jamk.fi

Shopify-sovelluskehitys

Jussi Ahola

Opinnäytetyö
Joulukuu 2020
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Ahola, Jussi	Opinnäytetyö, AMK	Joulukuu 2020
	Sivumäärä 55	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Shopify-sovelluskehitys

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Manninen Pasi, Niemi Kari

Toimeksiantaja(t)
-

Tiivistelmä

Opinnäytetyön tavoitteena oli tutustua Shopify-sovelluskehitykseen, sen ominaispiirteisiin ja siihen kuinka vaativaa Shopify-sovelluskehitys yleisesti ottaen on. Työstä oli tavoitteena saada lopputulokseksi Shopify-sovellus, joka on helposti jatkokehitettävissä ja tulevaisuudessa myös jatkokehityksen myötä julkaistavissa Shopify App Storeen.

Työ oli kehittämistutkimusta eli samaan aikaan kehitettiin tuotetta ja tutkittiin käsillä olevaa aihetta. Työ suoritettiin noin kuukauden aikajänteellä, viikoittain toistuvina intensiivisinä 3-4 työpäivän mittaisina sarjoina. Työn aikana sovelluksen ominaisuuksia suunniteltiin ja kehitettiin eteenpäin samanaikaisesti, jatkuvasti kartoittaen tärkeimpiä ja vaikuttavampia ominaisuuksia verkkokauppiaan arki huomioon ottaen. Näin ollen sovelluksesta saatiin toimiva ja tavallisimpiin verkkokauppiaan haasteisiin vastaava.

Kehitystyössä lähtökohtana oli, että sovelluksesta ei haluta kerralla valmista, vaan keskitytään siihen, että sovellus säilyy jatkokehityskelpoisena ja sovelluksen kehityksen aikana keretään perehtymään riittävästi Shopify-sovelluskehityksen ominaispiirteisiin ja sen haastavuuden arviointiin.

Sovelluksen kehityksen yhteydessä havaittiin, että Shopify-sovelluskehityksessä haasteita aiheuttaa Shopifyn App Bridge -kirjaston ymmärtäminen, sekä sovelluksen asennuksen yhteydessä hyödynnettävä OAuth-mekanismin asianmukainen implementointi. Kehitystyön aikana molemmat haasteista kuitenkin selvitettiin ja lopputuloksena saatiin toimiva Shopify-sovellus, jota käyttämällä verkkokauppias voi saavuttaa käyttötapauksesta riippuen jopa merkittävää ajallista säästöä muokatessaan verkkokauppansa tuotetietoa sovelluksen avulla.

Avainsanat (asiasanat)
Shopify, NodeJS, ReactJS, Shopify App Bridge, TailwindCSS, Sequelize.js, MySQL, JWT

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Ahola, Jussi	Bachelor's thesis	December 2020
		Language of publication: Finnish
	Number of pages: 55	Permission for web publication: x

Title of publication
Shopify App Development

Degree programme
Information and Communication Technologies

Supervisor(s)
Manninen Pasi, Niemi Kari

Assigned by
-

Abstract

The aim of the thesis was to gather information about Shopify app development, what special requirements it may hold, and how demanding Shopify app development is in general. The main goal of the thesis was to develop a working Shopify app, which could be easily developed further in the future, and possibly with some further development work, published in the Shopify App Store.

The project was a mixture of development and theory, which means that the theory was applied directly to the project, and the result was documented and studied. Development work was done in 3-4 day long sprints once per week, for one month in total. During development, the features for the app were planned and executed in rapid succession, and the requirements of the Shopify merchant were accounted as well. Developing the app in this way it was possible to create an app which answers the merchant's true needs and challenges in their day-to-day work.

The main idea was that the result should not be a final polished application with all the features, but more like bare-bones version of the application, which could be further developed in the future with ease. It was essential, that during the project time was spent evaluating the Shopify app development process in general, and its relative difficulty in comparison to other common development tasks.

During development it was clear that Shopify App Bridge library and OAuth process may cause some confusion when developing the app. This confusion was quickly cleared, and development showed great results with app, that could be installed and used by merchants, who could save time and resources by using the final application.

Keywords/tags (subjects)
Shopify, NodeJS, ReactJS, Shopify App Bridge, TailwindCSS, Sequelize.js, MySQL, JWT

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Johdanto

1.1 Työn taustaa

Työn motivaationa toimi halu tutkia Shopify-sovelluskehitystä, sekä tarve omalle Shopify-sovellukselle, jolla voidaan tehdä massamuokkauksia Shopify-verkkokauppojen tuotetietoon. Shopify App Storessa on saatavilla useita tietojen massamuokkaukseen pystyviä sovelluksia, mutta niiden joukosta puuttui sovellus, jolla voitaisiin suodattaa verkkokaupasta tuotteita erityisesti niiden sisältämän metatiedon perusteella. Tuotetiedon muokkaaminen käyttäjän valitsemiin tuotteisiin haluttiin mahdollistaa suoraan sovelluksen käyttöliittymässä.

1.2 Työn tavoite ja tutkimusmenetelmä

Työn tavoitteena oli tutustua sovelluskehitykseen Shopify-verkkokauppa-alustan yhteydessä. Työn kannalta oleellista oli selvittää, millaisia erityispiirteitä Shopify-sovelluskehitykseen liittyy ja kuinka haastavaa sovellusten kehittäminen Shopify-alustaa varten yleisesti ottaen on. Työstä oli tarkoituksena saada tulokseksi toimiva sovellus, jonka verkkokauppias voi asentaa omaan Shopify-verkkokauppaansa ja hallinnoida edistyneesti sen avulla oman verkkokauppansa tuotteita suoraan sovelluksen käyttöliittymässä. Sovelluksen julkaisua Shopify App Store -sovelluskauppaan ei käsitelty tässä opinnäytetyössä aihealueen rajauksellisista syistä.

Työ oli kehittämistutkimusta, jonka tarkoituksena oli luoda Shopify-sovellus. Sovelluksen kehittämisessä huomiota kiinnitettiin erityisesti sen jatkokehityskelpoisuuteen. Työn tarkoituksena ei siis ollut saada valmiiksi täysin valmista sovellusta kaikkine ominaisuuksineen, vaan hyvä ja toimiva peruspohja tulevaa jatkokehitystä var-

ten, jolloin ominaisuuksia tullaan lisäämään sovellukseen vaiheittain. Muutamia perusominaisuuksia sisällytettiin sovellukseen, jotta voitiin varmistua siitä, että sovelluksen logiikka on toimiva ja jatkokehityskelpoinen. Tällä tavalla toimien työssä voitiin keskittyä olennaiseen eli siihen mitkä ovat Shopify-sovelluskehityksen ominaispiirteet ja miten ne pitää ottaa huomioon sovelluksen arkkitehtuurissa sekä ominaisuuksien suunnittelussa.

2 Shopify

2.1 Yleistä

Shopify on verkkokauppa-alusta, jolla kuka tahansa voi helposti perustaa itselleen verkkokaupan. Shopify sisältää itsessään kaikki verkkokaupalle olennaiset ominaisuudet, kuten tilausten ja tuotteiden hallinnan sekä käyttäjätilit asiakkaille. Shopify-alustaa on mahdollista hyödyntää myös tavallisen kivijalkakaupan myyntipisteenä Shopify Point of Sale -ominaisuuden avulla. Verkkokaupassa olevia tuotteita on siis mahdollista myydä samanaikaisesti niin kivijalkaliikkeessä, että verkkokaupassa. (What is Shopify n.d.)

2.2 Shopify verkkokaupan osat

2.2.1 Yleistä

Shopify-verkkokauppa koostuu karkeasti kolmesta osasta. Ensimmäinen osa on verkkokaupan tuotetieto. Tuotetiedolla tarkoitetaan verkkokaupassa olevia tuotteita ja niihin liittyvää tietoa, kuten hintatiedot, varastosaldot, tuotteen kuvaus, tunnisteet, tuotekuvat ja muut mahdolliset lisätiedot.

Toinen osa on verkkokaupassa oleva teema. Teema käsittää verkkokaupan asiakkaalle näkyvät osat verkkokaupasta eli käyttöliittymän. Teeman avulla esitetään verkkokaupassa olevaa tuotetietoa asiakkaalle halutulla tavalla. Shopify-teeman voi joko ostaa valmiina tai kehittää itse alusta lähtien. Ostettua teemaa voi myös muokata itse, joten mikäli valmiista teemoista löytyy hyvä vaihtoehto, sen voi ostaa ja jatkokehittää mieleisekseen.

Kolmas osa on verkkokaupan hallintapaneeli, joka verkkokauppiaan näkökulmasta ikään kuin yhdistää tuotetiedon ja teeman toisiinsa. Verkkokaupan hallintapaneelin avulla on mahdollista muokata niin teeman asetuksia, kuin tuotetietoakin. Tuotteita voidaan ryhmitellä hallintapaneelissa erilaisiin kokoelmiin, jotka voidaan myöhemmin esittää teeman avulla verkkokaupan käyttäjälle.

2.2.2 Tuotetieto

Tuotetieto kuvaa sitä tietoa, joka kuuluu verkkokaupan tuotteille. Yksinkertaisimmillaan tämä voi tarkoittaa esimerkiksi yksittäisen tuotteen tietoja verkkokaupassa. Tuotetieto sanana voi käsittää myös laajemman joukon tuotteita eli on mahdollista puhua verkkokaupan tuotetiedosta samalla viitaten kaikkien verkkokaupan tuotteiden muodostamaan joukkoon ja niihin liittyvään tietoon.

Yhdellä tuotteella voi olla useita variantteja. Variantti tarkoittaa erilaista versiota kuin itse päätuotteesta (Variants n.d.). Variantti yleensä luodaan silloin, kun tuote on käytännössä sama kuin itse päätuote, mutta tuotteen joku ominaisuus, esimerkiksi väri, vaihtelee. Varianttien avulla tuotteita ei tarvitse luoda turhaan useita, vaan voidaan luoda yksi päätuote ja sille useita variantteja. Tämä helpottaa tuotetiedon hallintaa, sillä tuotetiedon hallinnassa voidaan hyödyntää tietoa siitä, että tuotteen ja

variantin välillä on yhteys. Tuotteen ja variantin välisessä yhteydessä pätee hyvin sääntö: variantti voi kuulua tuotteelle, mutta tuote ei voi kuulua variantille.

Tuotteilla sekä varianteilla voi olla metatietoa. Metatiedon tarkoitus on lisätä tuotteille sellaista lisätietoa, jolle ei ole Shopifyn oletustiedoissa muuta paikkaa. Metatiedolla voidaan siis rikastaa tuotteen tai kaupan tietoja entisestään ja tarjota verkkokaupan asiakkaalle mahdollisimman paljon hyödyllistä tietoa tuotteesta tai palvelusta, jota verkkokauppias tarjoaa.

Metatietoa voidaan Shopifyssä liittää mm. yksittäiseen tuotteeseen, varianttiin, Shopify-kauppaan, kokoelmaan, asiakkaaseen, blogiin tai sivuun. Shopifyllä ei ole tällä hetkellä omaa sovellusta tai keinoa, jolla metatietoa voitaisiin hallita, joten sitä varten tarvitaan ulkopuolisen tahon kehittämä sovellus. (The Metafield Object n.d.)

Metatieto jaotellaan nimiavaruuksittain loogisesti eri osiin, metatietokenttää luodessa nimiavaruuden voi valita itse. Metatietokenttää lisätessä tulee valita metatietokentälle nimiavaruus, avain, arvo sekä tietotyyppi. Valideja tietotyyppejä metatietokentille ovat: integer, string ja json_string.

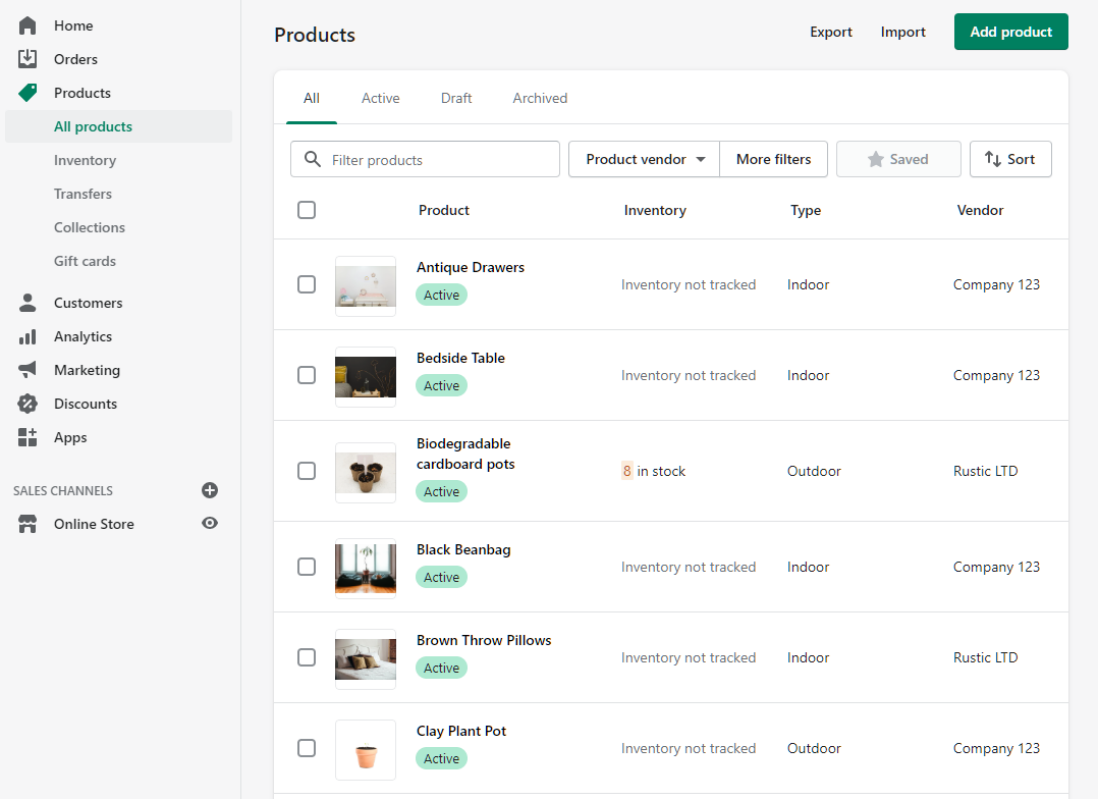
2.2.3 Teema

Shopify-teema tarkoittaa sivuston käyttöliittymää verkkokaupan asiakkaalle. Teeman tarkoitus on esittää verkkokaupan tuotetietoa ja sivuston muuta sisältöä asiakkaalle, yleensä mahdollisimman selkeällä tavalla, tähdäten hyvään ja suoraviivaiseen ostokokemukseen. Shopify-teemoja ei käsitellä tässä työssä tarkemmin, sillä ne eivät tässä tapauksessa liity Shopify-sovelluskehitykseen.

