Tampereen ammattikorkeakoulu

Väliohjelmiston käyttö ja hyödyt

Mika Vormisto

OPINNÄYTETYÖ
Marraskuu 2019

Tietojenkäsittely
Web-palvelut

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Web-palvelut

VORMISTO, MIKA:
Väliohjelmiston käyttö ja hyödyt

Opinnäytetyö 31 sivua, joista liitteitä 3 sivua
Marraskuu 2019

Opinnäytetyön taustalla oli asiakasprojekti, jossa rakennettiin verkkokaupan käyttöliittymä käyttäen väliohjelmistoa. Työn tilaajana oli Solteq Oyj, jonka projekti oli kyseessä. Työn tarkoituksena oli avata väliohjelmisto konseptina ja selvittää sen käyttötavat ja hyödyt nykyaikaisessa ohjelmistokehityksessä. Tavoitteena oli saada lukija kiinnostumaan väliohjelmistosta ja ymmärtämään se ohjelmistoarkkitehtuurisena ratkaisuna.

Opinnäytetyössä käsiteltiin käytettyjä teknologioita ja työkaluja, mutta keskityttiin erityisesti Solteq Oyj:n omistuksessa olleeseen väliohjelmistoon Perpendiculariin. Työssä kerrottiin väliohjelmiston teknisestä puolesta ja pohdittiin mahdollisia käyttötapauksia.

Työn tuloksena syntyi tiivis tietopaketti, jota tilaaja, muu ohjelmistokehittäjä tai ohjelmistoarkkitehti voi hyödyntää uuden ohjelmointiarkkitehtuurisen mallin oppimiseen ja perehdyttämiseen. Mahdollisena kehittämiskohtana havaittiin ainakin väliohjelmiston laajempi käyttötapauskartoitus.

Asiasanat: väliohjelmisto, palvelukerros, ohjelmistoarkkitehtuuri, verkkokauppa, käyttöliittymäkehitys

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Business Information Systems
Web Services

VORMISTO, MIKA:
The Usage and Advantages of Middleware

Bachelor's thesis 31 pages, appendices 3 pages
November 2019

The background of the study was a customer project, where an ecommerce platform was built using the Angular middleware. The purpose was to explain middleware as a concept, and to survey how it is used and when it should be used. The objective of the thesis was to have the reader fully understand how middleware works as a software architecture solution.

The technologies and tools that were used in the project were introduced, but the focus remained in the middleware Perpendicular. The result of the thesis was a document that can be used to educate Solteq staff and third-party developers. In order to further develop the study, a broader survey of possible use cases could be done.

Key words: middleware, service layer, software architecture, ecommerce

SISÄLLYS

ERITYISSANASTO

B2C	Business to Customer eli kuluttajalle suunnattu
B2B	Business to Business eli yritysten välinen
Back-end	Palvelinpuoli eli ohjelmiston näkymätön osuus, joka ajetaan yleensä palvelimella tai pilvessä. Esimerkkinä verkkokauppasovellus
Front-end	Selainpuoli eli ohjelmiston näkyvä osuus. Esimerkkinä JavaScript ja TypeScript
JavaScript	Ohjelmointikieli, joka on käytössä erityisesti Web-ympäristössä
JSON	JavaScript Object Notation eli tiedonvälityksessä yleisesti käytettävä tiedostomuoto
Ohjelmistokehys	Ohjelmoinnissa käytettävä apuväline, joka yleensä tarjoaa valmiita tietokoneohjelmissa käytettäviä toimintoja
REST	Representational State Transfer eli arkkitehtuurimalli, jonka avulla ohjelmointirajapintoja toteutetaan
Tekstieditori	Tietokoneohjelma, jolla muokataan tekstiä tai koodia. Tärkeä työkalu front-end -kehityksessä.
TypeScript	Vahvaa tyyppijärjestelmää tukeva ohjelmointikieli, joka on rakennettu JavaScriptin päälle

1 JOHDANTO

Aloitettuani Solteq Oyj:llä oman harjoittelujaksoni pääsin ensitöikseni suureen verkkokauppaprojektiin, jossa tehtävänäni oli rakentaa verkkokaupan käyttöliittymä Angular-ohjelmistokehyksellä. Olin samaan aikaan innoissani ja peloissani, sillä en kokenut olevani hyvä ohjelmoija ja ajatus oikeasta asiakasprojektista ahdisti.

Keskustelin samalla alalla työskentelevien ystävieni kanssa teknologioista ja työkaluista, joilla ensimmäistä projektiani olin päässyt tekemään. Minulle selvisi nopeasti, kuinka mainitsemani front-endin ja back-endin välissä ollut väliohjelmisto oli heille outo ja uusi käsite. Ystävieni hämmästely tuntui oudolta, sillä aloittelevana kehittäjänä olin luullut väliohjelmistojen käytön olevan arkipäivää. Oletin, että se on laajemmin tunnettu konsepti ja tavalla tai toisella muuallakin käytössä.

Projektin edetessä ohjelmointitaitoni luonnollisesti kehittyivät ja sain kokonaisvaltaisemman kuvan koko projektista ja siitä, miten eri kerrokset keskustelivat keskenään ja miksi. Tällöin aloin ymmärtämään käytössämme olevan väliohjelmiston tarjoamat mahdollisuudet ja hyödyt. Silloin heräsi ajatus tähän opinnäytetyöhön: miksi en avaisi konseptia asiasta kiinnostuneille enemmän?

Ollakseen varteenotettava kilpailija alalla kuin alalla on yrityksellä oltava vähintäänkin verkkosivut, joskus jopa mobiilisovellus. Monella alalla sivuihin yhdistyy kauppa, josta tuotteita tai palveluita voi ostaa. Tarve verkkosovelluksille on nykyisin kova. Näen siis, että verkkosovelluksien tekoon on aina uudet tehokkaammat toteutustavat tervetulleita – siksi koin, että tälle opinnäytetyölle on tarve.

Esittelen opinnäytetyössäni väliohjelmiston konseptina ja tutkin väliohjelmiston käyttöä ja sen hyötyjä Solteq Oyj:n asiakasprojektin näkökulmasta. Avaan lukijalle projektissa käytössä olleita työkaluja ja teknologioita mutta keskityn käytössä olleeseen väliohjelmistoon, Perpendiculariin. Käsittelen myös sitä, minkälaisiin käyttötarkoituksiin vastaavanlainen väliohjelmisto voisi sopia.

Tarkoituksenani on selkeyttää lukijalle käyttämäni väliohjelmiston toimintatapoja ja avata väliohjelmistoa sovellusarkkitehtuurisena ratkaisuna. Tavoitteenani on

herättää opinnäytetyölläni lukijan mielenkiinto väliohjelmistoja kohtaan ja saada aikaan keskustelua ja kysymyksiä väliohjelmistojen käytöstä, hyödyistä ja tulevaisuudesta. Koen, että projektien kokonaisvaltaista arkkitehtuuria mietittäessä väliohjelmiston käyttöä tulisi vakavasti harkita.

2 PROJEKTIN ESITTELY JA TAUSTAT

Projektin taustalla oli asiakasyrityksen halu uudistaa kokonaan heidän digitaalisen kaupankäynnin konseptinsa. Tarve oli kova, sillä asiakkaan aikaisempi verkkokauppa oli jo parhaat päivänsä nähnyt. Ne olivat visuaalisesti vanhentuneet ja vanhaa teknologiaa, jonka seurauksena ne olivat myös hitaat. Myyjien työkalut olivat myös vanhentuneet: hidasta ja rajoittunutta konsolia käytettiin asiakastietojen hallintaan. Myöskään mobiilisovellusta ei ollut, mutta kilpailun ja kysynnän takia sille koettiin tarvetta.

