Ville Juutila

MIKROPALVELUARKKITEHTUURI-POHJAINEN HEADLESS-VERKKO-KAUPPARATKAISU

Opinnäytetyö

Liiketalouden ammattikorkeakoulututkinto

Tietojenkäsittely

2022

Kaakkois-Suomen ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Tradenomi (AMK)
Tekijä	Ville Juutila
Työn nimi	Mikropalveluarkkitehtuuripohjainen headless-verkkokaupparatkaisu
Toimeksiantaja	Metatavu Oy
Vuosi	2022
Sivut	46 sivua
Työn ohjaaja	Janne Turunen

TIIVISTELMÄ

Tässä opinnäytetyössä oli tavoitteena selvittää, miten Saleor-verkkokaupparatkaisuun voidaan toteuttaa integraatioita muihin mikropalveluihin. Työn edetessä toteutettiin integraatio käyttäjänhallinnan ulkoistamiseksi Saleorin ja Keycloakin välille, sekä integraatio tuotetietohallintajärjestelmän ja Saleorin välille. Tuotetietohallintajärjestelmä-integraatiota varten kehitin itse sovelluksen, joka vie tuotetietoja tuotetietohallintajärjestelmästä Saleoriin. Käytännön toteutuksen lisäksi työssä perehdyttiin työn aikana esille nousseita oleellisia käsitteitä ja teknologioita.

Toimeksiantajayritys halusi kehittää Saleor-liiketoimintaansa. Taustalla oli tarve tutustua Saleoriin tarkemmin mm. sen laajennettavuuden osalta. Toimeksiantajayritys oli tunnistanut selkeän markkinatarpeen rajapintapohjaiselle ja laajennettavalle verkkokauppa-alustalle. Keskeiset asiakastarpeet ovat liittyneet mm. identiteetin- ja pääsynhallinnan sekä tuotetietojen integraatioihin. Saleorin perustuminen avoimeen lähdekoodiin sopii toimeksiantajayrityksen arvoihin ja suosimiin toimintatapoihin. Avointa lähdekoodia suosittiin myös jokaisessa tässä työssä käytetyssä sovelluksessa ja palvelussa.

Tämän opinnäytetyön tuloksena syntyi soveltuvuusselvityksen määritelmän täyttävä prototyyppi. Se helpottaa tulevaisuudessa mahdollisten Saleor-integraatioiden toteuttamista. Työn tuloksia testattiin vielä työn lopussa käytännössä. Testaus suoritettiin kirjautumalla sisään Saleoriin Keycloakin kautta ja tekemällä testiostos tuotteella, joka on tuotu Saleoriin tuotetietohallintajärjestelmästä toteuttamallani tuotetietohallintajärjestelmä-integraatiosovelluksella.

Työssä esitelty tapa integraatioiden toteuttamisesta on vain yksi esimerkki. Työn edetessä opitaan, että integraatioita on mahdollista toteuttaa lukematomin eri tavoin.

Asiasanat: ohjelmointi, ohjelmistoarkkitehtuuri, verkkokauppa, Saleor, headless

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Business Administration
Author	Ville Juutila
Thesis title	Microservice architecture based headless e-commerce solution
Commissioned by	Metatavu Oy
Time	2022
Pages	46 pages
Supervisor	Janne Turunen

ABSTRACT

The objective of this thesis was to examine the possibilities of integrating Saleor e-commerce platform with other microservices. During the process of this thesis, two integrations were made: one between Saleor and Keycloak to outsource user management and one between Saleor and a product information management system. For product information management system integration, I developed a software that imported products from a product information management system to Saleor. In addition to these integrations, essential concepts and technologies were studied.

The thesis commissioner wanted to improve their Saleor business. There was a need to explore Saleor more deeply, especially it's expandability. The commissioner had recognized a clear need in the e-commerce field for an API-based and expandable e-commerce platform. Essential customer needs have consisted of identity and access management and product information integrations. Among other things, Saleor is an open-source software and therefore suited well with the commissioner's principles and preferences. Open-source was also preferred in other software used in this thesis, and therefore all software used were open-source as well, including the product information management integration application that was developed in this thesis.

As a result of this thesis a proof-of-concept prototype was successfully developed. In the future it will hopefully help implementing possible Saleor integrations. The results were also evaluated in the end of this thesis. Tests were done by logging into Saleor via Keycloak and completing a test purchase of a product imported from the product information management system.

The approach of implementing Saleor integrations explained in this thesis was merely one example. This thesis process showed that implementing integrations into Saleor is possible in countless ways.

Keywords: programming, software architecture, e-commerce, Saleor, headless

SISÄLLYS

1 JOHDANTO

Opinnäytetyön tavoitteena on selvittää, miten olemassa oleva käyttäjähallinta- ja tuotetietohallintajärjestelmä integroidaan Saleor-verkkokaupparatkaisuun. Tavoitteena ei ole saada tuotettua valmista ratkaisua asiakkaalle vaan prototyyppi, jonka päälle mahdollisia tulevia ratkaisuja voidaan lähteä suunnittelemaan.

Tämän työn toimeksiantaja, Metatavu Oy, on mikkeliläinen ohjelmistokehitystä ja -ylläpitoa tarjoava yritys. Yritys pyrkii suosimaan töissään avoimen lähdekoodin ratkaisuja, johon myös tässä työssä pyritään. Yritys on halunnut saada selvitettyä mahdollisuuksia erilaisten integraatioiden toteuttamiseen Saleor-verkkokaupparatkaisuun, jota tässä opinnäytetyössä pyritään selvittämään.

Opinnäytetyön varsinainen raportointiosuus koostuu neljästä osasta. Kahdessa ensimmäisessä luvussa (luvut 2 ja 3) käsitellään työn toteutuksen kannalta oleellisia käsitteitä ja teknologioita. Luvussa 2 käsitellään viittä eriä tässä työssä esille nousevaa teknologiaa ja toteutustapaa. Luvussa 3 syvennytään käyttäjänhallintaan. Käyttäjänhallintaan tutustuttaessa tutustutaan myös tapaan, jolla tässä työssä tullaan toteuttamaan verkkokaupan käyttäjien autentikaatio ja auktorisointi.

Luvussa 4 suoritetaan Saleorin, ja käyttäjänhallinnan, eli Keycloakin, käyttöönotot. Molempien sovelluksien käyttöönotto raportoidaan vaiheittain ja käytettyihin tekniikoihin perehdytään. Käyttöönottojen jälkeen luvussa 5 toteutetaan ensin Saleorin ja Keycloakin välinen integraatio, jossa raportoidaan niin sovelluksien asetuksien määrittelemistä, kuin Saleorin asiakaskäyttöliittymän ohjelmointiakin. Seuraavaksi pohditaan tähän työhön soveltuvaa tuotetietohallintajärjestelmää, suoritetaan valitun tuotetietohallintajärjestelmän käyttöönotto sekä toteutetaan Saleorin ja tuotetietohallintajärjestelmän välinen integraatio, erillisenä sovelluksena. Viimeisenä tarkastellaan toteutettujen integraatioiden toimivuus käytännössä, suorittamalla testiostos Saleorissa. Viimeinen luku (luku 6) on päätäntö. Päätännössä pohditaan työn tavoitteiden saavuttamista, jatkokehitysideoita sekä omia ja toimeksiantajan mielipiteitä tästä työstä.

2 KESKEISET KÄSITTEET

2.1 Mikropalveluarkkitehtuuri

Mikropalveluarkkitehtuurilla tarkoitetaan tietojärjestelmäarkkitehtuuria, jonka osat on jaettu itsenäisesti toimiviin ja pieniin palveluihin (Hyvärinen 2019). Kyseistä arkkitehtuurimallia on viime vuosina hehkutettu paljon. Pitkäikäiseksi suunniteltujen vaativien ja laajojen ratkaisujen toteutustapaa mietittäessä voi mikropalveluarkkitehtuuri nousta esiin potentiaalisena lähestymistapana.

Mikropalveluarkkitehtuurissa määrittelemättömän kokoinen joukko pieniä palveluita muodostavat suuremman yhtenäisen kokonaisuuden, jossa palvelut kommunikoivat keskenään ohjelmointirajapintojen avulla. Esimerkki mikropalveluarkkitehtuurimallin mukaisesta sovelluksesta voisi olla verkkokauppa, jonka käyttäjätietojen hallinnasta vastaa yksi ja tuotetietojen hallinnasta toinen mikropalvelu. Toki myös käyttöliittymä on oma mikropalvelunsa.

Mikropalveluarkkitehtuuri on vikasietoisempi ja suorituskykyisempi kuin monoliittinen ratkaisu. Mikropalveluilla voi mahdollisesti olla omat välivarastonsa, jotka osaltaan mahdollistavat suorituskykyisemmän palvelun tarjonnan asiakkaalle. Myöskään yksittäisen palvelun käyttökatko ei automaattisesti estä muun palvelun käyttöä. (Hyvärinen2019.) Lisäksi Hyvärisen (2019) mukaan mikropalveluarkkitehtuurimalli mahdollistaa paremmin useamman kehitystiimin samanaikaisen kehittämisen. Toisistaan eriytetyt mikropalvelut helpottavat myös sitä, jos jossain vaiheessa sovelluksen elinkaaren aikana jokin palvelu halutaan, vaikka vaihtaa johonkin toiseen.

Toki mikropalveluarkkitehtuuri ei sovellu jokaiseen tilanteeseen. Kulloiseenkin sovelluskehitysprojektiin sopivinta arkkitehtuurimallia tulee aina arvioida ja pohtia tapauskohtaisesti. Hyvärisen (2019) mielestä mikropalveluarkkitehtuuri ei välttämättä ole paras mahdollinen vaihtoehto, jos sen vaatimia investointeja ei olla huomioitu hankkeen budjetoinnissa.

2.2 Ohjelmointirajapinta

Ohjelmointirajapinta (engl. Application Programming Interface, API) on eräs keino, jolla mahdollistetaan eri sovelluksien kommunikointi keskenään. Yksi

esimerkki ohjelmointirajapinnan toiminnasta voisi olla se, kun käyttäjä avaa Facebookin. Sivuston staattisia elementtejä (navigointipalkit yms.) lukuun ottamatta kaikki mitä käyttäjälle näytetään, haetaan palvelinsovellukselta ohjelmointirajapinnan kautta.

Rajapintaa voidaan ajatella ikään kuin palvelusopimuksena kahden sovelluksen välillä. Palvelusopimus määrittelee kuinka sovellukset kommunikoivat keskenään pyynnöin ja vastauksin. Kunkin palvelun rajapintadokumentaatio tarkentaa edellä mainittujen pyyntöjen ja vastausten rakenteet. (Amazon Web Services 2022a.)

Ohjelmointirajapintoja on useita erilaisia, mutta tässä työssä tullaan käyttämää REST- ja GraphQL-rajapintoja. Molemmat näistä käyttävät HTTP(S)-protokollaa kommunikoimiseen. REST-rajapinnat voivat käyttää kaikkia HTTP-metodeja, kun taas GraphQL-rajapinnat käyttävät vain GET- ja POST-metodeja.

Red Hatin (2020b) mukaan REST-rajapinta ei ole standardi tai protokolla, vaan pikemminkin arkkitehtuurityyli. REST-rajapinnassa määritellään, millä HTTP-metodeilla rajapinnan eri päätepisteisiin (engl. endpoint) voidaan tehdä pyyntöjä. Asiakas- ja palvelinsovellus kommunikoivat siis keskenään HTTP-protokollan mukaisesti. REST:n perusajatus on, että se on tilaton. (Amazon Web Services 2022a.) Tilattomuus tarkoittaa sitä, ettei palvelinsovellus tallenna asiakassovelluksen dataa pyyntöjen välissä. Pyynnöt ovat siis toisistaan riippumattomia, eikä edellinen pyyntö vaikuta seuraavaan. (Red Hat 2020.) REST-rajapinnat reagoivat ennalta määrättyihin päätepisteisiin eri tavoin. Esimerkiksi HTTP -standardin mukainen GET-pyyntö /comments-päätepisteeseen voisi palauttaa asiakassovellukselle kaikki kommentit, kun taas POST-pyyntö samaan päätepisteeseen voisi lisätä uuden kommentin. REST-rajapinnat palauttavat vastauksensa asiakassovellukselle ennalta määrätyssä muodossa. Tällöin monesti syntyy tilanteita, joissa asiakassovellus saa ns. turhaa dataa.

Vähemmän yleinen rajapinta tyyppi on GraphQL. GraphQL on vuonna 2012 Facebookin julkaisema avoimen lähdekoodin rajapinta kyselykieli (GraphQL 2022). GraphQL:n perusajatus on, että asiakassovellukselle palautetaan aina

vain se data mitä se kulloinkin pyytää. Siinä missä REST-rajapinnassa päätepisteet, sekä niiden palauttama data, on tarkasti määritelty, niin GraphQL-rajapinnassa ei ole kuin yksi päätepiste, johon asiakassovellus tekee pyyntöjä. GraphQL:ää käyttävät sovellukset voivat olla nopeita hitaalla mobiiliyhteydelläkin (GraphQL 2022).

Myös GraphQL-pyynnöt tapahtuvat HTTP-protokollan mukaisesti, mutta jokainen pyyntö on joko GET- tai POST-pyyntö. Asiakassovelluksen pyytäessä dataa rajapinnalta määrittelee se pyyntöön mitä dataa se kulloinkin tarvitsee. Tällöin vältytään REST:n tilanteelta, jossa saadaan turhaa dataa.

2.3 Headless

Mencelin (2020) mukaan headless on tulevaisuudenkestävin ja tehokkain tapa verkkosovellusten rakentamiseksi. Headless on terminä hieman kummallinen, mutta sen käyttö on vakiintunut alalla (Stalians 2022). Myös suomenkielisessä viestinnässä headless on terminä vakiintunut.

Aikana ennen mobiili- ja älylaitteita ainoa keino tehdä ostoksia verkkokaupassa oli tietokoneella, ja yrityksen verkkosivustojen kautta. Tällöin verkkosivuston presentaatio-taso, jota käyttäjä käyttää ja näkee, on niin sanotusti pää (engl. head). Pää on osana taustapalvelua, joka sisältää varsinaiset verkkokaupan toiminnot, kuten tuotetiedot, ostoskorin ja maksutoiminnallisuudet. (Stalians 2022.) Headless-arkkitehtuurin mukaisesti taas pää on irrotettu omaksi mikropalvelukseen (Vue 2022.)

Nykyään käyttöalustoja tietokoneen lisäksi verkkokaupalle on useita, kuten mobiili- ja älylaitteet. Jokainen käyttöalusta voi tarvita omanlaisensa presentaatio-tason, joka täytyy saada kytkettyä taustapalveluihin. Headlessin ajatuksena on, että jokainen käyttöliittymä voidaan kytkeä kiinni samoihin taustapalveluihin ohjelmointirajapintojen avulla. Tällöin jokaiselle alustalle voidaan toteuttaa halutun lainen käyttöliittymä, joka hyödyntää jo olemassa olevaa taustapalvelua varsinaisten verkkokauppatoimintojen tuottamiseksi. (Stalians 2022.)

Headless-arkkitehtuuri esiintyy edukseen, kun tarvitsee tehdä integraatioita. Se mahdollistaa integraatioiden tekemisen saumattomasti muiden palveluiden, kuten asiakkuudenhallinta- (engl. Customer Relationship Management, CRM), toiminnanohjaus- (engl. Enterprise Resource Planning, ERP) ja tuoteinformaatiojärjestelmien (engl. Product Information Management, PIM) välille. (Mencel 2022.)