2.2.4 Hallintapaneeli

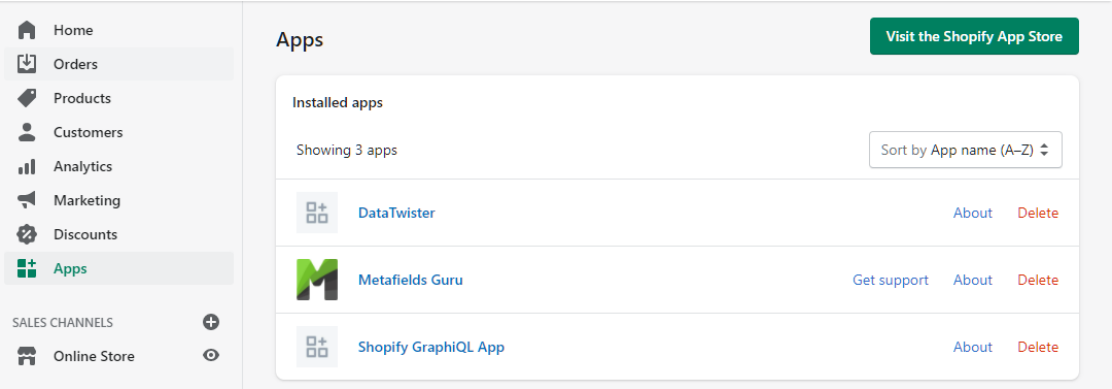
Shopify Admin -hallintapaneeli on työkalu, jolla verkkokauppias voi hallita verkkokauppansa eri osia ja asetuksia. Hallintapaneeliin täytyy kirjautua sisään ennen kuin sitä voidaan käyttää.

Hallintapaneelissa on mahdollista hallita kaikkia kaupan asetuksia, tilauksia, muokata ja luoda tuotteita, hallita asiakkaita, tarkastella sivuston analytiikkaa ja hallita markkinointia. (ks. kuvio 1). Hallintapaneelista käsin voidaan myös luoda alennuskuponkeja ja hallita kaupan teemaa ja teeman sisältämiä tiedostoja.



Kuvio 1. Tuotteiden listausnäkymä Shopify Admin -hallintapaneelissa

Hallintapaneelissa on oma osionsa sovelluksille, jotka on asennettu kauppiaan verkkokauppaan. Asennetut sovellukset löytyvät vasemman sivuvalikon kohdasta “Apps”. Verkkokauppias voi asentaa sovelluksia omaan kauppaansa Shopifyn App Storesta. Kun sovellus asennetaan kauppaan, niin se ilmestyy “Apps” osiossa näkyviin, josta asennettua sovellusta päästään käyttämään klikkaamalla sovelluksen nimestä (ks. kuvio 2).



Kuvio 2. Sovelluslistaus Shopify-hallintapaneelissa

2.3 Shopify rajapinnat

2.3.1 Admin API

Shopify Admin API on pääasiallinen väylä, jolla sovellukset ja palvelut kommunikoivat Shopifyn taustajärjestelmien kanssa. Rajapinnasta saadaan tietoa esimerkiksi kaupan tuotteista, tilauksista ja asiakkaista ja näitä tietoja on mahdollista hallita suoraan rajapintaa pitkin. Admin API on saatavilla kahtena eri versiona: REST-rajapintana ja GraphQL-rajapintana. (Shopify Admin API n.d.)

Shopify suosittelee käyttämään GraphQL-versiota rajapinnasta, sillä se nähdään monella tapaa tehokkaampana kuin tavanomainen REST-rajapinta. Tyypillisiä ongelmia REST-rajapintojen käytössä ovat mm. tarvittavan datan hakemisen raskaus. On hyvin tyypillistä, että rajapinnasta halutaan hakea sellaista dataa, joka vaatii useita HTTP-kutsuja käytettäessä REST-tyyppistä rajapintaa. REST-rajapinta palauttaa myös paljon sellaista dataa, jota sovellus ei välttämättä käytä. Tästä syntyy turhaa datan hakua ja siirtoa, joka ei ole tehokasta. GraphQL pyrkii ratkaisemaan nämä ongelmat palauttamalla vain sen tiedon, jota sovellus pyytää. GraphQL-rajapinnassa on myös vain yksi API-endpoint, josta data haetaan verrattuna siihen, että REST-API:ssa endpointteja on useita. (GraphQL and its benefits n.d.)

2.3.2 Storefront API

Shopify storefront API mahdollistaa Shopifyn verkkokaupan ominaisuuksien hyödyntämistä itse Shopify-verkkokaupan ulkopuolella. Storefront API:a voidaan hyödyntää esimerkiksi silloin, kun halutaan myydä Shopify-verkkokaupan tuotteita sellaisessa palvelussa, joka ei ole Shopify-järjestelmän päälle rakennettu. Tällaisia käyttötapauksia ovat esimerkiksi tuotteiden myyminen videopeleissä tai täysin Shopifystä irrallaan olevat sovellukset tai verkkosivustot. (Shopify Storefront API n.d.) Storefront API:a ei käytetä tämän työn sovelluksen toiminnassa, joten sen tarkempi tarkastelu suljetaan työstä pois.

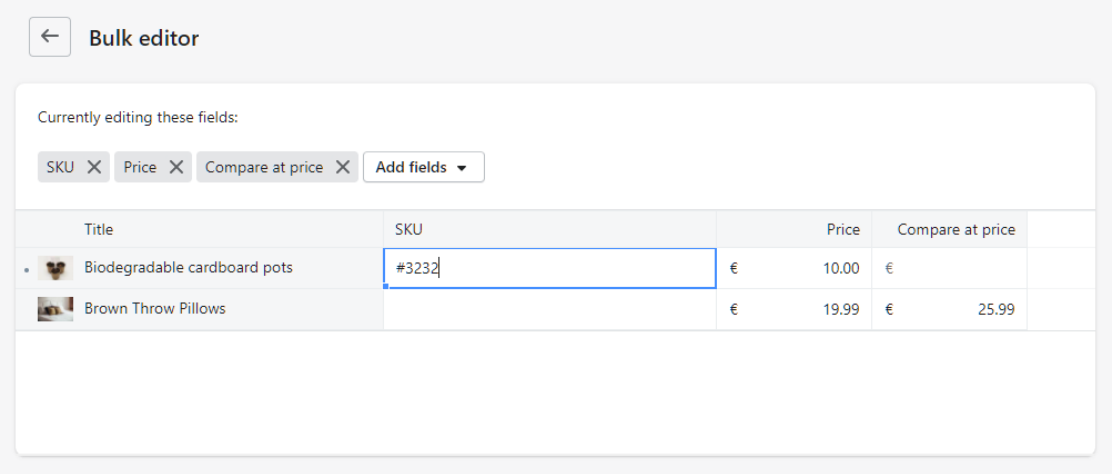
2.4 Tuotetiedon hallinta Shopifyn hallintapaneelissa

2.4.1 Tuotteiden lisäys, muokkaus ja poisto

Tuotteiden lisäys onnistuu navigoimalla hallintapaneelissa Products-kohtaan ja valitsemalla käyttöliittymästä “Add New Product”, jonka jälkeen täyttämällä tiedot tuotteelle. Tuotteiden muokkaus toimii pitkälti samalla tavalla kuin tuotteen luominen, mutta sen sijaan että valittaisiin “Add New Product”, niin klikataankin listauksessa

näkyvä tuote auki ja muokataan tiedot haluttuun muotoon. Tuotteiden poistaminen tapahtuu painikkeella ”Delete Product”, joka sijaitsee hallintapaneelissa tuotteen sivun alareunassa.

Tuotteiden hallinta toimii siis Shopifyssä pienissä määrissä varsin hyvin. Ongelmat nousevat siinä vaiheessa esiin, kun tuotteita pitäisi alkaa luomaan useita kymmeniä tai satoja kerralla tai muokata useita tuotteita samanaikaisesti. Shopifyllä on tuotelistauksessa hyvät perustason suodattimet, joilla tuotteita voidaan suodattaa esimerkiksi valmistajan tai tuotteessa olevan tunnisteen perusteella. Suodattimien avulla verkkokaupan tuotteista voidaan valita ne tuotteet, joita halutaan muokata. Shopifyn omalla massamuokkaimella (Bulk editor) voidaan muuttaa kaikkien valittujen tuotteiden tietoja samanaikaisesti (ks. kuvio 3).



Kuvio 3. Shopify bulk editor

Shopifyn massamuokkain on muuten hyvä, mutta sillä ei voida tehdä kovin edistyksellisiä muokkauksia, kuten esimerkiksi korvata tiettyjä tunnisteita toisilla tunnisteilla tai muokata tuotteiden kuvaustekstejä. Suurten datamäärien hallinta bulk editorilla

on myös hieman vaikeaa, sillä bulk editorin sivulle voidaan ladata kerralla vain rajallinen määrä tuotteita, joten useiden satojen tuotteiden muokkaus kerralla ei onnistu helposti.

2.4.2 Tuotetiedon hallintaan erikoistuneiden sovellusten hyödyt

Tuotetiedon hallintaa varten tehdyt sovellukset paikkaavat hyvin Shopifyn massamuokkaimen puutteita. Usein sovelluksilla voidaan tehdä muutoksia massana useisiin satoihin tuotteisiin kerralla melko vaivattomasti. Sovellusten käyttöliittymät ovat usein myös visuaalisesti miellyttävämpiä sekä paremmin toimivia, kuin Shopifyn oma massamuokkain.

Tarkoitusta varten tehdyt sovellukset mahdollistavat myös sellaisten asioiden tekemisen, mikä Shopifyn massamuokkain ei mahdollista ollenkaan. Esimerkiksi kuvaustekstien muokkaus ja tekstien korvaus toisella tekstillä eivät ole mahdollisia toimintoja Shopifyn massamuokkaimessa työn kirjoitushetkellä. Tällä hetkellä myöskään hintojen muokkaus prosenttimääräisesti ei ole mahdollista Shopifyn massamuokkaimessa.

Huomattava asia on myös se, että Shopifyn omat tuotteiden muokkausominaisuudet eivät tarjoa keinoa tuotteiden metatietojen lisäämiseen, muokkaamiseen tai poistamiseen. Tätä puutetta pyritään erityisesti täyttämään tämän työn aikana valmistuvassa sovelluksessa.

2.5 Shopify-sovellukset

2.5.1 Yleistä

Shopify-sovellus on palvelu, jonka verkkokauppias voi helposti ottaa käyttöön verkkokauppaansa asentamalla sen Shopifyn App Storesta. Jokainen Shopify-sovellus on

loppujen lopuksi vain verkkosivupohjainen palvelu, joka on sovellustyypistä riippuen joko upotettu Shopifyn hallintapaneeliin tai on kokonaan itsenäinen sovellus. Sovellusta, joka on upotettu Shopifyn hallintapaneeliin, kutsutaan upotetuksi sovellukseksi. Mikäli sovellus on täysin itsenäinen, sitä kutsutaan nimellä Standalone App. Teknisesti upotettu sovellus upotetaan hallintapaneeliin iframen avulla, kun taas Standalone App avautuu kokonaan uuteen välilehteen sovelluksen käynnistyessä.

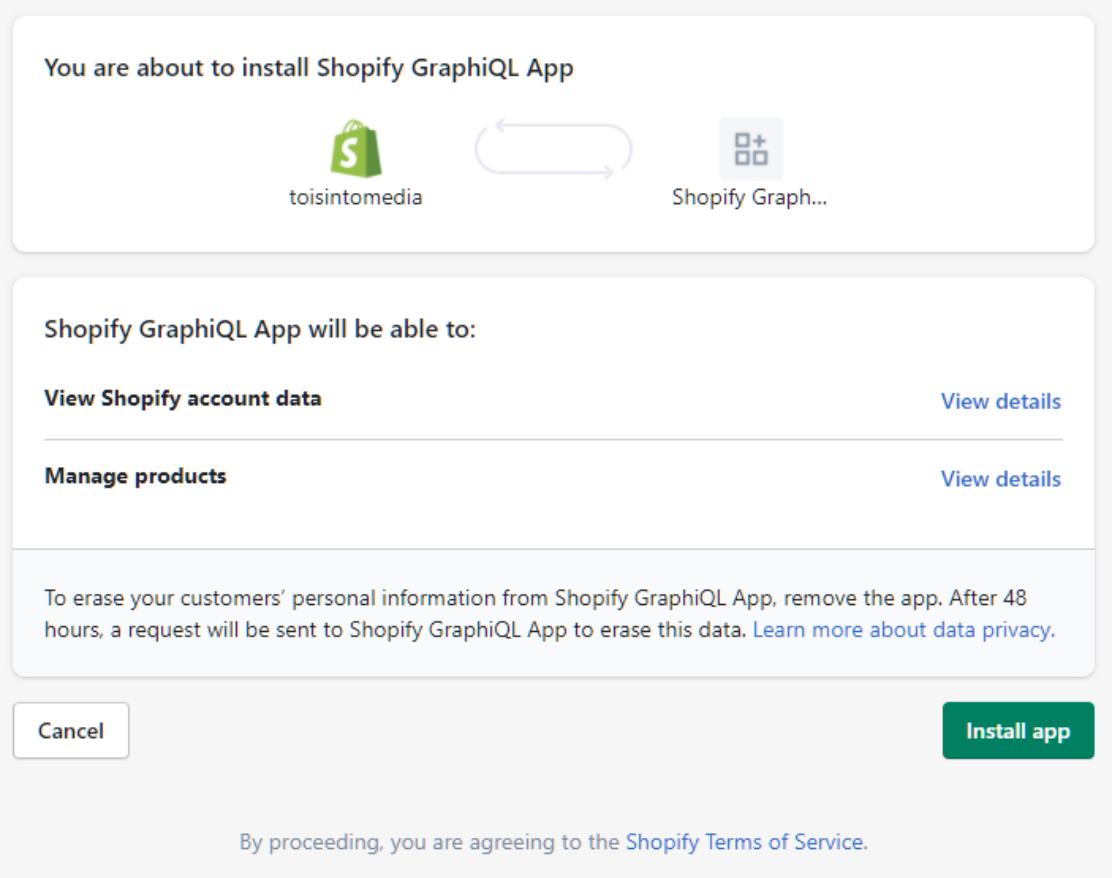
Sovellukset jaetaan kolmeen eri kategoriaan: yksityiset sovellukset (Private Apps), julkiset sovellukset (Public Apps) sekä räätälöidyt sovellukset (Custom Apps). Julkiset sovellukset jaetaan listattuihin ja listaamattomiin sovelluksiin. Listatut sovellukset ovat näkyvillä Shopifyn App Storessa hakutuloksissa ja eri sovelluskategorioissa. Listaamattomilla sovelluksilla on oma App Store -sivu, mutta ne eivät näyttäydy hakutuloksissa tai kategorialistauksissa. Molemmat sovellustyypit on mahdollista asentaa App Storesta suoraan verkkokauppaan. Sekä listaamattomien että listattujen sovellusten pitää läpäistä Shopifyn ”App Review Process”-hyväksymisprosessi, jotta sovellukselle saadaan luotua App Storeen oma sivu, jonka kautta sovellus voidaan asentaa verkkokauppaan. (App Types n.d.)