Syntyi hanke, jonka tavoitteena oli luoda digitaalisen liiketoiminnan työkalut, jotka kantavat pitkälle tulevaisuuteen. Käytännössä tämä tarkoitti sitä, että rakennettaisiin uusi B2C- ja B2B-verkkokauppa – täysin tyhjältä pöydältä uusia teknologioita käyttäen. Uusien verkkokauppojen lisäksi asiakas halusi, että Solteq toimittaisi pelkästään B2B-verkkokaupan käyttöliittymän päällä toimivan työkalun myyjiensä avuksi. Myyjän työpöydäksi kutsutun työkalun tarkoituksena oli toimia eräänlaisena toisena käyttöliittymänä asiakasyrityksen myyjien apuna asiakashallinnassa ja myynnissä. Sen avulla myyjä pystyisi käsittelemään useampia asiakkaita samaan aikaan ja esimerkiksi tallentamaan useampia koreja itselleen muistiin.

Verkkokauppojen lisäksi yritys halusi helpottaa asiakkaidensa ostoprosessia luomalla mobiilisovelluksen, mutta koska asiakasyrityksen myynti painottuu vahvasti B2B-puolelle, haluttiin sovellus tuottaa palvelemaan vain B2B-puolen asiakkaita. Kaiken edellä mainitun lisäksi myös tuote- ja asiakastiedonhallinnan ratkaisut kuuluivat hankkeen tilaukseen.

Kehitystiimimme, johon minun lisäksi kuului kaksi muuta käyttöliittymäkehittäjää, tehtäväksi tuli toteuttaa B2C- ja B2B-verkkokauppa kokonaisuudessaan mukaan lukien B2B-verkkokaupan päälle kehitettävä Myyjän työpöytä -työkalu. Asiakkaan toiveesta keskityimme ensin B2B-verkkokaupan ja Myyjän työpöydän kehittämiseen, sillä se oli heidän yritystoiminnalleen olennaisin asia. Pian projektin alettua saimme vielä neljännen käyttöliittymäkehittäjän käyttöömme, sillä tekemistä riitti. Kehitystiimiimme apuna oli myös kaksi back-end -kehittäjää ja testaaja, jotka tekivät töitä tasaisesti projektin kaikkiin toimitettaviin tuotteisiin.

Opinnäytetyössäni käsittelemäni asiat koskevat vain niitä asioita, joiden kehittämisessä olen ollut mukana. Aion sivuttaa hankkeen mobiilisovelluksen ja tuote- ja asiakastiedonhallinnan ratkaisut kokonaisuudessaan, sillä niistä minulla on hyvin vähän kokemusta.

3 TYÖKALUT JA TEKNOLOGIAT

Projektissa käytettävien työkalujen valinta tulee tehdä huolellisesti ja harkiten. Erityisesti verkkosovelluksia ja verkkokauppoja tehtäessä on yritettävä parhaansa mukaan ennustaa, mitkä ohjelmistokehykset ovat laajassa käytössä vielä useammankin vuoden päästä, jotta valitun ohjelmistokehyksen kehittämistä ei lopeteta ja mahdollisia tietoturvauhkia ei pääse syntymään. Ohjelmistokehyksen ei myöskään kannata olla liian uusi. Tällöin mahdolliset tietoturva-aukot eivät välttämättä ole vielä yleisessä tiedossa, yhteisön tuki on vähissä eikä ohjelmistokehyksen mahdollisesta suosiosta ole takeita (Nowak 2019). Väärän ohjelmistokehyksen valinta kerryttää teknistä velkaa jo ennen projektin varsinaista alkamista, mikä ei ole järkevää.

3.1 IBM WebSphere Commerce

Asiakkaan aikaisemman verkkokaupan taustalla oli IBM:n kehittämä WebSphere Commerce (jatkossa Commerce), joka on esimerkiksi Shopifyyn, WooCommerceen tai SAP Hybrikseen verrattavissa oleva isojen yritysten verkkokauppaohjelmisto. Commerce skaalautuu erinomaisesti suurille yrityksille tai konserneille, ja sen avulla asiakasyritys voi hallinnoida koko verkkokauppatoimintaansa erittäin tehokkaasti (IBM 2019a). Koska asiakkaalla oli aikaisempaa kokemusta Commercen kanssa työskentelystä, ei nähty mitään syytä vaihtaa toimivaa tuotetta toiseen. Kuitenkin versiopäivitys vanhasta 6.0 versiosta uuteen 9.0 versioon päätettiin samalla suorittaa, sillä se tarjosi asiakkaan kaipaamia uusia ominaisuuksia. Verkkokauppaohjelmiston versiopäivitys ei kuitenkaan vaikuttanut kehitystiimimme työtehtäviin mitenkään, sillä se tapahtui kulissien takana ennen varsinaista käyttöliittymän kehityksen alkamista. En aio keskittyä siihen opinnäytetyössäni.

3.2 Angular

Käyttöliittymää päätettiin lähteä rakentamaan Googlen kehittämällä Angular-ohjelmistokehyksellä, joka lukeutuu Reactin ja Vuen lisäksi suosituimpiin ohjelmistokehyksiin (Stack Overflow 2019, Elliott 2019). Mainitut ohjelmistokehykset ovat laajoja ja tehokkaita komponenttikeskeisiä ohjelmointikirjastoja, jotka tarjoavat käyttöliittymää kehittäville ihmisille erilaisia työkaluja. Jokainen ohjelmistokehys on tietysti erilainen ja tarjoaa erilaisia työkaluja samalla tietysti toimien hieman omalla tavallaan, mutta paljon yhtäläisyyksiäkin löytyy.

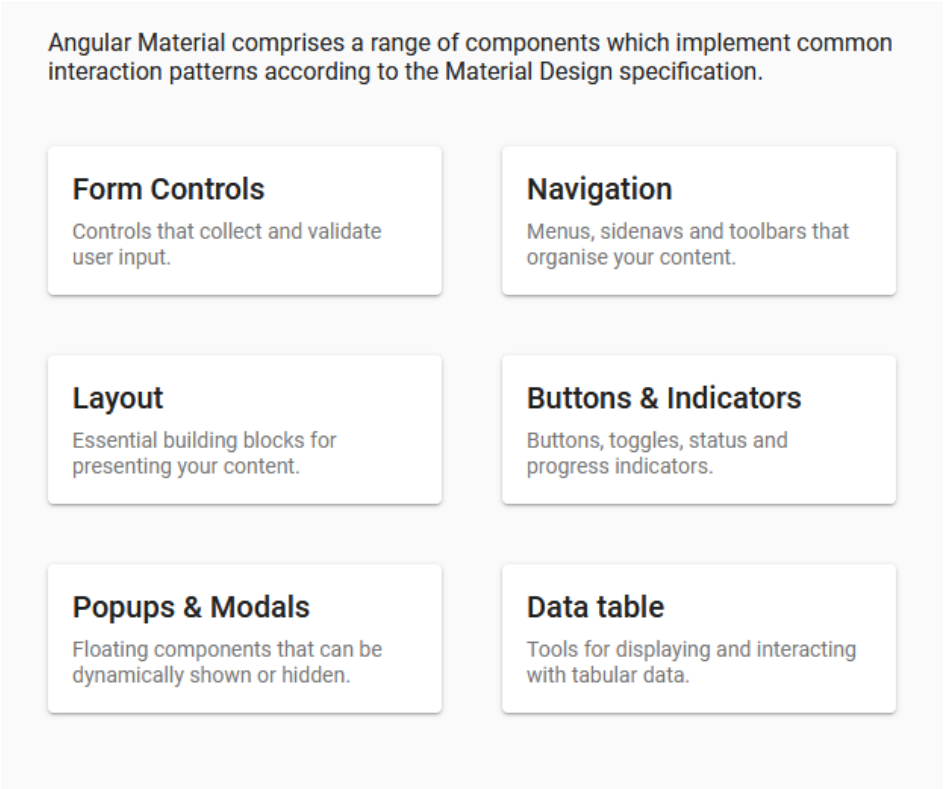
Tärkeimpänä syynä käyttöliittymän kehittämisteknologian valinnalle oli siis se, että Solteq oli aikaisemmin ostanut tanskalaisen yrityksen ja sen myötä saanut omistukseensa Commercen eli back-endin ja verkkokaupan käyttöliittymän eli front-endin sekä kommunikointia avustavan Angular-väliohjelmiston Perpendicularin. Sen tarkoituksena on toimia Commercen ja Angularin ”välikerroksena” näin helpottaen ja nopeuttaen käyttöliittymäkehittäjän työtä huomattavasti.

