentaminen ja polettien validointi olisi ollut luultavasti liian työlästä.

Jos toimeksiantajan tulevissa projekteissa tarvitaan identiteetin- sekä pääsynvalvonnan ratkaisua, on OpenID Connect hyvä valinta siihen. Lisäksi jos projektin pohjalla käytetään jo Spring Security -sovelluskehystä, on tämän opinnäytetyön tulokset hyödynnettävissä.

Lähteet

Bihis, C. 2015. Mastering OAuth 2.0. Packt Publishing.

Bradley, J., De Medeiros, B., Google., Jones, M., Microsoft., Mortimore, C., NRI., Ping Identity., Sakimura, N., & Salesforce. 2014. OpenID Connect -spesifikaatio. Viitattu 13.3.2019. https://openid.net/specs/openid-connect-core-1_0.html.

Bradley, J., Jones, M., Microsoft., NRI., Ping Identity., Sakimura, N. 2015. JWT standardi. Viitattu 12.3.2019. https://tools.ietf.org/html/rfc7519.

Hardt, D., Microsoft. OAuth 2.0 standardi. Viitattu 2.4.2019. https://tools.ietf.org/html/rfc6749.

Tietoja meistä. N.d. Tieto osio yrityksestä Combitech Oy:n verkkosivuilla. Viitattu 7.2.2019. https://www.combitech.fi/tietoja.

Sanso, J.R.S. 2017. OAuth 2 in Action. Manning Publications.