Yksityiset ja räätälöidyt sovellustyypit sopivat hyvin yksittäisen verkkokauppiaan kehitystarpeisiin vastaamiseen, kun taas julkiset sovellukset voidaan julkaista App Storessa suuremmalle yleisölle.

2.5.2 Sovellusten asennus verkkokauppaan

Sovellus voidaan asentaa Shopify-verkkokauppaan kahdella eri tavalla. Sovellus voidaan asentaa joko suoraan Shopifyn App Storesta sovelluksen omalta sivulta tai klikkaamalla sovelluksen kehittäjän tarjoamaa asennuslinkkiä. Tätä asennuslinkkiä voidaan käyttää esimerkiksi sovelluksen kehittäjän omilla verkkosivuilla, josta käyttäjä voidaan ohjata asentamaan sovellus omaan Shopify-verkkokauppaansa.

Sovellusta asennettaessa käyttäjältä tulee kysyä lupa käyttää verkkokaupan eri osien sisältämää tietoa, jotta sovellus saa tarvittavat käyttöoikeudet. Näistä käyttöoikeuksista puhutaan sanalla scope ja yleensä sovellusta asennettaessa sovellukselle annetaan pääsy yhteen tai useampaan scopeen. Mikäli sovellus haluaa hallita verkkokaupan tuotteita kauppiaan puolesta, niin sovelluksen täytyy pyytää käyttöoikeus tuotteiden lukemiseen ja kirjoittamiseen. Käyttöoikeuksien pyyntö suoritetaan Shopifyn hallintapaneelin sisällä (ks. kuvio 4), jonne käyttäjä ohjataan, kun sovellus halutaan asentaa.



Kuvio 4. Shopify GraphQL App -sovelluksen asennusruutu

3 Shopify-sovelluksen kehittäminen

3.1 Yleistä

Shopify-sovellusta kehittäessään kehittäjä voi käyttää haluamiaan teknologioita, sillä Shopify ei rajoita sitä millä ohjelmointikielellä sovellus voidaan toteuttaa. Ainut rajoite teknologioiden valinnassa on se, että sovelluksen tulee toimia verkkoselaimessa.

Shopifyllä on sovelluskehitystä varten tarjolla oma React-komponenttikirjasto Polaris, joka on tarkoitettu käytettäväksi React-sovellusten kehityksen yhteydessä. Kirjasto tarjoaa monia usein käytettyjä komponentteja, kuten tuotelistauksia ja oletusnäkymiä. Komponentit ovat valmiiksi tyylitelty Shopifyn hallintapaneelin tyyleihin sopivaksi. Yhtäläiset tyylit Shopifyn hallintapaneelin ja Shopify-sovellusten välillä helpottavat verkkokauppiasta sovelluksen käytössä, sillä kaikki elementit muistuttavat toisiaan ja ovat tutuilla paikoilla käyttöliittymässä. Tässä projektissa Polaris-komponenttikirjastoa ei kuitenkaan hyödynnetty, sillä sovelluksen tyylit haluttiin toteuttaa käyttäen TailwindCSS-kehystä.

3.2 Shopify Partners -hallintapaneeli

3.2.1 Sovelluksen luonti ja API-avaimet

Shopify-sovellukselle on saatavissa omat API-avaimet. API-avaimessa on julkinen sekä salainen osa, joista molemmat ovat muodoltaan pitkiä merkkijonoja. Avaimet saa joko luomalla Shopify partners -hallintapaneelissa uuden sovelluksen, tai hakemalla avaimet jo olemassa olevan sovelluksen asetuksista. Sovellus tarvitsee API-avaimiaan esimerkiksi silloin, kun verkkokauppias asentaa sovellusta omaan verkkokauppaansa

ja OAuth-prosessin aikana Shopifylle lähetetään pyyntö, jonka yhteydessä yksi vaadituista parametreista on sovelluksen API-avaimen julkinen osa.

3.2.2 Sovelluksen asetusten määrittäminen

Sovellusta varten Shopify Partners -hallintapaneelissa on useita eri asetuksia. Kehittäjän on mahdollista hallita sovellukseen liittyviä tietoja, kuten sovelluksen nimeä ja sen käyttämää ikonia, sovelluksen kehittäjän sähköpostiosoitetta, sovelluksen toimintaan tarvittavia URL-osoitteita ja muuta konfiguraatiota, kuten webhookkeihin liittyviä URL-osoitteita. Sovelluksella on oltava GDPR-asetuksen mukaisesti asetettuna vähintään kolme webhook-osoitetta: yksi asiakasdatan hakemista varten, yksi asiakasdatan poistamista varten, sekä yksi verkkokaupan datan poistamista varten.

Tyypillisimmät asetukset, joita hallintapaneelissa halutaan muokata, ovat sovelluksen käyttöön sallitut URL-osoitteet. Nämä osoitteet kertovat Shopifylle, mistä osoitteesta Shopify-sovellus voidaan tarjota iframe-kehykseen. Sovelluksella on kaksi URL-osoitetta, jotka täytyy asettaa: sovelluksen pääasiallinen osoite, josta sovellus tarjoillaan käyttäjälle sekä oAuth-prosessissa hyödynnettävä redirect-url. Shopify ohjaa käyttäjän sovelluksen asennuksen yhteydessä viimeiseksi redirect-urliin, jossa sovelluksen asentamisen viimeistely tapahtuu.

3.3 Käytetyt teknologiat ja kirjastot

3.3.1 Node.js

Node.js on runtime-ympäristö, joka on rakennettu Chromen V8 JavaScript-moottorin päälle (Nodejs n.d.). Node.js mahdollistaa JavaScript-koodin ajamisen palvelinympä-

ristössä. Tämä helpottaa ohjelmointityötä sillä kehittäjän ei tarvitse vaihtaa ohjelmointikielestä toiseen ohjelmoidessaan sovelluksen eri osia, vaan on mahdollista käyttää JavaScript-kieltä niin palvelimen päässä kuin itse selainsovelluksessakin.

Node.js on asynkroninen runtime-ympäristö, joten se soveltuu hyvin skaalautuvien web-sovellusten kehittämiseen, sillä se hallitsee hyvin useita samanaikaisia tehtäviä. (About Node.js n.d.)

Node.js valittiin sovelluksen backendin päätekniikaksi, sillä JavaScriptin käyttö sekä selaimessa että palvelimella koettiin tässä tapauksessa suurena etuna. Suuri määrä valmiita kehyksiä ja kirjastoja oli myös yksi syy, miksi Node.js valittiin käytettäväksi.

3.3.2 Express.js

Express.js on web-kehys Node.js-ympäristöön, joka tarjoaa yleisiä web-sovelluksen tarvitsemia ominaisuuksia, kuten HTTP-reititys, erilaisien HTTP-verbien käsittely reittien yhteydessä, view-moottoreiden integrointi osaksi sovellusta sekä reitteihin liitettävien middleware-funktioiden mahdollistaminen. (Introducing Express 2020)

Express.js valittiin sovellukseen sen keveyden, yksinkertaisuuden sekä selkeyden takia. Aikaisempi kokemus Express-kehyksen käyttämisestä web-projekteissa oli myös yksi valintaperusteista.

3.3.3 React.js

React.js (myöhemmin React) on Facebookin kehittämä JavaScript-kirjasto käyttöliittymien rakentamista varten. Reactin toiminta perustuu uudelleenkäytettäviin komponentteihin, jotka kirjoitetaan käyttäen JSX-syntaksia. (Introducing JSX n.d.)

3.3.4 Sequelize.js

Sequelize.js on kirjasto, jonka avulla sovellus kommunikoi MySQL-tietokannan kanssa. Sequelize.js:llä muodostetaan erilaisia kyselyitä, jotka hakevat tai kirjoittavat dataa tietokantaan.

3.3.5 Shopify App Bridge

Shopify App Bridge on JavaScript-kirjasto, joka mahdollistaa erilaisten shopify-toiminnallisuuksien hyödyntämisen helposti eri alustoissa, kuten Shopify POS -ympäristössä (Shopify App Bridge n.d.). Shopify App Bridge -kirjastoa hyödyntämällä sovelluksen on mahdollista hakea käyttöönsä oma session token, jolla sovelluksen tekemät verkkokutsut voidaan todentaa sovelluksen käyttämissä taustajärjestelmissä.

3.3.6 JWT

JSON Web Token on standardi, jota käytetään turvalliseen tiedonvälitykseen kahden eri osapuolen välillä (What is JSON Web Token n.d.). Tokenin sisältöön voidaan luottaa, sillä tieto salataan käyttäen salaista avainta. Mikäli salaista avainta ei ole tiedossa, niin tietoa tokenin sisällä ei voida muuttaa. Tokenin sisältämä tieto voidaan kuitenkin lukea ilman salaisen avaimen tietämistä.

3.3.7 MySQL

MySQL on tietokantajärjestelmä, jonka avulla relaatiotietokannassa olevaa tietoa voidaan hallita. Relaatiotietokanta sisältää tyypillisesti erilaisia tietomalleja, joiden välillä voi vallita erilaisia yhteyksiä. Sovelluksen käyttämä MySQL 8.0-tietokantainstanssi on hankittu DigitalOcean-palvelusta täysin ylläpidettynä versiona toiveena vähentää tietokantaan liittyvien tietoturvariskien määrää. Tietoturvaa parantaa ainakin se, että täysin ylläpidettyyn versioon asennetaan kaikki uusimmat päivitykset automaattisesti, ja siihen päästään käsiksi vain sallituista IP-osoitteista.

3.3.8 Tailwind CSS

Tailwind CSS on CSS-kehys, joka mahdollistaa CSS-tyylien kirjoittamisen yksinkertaisella ja ylläpidettävällä tavalla. TailwindCSS koostuu pienistä uudelleenkäytettävistä CSS-määrityksistä, joita yhdistämällä voidaan luoda monimutkaisia komponentteja ja käyttöliittymiä.

3.3.9 Git-versionhallinta

Koko sovelluksen kehittämisen ajan työskentelyssä hyödynnetään Git-versionhallintaa. Versionhallinnan avulla on mahdollista tallentaa sovelluksen ohjelmakoodi eri vaiheissa ja palata tiettyihin vaiheisiin tarvittaessa myöhemmin. Versionhallinta mahdollistaa sovelluksen eri ominaisuuksien kehittämisen erillään toisistaan, jolloin omaisuuksien valmistuttua eri ohjelmakoodit voidaan yhdistää sujuvasti takaisin yhteen.

4 Suunnitteluvaihe

4.1.1 Vaatimusmäärittely

Sovelluksen vaatimusmäärittely esitetään kuviossa 5. Kuvion alueissa on kuvattu otsikkona omaisuuden päätoiminto sekä sen alaisuuteen kuuluva varsinainen toiminnallisuus tarkemmin kuvattuna mahdollisine lisämäärityksineen.

Kuvio 5. Sovelluksen vaatimusmäärittely ominaisuustasolla

Sovelluksen vaatimukset voidaan jakaa karkeasti neljään eri osaan: sovelluksen asennus, datan synkronointi, tuotteiden valinta sekä operaatiot. Sovellukselle asetettiin vaatimukset siten, että niiden täytyttyä sovellusta on mahdollista käyttää verkkokaupan tuotteiden tuotetiedon hallintaan.

Sovelluksen asennus käsittää sovelluksen asentamisen mahdollistamisen verkkokauppiaalle. Tätä varten sovelluksella pitää olla tapa hoitaa käyttäjän asennuspyyntö.