Tämän lisäksi Angularin tarjoama komponenttikeskeinen malli sopii erinomaisesti laajan verkkokaupan kehittämiseen, sillä jokainen isompi tai pienempi kokonaisuus voi olla oma komponenttinsa, joka voidaan helposti käyttää uudelleen eri yhteydessä. Tämä tarkoitti käytännössä esimerkiksi sitä, että Lisää ostoskori -painike voitiin tulostaa kaikkine toiminallisuuksineen juuri sinne, missä sitä tarvittiin.

3.3 Angular Material

Angularin valintaa puolsi myös se, että Google tarjoaa Angularille valmiiksi kattavan Angular Material -komponenttikirjaston, joka sisältää ison määrän verkkokaupalle olennaisia käyttöliittymäelementtejä. Se on erinomainen kirjasto, joka tehostaa käyttöliittymäkehittäjän työtä huomattavasti (Arora ym. 2018). Yli kolmeenkymmeneen valmiiseen komponenttiin lukeutuu muun muassa erilaiset painikkeet, väriteemat, tekstikentät ja päivän valitsimet, jotka ovat yleisesti ottaen jopa laajasti muokattavissa kirjaston tarjoamin parametrein. Usein valmiiksi tarjotut muokkaukset olivat riittävät, mutta komponentteja pystyi muokkaamaan myös

itse omiin tarpeisiin sopiviksi (Google 2019). Kun käytössämme oli suuri määrä valmiita komponentteja, verkkokaupan rakentamista ei tarvinnut aloittaa täysin tyhjästä. Tällöin valmiita rakennuspalikoita käyttäen oli mahdollista tuottaa asiakkaalle konkreettista näytettävää jo projektin varhaisessa vaiheessa. Esimerkkejä Angular Material -komponenttikirjaston tarjoamista valmiista komponenteista kuvassa 1.



Kuva 1: Angular Material -komponenttikirjaston tarjontaa.

Komponenttikirjaston lisäksi Googlen Material sisältää kattavan ja avoimen ikonikirjaston, joka tarjosi meille kaikki verkkokaupassa yleisimmin käytetyt ikonit. Vaikka muitakin täysin ilmaisia ikonikirjastoja oli meille tarjolla, päädyimme kuitenkin lopulta käyttämään Material-kirjastoa erityisesti sen saumattoman Angular-yhteensopivuuden takia.

Aikaisemmin mainitut komponentti- ja ikonikirjastot noudattavat ulkonäöltään ja toiminnallisuuksiltaan Googlen suunnittelemaa Material Design -muotokieltä, jonka minimalistista filosofiaa hyödynsimme käyttöliittymän suunnittelussa suurelta osin. Material Design on alun perin julkaistu vuonna 2014 ja on siitä asti ollut suuressa suosiossa verkkosivujen suunnittelussa (Moss 2019). Se määrittelee,

kuinka erilaisten elementtien tulisi käyttäytyä suhteessa toisiinsa, esimerkiksi luoden varjon alleen tietyissä tilanteissa ja muuttaen muotoaan tai väriään käyttäjän toimintojen mukaisesti. Material Design tarjosi erinomaisen perustan verkkosivun ulkoasun suunnitellulle niin meidän suunnittelijallemme kuin kehitystiimillemmekin.

Kaiken kaikkiaan Googlen tarjoama Material-kokonaisuus oli erittäin hyödyllinen projektillemme. Työtuntien määrää, joka kyseisen kokonaisuuden itse rakentamiseen olisi kulunut, on vaikea lähteä edes arvioimaan, niin isosta paketista on kyse. Joka tapauksessa rahallista säästöä tuli niin asiakkaalle kuin Solteqille.

4 KEHITTÄMINEN

Kun verkkokaupan käyttäjä lisää tuotteen ostoskoriinsa painamalla Lisää ostoskoriin -painiketta, tulee käyttöliittymän kertoa siitä myös itse verkkokaupalle, tässä tapauksessa Commercelle. Commercen ja käyttöliittymän välinen viestintä tapahtuu REST-rajapinnan kautta, eli kehittäjän tulee ensin muotoilla kutsu tietyin parametrein. Kyseisessä tapauksessa kutsu voisi sisältää kaikessa yksinkertaisuudessaan esimerkiksi käyttäjän tunnisteen, tuotenumeron ja tuotteen kappalemäärän. Tämän jälkeen kutsu tulee ohjata oikeaan osoitteeseen ja sen jälkeen käsitellä Commercen vastaus. Liitteenä 1 on kulkukaavio, jossa havainnollistan edellä kuvatun toiminnon. Olen käyttänyt kulkukaavion havainnollistamisen mallina WebSphere Commerce 7.0 -rajapinnan dokumentaatiota (IBM 2019b).

Edellä kuvailtu operaatio on kiistämättä yksi verkkokaupan yksinkertaisimmista perustoiminnallisuuksista. Liitteen 1 kulkukaavio kuitenkin havainnollistaa sen, että kyseisen operaatio pitää yllättävän paljon sisällään logiikkaa ja muistettavaa, jolloin samalla on myös useampi mahdollisuus tehdä virheitä.

Ajallisesti ohjelmointiin ei kulu kovin pitkä aika. Liitteen 1 esimerkki on kuitenkin vain yksi sadoista mahdollisista kutsuista, joista jokainen on erilainen ja tarvitsee toimiakseen oikean määränpään ja oikeat parametrit. Näiden kaikkien tarkistaminen joka kerta saattaa käydä isommassa projektissa hyvin äkkiä todella hitaaksi. Tämän lisäksi tiettyjä kutsuja saatetaan tarvita useammassa paikassa, jolloin kutsujen koodin monistaminen on hyvin houkuteltavaa ja tietysti suotavaa.

On suositeltavaa tehdä edellä kuvatut toiminnot palveluiksi, jolloin niitä voi käyttää useassa eri paikassa helposti. Tämä pätee erityisesti vähänkään isommassa projektissa, jossa erilaisia palveluita voi olla useita kymmeniä. Miten projektin sisäinen palveluista muodostuva kirjasto sitten eroaa käytössämme olleesta väliohjelmistosta, Perpendicularista?

4.1 Perpendicular lyhyesti

Perpendicular on rakennettu käytännöstä kolmesta kerroksesta. Ensimmäinen kerros on käyttöliittymien käytössä oleva palvelukerros, joka käyttäytyy hyvin pitkälti samalla tavalla kuin palvelukirjasto. Käyttöliittymäkehittäjä kutsuu kerrosta käyttääkseen sen palveluita esimerkiksi saadakseen aikaan REST-kutsun lähettämisen. Perpendicularin palvelukerros on kuitenkin erilainen, sillä se ei varsinaisesti muotoile ja lähetä kutsua tai käsittele vastausta, vaan käskyttää toisia kerroksia.