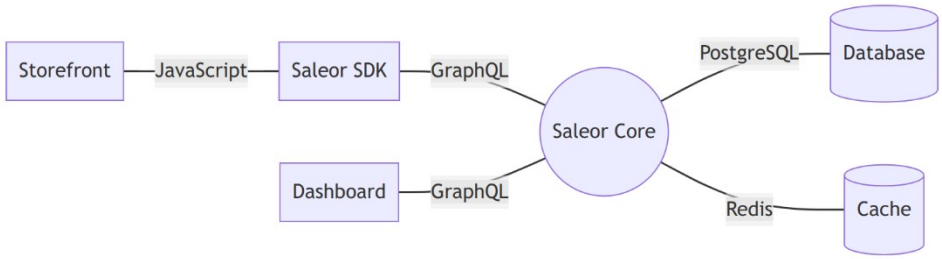
Radhakrishanin (2021) mukaan mikropalveluarkkitehtuurista on puhuttu vuodesta 2010-luvulta alkaen, ja vuonna 2020 tehdyn kyselyn perusteella 61% verkkokaupoista joko käyttävät, tai suunnittelevat, headless-ratkaisua. Tästä voitaisiin päätellä headless-ratkaisujen yleistyvän eri päätelaitteiden yleistyessä. Tässä osiossa käsiteltiin headless-termiä verkkokaupan näkökulmasta, mutta samalla menetelmällä voidaan toteuttaa muitakin verkkosovelluksia.

2.4 Saleor

Saleor on verkkokauppa-alusta. Se mahdollistaa modernien verkkokauppojen rakentamisen. Ylläpitäjän hallintapaneelin kautta voi helposti hallinnoida tuotteita, henkilöitä ja toiminnallisuuksia. Saleorin avulla voi ratkaista kaiken kokoisten jälleenmyyjien tarpeet. (Saleor 2022a.)

Saleorin (2022a) mukaan heidän selkeä etunsa on se, että sen ollessa headless, on sama verkkokauppa helppo toteuttaa sekä tyypillisenä verkkokauppana, että progressiivisena verkkosovelluksena (engl. Progressive web app, PWA). Progressiivinen verkkosovellus mahdollistaa laitteiden natiivien ominaisuuksien käyttämisen mobiililaitteen verkkoselaimessa. Tällöin sovellus tuntuu enemmän mobiilisovellukselta kuin verkkosivustolta. (Saleor 2022a.)

Saleor perustuu avoimeen lähdekoodiin ja on täten täysin ilmainen. Sen ydin on kirjoitettu Python-ohjelmointikielellä, Django-viitekehystä käyttäen. Kommunikaatio käyttöliittymän ja palvelinsovelluksen välillä tapahtuu GraphQL-ohjelmointirajapinnan avulla. (Saleor 2022a.) Saleor tarjoaa sellaisenaan valmiin palvelinsovelluksen sekä asiakas- ja hallintakäyttöliittymän, jota kukin voi muokata haluamakseen. Seuraavassa kuvassa esitellään Saleorin arkkitehtuuria tarkemmin.



Kuva 1. Saleorin arkkitehtuuri (Saleor 2022c)

Kuvan 1 keskellä on Saleorin ydin, ns. palvelinsovellus, joka tarjoaa GraphQL-ohjelmointirajapinnan. Se ei sisällä käyttöliittymää ja on kirjoitettu Python-ohjelmointikielellä. Tiedon tallentamiseen se käyttää PostgreSQL-relaatiotietokantaa. Joitain asioita se tallentaa välimuistiin Rediksen avulla, sen ollessa mahdollista. (Saleor 2022a.)

Seuraavana vasemmalla on hallintakäyttöliittymä. Sen avulla verkkokaupan henkilökunnan jäsenet ja ylläpitäjät voivat hallinnoida verkkokauppaa. Saleorin mukana tuleva hallintakäyttöliittymä on verkkoselaimessa toimiva React-sovellus. (Saleor 2022a.)

Viimeisenä on varsinainen verkkokauppa käyttöliittymä. Mukana tuleva käyttöliittymä on Next.js-viitekehykseen kirjoitettu React-sovellus. (Saleor 2022a.) Kumpaakin käyttöliittymää voi kustomoida haluamakseen ja molemmat kommunikoivat palvelinsovelluksen kanssa GraphQL-ohjelmointirajapinnan avulla HTTPS-standardien mukaisesti.

Saleorin ydintä on mahdollista muokata ja laajentaa haluamakseen. Dokumentaation mukaan ei ole kuitenkaan suotavaa muokata ydintä suoraan, sillä tämä voi vaikeuttaa mahdollisten virallisten päivitysten asentamista myöhemmin. Sen sijaan laajentaminen on suositeltavaa liitännäisten (engl. plugin) ja laajennussovellusten (engl. app) avulla.

Laajennussovellukset ovat erillisiä verkkosovelluksia, jotka toimivat webhookien ja palvelinsovelluksen GraphQL-ohjelmointirajapinnan avulla. Laajennussovelluksille voidaan myöntää pääsyoikeuksia tarpeen mukaan ja ne voivat

suorittaa miltei kaikki samat toiminnot mitä henkilökunnan jäsenetkin. Ne voivat myös saada tiedon erilaista tapahtumista webhookien avulla. (Saleor 2022a.) Laajennussovellus voisi olla esimerkiksi palvelu, joka lähettää verkkokaupan käyttäjälle vahvistusviestin sähköpostitse onnistuneen ostotapahtuman seurauksena.

Liitännäisten avulla on mahdollista suorittaa omaa koodia samassa prosessissa Saleorin kanssa. Liitännäisillä on suora pääsy mm. Saleorin käyttämään PostgreSQL-tietokantaan. Liitännäisten käyttäminen voi sisältää tietoturvariskejä, ja täten on suositeltavaa käyttää vain luotettavien lähteiden liitännäisiä. Saleorin tarjoamassa SaaS-alustassa (Software as a Service) ei ole turvallisuussyistä mahdollista käyttää liitännäisiä. (Saleor 2022a.)

2.5 Konttiteknologia

Kontittamisella (engl. containerization) tarkoitetaan sitä, että kaikki sovelluksen ajamiseen tarvittavat riippuvuudet, kuten kirjastot ja viitekehykset pakataan yhteen konttiin (Red Hat 2021a). Kontti on vakiintunut yksikkö, jossa sovellus toimii luotettavasti laiteympäristöriippumattomasti (Docker 2022).

Konttiin pakattua sovellusta voidaan ajaa missä tahansa käyttöjärjestelmässä ja -ympäristössä. Konttia voidaan ajatella eräänlaisena itsenäisenä kuplana. (Red Hat 2021a.)

Konttiteknologia tarjoaa vaihtoehdon sille, että sama ohjelma tarvitsee kirjoittaa uudestaan eri käyttöjärjestelmille. Se ehkäisee käännöksen aikana syntyvien virheiden ja bugien riskiä. Täten myös lisää tuottavuutta, kun samaa ohjelmaa ei tarvitse kirjoittaa useaan kertaan, eikä käännöksestä aiheutuvia bugeja joudu korjaamaan. (Red Hat 2021a.)

Konttiteknologia ja virtuaalikoneessa ajaminen ovat ajatukseltaan saman suuntaisia siinä, että molemmat tarjoavat eristetyn ympäristön ohjelman suorittamiselle. Suurimmat erot näiden kahden välillä ovat tiedostokoossa ja siirrettävyydessä. (Red Hat 2021a.) Kontit virtualisoivat käyttöjärjestelmän, kun taas virtuaalikoneet virtualisoivat fyysistä laitteistoa (Docker 2022).

3 KÄYTTÄJÄNHALLINTA

Tässä luvussa käsitellään käyttäjänhallinnan toteuttamista identiteetin- ja pääsynhallintaa tarjoavien palveluiden avulla, sekä käyttäjän ja tämän käyttöoikeuksien tunnistamista. Käyttäjätunnukset, salasanat ja käyttövaltuudet ovat tietojärjestelmien ja -verkkojen käyttäjille tuttuja. Käyttäjällä voi olla lukuisia toisistaan erillisiä käyttäjätunnus-salasana-yhdistelmiä eri palveluihin, joita tämä joutuu käyttämään sekä yksityishenkilönä, että organisaation jäsenenä. (Linden 2017a.)

3.1 Identiteetin- ja pääsynhallinta

Identiteetin- ja pääsynhallinta (engl. Identity and access management, IAM) on turvallisuussääntö, joka mahdollistaa oikeiden entiteettien, henkilöiden tai asioiden, käyttää oikeita resursseja häiriöttä, haluamillaan laitteilla. Identiteetin- ja pääsynhallinta koostuu järjestelmistä ja prosesseista, jotka tarjoavat järjestelmäylläpitäjille mahdollisuuden määrittää digitaalisen identiteetin jokaiselle entiteetille, autentikoida heidät sisäänkirjautumisen yhteydessä, auktorisoida oikeisiin resursseihin ja hallinnoida ja monitoroida näitä identiteettejä niiden elinkaarensa aikana. (IBM 2022.)

Identiteetin- ja pääsynhallinta ei ole enää vain yrityksen työntekijöitä varten. Organisaatioiden tulee voida tarjota turvalliset käyttöoikeudet myös muun muassa alihankkijoilleen, liikekumppaneilleen, mobiilikäyttäjille, sekä asiakkailleen. Digitaalisen murroksen myötä myös esineiden internetin (engl. Internet of Things, IoT) laitteet, robototit ja ohjelmointirajapinnat tarvitsevat omat digitaaliset identiteettinsä. (IBM 2022.)

Tavanomaisissa turvallisuusjärjestelmissä on usein yksi hajoamispiste, salasana. Jos käyttäjän salasana, tai vielä pahempaa, käyttäjän salasanan palauttamiseen käytettävä sähköpostiosoite murretaan, voi koko organisaatiosta tulla haavoittuvainen hyökkäyksille. Identiteetin- ja pääsynhallinta voi kaventaa hyökkäämispintaa ja ehkäistä hajoamisia tarjoamalla työkaluja vahinkojen huomaamiseen niiden sattuessa. (Onelogin 2022.)

Käyttäjien kirjautuessa kerran järjestelmään yhden identiteetin- ja pääsynhallinnan portaalin kautta ei heidän tarvitse enää murehtia oikeiden salasanojen

muistamisesta, eikä oikeista käyttöoikeuksista tehtäviensä suorittamiseen. Käyttäjät eivät ainoastaan saa käyttöönsä täydellisiä työkaluja tehtäviinsä, vaan heidän pääsyään resursseihin voidaan hallinnoida käyttäjäryhmien tai –roolien mukaan, yksittäisten henkilöiden sijasta. Tämä vähentää järjestelmäylläpitäjien työkuormaa, ja parantaa työntekijöiden tuottavuutta. (Onelogin 2022.)

Tässä opinnäytetyössä tullaan identiteetin- ja pääsynhallinta toteuttamaan Keycloak nimisellä sovelluksella. Vitalen (2019) mukaan Keycloak on RedHat Communityn ylläpitämä avoimen lähdekoodin identiteetin- ja pääsynhallinnan järjestelmä. Se tarjoaa laajat ominaisuudet, kuten kertakirjautumisen (engl. Single Sign-on, SSO), autentikaation ja auktorisoinnin, sosiaalisen median avulla kirjautumisen, monivaiheisen tunnistautumisen ja keskitetyn käyttäjänhallinnan. Keycloak tekee sovelluksien ja palveluiden suojaamisesta helppoa pienellä tai olemattomalla ohjelmoinnilla (Keycloak 2022a).

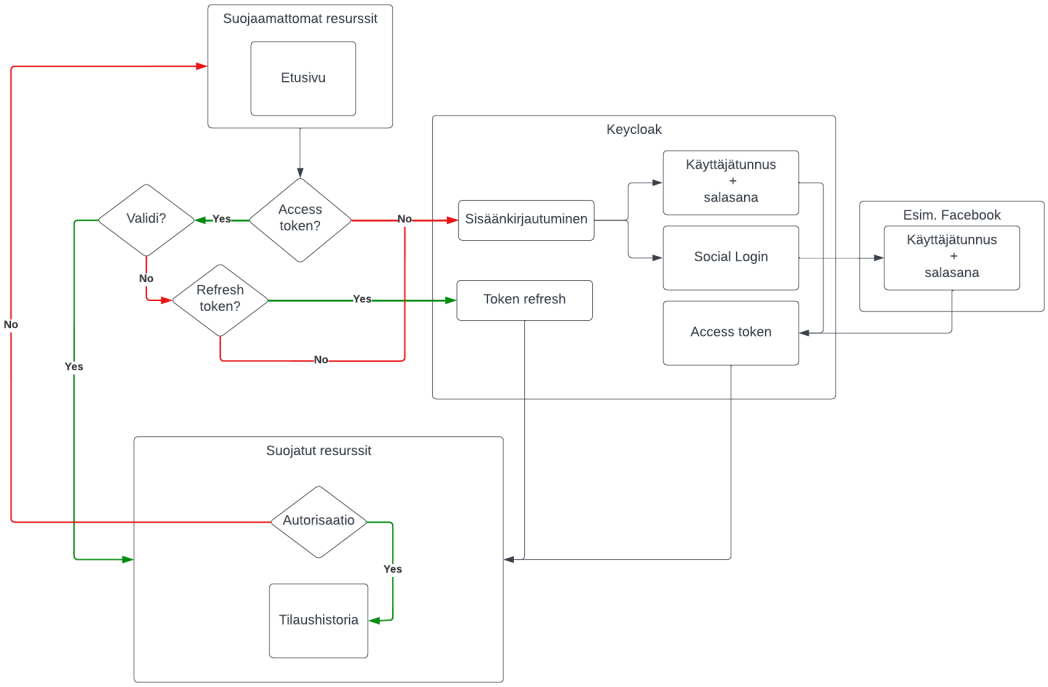
3.2 Autentikaatio ja auktorisointi

Autentikaatio (engl. authentication) ja auktorisointi (engl. authorization) ovat käyttäjän- ja identiteetinhallinnan yhteydessä usein kuultuja termejä. Vaikka edellä mainittuja termejä käytetään usein keskenään vaihdettuina, tarkoittavat ne kuitenkin pohjimmiltaan erilaisia toimintoja. Lyhyesti sanottuna, pääsy resurssiin on suojattu sekä autentikaatiolla, että auktorisoinnilla. Jos käyttäjä ei voi todentaa identiteettiään, ei tälle myönnetä pääsyä resurssiin. Vaikka käyttäjä voisi todentaa identiteettinsä, mutta häntä ei ole auktorisoitu kyseiseen resurssiin, ei tälle silloinkaan myönnetä pääsyä. (Auth0 2022a.)

Yksinkertaisilla termeillä kuvattuna, autentikaatio on prosessi, jossa todennetaan, kuka käyttäjä on, kun taas auktorisointi on prosessi, jossa todennetaan mihin tällä on pääsy. Näitä prosesseja kuvaava tosi elämän esimerkki voisi olla se, kun henkilö kulkee lentokentän turvatarkastuksen läpi, hänet autentikoidaan. Myöhemmin henkilön saapuu lentokentän lähtöportille, ja hän ojentaa tarkistuskorttinsa vastaanottovirkailijalle, jolloin tämä auktorisoi henkilön ja mahdollisesti sallii pääsyn lentokoneeseen. (Auth0 2022.)

Yksi tapa autentikoinnin ja auktorisoinnin toteuttamiseksi on OpenID Connect (OIDC). OIDC on autentikaatio protokolla, joka on OAuth 2.0-viitekehyksen laajennus. Siinä missä OAuth 2.0 on viitekehys auktorisointiprotokollien rakentamiseen ja sellaisenaan keskeneräinen, on OIDC valmis autentikaatio- ja auktorisointiprotokolla. OIDC käyttää suuresti Json Web Token-standardeja (JWT). JWT-standardit määrittelevät ns. identiteettitunnuksen ja pääsytunnuksen JSON-muodon, sekä keinot identiteettitunnuksen digitaaliseen allekirjoittamiseen ja salaamiseen kompaktilla ja verkkoystävällisellä tavalla. (Keycloak 2022b.)