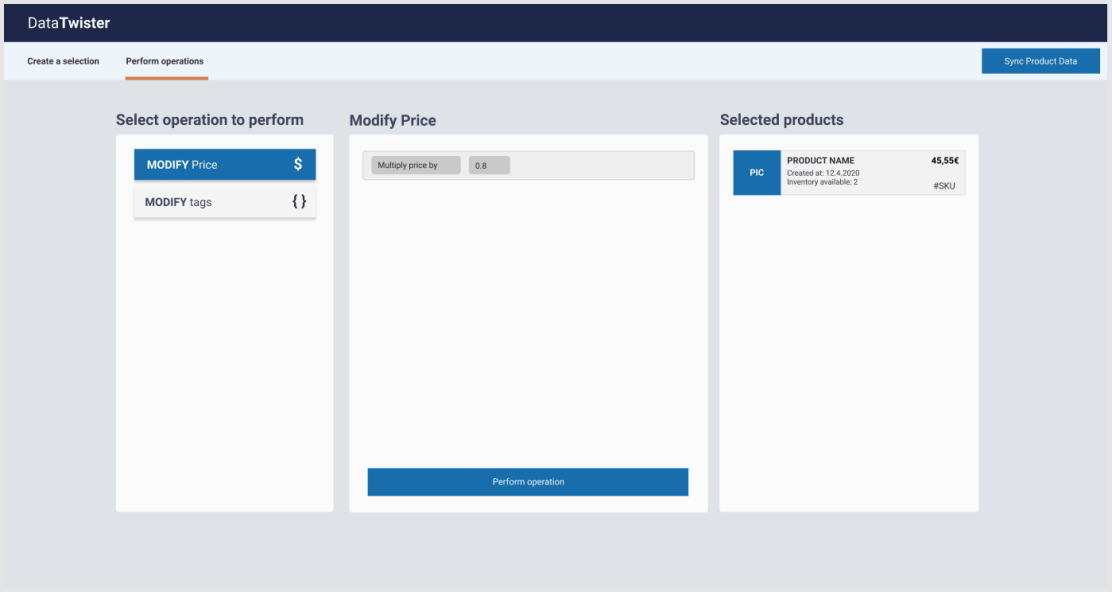
Datan synkronointi käsittää tuotteiden tietojen siirtämisen verkkokauppiaan Shopify-kaupasta sovelluksen omaan tietokantaan. Sovelluksen hakutoimintojen mahdollistamiseksi sovelluksen omaan tietokantaan tulee siirtää verkkokaupan tuotteet, tuotteiden variantit, sekä metatietokentät, jotka liittyvät joko tuotteisiin tai tuotteiden variantteihin.

Operaatioilla tarkoitetaan tuotevalinnan jälkeistä prosessia sovelluksessa. Esimerkki operaatiosta voi olla esimerkiksi tuotteen tunnisteiden muokkaaminen. Operaatiot

suoritetaan aina yhteen tai useampaan tuotteeseen kerralla ja operaatioiden aiheuttamat muutokset tallennetaan sekä verkkokauppiaan Shopify-verkkokauppaan että sovelluksen tietokantaan.

4.1.2 Graafisen käyttöliittymän suunnittelu

Graafinen käyttöliittymä (ks. kuvio 6) suunniteltiin työkalulla nimeltään Figma. Figmalla voidaan luoda helposti erilaisia prototyyppejä ja käyttöliittymähahmotelmia suoraan verkkoselaimessa. Sovelluksesta pyrittiin suunnittelemaan käyttökokemukseltaan mahdollisimman yksinkertainen ja visuaalisesti miellyttävä.



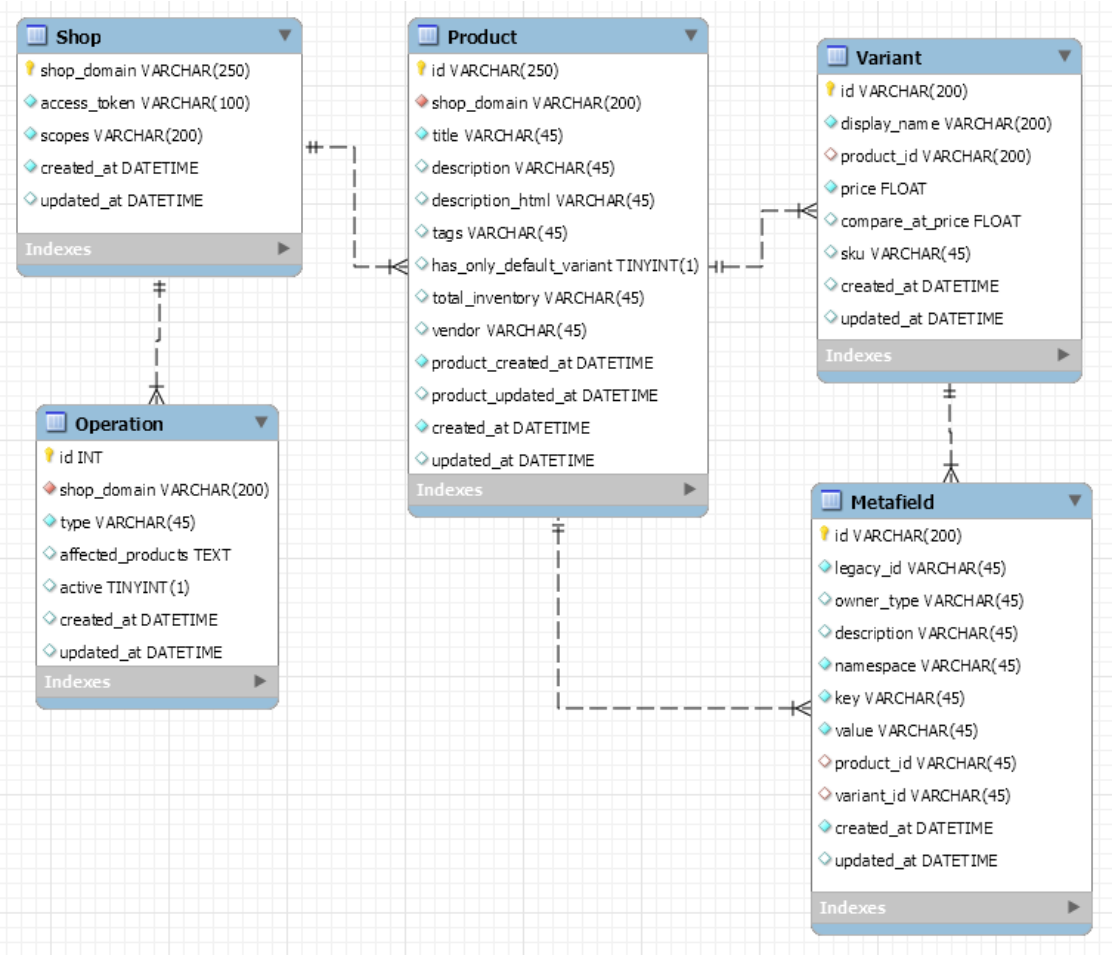
Kuvio 6. Sovelluksen graafinen käyttöliittymä suunniteltuna Figmassa

4.1.3 Tietokantasuunnittelu

Tietokannaksi valittiin relaatiotietokanta, sillä eri tietokantaan tallennettujen tietojen väliset yhteydet olivat sovelluksen toiminnan kannalta oleellisia. Erilaisista relaatiotietokannoista vertailtiin PostgreSQL- sekä MySQL-tietokantoja, joista sovelluksen käyttöön valikoitui MySQL 8.0. Molemmat vertailluista tietokannoista olisivat varmasti vastanneet sovelluksen tarpeisiin hyvin, mutta MySQL otettiin käyttöön aikaisemman kokemuksen perusteella.

Tietokannan suunnittelussa tietorakenteen pohjana hyödynnettiin Shopifyn rajapinnasta saatavan datan muotoa. Sovelluksen käyttötarkoitus huomioiden tietokannan tauluihin valittiin tarpeelliset tiedot, joita sovellus toiminnassaan hyödyntää.

Tietokantakuvauksessa (ks. kuvio 7) kuvataan tietokannan taulujen sisältämät tietokentät sekä taulujen väliset yhteydet. Taulujen välillä vallitsee yksi-moneen suhde, esimerkiksi tuotteella voi olla useita variantteja, mutta varianttia kohden voi olla vain yksi tuote. Sama sääntö pätee jokaiseen kuviossa esitettyyn yhteyteen. Tällä tavalla tietokannasta on helppo hakea esimerkiksi kaikki tuotteet sekä niille kuuluvat variantit ja metatietokentät.

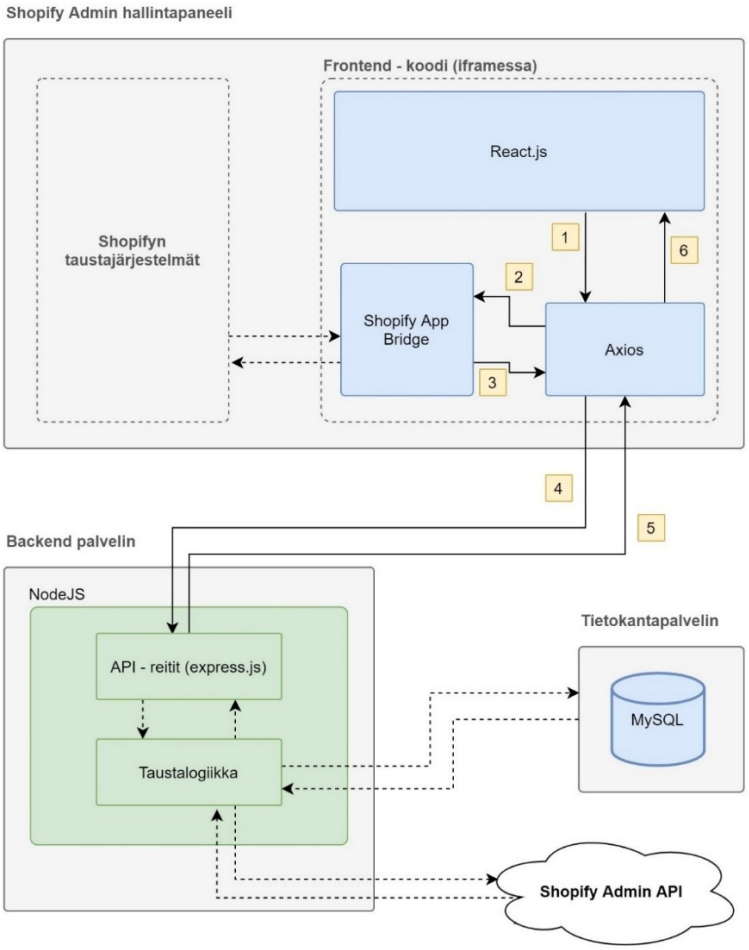


Kuvio 7. Tietokannan rakenne MySQL workbench 8.0 CE -työkalulla suunniteltuna

5 Tekninen toteutus

5.1 Arkkitehtuurikuvaus

Sovelluksen arkkitehtuurikuvauksessa (ks. kuvio 8) kuvataan järjestelmän eri osat ja niiden väliset yhteydet. Kuvauksessa havainnollistetaan yksittäisen verkkokutsun kulkema reitti keltaisten merkintöjen avulla.



Kuvio 8. Sovelluksen arkkitehtuurikuvaus

Frontendissä React käyttää Axios-kirjastoa HTTP-kutsujen lähettämiseen. Axios hakee jokaisen verkkokutsun yhteydessä Shopifyn App Bridgen kautta Session Tokenin Shopifyltä, joka liitetään kutsun Authorization-headeriin. Tokeniin on asetettu vanhenemisaika, joka on Shopifyllä oletuksena yksi minuutti siitä lähtien kun token on luotu. Tällä voidaan taata se, että vaikka session token joutuisi vääriin käsiin, niin sitä ei voitaisi käyttää kovin kauaa ennen kuin se vanhenee ja ei ole enää kelvollinen.

Kun session token on asetettu headeriin, Axios lähettää kutsun sovelluksen backendiin. Backendissä on erilaisia API-reittejä, jotka vastaanottavat kutsuja. Reititys on tehty hyödyntäen Express.js-kirjastoa. Kutsun saapuessa reittiin kutsussa oleva token varmistetaan käyttäen uudelleenkäytettävää middleware-funktiota ja riippuen siitä onko token validi, päästetään käyttäjän kutsu eteenpäin tai hylätään se. Token voidaan varmistaa sillä Shopifyn taustajärjestelmä allekirjoittaa tokenin salaisella avaimella, joka on vain Shopifyn ja sovelluksen backendin tiedossa. Reittiin voidaan asettaa useita middleware-funktioita peräkkäin, joten erilaisten tarkistusten suorittaminen reitityksen yhteydessä on melko suoraviivaista ja helposti organisoitavissa eri käyttötarkoitusten mukaan.

Kutsun edettyä jokaisen reitin middlewaren läpi, se ohjataan sovelluksen taustalogiikalle hoidettavaksi. Taustalogiikan valmistuttua tiedot palautetaan takaisin reititykseen, ja sieltä takaisin Axiosiin res-objektin send-funktion avulla. Axios palauttaa tiedot vastaavasti React-sovellukselle, joka hyödyntää vastaanotetun datan asianmukaisella tavalla, esimerkiksi esittämällä datan käyttöliittymässä käyttäjälle.

5.2 Frontend-kehitys

5.2.1 Yleistä

Frontend alustettiin käyttäen create-react-app nimistä työkalua. Create-react-app on Facebookin kehittämä ja ylläpitämä npm-paketti, joka mahdollistaa React-sovelluksen nopean luomisen ja yleisimpien kehitysprosessissa käytettävien työkalujen, kuten webpackin, automaattisen konfiguraation ja käyttöönoton. Tällä tavalla työhön käytettävissä olevat resurssit voidaan ohjata tehokkaasti sovelluksen kehittämiseen, testiympäristön pystyttämisen sijasta.

Kehitysvaiheessa sovelluksen frontendin esittäminen Shopifyn hallintapaneelissa mahdollistetaan ngrok-työkalun avulla. Käynnistettäessä ngrok luo verkko-osoitteen, jonne tuleva liikenne ohjataan isäntäkoneen paikallisen kehitysympäristön määrättyyn porttiin. Tällä tavalla kehitysympäristö voidaan paljastaa ulkoverkkoon julkisen HTTPS-protokollaa käyttävän verkko-osoitteen taakse. Ngrok tarvitaan kehitysympäristöön, sillä Shopifyn hallintapaneeliin voidaan upottaa sovellus vain, jos se käyttää suojattua HTTPS-yhteyttä.

5.2.2 React Hooks ja Context

Reactin versioissa 16.8 ja ylöspäin on mahdollista hyödyntää uutta hooks-omainaisuutta. React hooks mahdollistaa tavallisesti vain luokkapohjaiselle komponentille saatavilla olevan ”state”-toiminnallisuuden käyttämisen funktionaalisessa komponentissa. (Introducing Hooks n.d.) Ero luokkapohjaisen ja funktionaalisen komponentin välillä on se, että luokkapohjainen komponentti luodaan käyttämällä class-avainsanaa ja se pohjautuu Reactin Component-luokkaan, kun taas funktionaalinen komponentti on vain funktio, joka palauttaa JSX-koodia. (Functional Components vs Class Components in React, 16.12.2019)

React Context helpottaa tiedon välitystä React-sovelluksen sisällä ja se on suunniteltu jakamaan koko sovellukselle yleisesti hyödyllistä tietoa sovelluksen eri komponenteille. (When to Use Context n.d.).