Toinen kerros on niin kutsuttu välittäjäkerros, jota palvelukerros kutsuu tarpeen mukaan. Tälle kerrokselle ei normaalisti ole tarvetta, mutta se on yksi tärkeimmistä syistä, miksi Perpendicular on niin hyödyllinen työkalu. Koska eri rajapinnoille lähtevät kutsut muodostetaan välittäjäkerroksessa, sen voi muokata käyttämään eri rajapintaa tilanteen niin vaatiessa. Tämä ei kuitenkaan vaikuta käyttöliittymäkehittäjän tehtäviin, sillä kehittäjä ei itse kutsu suoraan välittäjäkerroksen toimintoja, vaan abstrahoivan palvelukerroksen toimintoja.

Rajapinnoilta saatujen vastausten käsittelyyn on tehty Perpendicularin kolmas kerros – tehdaskerros. Sen tehtävänä on rakentaa JSON-formaatissa olevasta vastauksesta Perpendicularin tarjoamia luokkia käyttöliittymäkehittäjien käyttöön. Rajapinnan vaihtuessa myös tehdaskerrosta joudutaan todennäköisesti muokkaamaan.

Perpendicular on luotu helpottamaan ja nopeuttamaan käyttöliittymäkehittäjän työtä. Sen tarjoamat palvelut mahdollistavat Commercen rajapinnan kanssa keskustelun yksinkertaisesti ja helpottavat rajapinnan vastauksen käsittelyä. Perpendicular hoitaa käytännössä kaiken raskaan työn ja pyrkii poistamaan suurimman osan liitteen 1 kulkukaavion sisällöstä. Kehittäjän ei tarvitse muistaa tai tietää, millä protokollalla rajapinnan kanssa keskustellaan. Kehittäjän ei tarvitse tietää, mikä on rajapinnan päätepisteen oikea osoite. Sen ei myöskään tarvitse muistaa, minkälaisena objektina rajapinta haluaa mahdolliset parametrit.

4.2 Perpendicularin käyttö

Havainnollistaakseni Perpendicularin käyttöä aion käydä läpi aikaisemmin esille tuomani esimerkkitapauksen tuotteen lisäämisestä käyttäjän ostoskoriin uudelleen – tällä kertaa Perpendicularin käyttäen. Kaikessa yksinkertaisuudessaan tuotteen lisääminen koriin onnistuu Cart-palvelua käyttäen yhdellä komennolla. Kuva 2 havainnollistaa tarvittavan logiikan vähyyden erinomaisesti.

```
/**
 * When called sets spinner to true and adds the current product to cart
 */
public addToCart() {
    const skuId = this.product.singleSKUCatalogEntryID;
    this.addingToCart = true;
    this.cartService.addToCart(skuId, this.quantity);
  }
```

KUVA 2: Lisää ostoskoriin -painikkeesta tapahtuva toiminto.

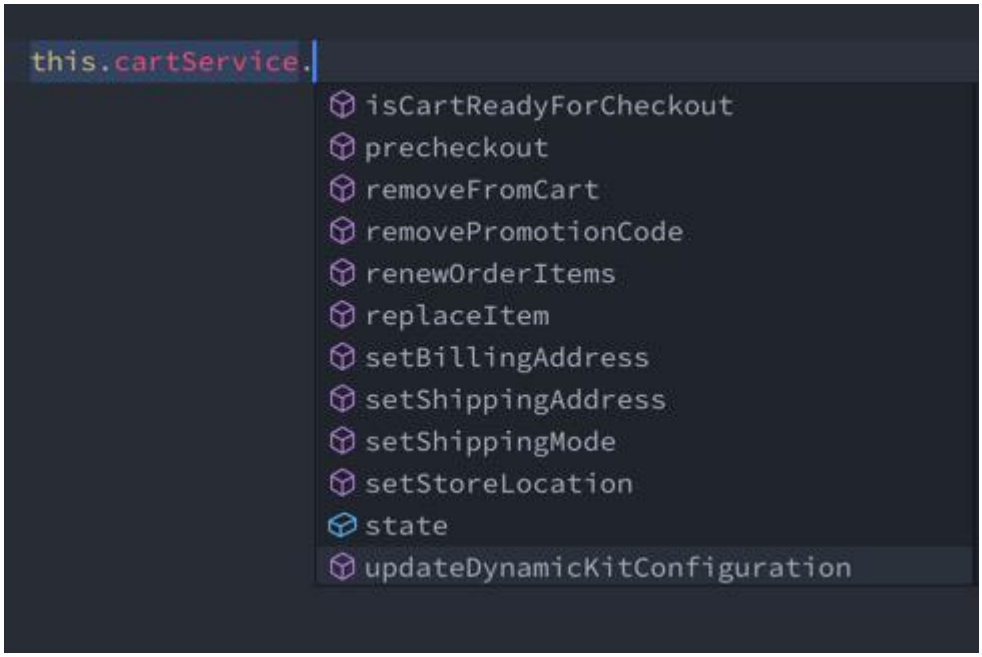
Kuvakaappaus on suoraan koodikannasta, jossa Cart-palvelua käytetään komponentin sisäisessä `addToCart`-toiminnossa. Kuvassa 2 olennaisin asia on käytännössä vain `this.cartService.addToCart`-kutsu, joka vain käskyttää Cart-palvelua lisäämään tuotteen koriin. Palvelun `addToCart`-toiminnossa itsessään on paljon logiikkaa, jonka avulla voidaan käyttäjälle esimerkiksi ilmoittaa onnistuneesta koriin lisäyksestä. Lopuksi palvelu kuitenkin itse käskyttää Cart-välittäjää, jossa kutsu lopulta rakentuu ja se lähetetään. Kuva 3 havainnollistaa, mitä välittäjän sisällä tapahtuu. Kutsun sisältö muodostuu välittäjän sisällä olevia aputoimintoja käyttämällä, ja lopulta kutsu rajapinnalle lähetetään. Kun vastaus saadaan, se palautetaan Cart-palvelulle, josta se palautuu kehittäjän käytettäväksi itse komponenttiin.

```
/**
 * Adds a (set) of items to the cart.
 */
public addToCart(skuId: string[], qty: number[]): Promise<Cart> {
  const url: string = this.getAddToCartURL();
  const payload: any = this.getAddToCartPayload(skuId, qty);
  const thePromise: Promise<Cart> = this.driver.post(url, payload, null);
  return thePromise;
}
```

KUVA 3: Cart-välittäjän addToCart-toiminto.

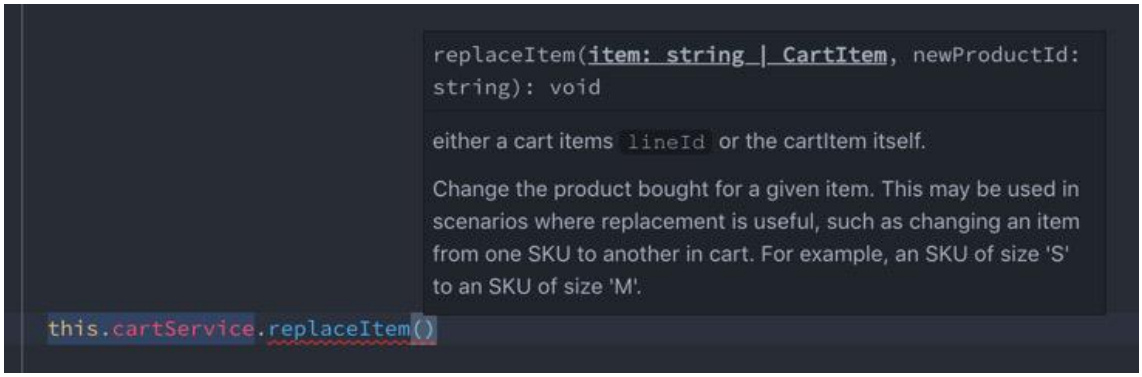
Haluan kuitenkin toistaa, että kuvassa 2 esittelemäni komento on itse kehittäjälle siis ainoa asia, mitä hänen tulee Perpendicularia käytettäessä kutsua. Muut kuvailemani asiat tapahtuvat Perpendicularin eri kerroksissa automaattisesti. Vertailemalla liitettä 1 ja kuvaa 2 on toivottavasti jokaiselle selvää, miksi Perpendicularia käyttämällä tekeminen helpottuu. Perpendicularin helposti muistettavat ja intuitiiviset komennot näkyvät suoraan kehittäjän työn nopeutumisena.

