Tampereen ammattikorkeakoulu

Maksunvälittäjän integrointi PrestaShop-verkkokauppa-alustaan

Sauli Maijala

OPINNÄYTETYÖ
Toukokuu 2020

Tietojenkäsittely
Web-palvelut

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Web-palvelut

MAIJALA, SAULI:
Maksunvälittäjän integrointi PrestaShop-verkkokauppa-alustaan

Opinnäytetyö 26 sivua
Toukokuu 2020

Opinnäytetyön tarkoituksena oli kehittää PrestaShop-moduuli, joka mahdollistaa maksunvälittäjä Avardan maksupalveluiden käytön PrestaShop-verkkokaupassa. Työn toimeksiantaja oli suomalainen verkkokaupparatkaisuja tarjoava Vilkas Group Oy. Työn tavoitteena oli tuoda toimeksiantajayritykseen PrestaShop-osaamista ja sitä kautta kehittää liiketoimintaa.

Opinnäytetyössä käsiteltiin PrestaShopin toimintaa esimerkiksi MVC-arkkitehtuurin kautta. Työn pääpaino oli moduulin kehitysprosessi -osiossa, jossa käsiteltiin kehitysprosessin lisäksi prosessin aikana ilmenneitä ongelmia ja ongelmien ratkaisukeinoja. Lisäksi käsiteltiin ohjelmistotestaamista, moduulijulkaisujen automatisointiprosessia ja tiedossa olevia ongelmia ja kehityskohteita.

Työn tuloksena syntyi PrestaShopin lisäosakaupassa julkaistava moduuli, ja sen ohella toimiva jatkuva integraatio -prosessi, joka helpottaa yrityksen PrestaShop-moduulikehitystä myös tulevaisuudessa. Moduulin kehitys jatkuu edelleen työssä esiteltyjen kehityskohteiden parissa. Tulevaisuuden kehityskohteita ovat esimerkiksi selkeämmät käännökset ja lokiviestit.

Asiasanat: verkkokauppa, maksunvälittäjä, prestashop, avarda

ABSTRACT

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

MAIJALA, SAULI:
Integration of a Payment Processor into the PrestaShop E-Commerce Platform

Bachelor's thesis 26 pages
May 2020

The purpose of this thesis was to develop a PrestaShop module that makes it possible to utilize the payment services of the payment processor Avarda in a PrestaShop online store. This thesis was commissioned by Vilkas Group Oy, a Finnish company, which provides e-commerce solutions. The purpose of this thesis was to bring PrestaShop expertise into the company.

The operation of PrestaShop is discussed, for example through the MVC architecture. The main focus of this thesis is in the section that explains the development process of the module, which included not only the development process itself, but also the problems encountered during the process and the means through which they were solved. In addition, software testing, the automation process of module releases, known problems and areas of development are discussed.

The result was a PrestaShop module that will be released on the PrestaShop Addons Marketplace, and a continuous integration process, which will aid the company’s future PrestaShop module development. The development of the module is ongoing amongst the areas of development presented in the thesis. The areas of development include clearer Finnish translations and log messages.

Key words: e-commerce, payment processor, prestashop, avarda

SISÄLLYS

LYHENTEET JA TERMIT

CRUD	Create, Read, Update, Delete: tietokantojen perustoiminnot (lisää, lue, päivitä, poista)
GitHub	Lähdekoodin hallinta- ja jakopalvelu git-versionhallintajärjestelmälle
JSON	JavaScript Object Notation, kieliriippumaton tieto- ja tiedostomuoto
Lokitus	Tapahtumalokin pitäminen (engl. logging)
Maksunvälittäjä	Tarjoaa verkkokaupan maksujärjestelmän, yhdistää erilaisia maksurajapintoja yhdeksi
PHP	Ohjelmointikieli, jota käytetään erityisesti web-kehityksessä
PhpStorm	Ohjelmointiympäristö PHP-ohjelmointikielelle
Symfony	PHP-ohjelmointikielellä kirjoitettu ohjelmistokehys ja joukko uudelleenkäytettäviä komponentteja
Taikanumero	Arvo, jolla on tarkoitus, jonka lähde ei käy suoraan ilmi ja jonka voisi korvata nimetyllä muuttujalla
Virheenjäljitin	Ohjelma, jota käytetään ohjelmointivirheiden selvittämiseen

1 JOHDANTO

Opinnäytetyön taustalla on toimeksiantajan suomalaisen verkkokaupparatkaisuja tarjoavan Vilkas Group Oy:n (jäljempänä ”toimeksiantajayritys”) ja pohjoismaiseen TF Bank AB:hen kuuluvan maksunvälittäjän Avardan (jäljempänä ”Avarda”) keskeinen projekti, jossa Avardan maksupalvelut integroidaan Vilkkaan käyttämiin verkkokauppa-alustoihin.