Identiteettitunnus (engl. identity token) tarjoaa käyttäjän identiteettitietoja käyttäjästä ja on määritelty OIDC-spesifikaatiossa. Pääsytunnus on tunnus, joka voidaan tarjota HTTP-pyynnön mukana, ja sen perusteella palvelin voi sallia tai evätä pääsyn palveluun. Pääsytunnus (engl. access token) on osa sekä OIDC-, että OAuth 2.0-spesifikaatioita. (Keycloak 2022c.) Monesti onnistuneen autentikaation jälkeen käyttäjälle myönnetään myös päivitystunnus (engl. refresh token). Auth0:n (2022) mukaan päivitystunnuksen avulla voidaan myöntää uusi pääsytunnus ilman käyttäjän vuorovaikutusta. Seuraavassa kuvassa käsitellään karkeasti autentikaation ja auktorisoinnin toteutuminen esimerkiksi verkkosovelluksessa.



Kuva 2. Esimerkki autentikaatiosta ja auktorisoinnista

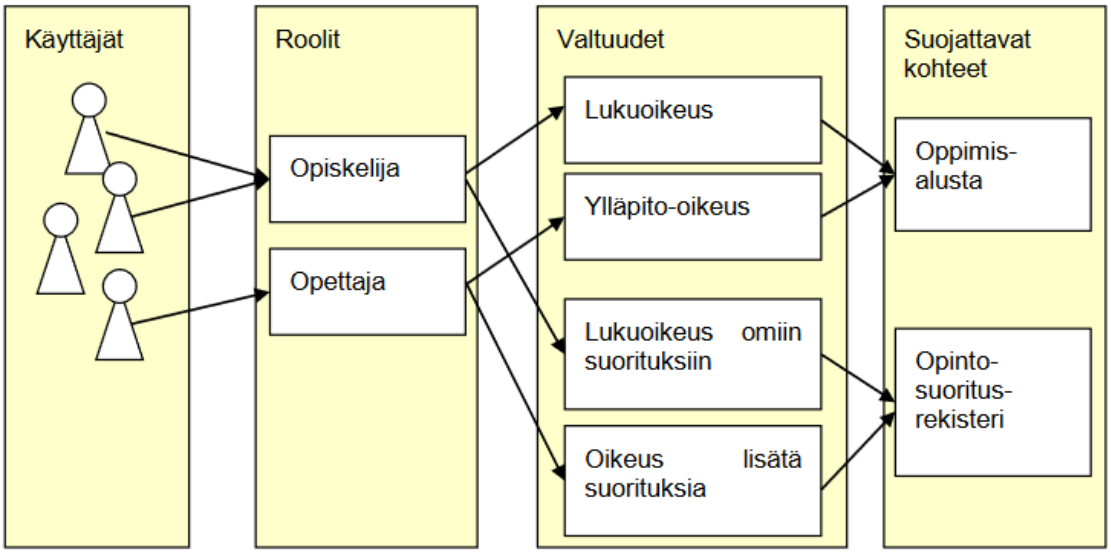
Kuvassa 2 on havainnollistettu karkeasti autentikaation ja auktorisoinnin toteutuminen Keycloakin tarjoaman käyttövaltuushallinnan avulla. Käyttäjän siirtyessä suojattuun resurssiin tarkistetaan, onko pyynnön mukana pääsytunnusta, jos on, niin tarkistetaan tämän voimassa olevuus. Mikäli pääsytunnus on voimassa, suoritetaan auktorisointi, ja tämän onnistuessa ohjataan käyttäjä suojattuun resurssiin.

Mikäli pyynnön yhteydessä välitetty pääsytunnus ei ole enää voimassa, tarkistetaan päivitystunnuksen voimassaolo. Uuden, voimassa olevan, pääsytunnuksen hakeminen päivitystunnuksen avulla toteutetaan tavanomaisesti niin, ettei käyttäjä huomaa tätä.

Pääsytunnuksen tietosisällössä on väitteitä, jotka sisältävät toteamuksia entiteetistä, tässä tapauksessa käyttäjästä (JWT 2022). Tämän avulla palvelin suorittaa auktorisoinnin. Auktorisoinnin onnistuessa käyttäjälle myönnetään pääsy suojattuun resurssiin.

Pääsynhallinta resursseihin voidaan toteuttaa myös rooli perustaisesti (engl. role-based access control, RBAC). Tällöin käyttäjän ja käyttövaltuuden väliin on luotu abstraktio, rooli. Käyttäjälle määritellyt roolit kuvaavat esimerkiksi tämän työtehtäviä organisaatiossa ja käyttövaltuudet on myönnetty rooleille.

(Linden 2017a.) Seuraavaksi kuva, jossa havainnollistetaan rooliperustaista pääsynvalvontaa.



Kuva 3. Esimerkki rooliin perustuvasta pääsynvalvonnasta (Linden 2017b)

Kuvassa 3 on esimerkki rooliin perustuvasta pääsynvalvonnasta korkeakoulun oppimisalustalla. Esimerkki sisältää kaksi oppimisalustaan liittyvää roolia, opiskelija ja opettaja (Linden 2017a). Käyttäjälle myönnetään pääsy resursseihin tämän roolin oikeuksien, eikä käyttäjän, pääsyoikeuksien mukaisesti.

4 SOVELLUKSIEN KÄYTTÖÖNOTTO

Tässä luvussa käydään läpi Saleorin demosovelluksen ja Keycloakin lokaalin käyttöönoton vaiheet. Saleorin käyttöönotto on itselleni uusi asia, mutta heidän verkkosivuillaan on aiheesta kattavan oloinen dokumentaatio.

4.1 Saleorin demosovelluksen käyttöönotto

Kuten mainittu aiemmin tässä työssä, tarjoaa Saleor sinänsä käyttövalmiin demosovelluksen. Seuraavaksi suoritetaan kyseisen demosovelluksen käyttöönotto Saleorin ohjeistuksen mukaisesti, ja se dokumentoidaan ja mahdolliset virheet raportoidaan.

Demosovellus on ladattavissa GitHubissa ja se on suunniteltu ajettavaksi Dockerissa. Git-versionhallinnan tai Dockerin asentaminen ja käyttöönotto eivät kuulu tämän työn laajuuteen, joten oletetaan että ne ovat tässä kohdin jo

valmiiksi asennettuna. Seuraavaksi kuvassa 4 käydään läpi Saleorin käyttöönottoon käytetyt komentorivit komennot.

```
git clone https://github.com/saleor/saleor-platform.git --recursive --jobs 3
docker compose build
docker compose run --rm api python3 manage.py migrate
docker compose run --rm api python3 manage.py collectstatic --noinput
docker compose run --rm api python3 manage.py populatedb
docker compose run --rm api python3 manage.py createsuperuser
docker compose up
```

Kuva 4. Saleor demosovelluksen käyttöönotto

Ensimmäinen komento on perinteinen git clone. Saleorin demosovelluksen repositorio koostuu alamoduuleista (engl. submodule), eli ikään kuin linkeistä toisiin Git-repositorioihin. Komennon lisäasetus --recursive ohjeistaa Gittiä kloonaamaan myös alamoduulit, --jobs taas kertoo Git:lle, kuinka monta alamoduulia tulee kloonata.

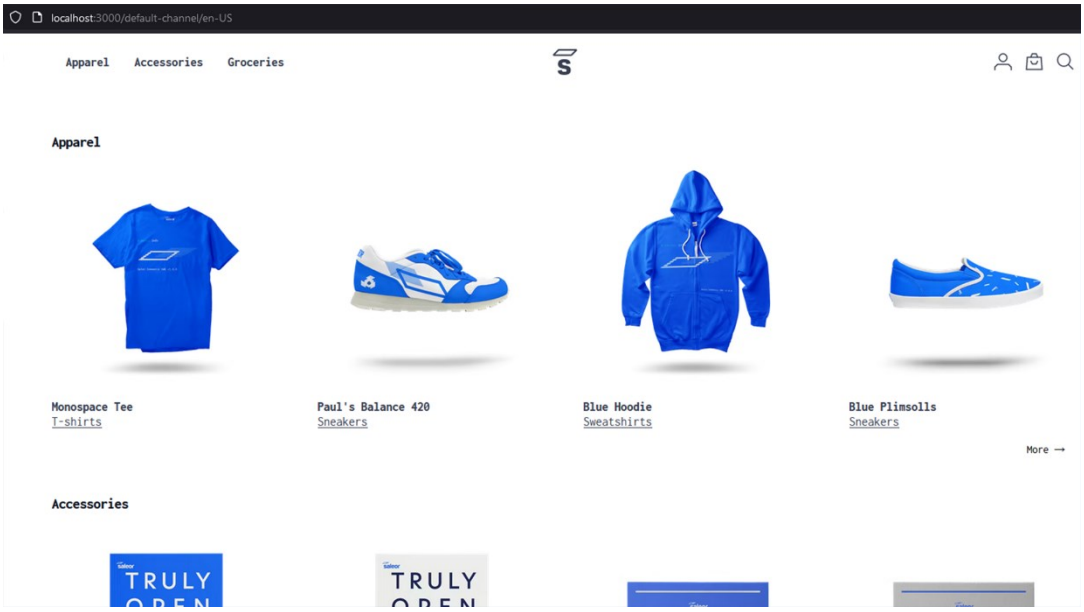
Seuraavat komennot liittyvät kaikki Docker Composeen. Docker Compose on työkalu useista konteista koostuvien sovellusten määrittämiseen ja ajamiseen. Build-komennolla nimensä mukaisesti kootaan määrityksen mukaiset palvelut (engl. service). Palvelut on määritelty erillisessä tiedostossa, jonka juuressa komentoriviä ajetaan. Määritystiedosto on oletuksena nimeltään docker-compose.yaml.

Run-komennolla voidaan ajaa yksittäisiä komentoja jonkin kontin sisällä. Lisäasetus –rm käskee Docker Composea sammuttamaan automaattisesti käynnistämänsä kontin skriptin suorituksen jälkeen. Sen jälkeen oleva api on Docker Composen määritystiedossa määritelty palvelun nimi. Seuraava python3 manage.py-komento käskee konttiin asennettua Python tulkkia suorittamaan manage.py-niminen tiedosto erilaisilla komentoriviargumenteilla. Komentoriviargumentit tässä tapauksessa käskevät suorittamaan tietokantamigraatiot, täyttämään tietokanta esimerkki datalla sekä luomaan ylläpitäjäkäyttäjän demosovellukseen. Seuraava collectstatic-komento on jokin Django viitekehyksen oma juttunsa, johon ei tässä vaiheessa koettu tarpeelliseksi perehtyä enempää.

Up-komento tekee saman kuin build, mutta myös käynnistää palvelut omissa konteissaan, määritystiedoston mukaisesti. Kontit jäävät oletuksena käyntiin

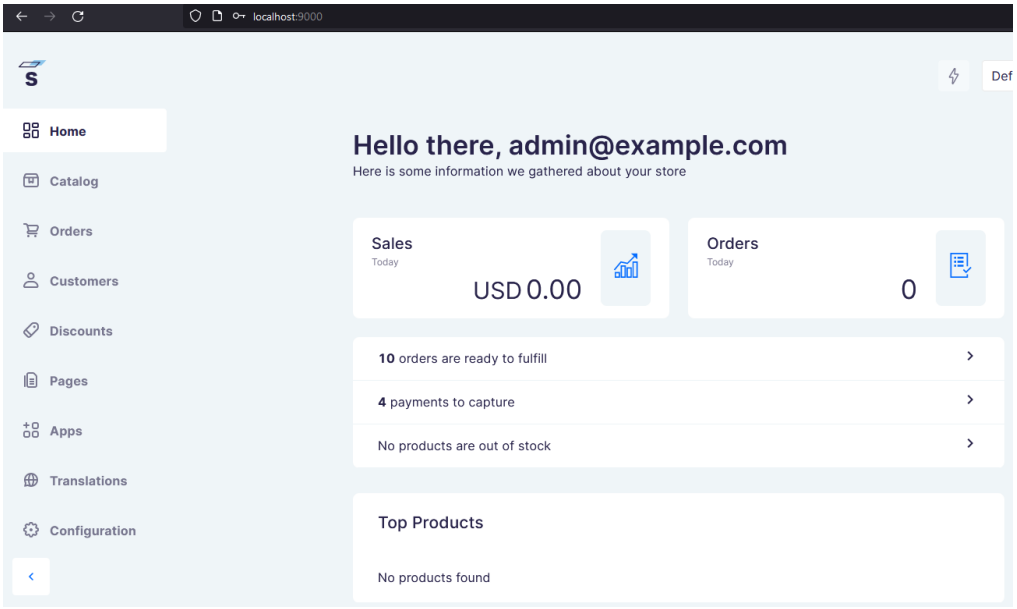
terminaali-ikkunaan, jossa komento ajettiin. Lisäasetuksella –detached olisi mahdollista käynnistää kontit taustalla.

Tämän jälkeen demosovellus on toiminnassa ja sellaisenaan käytettävissä. Asennuksen toteutettiin Windows-ympäristössä, eikä sen aikana törmätty ongelmiin vaan kaikki toimia kuten kuuluikin. Seuraavassa kuvassa on esitelty demosovelluksen asiakaskäyttöliittymää.



Kuva 5. Saleor-demosovelluksen asiakaskäyttöliittymä

Asiakaskäyttöliittymä on hyvin modernin näköinen ja vaikuttaa toimivan kuten kuuluukin. Kuvassa 5 näkyvät tuotteet ovat seurausta asennusvaiheessa ajetusta komennosta, joka täytti tietokannan esimerkki datalla. Tämä oli vapaaehtoista, mutta koin sen tätä opinnäytetyötä varten hyödylliseksi. Seuraavassa kuvassa 6 on esitelty demosovelluksen hallintakäyttöliittymää

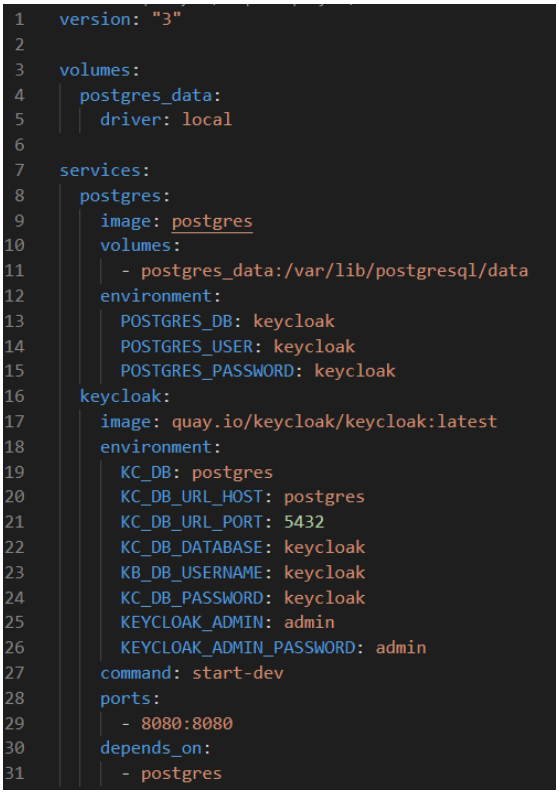


Kuva 6. Saleor demosovelluksen hallintakäyttöliittymä

Hallintakäyttöliittymä noudattaa samaa värimaailmaa, jota asiakaskäyttöliittymäkin. Asennusvaiheessa syötetty esimerkki data on näkyvissä hallintakäyttöliittymässäkin. Tämän käyttöliittymän kautta verkkokaupan ylläpitäjän on mahdollista hoitaa verkkokauppansa hallinnollisia asioita, kuten varastosaldojen ja maksutapahtumien seurantaa. Mielestäni mainitsemisen arvoista on se, että Saleorissa likimain jokainen asiakaskäyttöliittymässä näkyvä asia on lokalisoitavissa hallintakäyttöliittymän kautta.

4.2 Keycloakin käyttöönotto

Seuraavaksi otamme Keycloakin käyttöön. Tässä käytämme Saleorin demosovelluksen asentamisesta tutuksi tullutta Docker Composea. Internetissä on olemassa konttikuvakehakemistoja (engl. container image repository), josta on mahdollista ladata myös Keycloakin konttikuvake (engl. container image). Keycloakin aloitusoppaassa ohjeistetaan aloittamaan kaltaisemme demokäyttö docker run -komennolla. Itse koin kuitenkin kätevämmäksi luoda Docker Compose määritystiedoston, joka pitää sisällään palvelut sekä Keycloakin tarvitsemalle tietokannalle, että itse Keycloakille. Seuraavassa kuvassa on tämä Docker Composen määritystiedosto.