Koska Angular on TypeScript-pohjainen ohjelmistokehys, kehittäjälle uusien komentojen käyttäminen ei myöskään ole haaste. TypeScriptin ja hyvän kommentoinnin yhdistelmän ansiosta kehittäjä saa tekstieditoriinsa vihjeen siitä, miten ja milloin tiettyä komentoa kuuluisi käyttää. Esimerkiksi, jos kehittäjällä on tarve päivittää käyttäjän korissa oleva tuote toiseen, hän voi selata Cart-palvelun tarjoamia toimintoja kirjoittamalla toimintoa tukevaan tekstieditoriin pelkästään `this.cartService`. Kuvassa 4 näkyy tekstieditorin kehote, joka tulee esille tarjoten palvelun tuettuja komentoja.



KUVA 4: Tekstieditorin kehote.

Hyvin nimetyistä toiminnoista kehittäjä pystyy helposti päättelemään, että `replaceItem`-nimisellä komentokutsulla tuotteen pystyy vaihtamaan. Tämän jälkeen kehittäjä pystyy katsomaan tekstieditoristaan, mitä parametreja kyseinen komento tarvitsee toimiakseen. Tämä toiminnallisuus on havainnollistettu kuvassa 5.



KUVA 5: Kehittäjän avuksi kirjoitetut kommentit.

Perpendicularin Cart-palvelun `replaceItem`-toiminnon yläpuolelle kirjoitetut opastavat kommentit näkyvät kehittäjälle kuvan 5 mukaisesti tekstieditorissa. Kehitystiimimme käytössä ollut tekstieditori, Microsoftin kehittämä Visual Studio Code, kutsuu tätä kyseistä kehotetoimintoa IntelliSenseksi. Tämän toiminnon ansiosta monen komponentin ohjelmoiminen oli mielestäni hyvin suoraviivaista ja

nopeaa. Se mahdollisti sen, että pystyimme keskittymään enemmän itse verkkokaupan kehittämiseen, eikä työtunneista kulunut isoa osaa esimerkiksi dokumentaation lukemiseen. Välillä käyttöliittymän kehittäminen tuntui humoristisesti jopa liian helpolta, kun virheisiin ei annettu edes mahdollisuutta.

4.3 Perpendicularin laajentaminen

Vaikka Perpendicular tarjosikin meille erittäin kattavan kirjaston palveluita, tuli eteemme tilanteita, jolloin meidän oli järkevää laajentaa Perpendicularin tarjoamaa palvelua tavalla tai toisella. Näihin tilanteisiin lukeutui esimerkiksi hetket, jolloin huomasimme toistavamme tiettyä koodia komponenttien logiikassa aivan liikaa tai tiesimme, että tiettyä logiikkaa tullaan tarvitsemaan vielä useasti. Tällöin päätimme, että on helpompaa implementoida kyseinen logiikka suoraan palvelun tasolle, jolloin mahdollisia muutoksia ei ole tarpeellista suorittaa moneen eri komponenttiin.

Konkreettisena esimerkkitapauksena tällaisesta tilanteesta käytän laajennettua Cart-palvelua, joka suomennoksensa mukaisesti siis hoitaa kaikki ostoskoriin liittyvät asiat. Oletuksena kyseinen palvelu kyllä tarjosi kaikki tarvittavat toiminnot, mutta halusimme lisätä siihen logiikkaa varmistaaksemme verkkosivujen toiminnan ja parantaaksemme käyttäjäkokemusta.

Asiakkaan toiveesta organisaation käyttäjien tuli pystyä lisäämään kerralla monta tuotetta ostoskoriin esimerkiksi vanhasta tilauksesta tai käyttäjän itse luomasta ostoslistasta. Ongelmaksi muodostui nopeasti se, että tarpeeksi monen tuotteen epäsynkroninen lisääminen kerralla ostoskoriin ei toiminutkaan odotetusti, ja tiettyjä ehtoja olikin syytä tarkistaa. Tämän takia päädyimme laajentamaan Perpendicularin alkuperäistä Cart-palvelua.

Palveluiden laajentaminen tapahtuu myös helposti. Koska Angular käyttää TypeScript-ohjelmointikieltä ja palvelut itse asiassa ovat luokkia, voidaan niitä laajentaa normaalin TypeScript-luokan tapaan (Microsoft 2019). Oman palvelun tehtyämme sen käskettiin vain yksinkertaisesti laajentaa Perpendicularin oma Cart-palvelu kuvan 6 kuvailemalla tavalla.

```
export class CustomCartService extends CartService {
```

KUVA 6: Luokan laajentaminen

Mahdollistaaksemme laajennetun palvelun helpon käytön tuli meidän kertoa laajennetusta palvelusta myös Angularille. Käytännössä tämä tarkoitti sitä, että Angularia käsketään tarjoamaan juuri luomaamme laajennettua palvelua alkuperäisen palvelun sijasta. Moduuliin, jonka sisällä Cart-palvelu sijaitsi, lisättiin kuvan 7 komento.

```
@NgModule({
  imports: [ProvidersFactoryModule],
  providers: [
    { provide: ICartService, useClass: CustomCartService },
  ],
})
```

KUVA 7: Laajennetun palvelun tarjoaminen.

Tämän jälkeen Angular pystyi tarjoamaan muokkaamaamme palvelua aina, kun oli tarvetta käyttää Cart-palvelua (Kasagoni 2017). Tämän ansiosta meidän kehittäjien ei ollut tarvetta muistaa, tuleeko meidän käyttää alkuperäistä vai laajennettua palvelua. Tämä takasi sen, että käytimme aina oikeaa versiota tietystä toiminnosta.

Tämän jälkeen lisäsimme oman `addMultipleItemsToCart` -toimintomme, joka näkyy kuvassa 8.

```
/**
 * Custom function to add multiple items to cart
 * @param items array of the items that needs to be added.
 */
public addMultipleItemsToCart(items: any[], hideMsg?: boolean) {
  items
    .reduce((promise, item) => {
      return promise.then(() => {
        return this.cartProvider.addToCart([item.productCode], [item.quantity]);
      });
    }, Promise.resolve().then())
    .then(() => {
      const thePromise = Promise.resolve(null);
      const msg = new UIMessage(UIMessageType.SUCCESS, '_ORDER_TO_CART_SUCCESS', null, {
        cartName: (this._lastCart as CustomCart).description,
        hideMsg,
      });
      this.reloadAfterServiceCall(thePromise, msg);
    })
    .catch(() => {
      const thePromise = Promise.resolve(null);
      const msg = new UIMessage(UIMessageType.SUCCESS, '_ORDER_TO_CART_ERROR', null, {
        cartName: (this._lastCart as CustomCart).description,
        hideMsg,
      });
      this.reloadAfterServiceCall(thePromise, msg);
    });
}
```

KUVA 8: Laajennetun palvelun itse kirjoitettu toiminto.

Toiminnossa teimme jokaisesta tuotteen lisäyksestä ensin niin kutsutun lupauksen, jotta pystyimme lisäämään tuotteet koriin synkronisesti yksitellen käyttäen Perpendicularin Cart-palvelun tarjoamaa `addToCart`-toimintoa. Kun jokainen lupaus oli täytetty ja tuotteet oli lisätty koriin hallitusti, ilmoitimme siitä käyttäjälle.