Kontekstia hyödynnetään työssä jakamaan tietoa sovelluksen eri osille verkkokaupan tuotteista, sekä tarjoamaan funktioita sovelluksen käyttöön, joilla uusia tuotelistauksia voidaan hakea käytännöllisesti mistä tahansa komponentista sovelluksen sisällä.

5.2.3 Shopify App Bridgen liittäminen sovellukseen

Shopifyn App Bridge -kirjasto liitetään Reactiin, jotta sillä voidaan hakea Shopifyn järjestelmästä Session Token, kun sovellus on avattuna hallintapaneelissa. Jotta App Bridge voi toimia oikein, frontend sovelluksen tulee olla avattuna Shopifyn hallintapaneelissa iframe-kehyksen sisällä. Shopify App bridgen tarjoamaa session tokenia hyödynnetään sovelluksen autentikaatiossa jokaisen verkkokutsun yhteydessä backendiin.

5.2.4 Autentikaatiotokenin liittäminen verkkokutsuihin

Sovelluksen verkkokutsut tehdään Axios-nimisen JavaScript-kirjaston avulla. Axios mahdollistaa useiden axios-instanssien luomisen create-funktiota käyttämällä. Create-funktiolla voidaan luoda kätevästi erilaisia instansseja, joihin on määritetty ennalta erilaisia asetuksia, kuten kutsuun liitettävät header-tiedot.

Kuviossa 9 luodaan AuthAxios-niminen instanssi, johon liitetään jokaisen verkkokutsun lähtiessä ajettava funktio. Funktio ottaa vastaan config-objektin, joka sisältää tietoja lähtevästä kutsusta. Funktio suoritetaan ennen kuin verkkokutsu lähtee matkaan, joten sen avulla on mahdollista lisätä tietoa lähtevään verkkokutsuun ilman, että tietoa tarvitsee lisätä joka kerta manuaalisesti. Tässä tapauksessa kutsuun liitetään mukaan Shopifyn App Bridgeltä saatava session token. Jatkossa kun sovelluksesta halutaan tehdä verkkokutsuja, voidaan käyttää AuthAxios-objektia, jonka ansiosta session token liitetään automaattisesti jokaisen lähtevän kutsun header-osioon.

```
const AuthAxios = axios.create({ baseURL });

AuthAxios.interceptors.request.use(function (config) {
  return getSessionToken(window.app) // window.app is instance of App Bridge
    .then((token) => {
      config.headers["Authorization"] = `Bearer ${token}`;
      return config;
    });
});

export default AuthAxios;
```

Kuvio 9. Authorization-headerin asettaminen AuthAxios-instanssissa

5.2.5 Datan hakeminen palvelimelta

Frontend-sovellus käyttää tavanomaisia HTTP-kutsuja datan hakemiseen palvelimelta. Verkkokutsut tehdään käyttäen Axios-nimistä JavaScript kirjastoa ja sillä luotua AuthAxios-objektia. AuthAxios liittää jokaiseen verkkokutsuun automaattisesti Authorization-headeriin session tokenin, jolla sovellus tunnistautuu palvelimella.

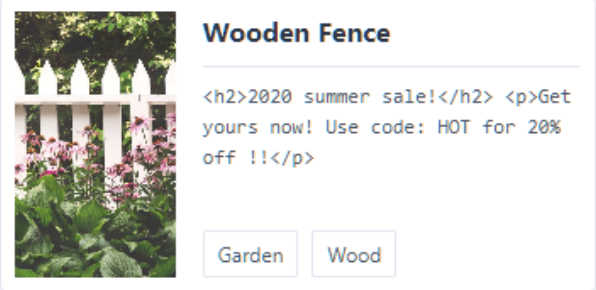
Verkkokutsujen lähetys tapahtuu Reactin eri komponenteissa, joista ne käynnistetään joko automaattisesti komponentin luomisen tai päivittymisen yhteydessä tai manuaalisesti, kun käyttäjä painaa esimerkiksi jotain tiettyä painiketta. Esimerkkifunktiossa (ks. kuvio 10) haetaan palvelimelta kaikki sovellukseen tallennetut tuotteet, jotka kuuluvat verkkokauppiaalle.

```
const getProducts = async () => {
  let products;
  try {
    products = await authAxios.get("/api/allproducts");
    console.log(products.data);
  } catch (error) {
    console.log(error); // TODO: show user some error
  }
  setProducts(products.data);
};
```

Kuvio 10. getProducts-funktio, joka hakee palvelimelta kaikki tietokantaan tallennetut tuotteet

5.2.6 Datan esittäminen käyttöliittymässä

Sovelluksen tieto esitetään käyttöliittymässä React-komponenttien avulla. Esitettävät tiedot sijaitsevat Reactin konteksteissa, jotka on luotu loogisesti eri tietoja varten. Esimerkiksi tiedot esitettävistä tuotteista sijaitsevat ProductContext-nimisessä kontekstissa. Kuviossa 11 näkyy sovelluksen käyttöliittymässä käytetty tuotekortti-komponetti, jonka esittämä data sijaitsee ProductContext-kontekstissa. Tiedot, joita tuotekorttikomponentti tarvitsee kontekstista ovat tuotteen nimi, tuotekuvaus, tunnisteet sekä tuotekuvan URL-osoite.



Kuvio 11. ProductContext-kontekstin tiedot esitettynä tuotekorttikomponentissa

React-komponentti käyttää haluttua kontekstia useContext-funktion avulla, jolle käytettävä konteksti annetaan argumenttina. React päivittää tuotelistauksen käyttöliittymässä automaattisesti, kun tieto käytettävän kontekstin sisällä muuttuu. Mikäli kontekstissa ei ole yhtään tuotetta, eli se on tyhjä, sovellus esittää tekstin “No products selected” (ks. kuvio 12). React heijastaa sovelluksen datan tilan käyttöliittymään ilman manuaalista työtä. Kun tieto kontekstissa muuttuu, React huolehtii sen muuttamisesta käyttöliittymään.

```
const SelectedProducts = () => {
  const { products } = useContext(ProductContext);
  const productList = products.map((p, inx) => {
    return <ProductCard data={p} key={inx} />;
  });

  return (
    <Container title="Selected Products">
      {products.length > 0 ? productList : "No products selected"}
    </Container>
  );
};
```

Kuvio 12. Tuotteiden esittäminen käyttöliittymässä ProductContext-kontekstin avulla

5.3 Backend-kehitys

5.3.1 Palvelimen vaatimukset

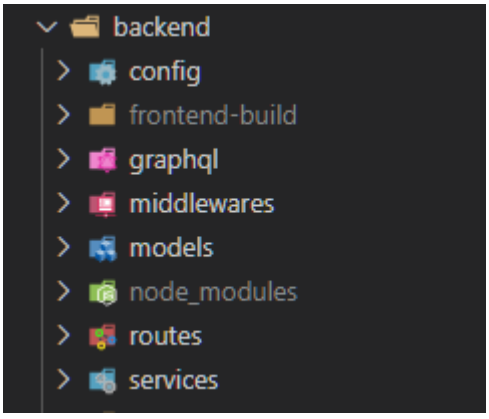
Sovellusta varten olevalle palvelimelle ei ole asetettu erityisen suuria vaatimuksia. Palvelimella tulee olla asennettuna Node.js, ja sille sovelluksen käyttöön suositeltu versio on 14 “Fermium” tai ylempi. Palvelimella tulee olla tallennustilaa sovelluksen tiedostoja ja mahdollista tietokantaa varten. Tässä tapauksessa sovelluksen tietokanta on hankittu ulkoisesta palvelusta tietoturvan ja käyttömukavuuden vuoksi.

Sovelluksen käytössä oleva Node.js 14 on LTS-versio, mikä tarkoittaa sitä, että kriittiset bugit korjataan 30 kuukauden ajan LTS-version julkaisusta. Tuotannossa olevien sovellusten suositellaan käyttävän vain LTS-versioita. (Releases n.d.)

5.3.2 Node.js-projektin alustus

Sovelluksen alustaminen tehdään luomalla ensimmäinen tiedosto, josta sovellus käynnistetään. Sovellusta varten luotiin server.js-niminen tiedosto, joka toimii lähtöpisteenä kaikkeen muuhun sovelluksen logiikkaan. Server.js-tiedosto sijaitsee backend-sovelluksen kansiorakenteen juuressa.

Sovelluksen eri osat jaotellaan loogisesti eri kansioihin, jotta sovelluksen kehitys ja ylläpito pysyy hallittavana. Mikäli kaikki sovelluksen koodi kirjoitetaan yhteen tiedostoon, sovelluksen koodista tulee nopeasti hyvin vaikeaa hallita ja ylläpitää. Sovelluksen lopulliseen kansiorakenteeseen päädyttiin pienen tutkimuksen sekä yrityksen ja erehdyksen kautta. Kuviossa 13 esitellään karkeasti sovelluksen backendin käyttämä kansiorakenne. Seuraavaksi käydään läpi jokaisen kansion tarkoitus ja sisältö, jotta voidaan ymmärtää paremmin sovelluksen toimintaa.



Kuvio 13. Backend-sovelluksen kansiorakenne

Config-kansio sisältää tietokantoihin ja datan hakuun liittyvää konfiguraatiota. Graphql-kansiossa on jaoteltuna mutaatiot sekä queryt Shopify Admin -rajapintaa varten. Middlewares-kansio sisältää Express.js-reittien käyttämät middleware-funktiot. Models-kansiossa on säilöttynä tietokantamallit, joita Sequelize.js käyttää toiminnassaan. Routes-kansiossa on Express.js-kirjaston käyttämät reitit, joiden erittely omiksi tiedostoikseen on pidemmän päälle järkevää. Services-kansio sisältää erilaisia funktioita, jotka sitovat Express.js-kirjaston reitityksen sovelluksen käyttämien tietokantayhteyksien kanssa yhteen. Backend-kansion juuressa sijaitsee myös niin sanottu env-tiedosto, josta voidaan lukea sovelluksen käyttöön erilaisia muuttujia, joita ei haluta sisällyttää itse ohjelmakoodiin, eikä sitä myöten versionhallintaan. Tällaisia muuttujia ovat esimerkiksi sovelluksen käyttämät API-avaimet niiden arkaluontoisuuden takia.

5.3.3 API-reittien määritys Express.js:n avulla

Sovelluksen tarjoamat API-reitit määritellään ensimmäisen kerran server.js-tiedostossa. Kuviossa 14 määritellään sovelluksen käyttöön erilaisia reittejä. App-objekti on tässä tapauksessa Express.js-kirjaston avulla luotu express-instanssi, jolle liitetään use-funktion avulla erilaisia toimintoja. Use-funktiolle annetaan argumenttina ensin reitti, josta halutaan vastaanottaa käyttäjän HTTP-kutsuja, sitten valinnaiset middleware-funktiot ja lopuksi itse funktio, joka käsittelee vastaanotetun kutsun.

```
// basic authentication and installation routes
app.use("/", authRoutes);

// API ROUTES
app.use("/api", [_test_addUserDetailsToReq, _addServices], apiRoutes);
```

Kuvio 14. Server.js-tiedostossa määritellyt reitit

Tässä tapauksessa authRoutes ja apiRoutes ovat Express.js-kirjaston Router-funktion avulla luotuja objekteja, jotka jatkokäsittelevät kutsun omalla logiikallaan eri tiedostoissa. Esimerkiksi kutsu, joka lähetetään osoitteeseen ”/api/sync”, käsitellään apiRoutes-reitittimen toimesta (ks. kuvio 15), sillä saapuvan verkkokutsun alkuosassa on ”/api”.

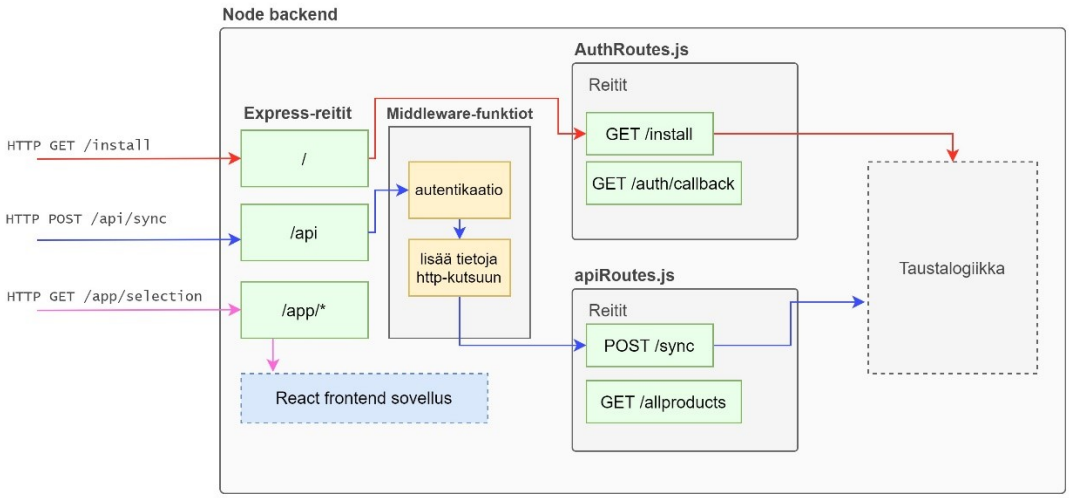
```
/**
 * This route is hit when user wants to sync his product shopify-product data with the DB
 */
apiRoutes.post("/sync", async (req, res) => {
  const { ProductServiceInstance, OperationServiceInstance } = req.services;

  // Create new bulk operation
  OperationServiceInstance.createOperation("bulk");

  try {
    await ProductServiceInstance.createBulkDataOperation();
    ProductServiceInstance.pollBulkStatus(); // start polling the status in the background
    return res.status(200).send();
  } catch (error) {
    console.log(error);
    return res.status(500).send("Something wrong when starting the sync");
  }
});
```

Kuvio 15. AuthRoutes-reitittimeen saapuvien HTTP-kutsujen jatkokäsittely

Kuviossa 16 on esitetty sovelluksen reititys korkealla tasolla. Kuviossa esitetään eri kutsujen kulkema polku riippuen kutsun polusta. Punaisella värillä oleva nuoli kuvaa kutsua osoitteeseen ”/install”. Kuviosta huomataan, että kutsun ei ole pakko kulkea middleware-funktioiden läpi, mikäli niin ei haluta. Punaisen nuolen kulkemalle reitille ei ole asetettu middleware-funktiota, joten se siirtyy suoraan AuthRoutes.js-tiedostossa sijaitsevan reitittimen hoidettavaksi.