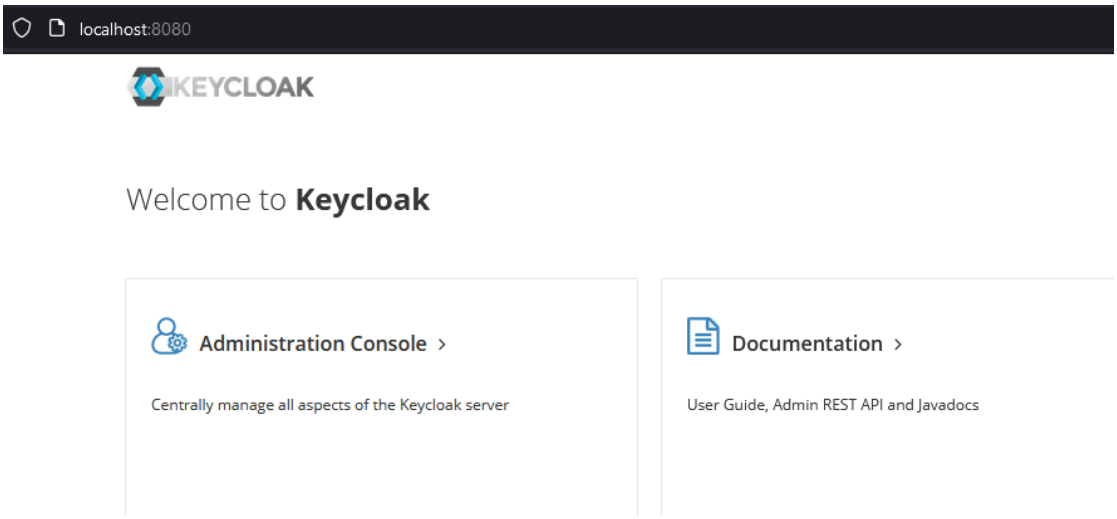


```yaml
version: "3"

volumes:
  postgres_data:
    driver: local

services:
  postgres:
    image: postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      POSTGRES_DB: keycloak
      POSTGRES_USER: keycloak
      POSTGRES_PASSWORD: keycloak
  keycloak:
    image: quay.io/keycloak/keycloak:latest
    environment:
      KC_DB: postgres
      KC_DB_URL_HOST: postgres
      KC_DB_URL_PORT: 5432
      KC_DB_DATABASE: keycloak
      KB_DB_USERNAME: keycloak
      KC_DB_PASSWORD: keycloak
      KEYCLOAK_ADMIN: admin
      KEYCLOAK_ADMIN_PASSWORD: admin
    command: start-dev
    ports:
      - 8080:8080
    depends_on:
      - postgres
```

Kuva 7. Docker Compose määritystiedosto Keycloakille

Kuvassa 7 on määritelmätiedosto Docker Composelle, joka pitää sisällään kaksi palvelua, PostgreSQL-tietokannan ja Keycloakin. Tämän työn laajuus huomioiden ei tässä kohdin perehdytä määrittelyyn sen kummemmin. Huomion arvoista on Keycloak-palvelun ympäristömuuttujista (engl. environment variables) *KEYCLOAK_ADMIN* ja *KEYCLOAK_ADMIN_PASSWORD*. Niillä määritellään nimiensä mukaisesti ylläpitäjäkäyttäjän käyttäjätunnus ja salasana. Työn edetessä huomattiin myös, että Keycloak ei, ainakaan helposti, suostu toimimaan muussa TCP-portissa kuin 8080. On syytä myös huomioida, ettei mikään Saleorin palveluista ole käynnissä samassa portissa missä Keycloak.

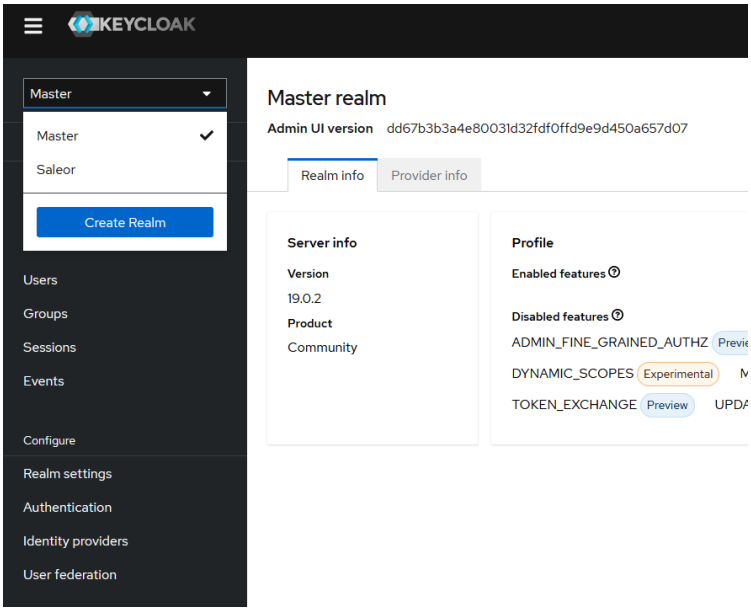
Nyt navigoimalla komentorivillä kuvan 7 määritystiedoston polkuun, ja suorittamalla komento docker compose up saadaan Keycloak käynnistymään. Seuraavaksi kuva Keycloakin hallintapaneelin aloitusnäkymästä.



Kuva 8. Keycloakin hallintapaneelin aloitusnäkymä

Kuvasta 8 voimme todentaa Keycloak-palvelumme olevan pystyssä ja toimivan portissa 8080. Samassa kuvassa näkyvä Administration Console-linkki vie ylläpitäjän sisäänkirjautumiseen. Sisäänkirjautuminen onnistuu Docker Compose määritystiedostossa määritetyillä ylläpitäjän tunnuksilla. Seuraavassa kuvassa Keycloakin hallintapaneeli sisäänkirjautumisen jälkeen.

Seuraavassa luvussa tullaan perehtymään Saleorin demosovelluksen käyttäjänhallinnan toteuttamiseen Keycloakin avulla, joten määrittelemme jo tässä vaiheessa Keycloakin niin valmiiksi kuin mahdollista.

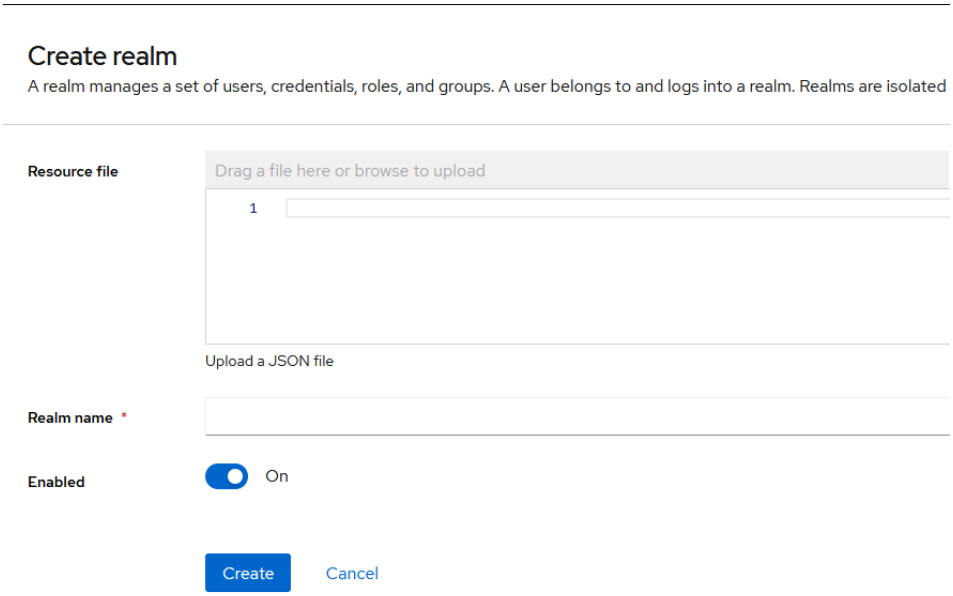


Kuva 9. Keycloakin hallintapaaneli sisään kirjauduttua

Kuvassa 9 nähdään miltä Keycloakin hallintapaneeli näyttää onnistuneen sisäänkirjautumisen jälkeen. Onnistuneen sisäänkirjautumisen jälkeen ylläpitäjä ohjataan aina Master-piiriin (engl. Master realm).

Piirit (engl. realm) hallinnoivat käyttäjiä, valtuutuksia, rooleja ja ryhmiä. Jokainen käyttäjä kuuluu johonkin piiriin. Piirit ovat eristettyjä toisistaan ja voivat ainoastaan hallinnoida omia käyttäjiään. (Keycloak 2022.)

Virallisen ohjeistuksen mukaan Master-piiriä tulee käyttää vain muiden piirien luomiseen ja hallinnoimiseen. Luomme siis uuden piirin. Kuvassa 8 näkyvän alas vetolaatikon saa esille klikkaamalla vasemmassa yläkulmassa näkyvää Master-tekstiä. Seuraavassa kuvassa Create Realm-painikkeen takaa aukeava näkymä.



Kuva 10. Keycloakin uuden piirin luominen

Kuvasta 10 näkyy, kuinka yksinkertaista uuden piirin luominen on. Syötetään vain nimi, sekä valitaan, onko piiri enabloituna. Halutessaan olisi myös mahdollista tuoda piirille määrittelyjä ulkoisesta JSON-tiedostosta, mutta se on tässä kohdin tarpeetonta.

Keycloakin asentaminen ja käyttöönotto oli äärimmäisen yksinkertaista. Kuten tämän osion alussa todettiin, ei tämän kaltaiselle käyttöönotolle ollut varsinaista aloitusopasta, mutta Keycloakin dokumentaatiosta löytyi helposti tarvittavat tiedot Docker Compose määritystiedoston luomista varten.

5 INTEGRAATIOIDEN TOTEUTUS

Tämän luvun aikana käydään läpi kuinka aikaisemmin käyttöönotettu Keycloak saadaan integroitua Saleoriin. Sen jälkeen valitaan toimeksiantajan ja omien arvojeni mukainen tuotetietohallintajärjestelmä, suoritetaan sen käyttöönotto ja integroidaan se Saleoriin.

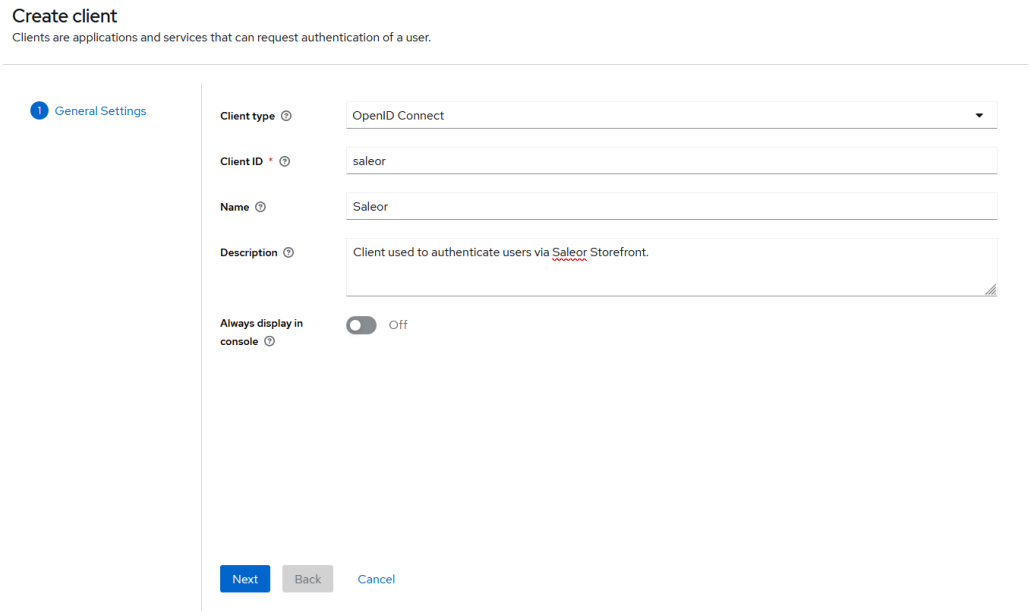
Keycloakin integroimisen ei pitäisi vaatia juurikaan ohjelmointia, vaan pääasiassa asetuksien säätämistä. Tuotetietohallintajärjestelmä-integraatio taas täytyy toteuttaa teoriaosuudessa käsiteltynä Saleorin laajennussovelluksena, ja se tulee vaatimaan enemmän ohjelmointia. Työn laajuuden vuoksi kuitenkaan varsinaista ohjelmointia ei tulla kovinkaan paljoa käsittelemään, vaan enemmänkin valmiin prototyypin arkkitehtuuria.

5.1 Keycloakin integroiminen

Nyt, kun Saleor ja Keycloak on saatu otettua käyttöön, toteutetaan seuraavaksi käyttäjänhallinnan ulkoistaminen Keycloakkiin.

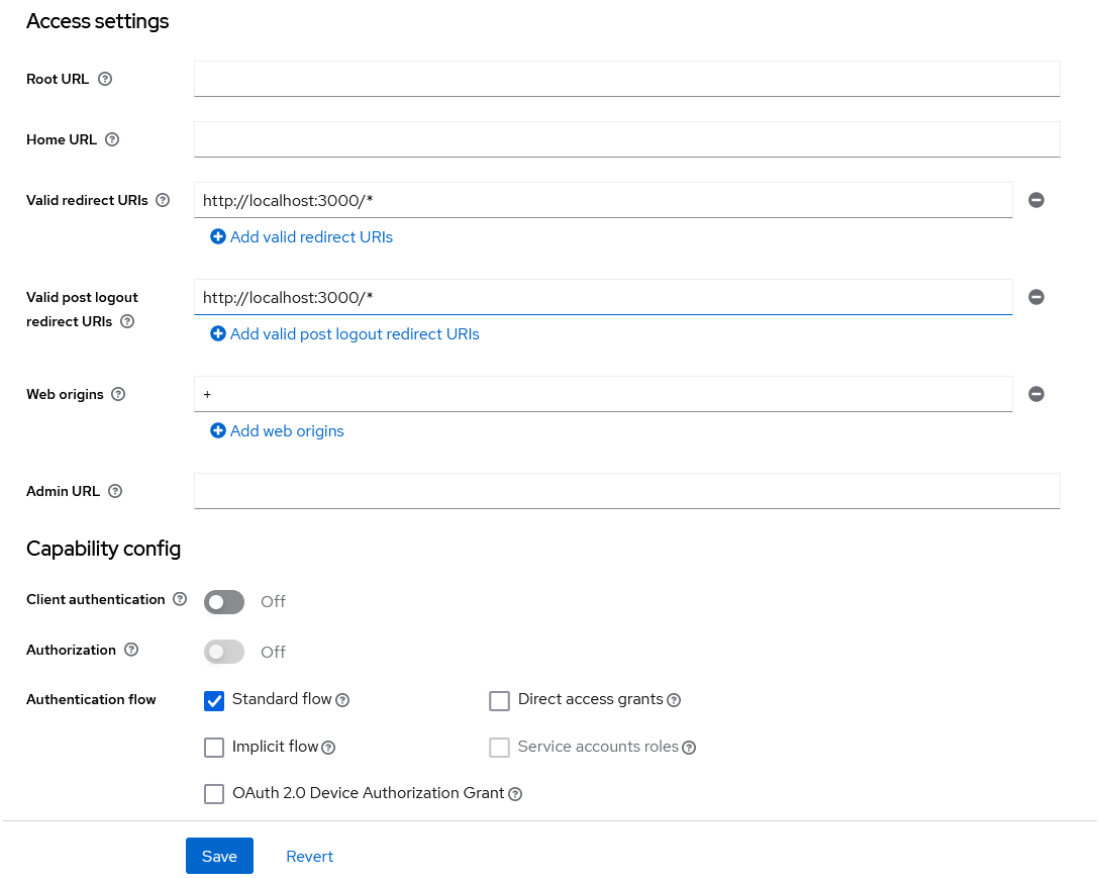
Keycloakin piireihin on mahdollista luoda asiakasohjelmia (engl. client), joiden välityksellä käyttäjien autentikaatio toteutetaan. Asiakasohjelmat ovat entiteettejä, jotka voivat pyytää Keycloakilta käyttäjän autentikaatiota. Useimmiten asiakasohjelmat ovat sovelluksia ja palveluita. (Keycloak 2022c.) Samoin kuin käyttäjätkin, ovat asiakasohjelmat piiri kohtaisia.