4.4 Vastausten käsittely

REST-rajapinnan kanssa työskenneltäessä hyvin olennainen asia on tietysti rajapinnalta saatujen vastausten käsittely. Yleisimmin käytössä oleva protokolla REST-rajapinnan kanssa työskenneltäessä on lähettää vastaukset JSON-formaatissa. Tätä protokollaa noudattaa myös Commerce. Kun käyttöliittymäkehittäjä haluaa saada käyttöönsä esimerkiksi verkkokaupan käyttäjän tiedot, hän pyytää niitä Profile-palvelulta. Kuten aikaisemmin kerroin, palvelukerros välittää pyynnön välittäjäkerrokselle, joka tekee itse rajapintakutsun. Välittäjäkerros lo-

pulta saa liitteen 2 mukaisen vastauksen, ja se välitetään Perpendicularin tehdaskerrokselle, joka rakentaa vastauksesta Profile-luokan mukaisen TypeScript-objektin.

Rajapinnasta riippuen rajapinnalta saatu vastaus voi olla välillä sekava tai sen attribuutit voivat olla huonosti nimettyjä. Liitteen 2 esimerkissä attribuutit itsessään ovat hyvin nimettyjä, mutta vastaus kokonaisuutena on melko pitkä ja epäselvä. Tehdaskerroksen käsittelyn jälkeen vastaus on muotoutunut liitteen 3 mukaiseksi. Vertailemalla liitteitä 2 ja 3 näkee, kuinka paljon selkeämpi tehtaan luoma TypeScript-objekti on. Koska kyseessä on TypeScript-objekti, antoi tekstieditorimme käyttöömme listan objektin attribuuteista kuvan 4 havainnollistamalla tavalla.

Tehdaskerrokseen on myös mahdollisuus itse lisätä tarvittavaa logiikkaa laajentamalla sitä palvelukerroksen tapaan. Tämä mahdollisti meille esimerkiksi kahden attribuutin arvon yhdistämisen yhdeksi attribuutiksi. Vastauksena saatu `firstName` ja `lastName` olisi ollut mahdollista yhdistää attribuutiksi `fullName`. Myös tuotteen tiedoista löytyvästä kuvasta pystyimme tehdaskerroksessa muodostamaan pienempiresoluutioisen kuvan, jota saimme käytettyä esikatselukuvina.

5 PROJEKTIN YHTEENVETO

Solteq on yritys, jolla on pitkä historia. Sen koko olemassaolon aikana on moni ohjelmointikieli syntynyt ja kuihtunut pois, mutta kehittäjät ovat mukautuneet tilanteen vaatimalla tavalla. Useammat kollegoistani olivat tehneet työhistoriansa aikana monia verkkokauppoja suoraan Commercen päälle – ilman minkäänlaisia Perpendicularin tyylisiä palvelukirjastoja. Loogisesti tämä tarkoitti sitä, että heillä on syvällinen tietämys ja tuntemus siitä, miten Commercen kanssa toimitaan.

Aloittaessani oman urani Solteqissa, minun ensimmäinen asiakasprojektini oli opinnäytetyössäni esittelemä asiakas. Se oli projekti, jossa käytettiin moderneja ja tehokkaita työvälineitä, joista moni kollegoistani oli aikaisemmin vain uneksinut. Koko kehitystiimillemme se oli myös ensimmäinen kerta, kun pääsimme käyttämään Perpendicularia, joten siinä mielessä olimme kaikki samalla viivalla.

Uutena harjoittelijana olin erittäin tyytyväinen ja tunsin oloni onnekkaaksi. Perpendicularin käyttäminen oli minulle helppoa, ja resursseja jäi muiden, vaikeampien asioiden oppimiseen.

5.1 Omat kokemukset

Kuten ilmaisin, Perpendicularin tarjoaman palvelukirjaston käyttö oli uuden kehittäjän näkökulmasta erittäin helppoa. Sillä oli kuitenkin omat ongelmansa, jotka kohtasin ennen pitkää projektin edetessä. Asia, jonka ensimmäisenä huomasin, oli Perpendicularin palvelukerroksen abstrahoinnin mukana tulleet ongelmat. Abstrahointi tarkoittaa sitä, että yritetään tehdä monimutkaisesta asiasta yksinkertainen, pelkistetty. Käytännössä tämä tarkoitti Perpendicularissa juuri sitä, että kaikki verkkokaupan kanssa tehtävä keskustelu oli purettu helposti käytettäviin palveluihin ja toimintoihin, kuten aikaisemmissa kappaleissa olen läpi käynyt. Tämän seurauksena saadakseni komponentin tekemään haluamani toiminnot minulle riitti yleensä se, että löysin siihen toimintoon liittyvän palvelun ja käytin palvelun tarjoamia valmiita toimintoja. Minun ei siis niin sanottujen kulissien taakse tarvinnut katsoa, ja se muodostui projektin edetessä ongelmaksi.

Tämä korostui erityisesti keskustellessani muiden kehitystiimin jäsenten kanssa esimerkiksi ongelmista tai uusista toiminnallisuuksista. Tällöin kollegani eivät välttämättä edes ymmärtäneet, että minä tiimin uusimpana tulokkaana en välttämättä tiennytkään, kuinka tekniset asiat oikeasti toimivat. En kuitenkaan sano, että edellä kuvailemani ongelma olisi ollut mitenkään vakava asia. Asiakkaan näkökulmasta se oli erittäin hyvä asia, sillä kokematonkin kehittäjä pääsi hyvään vauhtiin jo heti projektin alkuvaiheessa ja konkreettista tulosta syntyi nopeasti. Asia lähinnä vaivasi vain itseäni.

5.2 Perpendicularin mahdolliset käyttötapaukset

On täysin selvää, että jokaiseen projektiin väliohjelmisto isoine palvelukirjastoineen ei sovi. Jos kyseessä on pieni ja kiireinen projekti, en itse näe väliohjelmiston käyttöä järkevänä vaihtoehtona. Kun taas valmista kirjastoa ei vielä ole tarjolla, tulee tarkoin harkita, onko järkevää lähteä sellaista rakentamaan. Jos kuitenkin yritys rakentaa verkkokauppoja asiakkailleen toistuvasti ja käytettävät teknologiat pysyvät samoina, kannattaa asiaa ilman muuta harkita. Kun huomaa toistavansa samoja rajapintakutsuja päivästä toiseen useamman vuoden ajan, kannattaa miettiä, voisiko kuvailemastani väliohjelmistosta olla apua.

Jos on tiedossa, että samalla viitekehyksellä tullaan rakentamaan tulevina vuosina useampia isoja verkkokauppoja, on mielestäni erittäin perusteltua aloittaa väliohjelmiston rakentaminen. Sen rakentaminen itsessään onnistuu käytännössä usealla eri tavalla, sillä sitä voi rakentaa asiakasprojektin yhteydessä niin, että kirjasto kasvaa hiljalleen yhden tai useamman projektin myötä tarpeiden mukaan. Joka kerta kun uudelle kutsulle on tarvetta, se lisätään väliohjelmiston palvelukirjastoon tulevaa käyttöä varten.

Mielestäni hyvä vaihtoehto tälle on myös lähteä aktiivisesti rakentamaan väliohjelmistoa niin, että yritys kohdentaa työntekijöiden tunteja ohjelmiston kehittämiseen. Se on myös oivallinen tilaisuus hyödyntää mahdollisia harjoittelijoita antamalla heille oikeita työtehtäviä.