Sininen nuoli kuvaa kutsua, joka kulkee kahden erillisen middleware-funktion läpi. Tässä tapauksessa kutsu läpäisee molemmat funktiot ja päätyy jatkokäsittelyyn apiRoutes.js-tiedostossa määritellylle reitittimelle. Middleware-funktiossa on mahdollisuus keskeyttää kutsun eteneminen. Esimerkiksi tapauksessa, jossa huomataan, että käyttäjä ei ole autentikoinut itseään asiallisesti. Tässä tapauksessa kutsun eteneminen voidaan katkaista, joten kutsu ei päädy apiRoutes.js-tiedostossa määritellylle reitittimelle asti. Tällä tavalla voidaan helposti suojata iso määrä erilaisia reittejä, ilman että autentikaation todennukseen käytettävää logiikkaa tarvitsee monistaa useisiin paikkoihin. Nyt riittää, että jokainen kutsu, joka kulkee ”/api”-reittiin, varmistetaan yhdellä funktiolla jo ennen kuin se päästetään jatkoon.

Vaaleanpunainen nuoli kuvaa kutsua, jonka on tarkoitus hakea käyttöliittymää varten React-sovellus. Kaikki reitit, jotka alkavat ”/app” ohjataan suoraan palauttamaan staattinen frontend-sovellus sille määritellystä paikasta.



Kuvio 16. Sovelluksen reitityksen kokonaiskuva

5.3.4 JSON web tokenin todennus middleware-funktion avulla

Sovelluksen tiettyihin reitteihin saapuvat kutsut todennetaan tarkoitusta varten luodun middleware-funktion avulla. Middleware-funktio todentaa verkkokutsun mukana tulleen tokenin oikeellisuuden ja mikäli token on validi, päästää kutsun eteenpäin. Varmennettava token sijaitsee saapuneen kutsun otsakkeen authorization-osiossa, josta se poimitaan todennusta varten talteen (ks. kuvio 17).

Token on allekirjoitettu käyttäen avainta, joka on vain Shopifyn sekä sovelluksen backendin tiedossa, joten se voidaan purkaa käyttäen jsonwebtoken-kirjaston sisältämää verify-funktiota. Purettu token sisältää tietoja tokenin haltijasta, esimerkiksi verkkokauppiaan kaupan domain-nimen, jota hyödynnetään myöhempänä sovelluksen toiminnassa. Puretun tokenin sisältämään tietoon voidaan luottaa, sillä tokenin sisältämiä tietoja ei ole mahdollista muokata tietämättä salaista avainta. Tokenin purkaminen epäonnistuu, mikäli tokenin sisältämää tietoa on yritetty muokata sen luomisen jälkeen.

```
/**
 * Validates JWT bearer token in the authorization header
 * @param {*} req
 * @param {*} res
 * @param {*} next
 */
const validateJWT = (req, res, next) => {
  try {
    const token = req.headers.authorization.split(" ")[1];
    var decoded = jwt.verify(token, process.env.SHOPIFY_API_SECRET_KEY);
  } catch (err) {
    // err
    console.log(err);
    return res.sendStatus(401);
  }
  console.log("JWT SUCCESSFULLY DECODED:", decoded);

  if (decoded) next();
};
```

Kuvio 17. JWT-tokenin varmennukseen käytetty middleware-funktio

5.3.5 Sovelluksen asennus ja OAuth

Sovelluksen asennus alkaa, kun käyttäjä navigoi tarkoitusta varten sovelluksen backendissä määritettyyn osoitteeseen. Osoitteeseen tulee määrittää URL-parametriksi sen Shopify-kaupan osoite, jonne sovellus halutaan asentaa. URL-parametrit asetetaan osoitteeseen kysymysmerkin jälkeen nimi-arvo pareina. Esimerkki osoitteesta, jonne navigoimalla käyttäjä voi asentaa sovelluksen omaan kauppaansa: https://{sovelluksen_domain}/install?shop={shopify_kaupan_domain}

Kutsun saapuessa palvelimelle siitä tarkistetaan ensin shop-nimisen URL-parametrin olemassaolo ja mikäli parametri on olemassa, niin silloin käyttäjä ohjataan Shopifyn palveluun varmistamaan asennuspyyntö. Uudelleenohjauksen yhteydessä käyttäjän selaimeen tallennetaan nonce-eväste, jonka tarkoituksena on myöhemmin varmistaa, että asennuspyyntö on validi. Uudelleenohjausta varten käyttäjälle pitää muodostaa palvelimella oma asennusosoite, jonne käyttäjä uudelleenohjataan. Asennusosoitteeseen tulee määrittää URL-parametreiksi client_id, scopes, state, sekä redirect_uri. Client_id on käytännössä sovelluksen api-avain, jonka Shopify muodostaa sovellusta varten, kun sovellus luodaan Shopify Partners -hallintapaneelissa. Scopes kuvaa käyttöoikeuksia, joita sovellus pyytää käyttäjältä asennuksen yhteydessä, joita ovat esimerkiksi tuotteiden luku ja kirjoitusoikeus. State on aiemmin evästeeseen tallennettu nonce, jolla varmistetaan myöhemmin, että asennuspyyntö on validi. Redirect_uri on osoite, jonne käyttäjä ohjataan sen jälkeen, kun asennuspyyntö on hyväksytty käyttäjän toimesta Shopify-hallintapaneelissa.

Käyttäjän hyväksyttyä asennuspyynnön Shopify ohjaa käyttäjän aiemmin määritettyyn redirect_uriin. Uudelleenohjauksen mukana Shopify on asettanut URL-parametreja, jotka sovelluksen täytyy tarkastaa, jotta sovelluksen asennus voidaan

viimeistellä. Parametrien määrä voi vaihdella, mutta tärkeimpinä niistä ovat hmac, state, code sekä shop.

Sovelluksen tulee tehdä kolme tarkistusta, jotta sovelluksen asennus voi jatkua: Ensin tulee tarkastaa, että URL-parametrista saatu state vastaa käyttäjän evästeessä olevaa noncea. Toiseksi tulee varmistaa, että kutsun URL-parametrina oleva HMAC-sanoma on validi. Kolmanneksi sovelluksen tulee tarkastaa, että hostname-parametri on validi, päättyy merkkijonoon ”.myshopify.com” ja sisältää vain seuraavia sallittuja merkkejä: akkoset a-z, numerot 0-9, pilkut, väliviivat. (Authenticate with OAuth n.d.)

HMAC-sanoman tarkastus tapahtuu sitä varten olevalla funktiolla (ks. kuvio 18). Funktio ottaa kaikki URL-parametrit ja asettaa ne objektiin, jonka jälkeen objektista tehdään merkkijono. Tämän jälkeen tätä merkkijonoa käytetään crypto-kirjaston avulla muodostamaan uusi HMAC-sanoma, jota varten tarvitaan salainen API-avain, joka on vain Shopifyn sekä sovelluksen backendin tiedossa. Tästä prosessista syntyvää merkkijonoa verrataan URL-parametrissa olevaan HMAC-sanomaan ja mikäli ne vastaavat toisiaan, pyyntö voidaan todeta validiksi.

```
const map = Object.assign({}, req.query);
delete map["hmac"];
const message = querystring.stringify(map); // removed hmac from message so we can digest it

const generatedHash = crypto
  .createHmac("sha256", process.env.SHOPIFY_API_SECRET_KEY)
  .update(message)
  .digest("hex");
```

Kuvio 18. Osa hmac-parametrin tarkastusfunktiosta

Mikäli kaikki tarkastukset läpäistään, niin sovelluksen asennus voidaan viimeistellä hakemalla sovelluksen asennukselle oma API-avain Shopifyn rajapinnasta.

API-avaimen hakua varten sovellus lähettää HTTP-kutsun osoitteeseen https://{kaupan_domain}/admin/oauth/access_token ja kutsun mukana lähetetään client_id, client_secret sekä code, jonka saatiin edellisessä vaiheessa redirect_urlin parametreista. Vastauksena lähetettyyn kutsuun Shopify lähettää sovelluksen asennukselle oman API-avaimen, jonka avulla sovellus voi jatkossa hakea ja muokata tietoja verkkokauppiaan puolesta. Vastaanotettu API-avain on yksilöllinen jokaiselle asennuskerralle, joten se ei enää muutu sen jälkeen, kun sovellus on kerran asennettu. Tämän ansiosta avain on mahdollista tallentaa tietokantaan ja hakea se sieltä aina kun sitä tarvitaan API-kutsujen tekemistä varten.

Mikäli sovellus poistetaan verkkokaupasta ja asennetaan uudelleen, niin API-avain on eri kuin edellisella asennuskerralla, joten sitä ei voida pitää muuttumattomana eri asennusten välissä. Mikäli sovellus vaatii syystä tai toisesta lisäoikeuksia sen jälkeen, kun sovellus on ensimmäisen kerran asennettu, niin sen täytyy käydä edellä kuvattu OAuth-prosessi uudestaan läpi, jotta se saa haltuunsa uusia oikeuksia vastaavan API-avaimen.

5.3.6 Sequelize.js:n hyödyntäminen tietokantaoperaatioissa

Tietokantaoperaatioissa sovellus käyttää Sequelize.js-nimistä kirjastoa avukseen. Sequelizella on mahdollista luoda tietokannasta eräänlainen malli suoraan sovelluksen ohjelmakoodiin, jonka läpi tietokantaa voidaan käyttää vaivatta ja melko monipuolisesti. Sequelize.js mahdollistaa tietokantaan taulujen luomisen, tietojen haun, päivittämisen ja poistamisen ilman, että ohjelmakoodissa tarvitsee kirjoittaa varsinaisesti yhtään omia SQL-komentoja.

Mallit tietokannan eri tauluista luodaan omissa tiedostoissaan, jotka sijaitsevat backend-sovelluksen models-kansiossa. Mallille määritetään nimi, sekä kentät, jotka

liittyvät kyseiseen malliin, niiden datatyypit, sekä mahdolliset lisämääritykset (ks. kuvio 19). Kun malli on luotu, sitä voidaan alkaa käyttämään sovelluksen toimesta ja eri mallien välille voidaan alkaa muodostamaan yhteyksiä.

```
const db = require("./../config/db");
const { DataTypes } = require("sequelize");

module.exports = db.define("Shop", {
  shop_domain: {
    type: DataTypes.STRING,
    primaryKey: true,
    unique: true,
  },
  access_token: {
    type: DataTypes.STRING,
  },
  scopes: {
    type: DataTypes.STRING,
  },
  pending_operations: {
    type: DataTypes.STRING,
  },
});
```

Kuvio 19. Sequelize.js-malli shop-taulusta

Mallien väliset yhteydet voidaan muodostaa, kun kaikki tarvittavat mallit ovat määritettynä ja ne on tuotu samaan tiedostoon. Mallien välille on mahdollista luoda yksi-yhteen, yksi-moneen tai moni-moneen suhteita (Associations n.d.). Tässä tapauksessa sovellus käyttää yksi-moneen suhteita jokaisen taulun välillä (ks. kuvio 20). Suhteiden muodostamisen yhteydessä määritetään usein foreign key, jolla eri taulut linkittyvät toisiinsa. Mikäli tätä määritystä ei tehdä, sequelize.js muodostaa nämä yhteydet automaattisesti taulujen nimien perusteella mallien synkronoinnin yhteydessä. Selkeyden vuoksi määritys tehtiin kuitenkin itse ja foreign-key kolumnit nimettiin manuaalisesti.

```
const Shop = require("./Shop");
const Product = require("./Product");
const Variant = require("./Variant");
const Metafield = require("./Metafield");
const Operation = require("./Operation");

// establish relationships between models
Shop.hasMany(Product, { foreignKey: "shop_domain" });
Product.belongsTo(Shop, { foreignKey: "shop_domain" });

Shop.hasMany(Operation, { foreignKey: "shop_domain" });
Operation.belongsTo(Shop, { foreignKey: "shop_domain" });

Product.hasMany(Variant, { foreignKey: "product_id" });
Variant.belongsTo(Product, { foreignKey: "product_id" });

// metafield associations
Product.hasMany(Metafield, { foreignKey: "product_id" });
Metafield.belongsTo(Product, { foreignKey: "product_id" });

Variant.hasMany(Metafield, { foreignKey: "variant_id" });
Metafield.belongsTo(Variant, { foreignKey: "variant_id" });
```

Kuvio 20. Sequelize.js-mallien väliset yhteydet

Kun määritykset on tehty onnistuneesti, voidaan tietoa hakea sequelize.js-kirjaston avulla siten, että voidaan hakea esimerkiksi tuote ja kaikki siihen liittyvät variantit kerralla. Tämä on mahdollista, sillä tuotteen ja variantin välillä pätee yksi-moneen suhde. Yksittäisen variantin haussa on myös mahdollista sisällyttää siihen liittyvä tuote mukaan hakutuloksiin. Nämä yhteydet ovat kriittisiä sovelluksen toiminnan kannalta, sillä tietoa täytyy voida hakea eri tavoilla, kuitenkin tietojen väliset yhteydet säilyttäen.