Asiakasohjelmien hallinnoiminen tapahtuu valitsemalla kuvassa 9 näkyvästä valikosta Clients. Tätä ennen on tärkeää tarkistaa, minkä piirin asetuksissa on. Kulloinkin valittu piiri näkyy aina valittuna valikon yläreunassa. Asiakasohjelmien listauksessa voi siirtyä hallinnoimaan kunkin asiakasohjelman asetuksia. Create client-nappia painamalla päästään luomaan uusi asiakasohjelma. Tätä tarkastellaan seuraavassa kuvassa.



Kuva 11. Uuden asiakasohjelman luominen Keycloakkiin

Kuvassa 11 on määritelty aloitusasetukset uudelle asiakasohjelmalle. Client type määrittelee protokollan, jota asiakasohjelma tulee käyttämään. Tässä tapauksessa valitsemme jo aiemmin käsitellyn OpenID Connect:n. *Client ID* on asiakasohjelman yksilöivä nimi. Name ja description ovat nimensä mukaisesti asiakasohjelman nimi sekä kuvaus. Kentistä vain Client ID on pakollinen. Loppuihin asiakasohjelman asetuksiin pääsee käsiksi painettuaan kuvassa näkyvää Next-nappia. Seuraavassa kuvassa käsitellään lisää asiakasohjelman asetuksia.



Kuva 12. Keycloakin asiakasohjelman pääsy- ja kyvykkyysasetukset

Kuvassa 12 ylimpänä näkyvät pääsyasetukset (engl. access settings) käytännössä määrittävät miltä verkkosivuilta/mistä lähteistä kyseinen asiakasohjelma on oikeutettu asioimaan Keycloakin kanssa. Asiakasohjelman pyytäessä autentikaatiota, välitetään OIDC-standardin mukaisesti Keycloakille redirect url, eli verkko-osoite, johon Keycloak uudelleen ohjaa verkkoselaimen sisäänkirjautumisen jälkeen. Tässä tapauksessa Saleorin asiakaskäyttöliittymä toimii omalla tietokoneellani portissa 3000, joten määrittelen sen ainoaksi sallituksi uudelleen ohjaus osoitteeksi. Asiakasohjelman kirjatessa käyttäjää ulos, välitetään pyynnön mukana taas redirect url. Web origins määrittelee lähteet, joista CORS (Cross-Origin Resource Sharing) pyynnöt ovat sallittuja. Jos halutaan sallia kaikki uudelleen ohjaus osoitteet, voidaan käyttää +-merkkiä, kuten kuvassa. Mozillan (2022) mukaan CORS on HTTP-otsikko pohjainen turvallisuusmekanismi, jolla verkkoselaimet pyrkivät estämään skriptilähtöiset HTTP-pyynnöt eri lähteisiin.

Kyvykkyysasetuksissa (engl. capability config) määritellään tavat, joilla asiakasohjelma voi Keycloakilta pyytää autentikaatiota. Client authentication mää-

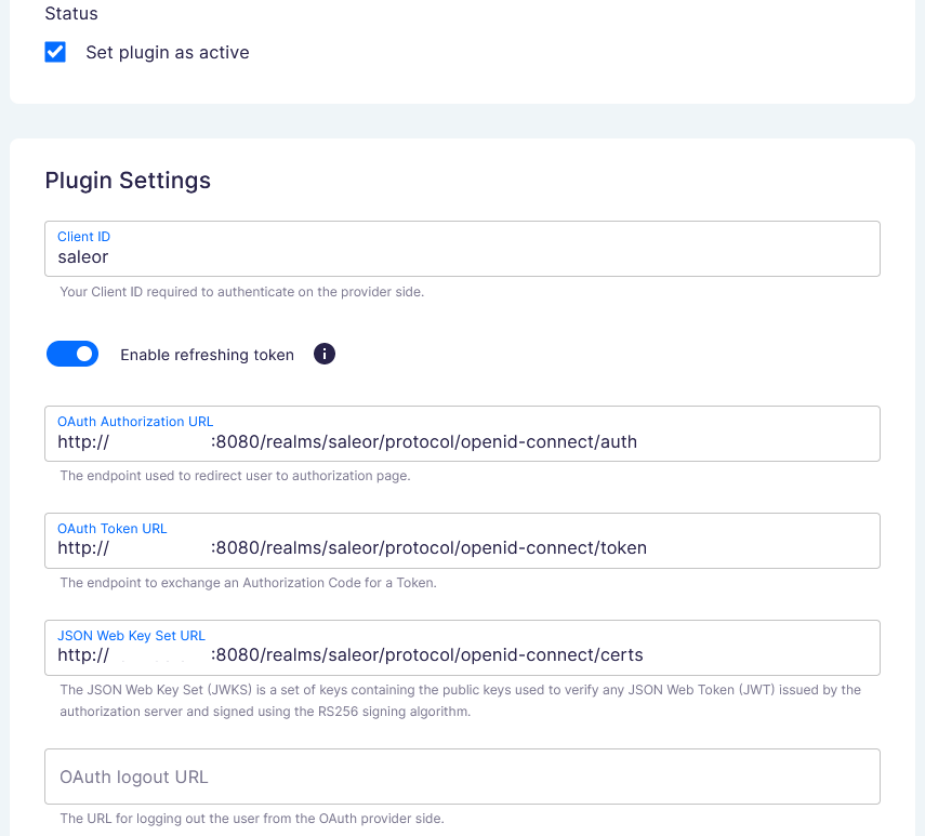
rittää, tuleeko asiakasohjelma pyytämään autentikaatiota ohjelmallisesti. Meidän tapauksessamme ei. Standard Flow sallii OIDC ja OAuth2 määritelmien mukaisen Authorization Code-flow:n. Tähän tutustumme syvemmin myöhemmin tässä luvussa.

Tässä vaiheessa Keycloak asiakassovelluksineen on valmis käytettäväksi. Seuraavaksi siirrytään konfiguroimaan Saleoria käyttämään Keycloakkia oman käyttäjänhallintansa sijasta.

Kuten teoriaosuudessa mainittiin, on Saleoria mahdollista laajentaa liitännäisiä ja sovelluksia käyttäen. Keycloak integraatiota varten käytän Saleorin kehitystiimin itsensä toteuttamaa OpenID Connect-liitännäistä. Saleorin (2022b) mukaan OpenID Connect-liitännäinen mahdollistaa integraation toteuttamisen OAuth-tarjoajan välille ja autentikaatio-prosessin siirtämisen ulkoiselle palveluntarjoajalle.

Integraatio on mahdollista toteuttaa kahdella tavalla, toisessa Saleor pyytää autentikaatiota aiemmin mainitun Authorization Code-flow:n mukaisesti ja toisessa Saleor ainoastaan tarkastaa pääsytunnuksen voimassaolon. Toteutetaan näistä ensimmäisen, Authorization Code-flow:n mukainen integraatio.

Aloitetaan prosessi navigoimalla kuvassa 8 nähtyyn Saleorin hallintakäyttöliittymään. Hallintakäyttöliittymässä siirrytään vasemmasta ala nurkasta Configuration-sivulle ja sieltä kohtaan Plugins. Plugins-näkymässä on kaikki kyseiseen Saleor instanssiin asennetut liitännäiset listattuna. Klikkaamalla haluamaansa liitännäistä, pääsee kyseisen liitännäisen asetuksia hallinnoimaan. Seuraavassa kuvassa käsitellään OpenID Connect-liitännäisen asetuksia.



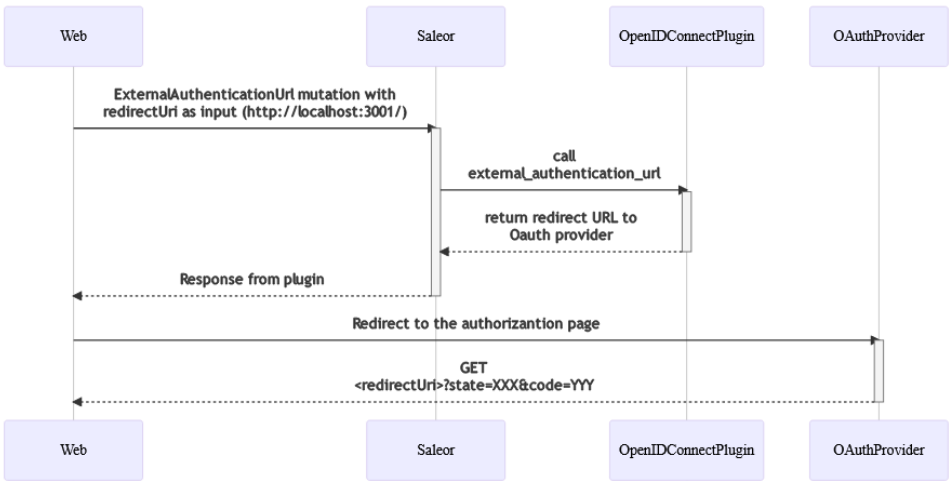
Kuva 13. Saleorin OIDC-liitännäisen asetukset

Kuvassa 13 on nähtävillä tämän työn kannalta oleellisimmat OpenID Connect-liitännäisen asetukset. Ylimmästä valintalaatikosta määritellään, onko kyseinen liitännäinen käytössä. Kuvassa näkyvistä verkko-osoitteista on tietoturvallisuussyistä peitetty tietokoneeni ulkoinen IP-osoite. Kuvan alapuolella on vielä tekstikenttä, johon Keycloakissa määritelty asiakasohjelman salasana on syötetty.

Client ID tulee olla sama, kuin Keycloakkiin luomamme asiakasohjelman ID. Enable refreshing token-kytkin määrittää käytetäänkö OIDC-määritelmän mukaisia päivitystunnuksia, jolloin asiakkaan pääsytunnus voidaan vanhentuessaan päivittää taustalla uuteen. Ilman, että käyttäjän tarvitsee kirjautua uudestaan sisään. Saleorin (2022b) dokumentaation mukaan toteuttaaksemme haluamamme kaltaisen integraation, tulee kuvassa näkyvät tekstikentät olla täytettynä oikein. OAuth Authorization URL kertoo Saleorille, mihin verkko-osoitteeseen sen tulee ohjata käyttäjät sisäänkirjautumista varten, OAuth Token URL taas mistä verkko-osoitteesta Saleor voi onnistuneen sisäänkirjautumisen jälkeen pyytää pääsytunnusta ja JSON Web Key Set URL kertoo, mistä Saleor löytää pääsytunnusten salauksen purkuun tarvittavat salausavaimet.

Saleorin dokumentaatiosta löytyvät ohjeet OpenID Connect-liitännäisen määrittelyyn olivat hyvät ja selkeät, mutta koin silti tähän asti isoimmat vaikeudet tämän vaiheen kanssa. Saleorin mikropalvelut toimivat omassa Docker Compose ympäristössä, ja Keycloak omassaan. Tämä aiheutti päänvaivaa, kun tavanomaisesti samalla tietokoneella käynnissä olevat verkkopalvelimet löytävät toisensa localhost-verkko-osoitteesta, mutta tässä yhteydessä se ei ollutkaan niin. Se johtui siitä, että Docker Composen sisällä olevat palvelut ovat omassa erillisessä konttiverkossaan, jolloin localhost viittaa samassa konttiverkossa toimivaan palveluun. Ymmärtääkseni olisi jotenkin mahdollista viitata toisen Docker Compose ympäristön verkkoon, mutta koin helpoimmaksi ratkaisuksi käyttää tietokoneeni ulkoista IP-osoitetta.

Seuraavaksi toteutetaan tarvittavat muutokset Saleorin demo-käyttöliittymän koodiin, jotta Keycloak-integraatio on valmis. Seuraavassa kuvassa 14 havainnollistetaan Authorization Code-flow:n alkua, jolla tulemme suorittamaan käyttäjän autentikaatio.



Kuva 14. Generating authentication URL – Workflow (Saleor 2022d)

Authorization Code-flow alkaa tässä tapauksessa sillä, että Saleorin asiakaskäyttöliittymä tekee GraphQL-rajapintakutsun Saleorin ydinsovellukselle, jonka mukana on uudelleen ohjausosoite (redirectUri), jonne käyttäjä halutaan ohjata takaisin Saleoriin onnistuneen sisäänkirjautumisen jälkeen.

OpenID Connect-liitännäinen muodostaa autentikaatio-osoitteen (engl. authentication url, koodissa authentication_url), eli verkko-osoitteen, jossa käyttäjä käy kirjautumassa sisään. Autentikaatio-osoitteeseen lisätään kyselyparametrina (engl. query parameter) mm. asiakasohjelman ID:n ja tila (engl. state). Tila on tietoturvallisuussyistä tätä kyseistä autentikaatiota varten luotu satunnainen tekstinpätkä. Auth0:n (2022b) mukaan pääasiallisin syy tilaparametrin käyttöön on sivustojenvälisten pyynnön väärennysten (engl. cross-site request forgery, CSRF) ehkäiseminen. Palvelinsovellus palauttaa oikean muotoisen autentikaatio osoitteen GraphQL-rajapintakutsun vastauksena.

Seuraavaksi selain uudelleen ohjaa käyttäjän autentikaatio-osoitteeseen Keycloakkiimme, jossa käyttäjä kirjautuu sisään, tai halutessaan luo uuden käyttäjän.