Hyvin suunniteltu väliohjelmisto muovautuu pienellä työmäärällä myös toisiin toimimaan muiden rajapintojen kanssa, ja tällöin käyttöliittymäkehittäjän ei välttämättä edes tarvitse välittää siitä, mitä rajapintaa vasten kehitetään verkkokauppaa. Kun yrityksen itse kehittämän väliohjelmiston yhdistää vaikkapa aikaisempien projektien sivutuotoksena syntyneeseen komponenttikirjastoon, on yrityksellä hallussa todella hyvä myyntivaltti. Tällöin esimerkiksi uudelle asiakkaalle prototyyppikaupan valmistaminen ja esitteleminen on erittäin nopeaa. Suurin ja aikaa vievin osuus projektissa on tällöin vain kaupan kustomointi asiakkaan toiveiden mukaan. Verkkokaupat ovat kuitenkin melko standardisoituja, sillä käyttäjät olettavat tiettyjen toimintojen löytyvän jokaisesta kaupasta. Jos verkkokauppa poikkeaa oletetusta, se saattaa näkyä asiakaskatona.

5.3 Huomioon otettavaa

Jotta väliohjelmistoa voi käyttää mahdollisimman tehokkaasti, tulisi sen olla oma irrallinen pakettinsa. Paketilla tarkoitetaan sitä, että se on täysin oma projektinsa ja sen lähdekoodi on irrallaan muusta projektista. Tällöin sen voi yhdistää tarvittaviin projekteihin riippuvuutena, jolloin kaikki välikerrokseen tehdyt päivitykset jakautuvat myös sitä käyttäviin projekteihin versionhallintajärjestelmien kautta. Helppokäyttöisyyden lisäksi tämä ratkaisu lisää myös väliohjelmiston tietoturvaa, sillä vanhat projektit on nopeasti päivitetty ja mahdolliset aukot tietoturvassa on korjattu.

Riippuvuuden tarjoamien hyötyjen edellytys on tietysti se, että pakettia oikeasti päivitetään. Projekteissa huomatut puutteet tai virheet tulee aina korjata suoraan väliohjelmistoon, eikä missään nimessä jokaiseen projektiin erikseen. Puuttuvien ominaisuuksien kohdalla tulee aina miettiä, ovatko ne mahdollisesti tarpeellisia myös tulevissa projekteissa vai pelkästään sen hetkisessä projektissa. Tulevia projekteja avustavat ominaisuudet tulee tietysti lisätä suoraan väliohjelmistoon. Jos väliohjelmisto on esimerkiksi yrityksessä monessa projektissa käytössä tai sitä suunnitellaan käytettäväksi lähitulevaisuudessa yhdessä tai useammassa projektissa, on erityistä syytä muistaa ylläpitää väliohjelmistoa eikä päästää teknistä velkaa kertymään.

Väliohjelmisto tulisi mielestäni aina rakentaa tulevaisuus mielessä, jolloin on tärkeää miettiä tarkkaan toteutettava infrastruktuuri. Perpendicularin esitellyt kerrokset ovat vain esimerkki siitä, miten väliohjelmiston voi rakentaa. Tärkein asia kuitenkin on mielestäni se, että käyttöliittymäkehittäjän käyttämä kerros on abstrahoitu. Tällöin väliohjelmiston yhdistäminen muihin rajapintoihin on mahdollisimman helppoa ja käyttöliittymäkehittäjä voi siirtyä projektista toiseen vaivattomasti.

6 POHDINTA

Opinnäytetyön tarkoituksena oli selkeyttää lukijalle väliohjelmiston käyttöä. Tuloksena oli opastava dokumentti, jota voi hyödyntää uuden ohjelmointiarkkitehtuurisen mallin oppimiseen ja perehdyttämiseen. Työssä käydään läpi väliohjelmisto konseptina ja arkkitehtuurisena ratkaisuna mielestäni perusteellisesti. Sen tarjoamat hyödyt tuodaan esiin ja mahdollisia käyttötapauksia esitellään. Tämän avulla esimerkiksi yrityksen tai kehitystiimin on mahdollista pohtia väliohjelmiston rakentamista ja käyttöönottoa.

Jälkikäteen ajatellen opinnäytetyön aiheen olisi voinut rajata vielä tarkemmin, jolloin aiheeseen olisi ollut mahdollisuus syventyä enemmän. Työn luotettavuuden nostattamiseksi useamman lähteen käyttö olisi ollut suotavaa. Mielestäni opinnäytetyö kuitenkin onnistui hyvin. Sitä tehdessä sain syvennettyä osaamistani vielä entisestään front-endin, back-endin ja ohjelmistoarkkitehtuurin osalta.

Mahdollisena jatkotutkimuskohteena on ainakin väliohjelmiston laajempi käyttötapauskartoitus. Olisi mielenkiintoista tutkia, minkälaisissa tapauksissa kyseisestä arkkitehtuurista olisi hyötyä.

LÄHTEET

Arora, C., Hennessy, K., Noring, C., Uluca, D. 2018. Building Large-Scale Web Applications with Angular. O'Reilly Media.

Elliott, E. 2019. Top JavaScript Frameworks and Topics to Learn in 2019. Medium. Luettu 26.11.2019. https://medium.com/javascript-scene/top-javascript-frameworks-and-topics-to-learn-in-2019-b4142f38df20

Google, Angular Material komponenttien kustomoinnin -dokumentaatio. Luettu 10.10.2019.https://material.angular.io/guide/customizing-component-styles

IBM, WebSphere Commerce version 7.0 korirajapinnan dokumentaatio. Luettu 15.9.2019b. https://www.ibm.com/support/knowledgecenter/en/SSZLC2_7.0.0/com.ibm.commerce.startersto-res.doc/code/rsm_cart_fep8.htm#addOrderItem

IBM, WebSphere Commerce version 9.0 yleiskatsaus. Luettu 2.10.2019a. https://www.ibm.com/support/knowledgecenter/en/SSZLC2_9.0.0/com.ibm.commerce.admin.doc/concepts/covoverall.htm

Kasagoni, S. 2017. Building Modern Web Applications using Angular, 1st edition. O'Reilly Media.

Microsoft, TypeScript luokkien dokumentaatio. Luettu 16.9.2019. https://www.typescriptlang.org/docs/handbook/classes.html