5.3.7 Tuotteiden hakeminen Shopifyn graphQL-rajapinnasta

Tuotteet haetaan massana Shopifyn graphQL-rajapintaa hyödyntäen. Kun käyttäjä painaa käyttöliittymästä synkronointipainiketta, niin frontend lähettää kutsun sovelluksen backendiin, joka taas puolestaan lähettää kutsun Shopifyn graphQL-rajapintaan, jossa luodaan tuotteiden hausta massaoperaatio. Massaoperaation luomisen yhteydessä tietokantaan tallennetaan tieto siitä, että käyttäjällä on massaoperaatio, joka ei ole vielä valmis. Frontend-koodi kyselee tätä tietoa backendiltä jatkuvasti tietyin väliajoin, jotta se saa tietää milloin operaatio on valmis ja uudet tiedot ovat ladattavissa käyttöliittymään.

Shopifyn graphQL-rajapinnan massaoperaatio-ominaisuus mahdollistaa tässä tapauksessa monien tuotteiden ja niiden varianttien tietojen hakemisen yhdellä kutsulla. Tavallisesti useiden tuotteiden hakuun tarvitaan useita verkkokutsuja ja logiikkaa kutsujen hallitsemiseen, mutta nyt Shopifyn massaoperaatio-ominaisuus hoitaa suuren osan tästä työstä automaattisesti (Perform bulk operations with the GraphQL Admin API n.d.). Massaoperaation suorittaminen vastaa suuren datamäärän hakemisessa esiintyviin haasteisiin ja sopii käyttötarkoitukseen todella hyvin.

Massaoperaatio luodaan lähettämällä mutaatio graphQL-rajapintaan, jossa määritellään kaikki halutut tiedot, jotka rajapinnan halutaan palauttavan. Massaoperaation luomisen jälkeen Shopify palauttaa osoitteen, josta massaoperaation tilaa voidaan kysellä. Kysely palauttaa massaoperaation tilan ja erilaisia tietoja operaatiosta. Mikäli massaoperaation tila on valmis, voidaan massaoperaation tuottama tiedosto ladata linkistä, jonka kysely palauttaa. (Bulk query workflow n.d.)

Massaoperaatio palauttaa tiedot JSONL-muodossa. JSONL eroaa tavallisesta JSON-muodosta siten, että jokainen tiedoston rivi on oma validi JSON-objektinsa. JSONL-tiedostomuoto mahdollistaa suurten tiedostojen lukemisen hyvin tehokkaasti, sillä

ne voidaan käsitellä rivi riviltä, ilman että koko tiedostoa tarvitsee lukea ensin sovelluksen muistiin. (The JSONL data format n.d.)

Tuotteet käsitellään ja tallennetaan tietokantaan ja kun kaikki tuotteet on käsitelty ja tallennettu, niin tietokannassa oleva bulk-operaatio asetetaan valmistuneeksi. Näin ollen käyttäjä saa tiedon valmistuneesta operaatiosta, kun frontend-koodi seuraavan kerran kysyy operaation tilaa.

5.3.8 Tuotteiden tallennus tietokantaan

GraphQL-rajapinnasta saatu JSONL-tiedosto (ks. kuvio 21) tallennetaan palvelimella tiedostoon ja käsitellään rivi riviltä. Ensin tiedostosta löytyvät objektit lajitellaan tuotteisiin, variantteihin sekä metatietokenttiin, jotta ne voidaan lisätä tietokantaan oikeassa järjestyksessä. Tuotteiden, varianttien ja metatietokenttien väliset yhteydet luodaan siinä vaiheessa, kun ne tallennetaan tietokantaan.

```
{"id": "ID_HERE", "type":"product", "price": "25.00"}
{"id": "ID_HERE", "type":"product", "price": "45.00"}
{"id": "ID_HERE", "type":"product", "price": "65.00"}
```

Kuvio 21. Esimerkki JSONL-tiedoston käyttämästä muotoilusta

Ensin tallennetaan tuotteet, sillä tuotevarianttien lisääminen ei onnistu ilman, että variantilla on omaa päätuotetta, jolle variantti kuuluu. Tuotteiden lisäämisen jälkeen lisätään tuotteiden variantit ja viimeisenä metatietokentät. Metatietokentät lisätään viimeisenä, sillä yksittäinen metatietokenttä voi kuulua joko tuotteelle tai variantille,

muttei molemmille. Kaikki operaatiot suoritetaan niin, että mikäli tuote, variantti tai metatietokenttä löytyy jo tietokannasta, niin sen tiedot päivitetään vastaamaan uusinta mahdollista tietoa. Tuotteiden tallennuksen jälkeen JSONL-tiedosto poistetaan palvelimelta.

5.3.9 Tuotteiden haku tietokannasta

Tuotteiden haku tapahtuu erilaisten hakuvalitsimien avulla (ks. kuvio 22). Käyttäjä voi muodostaa hakuvalitsimia sovelluksen käyttöliittymässä, josta ne lähetään sovelluksen backendille käsiteltäväksi. Sovellukseen on määritetty valintoja varten “/api/selection”-reitti, jonne hakuvalitsimet lähetään käyttöliittymästä.

```
1  [{"mode":"include","attribute":"p-metafields","operator":"contains","value":"plant"}]
2
```

Kuvio 22. Esimerkki hakuvalitsimesta

Hakuvalitsimet käsitellään sovelluksen ProductService.js-tiedostossa, jonne kutsu ohjataan reitityksestä. Kutsun saavuttua sovellus tarkastaa, minkä tiedon perusteella käyttäjä haluaa tuotteita valita. Mahdollisia vaihtoehtoja ovat: tuotteen kuvaus, tuotteen tunnisteet tai tuotteen metatietokentät. Mikäli valitsin ei ole mikään ennalta määrätyistä, sovellus palauttaa virheen.

Tietokantakysely muodostetaan hakuvalitsimen käsittelyn jälkeen. Jokainen valitsin sisältää tiedon siitä, millä termillä tietokannasta halutaan tietoa hakea. Sovellus muodostaa Sequelize.js:n avulla tietokantakyselyn, joka palauttaa käyttäjän haluamat tuotteet sovellukselle. Sovellus palauttaa sen jälkeen tuotteet reititykseen, josta ne palautuvat lopuksi sovelluksen käyttäjälle käyttöliittymään, jossa se esitetään visuaalisesti esimerkiksi tuotelistauksena.

5.3.10 Operaatioiden ajo käyttäjän valitsemiin tuotteisiin

Käyttäjä voi suorittaa erilaisia operaatioita tuotteisiin, jotka hän on valinnut käyttäen hakuvalitsimia. Operaatioilla tarkoitetaan käytännössä muokkauksia, jotka kohdistuvat tuotteeseen liittyvään tuotetietoon, kuten hintaan tai tuotekuvaukseen. Kaikki operaatiot muodostetaan käyttöliittymässä, josta ne lähetetään sovelluksen backendille käsiteltäväksi.

Käyttöliittymästä lähetetty operaatio vastaanotetaan palvelimella ja sen käsittely aloitetaan tarkastamalla mistä vastaanotetussa operaatiossa on kyse. Vastaanotettu tieto operaatiosta sisältää tiedot suoritettavasta toimenpiteestä, sekä tuotteista johon toimenpide kohdistuu. Mahdollisia tuotteisiin liittyviä toimenpiteiden tyyppejä ovat tuotteen tuotekuvauksen muokkaus, tekstin korvaus tuotekuvauksessa, tuotekuvauksen asettaminen, tuotteen hinnan muutos sekä tuotteen tunnisteiden lisäys ja poisto. Kun toimenpiteen tyyppi on selvillä, sovellus muodostaa Shopify-verkkokauppaan päivitettävät tiedot operaation tietojen perusteella. Sovellus päivittää tiedot Shopify-verkkokauppaan graphQL-rajapinnan kautta, ja onnistuneen päivityksen jälkeen se päivittää tiedon myös omaan tietokantaansa, jotta tiedot Shopifyssä sekä tietokannassa vastaavat toisiaan tarkasti.

Onnistuneen päivityksen jälkeen sovelluksen käyttöliittymä päivitetään vastaamaan päivitettyä tilaa. Käytännössä tämä tarkoittaa valittujen tuotteiden tuotelistauksen päivittämistä käyttöliittymässä.

5.4 Haasteet

5.4.1 OAuthin ymmärtäminen

Yksi työn suurimmista haasteista oli ymmärtää sovelluksen asennuksen yhteydessä käytettävää OAuth-prosessia ja sen toimintaa. Käyttäjän uudelleenohjaus eri verkko-osoitteiden välillä ja erilaisten tarkistusten tekeminen ohjausten välissä tuntui haastavalta ja vaativalta saada toimimaan oikein. Ajan kanssa prosessista kuitenkin muodostui selkeä ja toimiva kokonaisuus, jonka rakentamisesta käsin opittiin paljon.

5.4.2 App Bridgen osuus sovelluskokonaisuudessa

Shopify App Bridgen osuus ja vastuut olivat koko projektin ajan suhteellisen haastavia ymmärtää ja sisäistää. Erityisesti projektin alkuvaiheessa oli vaikea hahmottaa mitä App Bridge -kirjasto tarkalleen ottaen tekee ja mihin sitä sovelluksen toiminnassa tarvitaan. Pala palalta App Bridge -kirjaston käyttö kuitenkin kirkastui ja sen vastuualueet alkoivat hahmottua tarkemmin.

6 Jatkokehitys

6.1 Sovelluksen julkaisu Shopifyn App Storessa

Tulevaisuudessa sovellus voitaisiin julkaista Shopifyn App Storessa ja sillä tavalla tarjota sovelluksen toiminnallisuutta suurelle joukolle verkkokauppiaita. Julkaisu mahdollistaisi myös liiketoiminnan rakentamisen sovelluksen ympärille. Tätä ennen sovelluksen tulee kuitenkin täyttää Shopifyn kriteerit julkaisukelpoiselle sovellukselle ja sitä varten sovellusta tulee vielä kehittää eteenpäin niin käyttöliittymän kuin backend-sovelluksenkin puolesta.

6.2 Käyttöliittymän ominaisuuksien kehitys

Sovelluksen käyttöliittymään voidaan vähäisellä kehitystyöllä tuoda uusia ominaisuuksia käyttäjän saataville. Esimerkiksi tuotteiden valintoihin ja operaatioiden suorittamiseen saadaan helposti luotua lisätoiminnallisuuksia suhteellisen vähäisellä vaivalla. Sovelluksen backendiin tulee uusien ominaisuuksien myötä tehdä hieman lisäyksiä ja huomattavaa on, että palvelimen osa sovelluksesta mukautuu tulevaisuuden muutoksiin melko vaivatta.

Jatkokehityksen näkökulmasta tuotteiden valinta ja operaatiot ovat merkittävä ja mielenkiintoinen kohde. Tuotteiden valintoihin voidaan tuoda erilaisia mahdollisuuksia, kuten useiden eri valintojen yhdenaikainen käyttäminen ja mahdollisuus poissulkea tuotteita valinnasta tietyn ehdon täyttyessä. Näillä ominaisuuksilla tuotteiden valinnasta saadaan huomattavasti joustavampi ja samalla tehokkaampi.

Tuotteisiin suoritettaviin operaatioihin on mahdollista luoda tuki useammalle samanaikaiselle operaatiolle. Tällä tavoin käyttäjä voi valita useita eri operaatioita, jotka ajetaan valittuihin tuotteisiin samanaikaisesti. Usean operaation käyttäminen samaan aikaan säästää käyttäjältä huomattavasti aikaa, sillä tällä hetkellä käyttäjän pitää odottaa edellisen operaation valmistumista, jotta uusi operaatio voidaan aloittaa.

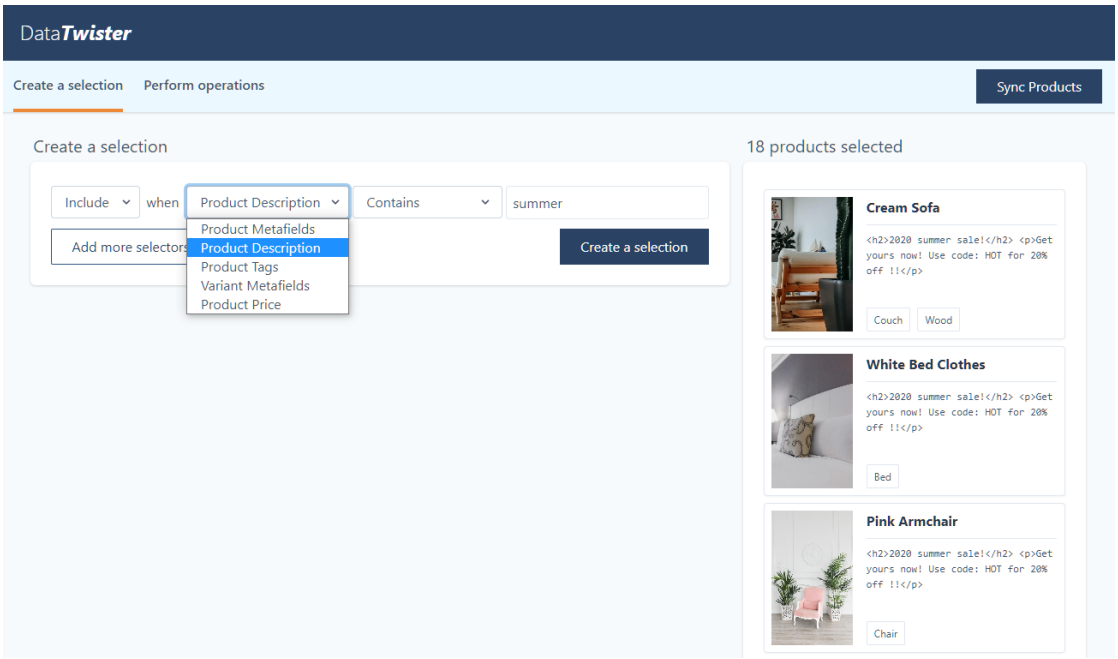
Sovelluksen käyttöliittymää voidaan jatkossa parantaa ilmoittamalla käyttäjälle selkeästi, missä tilassa sovellus milläkin hetkellä on. Tätä varten käyttäjälle tulee esittää selkeästi lataamista ilmaisevat ikonit sekä erilaiset ilmoitukset siitä, miten käyttäjän suorittama toimenpide on sujunut. Tällä hetkellä sovellus ei näytä käyttäjälle ilmoituksia.