Onnistuneen sisäänkirjautumisen jälkeen Keycloak uudelleen ohjaa selaimen aiemmin määriteltyyn uudelleen ohjausosoitteeseen, kyselyparametreina mm. aiemmin käsitelty tila ja auktorisointikoodi (engl. authorization code, code). Tilan ja koodin avulla Saleor voi pyytää Keycloakilta pääsytunnusta sisään kirjautuneelle käyttäjälle. Seuraavassa kuvassa näytetään, kuinka edellä kuvattu vaihe toteutettiin Saleorin asiakaskäyttöliittymään.

```
/**
 * Handles retrieving external authentication URL from Saleor API
 * and redirects user to it.
 */
const handleRedirectTologin = async () => {
  const redirectUri = window.location.href;

  try {
    const { data } = await getExternalAuthUrl({
      pluginId: "mirumee.authentication.openidconnect",
      input: JSON.stringify({ redirectUri })
    })
    const { authorizationUrl } = JSON.parse(data?.externalAuthenticationUrl?.authenticationData);

    window.location.assign(authorizationUrl);
  } catch (error) {
    console.error("Error happened while handling redirection to authentication provider: ", error);
  }
};
```

Kuva 15. handleRedirectToLogin-funktio Saleorin asiakaskäyttöliittymässä

Tavoitteena on, että käyttäjä ohjataan takaisin tämänhetkiselle sivulle onnistuneen sisäänkirjautumisen jälkeen, joten tämänhetkinen osoite määritellään aiemmin käsitellyksi uudelleen ohjaus osoitteeksi. GetExternalAuthUrl-funktio

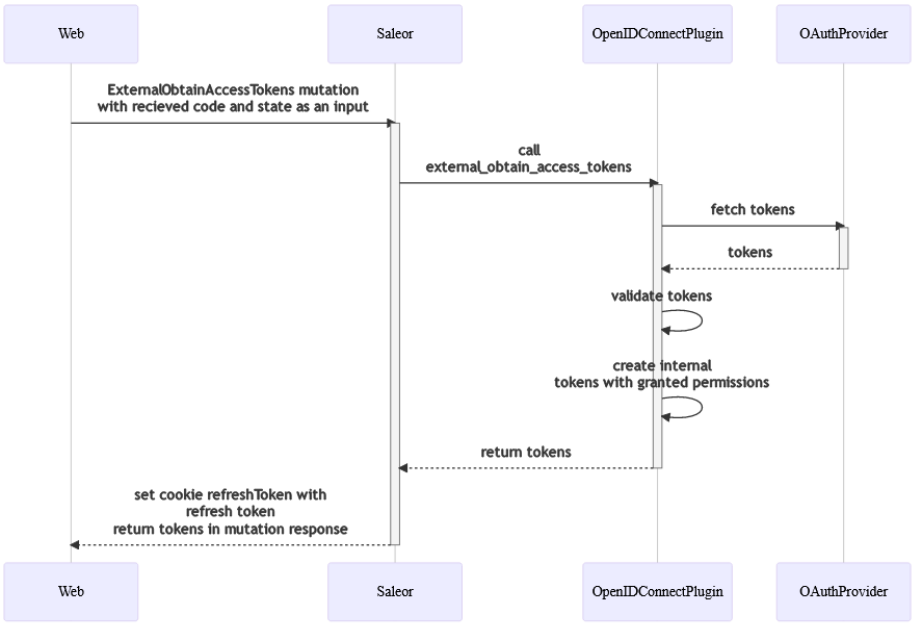
on Saleorista löytyvä funktio yksinkertaistamaan GraphQL-rajapintakutsun tekemistä. Kyseisen funktion palauttama arvo on tekstimuodossa oleva JSON-olio (engl. JSON string), josta palvelinsovelluksen muodostama sisäänkirjautumisosoite saadaan. Tämän jälkeen selain navigoidaan sisäänkirjautumisosoitteeseen. Seuraavassa kuvassa käsitellään miltä getExternalAuthUrl GraphQL-rajapintakutsu näyttää.

```
mutation ($input: JSONString!) {
  externalAuthenticationUrl(
    pluginId:"mirumee.authentication.openidconnect",
    input: $input
  ){
    authenticationData
    accountErrors{
      field
      message
    }
  }
}
```

Kuva 16. externalAuthenticationUrl GraphQL-rajapintakutsu (Saleor 2022e)

GraphQL-rajapintakutsuja on kahdenlaisia, kyselyjä (engl. query) ja mutaatioita (engl. mutation). Kyselyillä haetaan olemassa olevaa dataa ja mutaatioilla luodaan uutta tai muokataan olemassa olevaa dataa. Tässä tapauksessa luodaan uusi uudelleen ohjausosoite.

Rajapintakutsun tyypin lisäksi ensimmäisellä rivillä on määritelty kyselyssä käytetyt muuttujat (kts. kuva 16). Huutomerkki muuttujan tietotyypin perässä kertoo pakollisuudesta. REST-rajapintakutsuista poiketen GraphQL-rajapintakutsut tallennetaan yleensä omiin tiedostoihinsa ja GraphQL-viitekehykset generoivat tallennetuista rajapintakutsuista koodia yksinkertaistaakseen kutsujen tekemistä (kts. kuva 15). Täten GraphQL-rajapintakutsut ovat tapana nimetä, tässä tapauksessa externalAuthenticationUrl. Seuraavaksi kerrotaan mitä rajapintakutsun halutaan palauttavan. Tässä tapauksessa authenticationData-olio kokonaisuudessaan ja accountErrors-olion ominaisuudet field ja message. Nämä oliot ovat siis Saleorin GraphQL-rajapinnassa määriteltyjä. Seuraavassa kuvassa käsitellään Authorization Code-flow:n jatkoa, eli pääsytunnuksen hakemista sisäänkirjautumisen jälkeen.



Kuva 17. Obtaining access token – Workflow (Saleor 2022f)

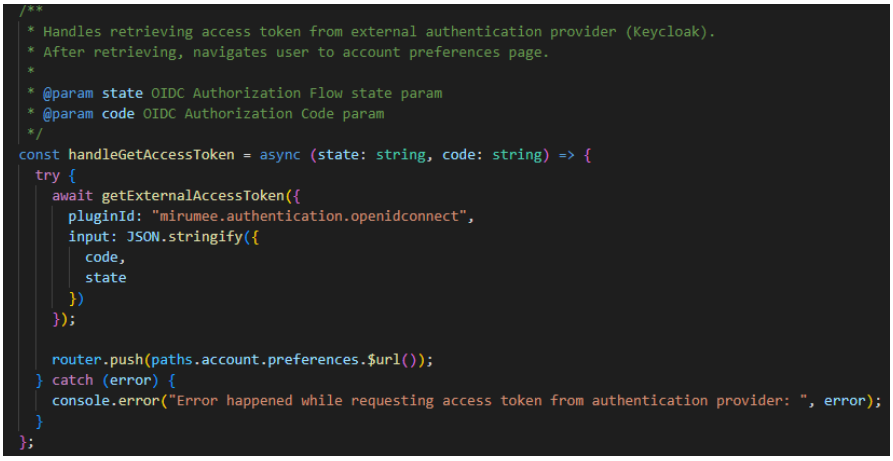
Kuvasta 17 on nähtävissä, kuinka aiemmin sisään kirjautuneelle käyttäjälle haetaan pääsytunnus. Onnistuneen sisäänkirjautumisen jälkeen Keycloak palautti aiemmin käsitellyt tilan ja koodin. Nyt niillä on mahdollista pyytää Keycloakilta pääsytunnusta käyttäjälle. Kuvan 16 kaltaisella GraphQL-rajapintakutsulla Saleorin palvelinsovellukselle annetaan muuttujina tila ja koodi, jotka palvelinsovellus välittää eteenpäin Keycloakille REST-rajapintakutsuna.

Palvelinsovellus validoi Keycloakin palauttaman pääsytunnuksen, ja luo uuden ns. Saleorin sisäisen pääsytunnuksen, joka pitää sisällään mm. Saleorin omia käyttöoikeuksia erilaisiin ylläpidollisiin toimintoihin, kuten laskutuksen seurantaan.

Tähän väliin olisi mahdollista kehittää oma laajennuksensa Saleoriin, jolla voitaisiin määritellä mm. käyttäjärooli tai -ryhmä perustaisia hinnoitteluja ja tuotelistauksia. Keycloakissa käyttäjä voitaisiin lisätä ryhmään, joka vastaisi esimerkiksi asiakasyritystä. Mahdollinen laajennus voisi näiden ryhmien perusteella määritellä aiemmin mainittuja asioita.

GraphQL-rajapintakutsun vastauksena palautetaan pääsy- ja päivitystunnukset sekä aiemmin tässä luvussa käsitelty CSRF-tunnus, jolla ehkäistään sivus-

tojenvälisiä pyyntöjen väärennyksiä. Rajapintakutsu palauttaa myös sellaisenaan sisään kirjautuneen käyttäjän yksilöintitunnuksen (engl. id) ja sähköpostiosoitteen. Seuraavassa kuvassa 18 käsitellään, kuinka toteutin tämän vaiheen ohjelmallisesti.



```
/**
 * Handles retrieving access token from external authentication provider (Keycloak).
 * After retrieving, navigates user to account preferences page.
 *
 * @param state OIDC Authorization Flow state param
 * @param code OIDC Authorization Code param
 */
const handleGetAccessToken = async (state: string, code: string) => {
  try {
    await getExternalAccessToken({
      pluginId: "mirumee.authentication.openidconnect",
      input: JSON.stringify({
        code,
        state
      })
    });

    router.push(paths.account.preferences.$url());
  } catch (error) {
    console.error("Error happened while requesting access token from authentication provider: ", error);
  }
};
```

Kuva 18. getExternalAccessToken-funktio Saleorin asiakaskäyttöliittymässä.

Pääsytunnuksen hakemisen ohjelmallinen toteutus on hyvin samankaltainen kuin autentikaatio-osoitteen hakeminen. Tämä funktio saa parametrina tilan ja koodin, jotka annetaan muuttujina GraphQL-rajapintakutsulle. GetExternalAccessToken-funktio tallentaa rajapintakutsun vastauksen Reduxiin, jossa se on saatavilla kaikkialla käyttöliittymäsovelluksessa. Onnistuneen rajapintakutsun jälkeen selain navigoidaan käyttäjänhallinta-sivulle Next.js-viitekehyksen router-olion avulla.

Molemmat käsitellyt koodit ovat samassa React-komponentissa, joka oli alun perin Saleorin oma sisäänkirjautumissivu. Edellä nähtyjen koodien lisäksi toteutin myös yksinkertaisen funktion, joka kutsuu jompaakumpaa tässä esitellyistä funktioista kulloinkin oikeassa tilanteessa. Seuraavassa kuvassa esittelen pääsytunnuksen hakemisen GraphQL-rajapintakutsun.

```
mutation ($input: JSONString!) {
  externalObtainAccessTokens(
    pluginId:"mirumee.authentication.openidconnect",
    input: $input
  ){
    token
    refreshToken
    csrfToken
    user{
      id
      email
    }
    accountErrors{
      field
      code
      message
    }
  }
}
```

Kuva 19. externalObtainAccessTokens GraphQL-rajapintakutsu (Saleor 2022g).

Kuvasta 19 nähdään, että rajapintakutsu on syntaksiltaan samanlainen kuin aiemminkin esitelty. Tässä rajapintakutsussa korostuu mielestäni GraphQL:n hienous. User-olio pitää sisällään lukemattomasti enemmän ominaisuuksia, kuin vain yksilöintitunnuksen ja sähköpostin, mutta vain ne on koettu tarpeelliseksi saada vastauksena tältä rajapintakutsulta.

Tässä vaiheessa käyttäjä voi rekisteröityä Saleorin käyttäjäksi ja kirjautua sisään kokonaan Keycloakin kautta. Käyttäjän on myös mahdollista suorittaa ostoksia ja tallentaa itselleen postitus- ja laskutusosoitteita.

Toteutin lisäksi pienen muutoksen käyttäjän olemassa olevaan uloskirjautumis-logiikkaan. Ilman ulkoistettua käyttäjänhallintaa käyttäjän kirjautuessa ulos, tulee selaimen muistiin tallennetut pääsy- ja päivitystunnukset poistaa. Kun käyttäjänhallinta on ulkoistettu, tulee käyttäjä myös kirjata ulos ulkoistetusta käyttäjänhallinnan sovelluksesta. Se onnistuu samoin kuin autentikaatio osoitteenkin kanssa, palvelinsovellus muodostaa oikean muotoisen uloskirjautumis osoitteen, johon siirtymällä Keycloak kirjaa käyttäjän ulos. Tähänkin Saleorista löytyi valmis tuki, ja valmiiksi kirjoitetut GraphQL-rajapintakutsut sitä varten. Ongelmaksi muodostui kuitenkin Keycloak. Ilmeisesti OIDC-standardi on päivittynyt jossain vaiheessa, jonka seurauksena käyttämäni Keycloakin uusin versio vaati uloskirjautumista varten antamaan sellaista dataa, jota minulla ei ollut saatavilla. Ratkaisin lopulta ongelman vaihtamalla Keycloakin

vanhempaan versioon. Aiemmin tässä työssä käsitellyt vaiheet eivät sen seurauksena muuttuneet mitenkään. Ainoastaan Keycloakin hallintakäyttöliittymä vanhemmassa versiossa on hieman erilainen.

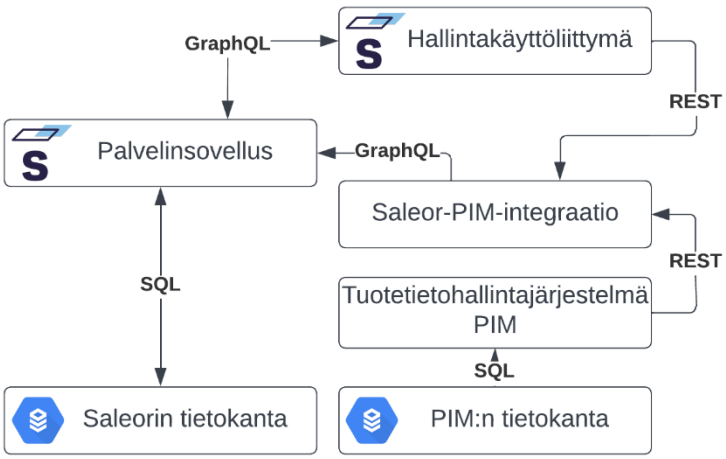
5.2 Tuotetietohallintajärjestelmän integroiminen

Akeneon (2022) mukaan tuotetietohallintajärjestelmä on sovellus, joka mahdollistaa nimensä mukaisesti yrityksen tuotetietojen hallinnoimisen keskitetysti yhdessä paikassa. Tuotetietohallintajärjestelmä helpottaa tuotteiden ja niiden tietojen viemistä verkkokaupan ja muiden tarvittavien alustojen välillä.

Toimeksiantajayrityksellä ei ole vakiintunutta ratkaisua tuotetietohallintajärjestelmäksi, joten tässä työssä käyttämäni tuotetietohallintajärjestelmä on itse valitsemani. Toisaalta koen tämän hyvänä asiana tämän opinnäytetyön kannalta, sillä näin saadaan selvitettyä, onko halutun kaltainen integraatio mahdollista toteuttaa käytännössä minkä tahansa tuotetietohallintajärjestelmän kanssa.

Kriteereinä valinnalle pidin tärkeänä toimeksiantajayrityksen ja omien arvojeni mukaisesti avointa lähdekoodia. Jotta tavoitteen mukainen integraatio olisi mahdollista toteuttaa, täytyy tuotetietohallintajärjestelmällä olla jonkinlainen ohjelmointirajapinta. Toivottavaa oli myös, että sovellusta olisi mahdollista ajaa Dockerissa, kehitystyötä helpottaakseen. Vaihtoehtoja ei loppujen lopuksi löytynyt kuin muutama. Jokainen vaihtoehto sisälsi jonkin asteiset käyttöönotto ohjeistukset, mutta jokaisessa ne olivat useita vuosia vanhoja, vaikka sovelluksien viimeisimmät versiot oli julkaistu tämän vuoden puolella. Kokeilin jokaista vaihtoehtoa, mutta vain yhden sain loppujen lopuksi toimimaan.

Tuotetietohallintajärjestelmäksi valikoitui Akeneo PIM. Crasmanin (2022) mukaan Akeneo PIM:istä löytyvät kaikki tuotehallinnan toiminnallisuudet. Akeneosta löytyi kohtalaisen kattava käyttöönotto ohjeistus, mutta siinä erikseen mainittiin, ettei sovellus toimi käyttämässäni Windows-ympäristössä ajetussa Dockerissa. Onnistuin kuitenkin aikani yritettyä saamaan sen onnistuneesti toimimaan. En kuitenkaan koe tarpeelliseksi esitellä kyseistä sovellusta tämän enempää, koska tämän työn kannalta käytetty tuotetietohallintajärjestelmä ei ole oleellista. Seuraavassa kuvassa esittelen toteuttamani integraation arkkitehtuuria.



Kuva 20. Saleor-tuotetietohallintajärjestelmä integraation arkkitehtuuri.

Kuvassa 20 on havainnollistettu jokainen integraatioon liittyvä mikropalvelu, ja niiden välinen kommunikaatio on piirretty viivoin. Viivan päällä oleva selite kuvaa, miten mikropalvelujen välinen kommunikaatio tapahtuu.

Laajennussovelluksien avulla on mahdollista tuoda lisäominaisuuksia Saleorin hallintakäyttöliittymään. Laajennussovellukseen voi luoda käytännössä minkälaisen tahansa verkkosovelluksen, joka esitetään iframe-elementissä Saleorin hallintakäyttöliittymässä, laajennussovelluksen hallintanäkymässä. Täten voidaan mahdollistaa tässäkin työssä toteutettu, käyttäjän käynnistämä tuotetietojen vienti tuotetietohallintajärjestelmästä Saleoriin.