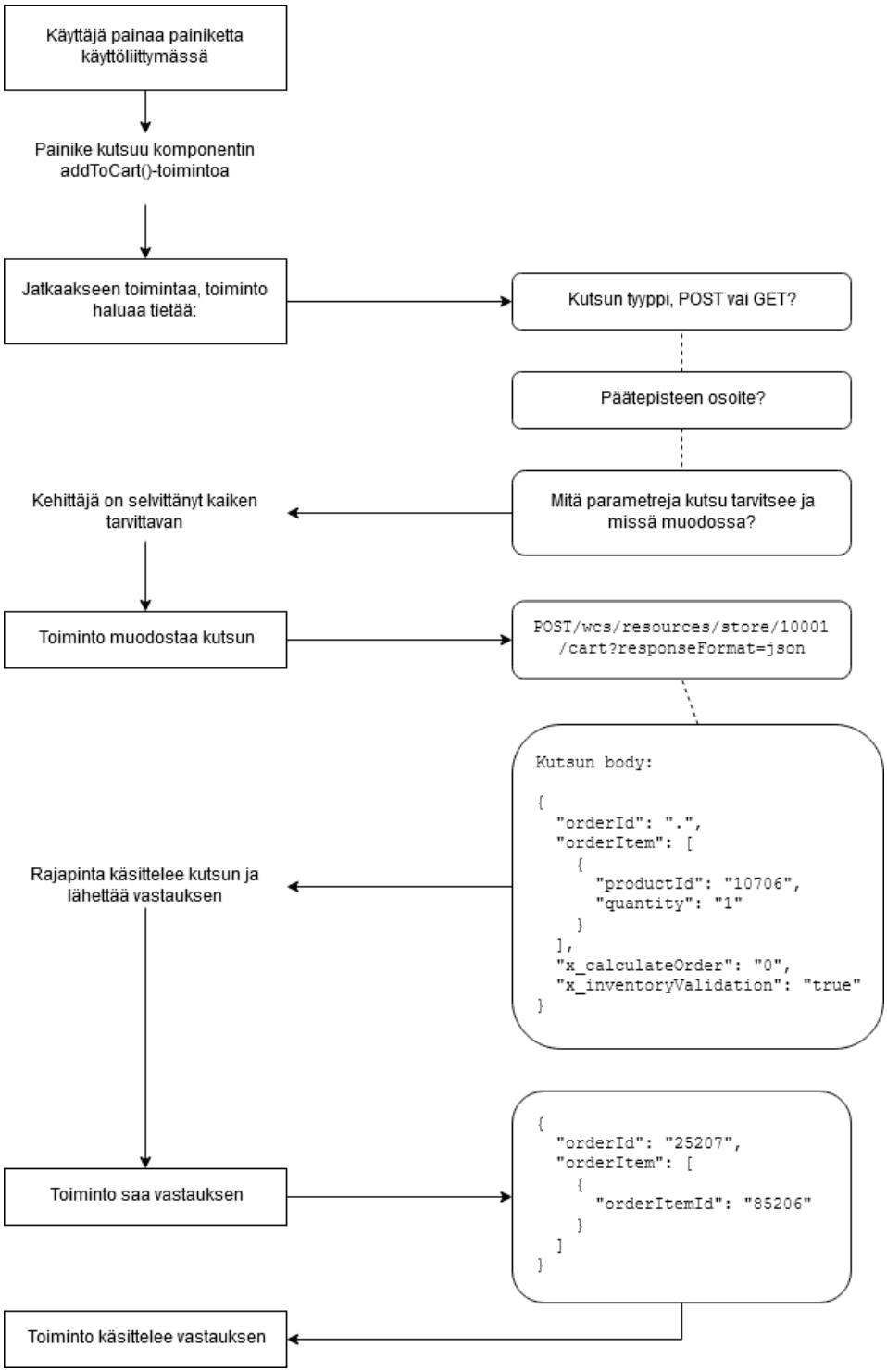
Moss, B. 2019. Why Don't We Just Use Material Design? Webdesigner Depot. Luettu 26.11.2019. https://www.webdesignerdepot.com/2019/04/why-dont-we-just-use-material-design/

Nowak, M. 2019. Choosing a Technology Stack for Your Web Application Development. Monterail. Luettu 26.11.2019. https://www.monterail.com/blog/web-development-technology-stack

Stack Overflow, Developer Survey Results 2019. Luettu 26.11.2019. https://insights.stackoverflow.com/survey/2019#technology-_-web-frameworks

LIITTEET

Liite 1. Logiikan kulkukaavio tuotteen lisäämisestä ostoskoriin.



Liite 2. Commerce rajapinnalta saatu JSON-tiedosto.


```
{
  "householdSize": "0",
  "organizationDistinguishedName": "o=default organization,o=root organization",
  "registrationStatus": "RegisteredPerson",
  "lastUpdate": "2019-09-12T08:53:59.268Z",
  "receiveSMSNotification": "false",
  "maritalStatus": "Other",
  "phone1": "05012345670",
  "contact": [
    {
      "addressLine": ["Itäinenkatu 11", "", ""],
      "lastName": "Vormisto",
      "nickName": "Billing address",
      "zipCode": "33210",
      "primary": "false",
      "addressType": "ShippingAndBilling",
      "firstName": "Mika",
      "phone1": "12 12 12 12",
      "email1": "mika.vormisto@solteq.com",
      "addressId": "26221186310",
      "country": "Finland",
      "city": "Tampere"
    },
    {
      "addressLine": ["Itäinenkatu 11", "", ""],
      "lastName": "Vormisto",
      "nickName": "Primary Address",
      "zipCode": "33100",
      "primary": "true",
      "addressType": "ShippingAndBilling",
      "firstName": "Mika",
      "phone1": "05012345670",
      "email1": "mika.vormisto@solteq.com",
      "addressId": "19403388430",
      "city": "Tampere"
    }
  ],
  "addressId": "19402549899",
  "registrationDateTime": "2019-08-13T05:00:26.252Z",
  "resourceId": "",
  "contextAttribute": [
    {
      "attributeName": "marketingTrackingConsentTimestamp",
      "attributeValue": [
        {
          "value": ["2019-09-12 08:53:58.881"],
          "storeId": "10201"
        }
      ]
    },
    {
      "attributeName": "marketingTrackingConsent",
      "attributeValue": [
        {
          "value": ["0"],
          "storeId": "10201"
        }
      ]
    },
    {
      "attributeName": "privacyNoticeTimestamp",
      "attributeValue": [
        {
          "value": ["2019-09-12 08:53:59.339"],
          "storeId": "10201"
        }
      ]
    },
    {
      "attributeName": "privacyNoticeVersion",
      "attributeValue": [
        {
          "value": ["1"],
          "storeId": "10201"
        }
      ]
    }
  ],
  "nickName": "mika.vormisto@solteq.com",
  "orgizationId": "-2000",
  "accountStatus": "Enabled",
  "description": "-",
  "userId": "1000048006",
  "primary": "false",
  "gender": "Male",
  "firstName": "Mika",
  "distinguishedName": "uid=mika.vormisto@solteq.com,o=default organization,o=root organization",
  "resourceName": "person",
  "preferredCurrency": "DKK",
  "lastName": "Vormisto",
  "challengeQuestion": "-",
  "passwordExpired": "false",
  "addressType": "ShippingAndBilling",
  "email1": "mika.vormisto@solteq.com",
  "profileType": "Consumer",
  "registrationApprovalStatus": "Approved",
  "numberOfChildren": "0",
  "logonId": "mika.vormisto@solteq.com"
}
```

Liite 3. Perpendicularin Profile-luokka.


```
{
  "marketingConsent": false,
  "addressBook": [
    {
      "isShippingAddress": true,
      "isBillingAddress": true,
      "nickName": "Billing address",
      "address1": "Itäinenkatu 11",
      "address2": "",
      "firstName": "Mika",
      "lastName": "Vormisto",
      "email": "mika.vormisto@solteq.com",
      "phone": "12 12 12 12",
      "zip": "3310",
      "city": "Tampere",
      "country": "Suomifinland",
      "id": "26221186310",
      "isPrimary": false,
      "ownerId": ""
    },
    {
      "isShippingAddress": true,
      "isBillingAddress": true,
      "nickName": "Primary Address",
      "address1": "Itäinenkatu 11",
      "address2": "",
      "firstName": "Mika",
      "lastName": "Vormisto",
      "email": "mika.vormisto@solteq.com",
      "phone": "05012345670",
      "zip": "33210",
      "city": "Tampere",
      "id": "19403388430",
      "isPrimary": true,
      "ownerId": ""
    }
  ],
  "primaryAddress": {
    "isShippingAddress": true,
    "isBillingAddress": true,
    "nickName": "Primary Address",
    "address1": "Itäinenkatu 11",
    "address2": "",
    "firstName": "Mika",
    "lastName": "Vormisto",
    "email": "mika.vormisto@solteq.com",
    "phone": "05012345670",
    "zip": "33210",
    "city": "Tampere",
    "id": "19403388430",
    "isPrimary": true,
    "ownerId": ""
  },
  "attributes": {},
  "salutation": "",
  "businessTitle": "",
  "firstName": "Mika",
  "middleName": "",
  "lastName": "Vormisto",
  "phone": "05012345670",
  "email": "mika.vormisto@solteq.com",
  "gender": "Male",
  "maritalStatus": "Other",
  "householdSize": 0,
  "numberOfChildren": 0,
  "photo": "",
  "marketingConsentLastUpdated": "2019-09-12T05:53:58.881Z",
  "privacyPolicyAcceptLastUpdated": "2019-09-12T05:53:59.339Z",
  "privacyPolicyAcceptVersion": "1",
  "parentOrganizationId": "-2000",
  "userId": "1000048006"
}
```