7 Tulokset

7.1 Konkreettiset tulokset

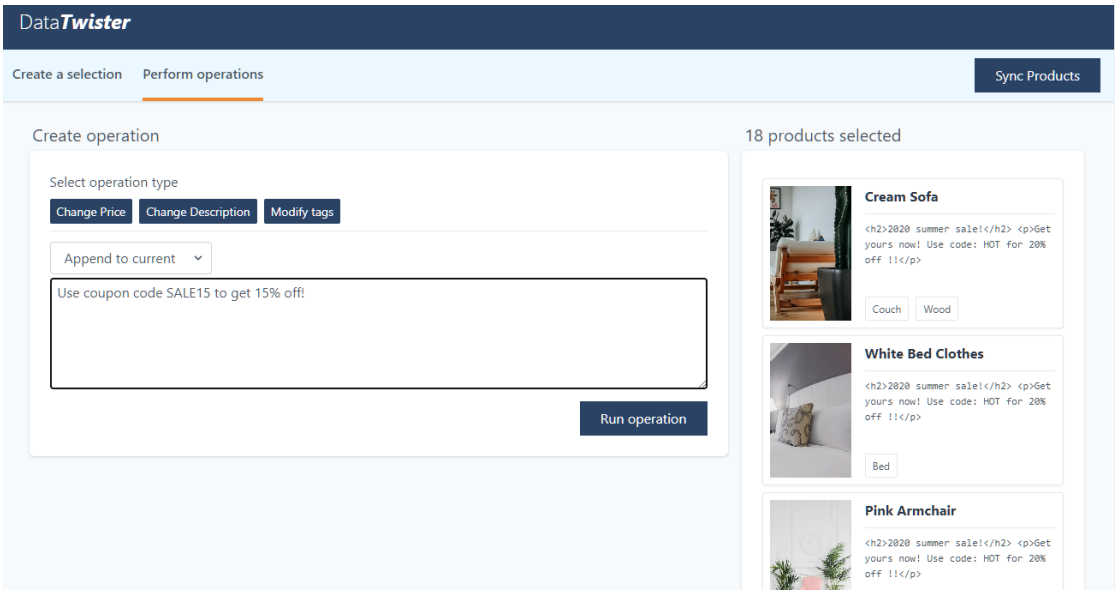
Tuloksena työstä saatiin toimiva sovellus, jonka jatkokehitys on paitsi mahdollista, että myös miellyttävää. Verkkokauppiaan on mahdollista asentaa sovellus omaan verkkokauppaansa, synkronoida verkkokaupan tuotteet sovelluksen tietokantaan ja suorittaa erilaisia muokkauksia valitsemiinsa tuotteisiin.

Kuviossa 23 esitellään valmiin sovelluksen valintanäkymä, jossa verkkokaupan tuotteita voidaan valita erilaisin kriteerein. Sivun vasemmalla puolella ovat näkyvillä tuotteiden valintoihin vaikuttavat kriteerit, joita muuttamalla käyttäjä voi vaikuttaa siihen mitkä tuotteet sovellus valitsee käsiteltäväksi. Oikealla puolella käyttäjälle esitetään listaus tuotteista, jotka nykyisillä valintakriteereillä ovat valittuina.



Kuvio 23. Tuotteiden valintanäkymä sovelluksessa

Tuotteiden valinnan jälkeen käyttäjän on mahdollista suorittaa valitsemiinsa tuotteisiin erilaisia operaatioita, kuten tuotteen tietojen muutoksia. Kuviossa 24 esitellään sovelluksen käyttöliittymän osa, jossa operaatiot suoritetaan valittuihin tuotteisiin. Kuvio kuvaa tilannetta, jossa suoritetaan operaatio, joka lisää valittujen tuotteiden tuotekuvaukseen perään käyttäjän asettaman tekstin.



Kuvio 24. Operaatio tuotekuvauksen muokkaamista varten

Sovellus toimii pohjana myös muille jatkossa kehitettäville Shopify-sovelluksille, sillä sen perustoiminnallisuudet, kuten asennus ja sovelluksen käyttäjän autentikaatio, säilyvät uudelleenkäytettävänä useiden sovellusten välillä.

Työstä saatiin paljon tietoa Shopify-sovelluskehityksestä ja sen ominaispiirteistä. Tätä tietoa voidaan jatkossa hyödyntää myös Shopify-kehityksen ulkopuolella, josta hyvänä esimerkkinä toimii yleisesti käytössä olevan OAuth-mekanismi, jonka toimintaan tässä työssä päästiin tutustumaan.

7.2 Kerätty osaaminen

Sovellusta kehittäessä opittiin paljon erilaisista teknologoista ja mekanismeista. Merkittävin opittu asia oli mielestäni Shopify-sovelluskehityksen kokonaisuuden hahmottaminen ja siihen liittyvien asioiden liittyminen toisiinsa. Teknisestä kulmasta Shopify App Bridge -kirjaston osuus sovelluksen toiminnassa oli kaikista haastavin sisäistää, mutta loppujen lopuksi se paljastui olevan myös yksi yksinkertaisimmista asioista koko projektissa. OAuth-mekanismin toiminnan osaamisesta on varmasti tulevaisuudessa myös hyötyä, joten se on varmasti yksi projektin parhaista oppimiskokemuksista.

Sovelluksen käyttöliittymän tyylittelyn suhteen oli todella palkitsevaa ottaa käyttöön TailwindCSS-kehys. Tailwindin käyttö sovelluksessa mahdollisti CSS-tyylien hallinnan helposti ja vaivattomasti. Luotujen tyylien ylläpito ja muokattavuus säilyivät suhteellisen helppona, vaikka käyttöliittymän rakenne muuttui kehityksen myötä monimutkaisemmaksi.

MySQL-tietokannan ulkoistaminen DigitalOceanin palveluun koettiin sovelluksen kehityksen ja ylläpidon kannalta hyvänä asiana. Tietokannan ollessa täysin ulkoisen tahon toimesta ylläpidetty, voitiin sovelluksen kehityksessä keskittyä olennaiseen ja tulevaisuudessa tuotantoversioon siirryttäessä voidaan varmistua tietokannan tietoturvasta.

Sequalize.js-kirjaston käyttö tietokantaoperaatioissa oli uusi asia, joka sovelluksen kehityksessä tuli vastaan. Projektin aikana havaittiin, että vastaavanlaisen kirjaston käyttäminen on järkevää, sillä se yksinkertaistaa ja tehostaa tietokannan käyttämistä erilaisissa tilanteissa. Monimutkaisten tietokantahakujen teko oli sujuvaa ja tietojen tallennus tietokantaan sujui Sequelize-kirjastoa käyttäen helposti.

7.3 Sovelluksen tuomat hyödyt

7.3.1 Säästetty aika erilaisisssa operaatioissa

Sovelluksen tuomat edut riippuvat hyvin paljon siitä, mitä sovelluksella halutaan tehdä. Mikäli tuotteita on käsittelyssä useita kymmeniä tai satoja kerralla ja sovellusta käytetään sellaiseen toimintoon, joka Shopifyn oman hallintapaneelin kautta on joko vaikeasti tai jopa mahdotonta toteuttaa, niin sovellus tuo silloin merkittävää hyötyä verkkokauppiaalle. Tästä esimerkkinä toimii hyvin useiden tuotteiden kuvaksen muokkaaminen kerralla, mikä on Shopifyn hallintapaneelin kautta tällä hetkellä mahdotonta. Tässä tapauksessa sovelluksella saavuttaa huomattavan säästön ajassa, joka tuotteiden hallintaan kuluu. Mikäli sovelluksella tehdään jotain, mikä on mahdollista Shopifyn omilla työkaluilla, kuten tuotteiden tunnisteiden muokkaus, saattavat sovelluksen käytöstä saatavat hyödyt jäädä melko pieniksi varsinkin, jos käsittelyssä on pieni määrä tuotteita. Sovelluksen jatkokehityksen myötä sovellus kuitenkin pystyy vastaamaan yhä useampiin tilanteisiin, joissa verkkokauppiaan aikaa voidaan säästää.

7.3.2 Jatkokehityskelpoisuus

Sovelluksesta saatiin tehtyä hyvin jatkokehityskelpoinen ja suhteellisen helposti laajennettava. Tämä saavutettiin siten, että sovelluksen taustalogiikka ja käyttöliittymän komponentit ovat jaoteltuna loogisesti eri osiin, jossa niiden kehittäminen ja uudelleenkäyttäminen on mahdollista ja melko vaivatonta.

Sovelluksessa on potentiaalia siihen, että se voidaan tulevaisuudessa julkaista Shopify App Storessa useiden eri verkkokauppiaiden asennettavaksi, jolloin se tarjoaa toiminnallisuuttaan potentiaalisesti useille sadoille tai tuhansille verkkokaupoille samanaikaisesti.

8 Pohdinta

8.1 Tavoitteiden toteutuminen

Työlle asetetut tavoitteet toteutuivat hyvin. Kaikki vaatimusmäärittelyssä olevat ominaisuudet eivät valmistuneet lopulliseen versioon sovelluksesta, mutta kaikki oleelliset ominaisuudet saatiin toteutettua ja sovelluksen toimintalogiikka saatiin varmennettua.

Loppujen ominaisuuksien kehittäminen sujuu käytännössä samalla tavalla, kuin sovellukseen jo toteutetut ominaisuudet, joten puuttuvien ominaisuuksien toteuttamisesta ei juuri olisi tässä tapauksessa oppimisen kannalta merkittävää hyötyä.

8.2 Onnistumiset ja haasteet

Projektin ehdottomasti suurin onnistuminen oli se, että siitä saatiin lopputuloksena toimiva ja jatkokehityskelpoinen sovellus. Projektin alussa Shopify-sovelluskehitys oli tuntematon polku, jonka tutkimisessa riitti työtä. Tutkiminen kannatti ja se tuotti myös hedelmää erilaisten oppien, havaintojen ja kehityskelpoisen sovelluksen muodossa. Oli hienoa huomata, miten saumattomasti käyttöliittymässä toimivan React-sovelluksen sekä palvelimella toimivan Node-sovelluksen sai toimimaan yhteen ja

luotua niiden avulla sulavasti toimivan kokonaisuuden Shopify-sovelluksen muodossa.

Työssä merkittävin haaste oli kova aikataulupaine, jonka takia työssä jouduttiin karsimaan ominaisuuksia, jotka olisivat muuten olleet mielekkäitä toteuttaa. Näihin ominaisuuksiin kuuluvat mm. useiden tuotevalitsimien yhdenaikainen mahdollistaminen, sekä usean operaation samanaikainen suorittaminen. Myöskään kaikkiin Shopify-sovelluskehityksen erityispiirteisiin ei keretty paneutua niin tarkasti, kuin aluksi oli tarkoitus.

Työn heikkona puolena oli se, että Shopify-sovelluskehityksen vaativuuteen ei keretty perehtyä tarkemmin, ja vertailemaan sitä muuhun tavanomaiseen kehitystyöhön. Vaikka työssä kuvataan joitain Shopify-sovelluksen kehityksessä vastaan tulevia asioita ei lukijalle silti muodostu suoraan tarkkaa kuvaa siitä, kuinka haastavaksi Shopify-sovelluskehityksen voisi luokitella yleisesti ottaen. Luotettavan luokittelun muodostaminen on varmasti myös melko haastavaa, sillä erilaisia ohjelmointitehtäviä on vaikea verrata toisiinsa.

9 Lähteet

About Node.js. Node.org. Viitattu 15.11.2020
https://nodejs.org/en/about/

App Types. N.d. Shopify.com. Viitattu 8.11.2020
https://help.Shopify.com/en/manual/apps/app-types.

Associations. N.d. Sequelize.org. Viitattu 4.12.2020
https://sequelize.org/master/manual/assocs.html

Authenticate with OAuth. Shopify.dev. Viitattu 4.12.2020
https://shopify.dev/tutorials/authenticate-with-oauth

Bulk query workflow. N.d. Shopify.dev. Viitattu 4.12.2020
https://shopify.dev/tutorials/perform-bulk-operations-with-admin-api #bulk-query-workflow

GraphQL and its benefits. N.d. Shopify.dev. Viitattu 8.11.2020
https://Shopify.dev/concepts/graphql/benefits.

Introducing Express. 14.11.2020. Developer.mozilla.org. Viitattu 15.11.2020
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

Introducing Hooks. N.d. Reactjs.org. Viitattu 15.11.2020
https://reactjs.org/docs/hooks-intro.html

Introducing JSX. N.d. Reactjs.org. Viitattu 15.11.2020
https://reactjs.org/docs/introducing-jsx.html

Nodejs. N.d. Nodejs.org. Viitattu 15.11.2020
https://nodejs.org/en/

Perform bulk operations with the GraphQL Admin API. N.d. Shopify.dev. Viitattu 4.12.2020
https://shopify.dev/tutorials/perform-bulk-operations-with-admin-api

Releases. N.d. Nodejs.org. Viitattu 22.11.2020
https://nodejs.org/en/about/releases/

Shopify Admin API. N.d. Shopify.com. Viitattu 8.11.2020.
https://Shopify.dev/docs/admin-api.

Shopify App Bridge. N.d. Shopify.dev. Viitattu 2.12.2020
https://shopify.dev/tools/app-bridge

Shopify Storefront API. N.d. Shopify.dev. Viitattu 8.11.2020
https://Shopify.dev/docs/storefront-api.

The JSONL data format. N.d. Shopify.dev. Viitattu 4.12.2020
https://shopify.dev/tutorials/perform-bulk-operations-with-admin-api#the-jsonl-data-format

The Metafield Object. N.d. Shopify.dev. Viitattu 8.11.2020
https://Shopify.dev/docs/themes/liquid/reference/objects/metafield.

Variants. N.d. Help.Shopify.com. Viitattu 15.11.2020
https://help.Shopify.com/en/manual/products/variants

What is JSON Web Token. N.d. jwt.io. Viitattu 2.12.2020
https://jwt.io/introduction/

What is Shopify. 1.3.2020, Shopify.com. Viitattu 15.11.2020
https://www.Shopify.com/blog/what-is-Shopify

When to Use Context. N.d. Reactjs.org. Viitattu 15.11.2020
https://reactjs.org/docs/context.html#when-to-use-context