Tämä opinnäytetyö liittyy kyseisen projektin PrestaShop-osuuteen. PrestaShop on PHP-ohjelmointikielellä kirjoitettu avoimen lähdekoodin verkkokauppa-alusta. PrestaShop on toimeksiantajayrityksessä uusi verkkokauppa-alusta, ja tämä opinnäytetyöhön liittyvä projekti on ensimmäinen moduuli, joka PrestaShop-alustalle on yrityksessä kirjoitettu. Opinnäytetyön tavoitteena onkin kartoittaa PrestaShop-moduulikehitystä, ja sitä kautta tuoda yritykseen PrestaShop-osaamista. PrestaShop ei ollut itsellenikään tuttu alusta ennen projektin alkamista.

Opinnäytetyössä esitellään PrestaShopin toimintaa, moduulin kehitysprosessia ja kehitysprosessin aikana ilmenneitä haasteita. Opinnäytetyössä kerrotaan myös, miten esimerkiksi ohjelmistotestaaminen ja jatkuva integraatio onnistuivat PrestaShop-moduulin kanssa.

2 PRESTASHOPIN TOIMINTA

2.1 MVC-arkkitehtuuri

MVC-arkkitehtuuri (*Model-view-controller*, eli malli-näkymä-käsittelijä) on ohjelmistoarkkitehtuurimalli, jossa käyttöliittymän vuorovaikutus jaetaan kolmeen erilliseen rooliin (Fowler 2003, 330). PrestaShopin rakenne perustuu MVC-arkkitehtuuriin (Borowicz 2019), ja se on sitä kautta vahvasti läsnä myös moduulikehityksessä.

MVC-arkkitehtuurissa malli (engl. model) on objekti, joka edustaa osaa ohjelmiston tietorakenteesta. Näkymä (engl. view) edustaa mallia käyttöliittymässä, ja se voi olla esimerkiksi HTML-sivu, jossa on tietoa mallista. Käsittelijä (engl. controller) ottaa vastaan käyttäjän syötteen, muokkaa mallia syötteen mukaan ja sitä kautta aiheuttaa näkymän päivityksen. (Fowler 2003, 330.)

Tässä opinnäytetyössä käsitellään MVC-arkkitehtuurin osista lähinnä käsittelijöitä, sillä käsittelijöiden ohjelmointiin käytettiin huomattavasti enemmän aikaa kuin MVC-arkkitehtuurin muihin osiin. Moduulissa on mukana jopa sellaisia käsittelijöitä, jotka eivät sisällä mitään näkymää, vaan esimerkiksi ohjaavat käyttäjän heti johonkin toiseen käsittelijään.

2.2 Koukut

Koukku (engl. hook) on ohjelmistoon sisältyvä ominaisuus, jonka avulla ohjelmoijat voivat lisätä omia ominaisuuksiaan ohjelmistoon (Nguyen 2004). Myös PrestaShop tarjoaa koukkuja, joiden avulla voidaan laajentaa sen toiminnallisuutta, ja iso osa PrestaShopin moduulikehityksestä tapahtuukin juuri koukkujen avulla.

PrestaShop tarjoaa kahdenlaisia koukkuja: ”toimintokoukkuja” ja ”esityskoukkuja”. Toimintokoukkuja käytetään koodin ajamiseen tietyissä olosuhteissa (esi-

merkiksi tunnistamaan tilauksen tilan muutos, jotta voidaan lähettää sähköpostiviesti asiakkaalle tilauksen yhteydessä). Esityskoukkuja käytetään lisäämään sisältöä sivulle. (PrestaShop 2020a.)

Koukkuja käytetään PrestaShopissa siten, että ensin halutut koukut rekisteröidään *Module::registerHook*-funktiolla, jonka kutsu lisätään yleensä moduulin asennusfunktioon. Jokaista rekisteröityä koukkua kohden täytyy myös luoda julkinen käsittelijämetodi, joka nimetään koukun mukaan (hook-etuliite, jonka jälkeen koukun nimi). Metodi saa yhden ainoan argumentin, eli listan, joka sisältää sille koukun ajon yhteydessä annetut parametrit. (PrestaShop 2020a.)