Muutoin mikropalveluiden välinen kommunikaatio tapahtuu REST- ja GraphQL-rajapintojen avulla lukuun ottamatta mikropalveluiden ja tietokantojen välistä kommunikaatiota.

Laajennussovellukset tulee rekisteröidä Saleoriin. Rekisteröiminen tapahtuu Saleorin GraphQL-rajapinnan kautta. Saleorin tarjoamassa SaaS-palvelussa laajennussovelluksen rekisteröiminen onnistuisi Saleorin komentorivityökalun avulla, mutta omalla tietokoneella Saleoria ajaessa täytyy rekisteröiminen hoitaa ns. käsin GraphQL-rajapintakutsuna. Tämän rajapintakutsun mukana kerrotaan Saleorille laajennussovelluksen nimi, verkko-osoite mistä kyseisen laajennussovelluksen manifesti on saatavilla sekä käyttöoikeudet mitä laajennussovellus tulee tarvitsemaan. Käyttöoikeuksia on useita, mutta toteuttamani

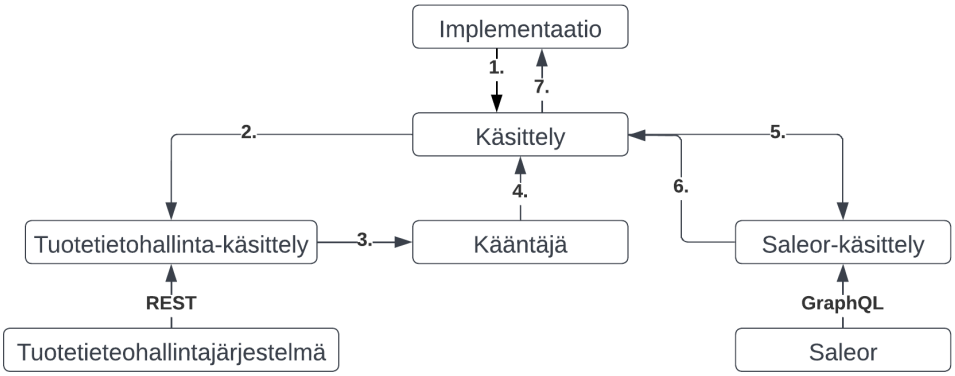
laajennussovellus tarvitsee vain MANAGE_PRODUCTS-käyttöoikeuden. Manifestissa määritellään tietoa laajennussovelluksen versioinnista, nimestä, sekä verkko-osoitteet josta laajennussovelluksen rekisteröiminen onnistuu ja mistä mahdollinen hallintakäyttöliittymän laajennus löytyy. Onnistuneen rekisteröinnin päätteeksi Saleor tekee REST-rajapintapyynnön aiemmin mainittuun osoitteeseen, josta rekisteröiminen onnistuu. Pyynnön mukana välitetään laajennussovellukselle myönnetty pääsytunnus, joka on voimassa ikuisesti, ja jonka avulla laajennussovellus voi tehdä pyyntöjä Saleorin GraphQL-rajapintaan. Tämä pääsytunnus tulee tallentaa jonnekin sovelluksen saataville, koska sitä ei ole mahdollista saada uudestaan. Tekemässäni laajennussovelluksessa se tallennetaan sovelluksen ympäristömuuttujiin.

Integraatio-mikropalvelun toteutan niin sanottuna lambda-funktiona, joka kirjoitetaan Kotlin-ohjelmointikielellä ja Quarkus-kirjastolla. Mikropalvelulla on oma REST-rajapinta, johon oikean pyynnön suorittaminen käynnistää tuotetietojen viemisen Saleoriin.

Jottei tämän työn pituus karkaisi, en käy yksityiskohtaisesti läpi integraatio-mikropalvelun lähdekoodia. Tekemäni sovelluksen lähdekoodi on kuitenkin saatavilla GitHubissa ja dokumentaatiossa olen kuvannut myös sovelluksen toteutusta.

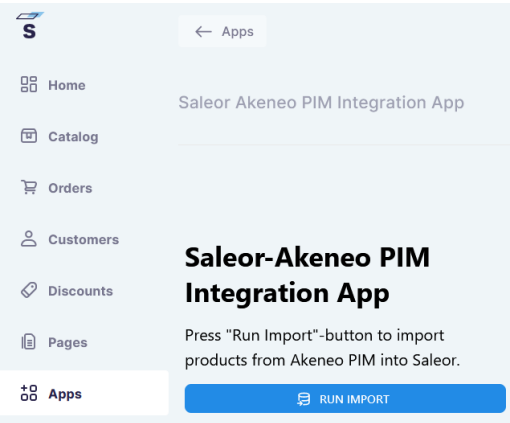
Tässä työssä jokaista mikropalvelua ajetaan omalla tietokoneellani Dockerissa, mutta toimeksiantajayrityksessä palveluiden ylläpito hoidetaan pääsääntöisesti AWS:sä (Amazon Web Services), toki Docker-kontteina sielläkin.

AWS:n (2022) mukaan lambda-funktion ajatuksena on, että ylläpidosta maksetaan vain silloin kun palvelua ajetaan. Palvelu on siis oletuksena skaalattu alas, kunnes jokin ennalta määritetty tapahtuma laukaisee palvelun käynnistämisen, operaationsa suorittamisen ajaksi. Tässä tapauksessa laukaiseva tapahtuma on HTTP-pyyntö johonkin sovelluksen päätepisteeseen. Seuraavassa kuvassa käsitellään integraatio-mikropalvelun arkkitehtuuria.



Kuva 21. Saleor-tuotetietohallintajärjestelmä integraatio mikropalvelun arkkitehtuuri.

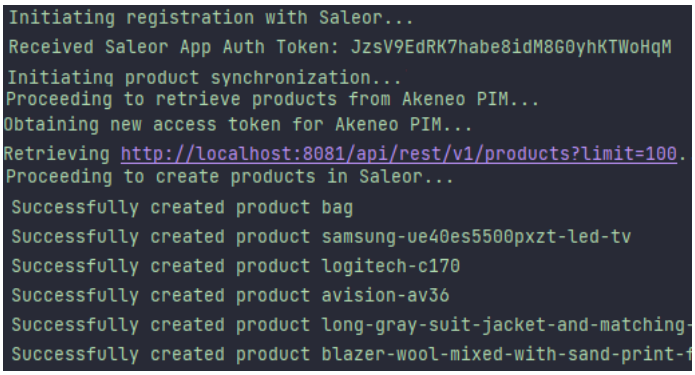
Kuvassa 21 ylimpänä oleva implementaatiotaso hoitaa saapuvien REST-rajapintakutsujen käsittelyn auktorisoinnin, ja onnistuneen auktorisoinnin jälkeen kutsuu käsittelytasoa. Käsittelytason vastuulla on varsinaisen logiikan toteuttaminen. Käsittelytaso kutsuu ensin tuotetietohallintakäsittelytasoa, joka tekee REST-rajapintakutsuja Akeneo PIM:n rajapintaan. Akeneo PIM:n rajapinnan palauttaessa tuotetietonsa vastauksena rajapintakutsuun, välitetään ne kääntäjälle. Koska data, tässä tapauksessa tuotetiedot, ovat eri lähteissä eri muodossa, hoitaa kääntäjä datan kääntämisen oikeaan, Saleorin hyväksymään, muotoon. Kääntäjä palauttaa oikean muotoiset tuotetiedot käsittelytasolle, joka välittää ne eteenpäin Saleor-käsittelytasolle. Saleor-käsittelytason tehtävänä on suorittaa kutsuja Saleorin GraphQL-rajapintaan. Onnistuneen operaation jälkeen visualisoinnin vuoksi integraatio-mikropalvelu palauttaa vastauksena saamaansa REST-rajapintakutsuun onnistuneesti vietyjen tuotetietojen määrän. Seuraavassa kuvassa katsotaan, miltä toteuttamani integraatiosovellus näyttää Saleorin hallintakäyttöliittymässä.



Kuva 22. Saleor-tuotetietohallintajärjestelmä integraatio Saleorin hallintakäyttöliittymässä.

Kuvassa 22 oikealla alhaalla näkyvä osio on osa integraatiosovellusta. Se sijaitsee laajennussovelluksen rekisteröinnin yhteydessä määritellyssä hallintakäyttöliittymän laajennusosoitteessa. Kuten aiemmin mainitsin, voi hallintakäyttöliittymän laajennukseksi toteuttaa miltei minkälaisen verkkosovelluksen tahansa, koska se esitetään iframe-elementissä.

Tässä tapauksessa toteutin yksinkertaisen React-sovelluksen, joka suorittaa Run import-napin painalluksella POST-pyynnön integraatiolaajennuksen REST-rajapintaan. Tämä käynnistää tuotetietojen viemisen Akeneo PIM:stä Saleoriin. Vieminen olisi myös mahdollista toteuttaa esimerkiksi ajastetusti, mutta tässä työssä toteutin sen nappina, jotta toimivuutta on selkeämpi testata ja havainnollistaa. Seuraavassa kuvassa katsotaan mitä integraatiosovelluksessa tapahtuu.



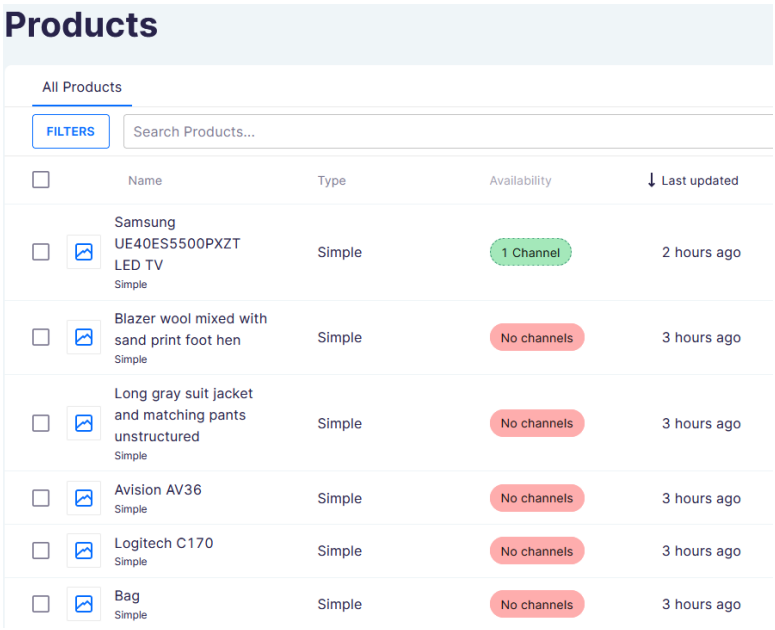
Kuva 23. Saleor-tuotetietohallintajärjestelmä integraatiosovelluksen loki.

Kuvan 23 kaksi ensimmäistä riviä liittyvät integraatiosovelluksen rekisteröimiseen Saleoriin. Saadessaan POST-pyynnön osoitteeseen, jossa rekisteröiminen onnistuu, tulostetaan tästä ilmoitus komentoriville. Onnistuneen rekisteröitymisen jälkeen Saleorin myöntämä pääsytunnus tulostetaan myös, josta sen kopioiminen integraatiosovelluksen ympäristömuuttujiin onnistuu.

Painettaessa kuvassa 23 näkyvää Run import-nappia, integraatiosovellus saa POST-pyynnön tuotetietojen viemisen käynnistävään osoitteeseen, josta tulostetaan myös ilmoitus komentoriville. Seuraavaksi käsittelytaso kertoo aloittavansa tuotetietojen haun Akeneo PIM:n REST-rajapinnasta. Tämän jälkeen tulee hakea Akeneo PIM:ltä voimassa oleva pääsytunnus. Voimassa olevalla pääsytunnuksella voidaan hakea varsinaiset tuotetiedot.

Tuotetietojen haun jälkeen kääntäjä muuntaa saadun datan Saleorin hyväksymään muotoon, josta ilmoituksen tulostaminen komentoriville on unohtunut. Sekä Saleorissa, että erinäisissä tuotetietohallintajärjestelmissä on mahdollista määrittää tuotteille lukematon määrä erilaisia ominaisuuksia. Koska tässä työssä toteutetaan prototyyppi, kääntäjä kääntää vain toiminnan kannalta oleellisimmat tuotetiedot.

Kääntäjän jälkeen tuotetiedot viedään Saleoriin GraphQL-rajapinnan kautta, ja jokaisen jälkeen komentoriville tulostetaan ilmoitus siitä, onnistuiko vienti vai ei. Tässä tapauksessa jokaisen tuotetiedon vieminen onnistui. Seuraavassa kuvassa tarkastellaan Saleorin tuotelistausta viennin jälkeen.



Kuva 24. Viedyt tuotteet Saleorissa.

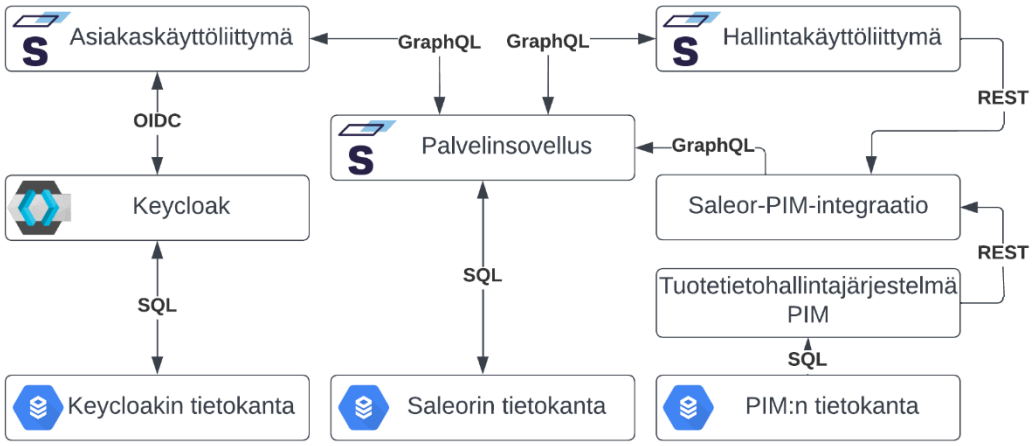
Kuvasta 24 nähdään Saleorin hallintakäyttöliittymästä, että onnistuneen viennin jälkeen tuotetiedot on oikeasti viety Akeneo PIM:stä Saleoriin. Jokainen tuote on yksinkertaistamisen vuoksi määritelty olemaan tuotetyyppiä Simple. Viemisen jälkeen täytyi käydä vielä Saleorin hallintakäyttöliittymässä lisäämässä tuote saatavaksi jollekin verkkokaupan kanavalle. Tuotteiden kuvien/muiden medioiden viemistä ei tässä työssä toteutettu.

Nyt myös tuotetietohallintajärjestelmä on onnistuneesti integroitu Saleoriin. Integraatiosovelluksen tekeminen osoitti, että lähes tulkoon mitä tahansa järjestelmiä on mahdollista integroida Saleoriin. Valmis tuotetietohallintajärjestelmä-

integraatio hoitaisi myös ainakin varastosaldojen päivittämisen tilausten tekemisen yhteydessä, sekä synkronoisi tuotetietojen muokkaamisen Saleorin ja tuotetietohallintajärjestelmän välillä. Molemmat ominaisuudet ovat oppimani perusteella hyvinkin mahdollista toteuttaa. Miltei jokaisesta, asiakas- tai hallintakäyttöliittymässä, tehdystä tapahtumasta Saleor lähettää webhookin, johon integraatiosovelluksen olisi mahdollista reagoida halutulla tavalla. Seuraavassa luvussa käydään vielä läpi Saleorin toimintaa ostotapahtumassa toteuttamieni integraatioiden jälkeen.

5.3 Työn tulosten tarkastelu

Tarkastellaan ensin tämän opinnäytetyön tuloksena syntyneen toteutuksen arkkitehtuuria. Sen jälkeen tarkastellaan Saleorin toimintaa integraatioiden jälkeen. Seuraavassa kuvassa esitellään valmiin toteutuksen arkkitehtuuria.



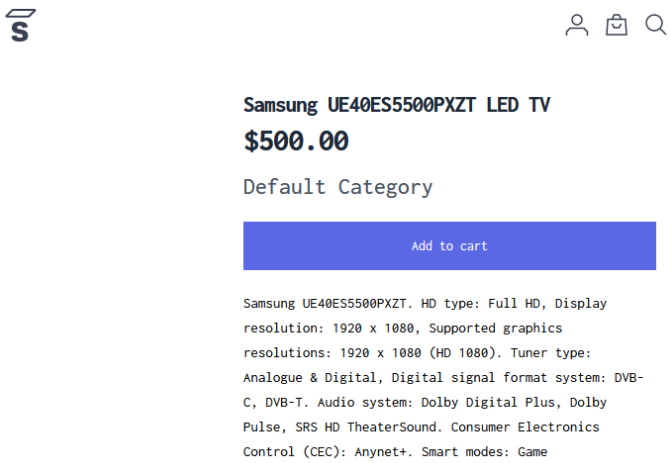
Kuva 25. Valmiin prototyypin arkkitehtuurikaavio

Integraatioiden jälkeen tavoitteena on olla kuvan 25 mukaista mikropalveluarkkitehtuuria toteuttava prototyyppi. Jokainen laatikko on oma mikropalvelunsa. Mikropalveluiden väliseen kommunikointiin käytetty tapa on kirjoitettu mikropalveluiden välisiin nuoliin. Tietokantaoperaatiot eivät sinänsä tähän työhön liity, mutta arkkitehtuurikaaviossa on hyvä havainnollistaa, että palveluilla on omat tietokantansa.

Saleorin mikropalveluiden välinen kommunikointi on jo toteutettu Saleorin demosovelluksessa. Keycloak-integraation jälkeen asiakaskäyttöliittymän ja

Keycloakin välinen kommunikointi tapahtuu OIDC-standardin mukaisesti. Tuotetietohallintajärjestelmä-integraation jälkeen olisi tarkoitus, että hallintakäyttöliittymän kautta voidaan lähettää REST-rajapintakutsu integraatiosovellukselle, joka käynnistää tuotetietojen hakemisen tuotetietohallintajärjestelmästä, ja vie ne Saleoriin.

Tässä työssä toteutettujen integraatioiden jälkeen käyttäjän tulisi voida rekisteröityä ja kirjautua sisään Saleoriin Keycloakin kautta, sekä voida ostaa tuote tai tuotteita, jotka on tuotu tuotetietohallintajärjestelmästä. Sisäänkirjautumis- ja rekisteröitymislomakkeet ovat melko varmasti ennestään tuttuja, joten siirrytään suoraan varsinaiseen ostotapahtumaan. Seuraavassa kuvassa tarkastellaan tuotetietohallintajärjestelmästä tuotua tuotetta Saleorin asiakaskäyttöliittymässä.

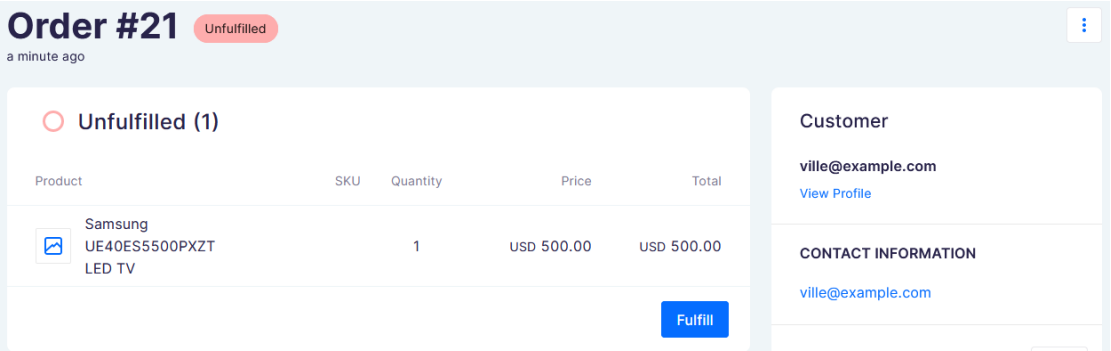


Kuva 26. Viety tuote Saleorin asiakaskäyttöliittymässä.

Kuvassa 26 nähdään tuotetietohallintajärjestelmästä tuotu tuote Saleorin asiakaskäyttöliittymässä. Tuote on mahdollista lisätä ostoskoriin, minkä jälkeen se on mahdollista ostaa. Integraatiosovelluksen yksinkertaistamisen vuoksi jokainen tuotetietohallintajärjestelmästä tuotu tuote kuuluvat samaan Default Category-tuotekategoriaan. Tuotteen nimi, hinta ja tuotekuvaus on kuitenkin sellaisenaan tuotu tuotetietohallintajärjestelmästä.

Saleorin demosovelluksessa on mahdollista suorittaa testiostotapahtuma. Ostotapahtuma ei poikkea mitenkään tavallisesta verkkokauppaostotapahtu-

masta, joten siihen ei tässä työssä paneuduta sen enempää. Sen sijaan seuraavassa kuvassa tarkastellaan tehtyä ostotapahtumaa Saleorin hallintakäyttöliittymässä.



Kuva 27. Toteutunut ostotapahtuma Saleorin hallintakäyttöliittymässä.

Kuvasta 27 on nähtävissä, että ostotapahtuma on suoritettu käyttäjänä, jonka olen rekisteröitynyt Keycloakkiin ja, että ostettu tuote on tosiaan tuotetietohallintajärjestelmästä tuotu tuote. Kuvasta on rajattu pois toimitusmuoto- ja osoite, mutta ne tallennetaan kuitenkin Saleorin omaan tietokantaan, josta ne yksilöidään Keycloakin käyttäjiin. Täten asiakkaan on mahdollista tarkastella muun muassa tallentamiaan toimitusosoitteita omasta hallinnastaan.

Kaiken kaikkiaan Keycloak ja tuotetietohallintajärjestelmä on onnistuneesti integroitu Saleoriin ja työn tavoite on saavutettu. Kuten edellisessä luvussa mainitsin, vaatii tuotetietohallintajärjestelmä vielä huomattavasti lisää töitä, jotta sen käyttöä voitaisiin ajatella kaupallisella tasolla. Tätä työtä tehdessä opin, että Saleoria on mahdollista laajentaa todella paljon laajennussovelluksien avulla. Jokaiselle Saleorin hallintakäyttöliittymän sivulle on mahdollista tuoda laajennuksia, joilla lisätä toiminnallisuuksia tai integraatioita. Halutessaan on myös täysin mahdollista käyttää Saleorista pelkkää palvelinsovellusta, jonka päälle rakentaa itselleen sopivat käyttöliittymät.

6 PÄÄTÄNTÖ

Päädyin valitsemaan tämän opinnäytetyön aiheen toimeksiantajan ehdottamista vaihtoehdoista aiheen monimuotoisuuden ja Saleorin verrattain nuoren iän vuoksi. Oli mielenkiintoista, kun pääsi käsittelemään laaja-alaisesti erilaisia

ohjelmistokehityksen osa-alueita sekä uutta teknologiaa. Uudesta teknologiasta koin harmiksi sen, että oikeasti käyttökelpoisia lähteitä muutamista aiheista oli todella hankala löytää.

Opinnäytetyön tulokset auttavat toimeksiantajaa kehittämään Saleoriin perustuvaa tarjontaa, teknisiä kyvykkyyksiä sekä ratkaisujen konseptointia. Selvitetyt kysymykset ovat keskeisessä roolissa siinä, että Metatavulla ymmärretään paremmin Saleorin mahdollisuuksia, arkkitehtuuria ja laajentamista. (Lampi 2022.)

Toteuttamaani tuotetietohallintajärjestelmää-integraatiota pitäisi oikeassa käytössä jatkokehittää jonkin verran. Ensinnäkin verkkokaupassa toteutuneiden ostosten seurauksena varastosaldot tulisi päivittyä automaattisesti tuotetietohallintajärjestelmään. Myös tuotetietojen muokkaamisen Saleorissa tulisi synkronoitua tuotetietohallintajärjestelmään. Molemmat edellä mainituista ominaisuuksista pitäisi olla kohtalaisen kivuttomia toteuttaa, mutta tässä opinnäytetyössä oli tavoitteena toteuttaa vain niin sanotun soveltuvuusselvityksen (engl. proof of concept) määrittelyn täyttävä prototyyppi. Työn edetessä pohdittiin myös mahdollisuutta verkkokaupan käyttäjärooliperustaisen kustomoinnin toteuttamisesta, mutta siitä luovuttiin työn laajuuden rajaamiseksi. Myös opinnäytetyön tuloksen julkaisemista pilveen pohdittiin opinnäytetyön jälkeen, josta sitä voitaisiin käyttää demonstraationa integraatioiden toteuttamisesta Saleoriin.

Opinnäytetyössä käytetyt ohjelmointiteknologiat olivat GraphQL:ää lukuun ottamatta itselleni entuudestaan tuttuja työni ja opiskelujeni puolesta, mutta koin silti omatoimisen tekemisen opettaneen minua todella paljon lisää. Ennen opinnäytetyön aloittamista en ollut kuullut Saleorista muuta kuin nimen, mutta opinnäytetyön edetessä koin pääseväni kohtuullisen hyvin sisälle siihen minkälainen kokonaisuus on kyseessä. Toivottavasti myös toimeksiantaja pääsee hyödyntämään tätä kokemusta.

LÄHTEET

Akeneo. 2022. What is a PIM and why do you need one? WWW-dokumentti. Saatavissa: https://www.akeneo.com/what-is-a-pim/ [viitattu 27.10.2022]

Amazon Web Services. 2022a. What is an API? WWW-dokumentti. Saatavissa: https://aws.amazon.com/what-is/api/ [viitattu 26.8.2022]

Amazon Web Services. 2022b. What is AWS Lambda? WWW-dokumentti. Saatavissa: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html [viitattu 28.10.2022]

Auth0. 2022a. Authentication vs. Authorization. WWW-dokumentti. Saatavissa: https://auth0.com/docs/get-started/identity-fundamentals/authentication-and-authorization [viitattu 4.9.2022]

Auth0. 2022b. Prevent Attacks and Redirect Users with OAuth 2.0 State Parameters. WWW-dokumentti. Saatavissa: https://auth0.com/docs/secure/attack-protection/state-parameters [viitattu 18.10.2022]

Crasman. 2022. Akeneo PIM – rikkaan tuotetiedonhallinta. WWW-dokumentti. Saatavissa: https://www.crasman.fi/palvelut/teknologiat/akeneo-pim-tuotetiedonhallinta [viitattu 28.10.2022]

Docker. 2022. What container? WWW-dokumentti. Saatavissa: https://www.docker.com/resources/what-container/ [viitattu 1.10.2022]

GraphQL. 2022. WWW-dokumentti. Saatavissa: https://graphql.org [viitattu 26.8.2022]

Hyvärinen, J. 2019. Mikropalveluarkkitehtuuri - milloin ja milloin ei? Blogi. Saatavissa: https://digitalillustrated.com/mikropalveluarkkitehtuuri-milloin-ja-milloin-ei/ [viitattu 31.8.2022]

IBM. 2022. Identity access management. WWW-dokumentti. Saatavissa: https://www.ibm.com/topics/identity-access-management [viitattu 14.9.2022]

JWT. 2022. Introduction to JSON Web Tokens. WWW-dokumentti. Saatavissa: https://jwt.io/introduction [viitattu 17.9.2022]

Keycloak. 2022a. Open Source Identity and Access Management. WWW-dokumentti. Saatavissa: http://www.keycloak.org [viitattu 14.9.2022]

Keycloak. 2022b. Securing Apps. WWW-dokumentti. Saatavissa: https://www.keycloak.org/docs/latest/securing_apps [viitattu 4.9.2022]

Keycloak. 2022c. Server Administration Guide. WWW-dokumentti. Saatavissa: https://www.keycloak.org/docs/latest/server_admin/ [viitattu 17.9.2022]

Lampi, M. 2022. Liiketoiminta- ja kehitysjohtaja. Haastattelu 7.11.2022. Metatavu Oy.

Linden, M. 2017a. Esimerkki rooliin perustuvasta pääsynvalvonnasta, 13. WWW-dokumentti. Saatavissa: https://core.ac.uk/download/pdf/250168612.pdf [viitattu 2.10.2022]

Linden, M. 2017b. Identiteetin- ja pääsynhallinta. Tampereen teknillinen yliopisto. Tietotekniikan laboratorio. Raportti. Saatavissa: https://core.ac.uk/download/pdf/250168612.pdf [viitattu 2.10.2022]

Mencel M. 2020. Why headless is the future of e-commerce? Blogi. Saatavissa: https://saleor.io/blog/why-headless-is-the-future-of-ecommerce-122/ [viitattu 1.10.22]

Mozilla. 2022. Cross-Origin Resource Sharing. WWW-dokumentti. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS [viitattu 17.10.2022]

Onelogin. 2022. What is identity and access management (IAM)? WWW-dokumentti. Saatavissa: https://www.onelogin.com/learn/iam [viitattu 14.9.2022]

Radhakrishnan, V. 2021. E-Commerce industry and its effect on the world today. *International Research Journal on Advanced Science Hub* 2, 23–29. Verkkolehti. Saatavissa: http://dx.doi.org/10.47392/irjash.2021.026 [viitattu 1.11.2022]

Red Hat. 2021a. What is containerization? WWW-dokumentti. Saatavissa: https://www.redhat.com/en/topics/cloud-native-apps/what-is-containerization [viitattu 1.10.2022]

Red Hat. 2020b. What is a REST API? WWW-dokumentti. Saatavissa: https://www.redhat.com/en/topics/api/what-is-a-rest-api [viitattu 2.11.2022]

Saleor. 2022a. Before you start. WWW-dokumentti. Saatavissa: https://docs.saleor.io/docs/3.x/dashboard/before-you-start [viitattu 17.9.2022]

Saleor. 2022b. OpenID Connect (OIDC). WWW-dokumentti. Saatavissa: https://docs.saleor.io/docs/3.x/developer/available-plugins/openid-connect [viitattu 17.10.2022]

Saleor. 2022c. Architecture. WWW-dokumentti. Saatavissa: https://docs.saleor.io/docs/3.x/developer/getting-started/architecture [viitattu 17.9.2022]

Saleor. 2022d. Generating authentication URL – Workflow. WWW-dokumentti. Saatavissa: https://docs.saleor.io/docs/3.x/developer/available-plugins/openid-connect#saleor-as-a-client-for-authorization-server [viitattu 17.10.2022]

Saleor. 2022e. ExternalAuthenticationUrl GraphQL-rajapintakutsu. WWW-dokumentti. Saatavissa: https://docs.saleor.io/docs/3.x/developer/available-plugins/openid-connect#saleor-as-a-client-for-authorization-server [viitattu 20.10.2022]

Saleor. 2022f. Obtaining access token – Workflow. WWW-dokumentti. Saatavissa: https://docs.saleor.io/docs/3.x/developer/available-plugins/openid-connect#saleor-as-a-client-for-authorization-server [viitattu 20.10.2022]

Saleor. 2022g. ExternalObtainAccessTokens GraphQL-rajapintakutsu. WWW-dokumentti. Saatavissa: https://docs.saleor.io/docs/3.x/developer/available-plugins/openid-connect#saleor-as-a-client-for-authorization-server [viitattu 20.10.2022]

Stalians, S. 2022. What is headless commerce — and is it right for you? Blogi. Saatavissa: https://blogs.oracle.com/cx/post/defining-headless-commerce [viitattu 1.10.2022]

Vitale, T. 2019. Introducing Keycloak for Identity and Access Management. Blogi. Saatavissa: https://www.thomasvitale.com/introducing-keycloak-identity-access-management/ [viitattu 14.9.2022]

Vue. 2022. Headless architecture: What Is It, And Why Is It The Future? WWW-dokumentti. Saatavissa: https://vuestorefront.io/headless-architecture [viitattu 2.11.2022