Tässä opinnäytetyössä käytetään lähinnä toimintokoukkuja, ja niiden sisältä löytyykin aika paljon koodia. Yhtä esityskoukkua käytetään tilaussivun tilaustietolaatikon näyttämiseen. Myös toinen esityskoukku on rekisteröitynä, mutta sitä käytetään käytännössä kuin toimintokoukkua, joka ajetaan jokaisen tilaussivun sivunlatauksen yhteydessä.

3 MODUULIN RAKENTAMINEN

3.1 Moduuligeneraattori

PrestaShop-moduulit vaativat tietynlaisen rakenteen toimiakseen. Ajan säästämiseksi moduulille luotiin valmis ”luuranko” käyttäen PrestaShopin tarjoamaa moduuligeneraattoria. Moduuligeneraattori luo kaikki tarvittavat tiedostot sille, että PrestaShop tunnistaa paketin moduuliksi, mutta moduuli ei vielä varsinaisesti tee mitään. Kyseisen moduulin mukana tulee myös joitain esimerkkitiedostoja aloittelevan kehittäjän avuksi.

Generaattorin luoman luurangon mukana tulee jonkin verran ylimääräistä koodia, ja itse koodi on myös paikka paikoin vähän vanhentunutta PrestaShopin nykyisiin kehitysstandardeihin nähden, mutta sen käyttö säästää silti merkittävästi aikaa, vaikka koodin joutuukin käymään läpi ja päivittämään nykyisten standardien tasolle.

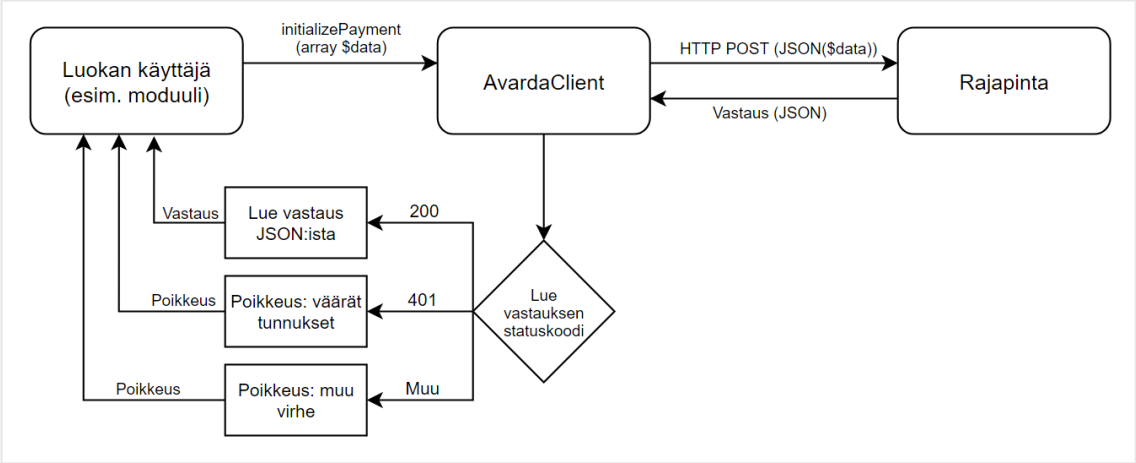
3.2 Rajapinta-asiakasohjelma

Toimeksiantajayrityksen neuvosta ohjelmoin moduulin ensimmäiseksi varsinaiseksi osaksi kevyen rajapinta-asiakasohjelman, jonka kautta hoidetaan kommunikaatio Avardan rajapinnan kanssa. Käytännössä asiakasohjelma on luokka, joka sisältää oman metodin jokaista rajapinnan tarjoamaa toimintoa varten.

Pääosin metodit vastaanottavat dataa moduulilta PHP-ohjelmointikielen ymmärtämässä muodossa (listamuuttuja), muuttavat sen Avardan rajapinnan ymmärtämään muotoon (JSON) ja lähettävät sen eteenpäin. Luokka ei sisällä mitään varsinaista virheidenkäsittelyä, vaan se jätetään luokan käyttäjälle. Myös rajapinnan palauttamat virheet välitetään poikkeuksien (engl. exception) mukana suoraan asiakasohjelman käyttäjälle. Luokan toimintaa on visualisoitu kuvassa 1.

Rajapintakutsujen tekemiseen luokassa käytetään Symfony-ohjelmistokehykseen kuuluvaa HttpClient-komponenttia. Komponentti valittiin käytettäväksi sen

helppokäyttöisyyden takia, ja sen takia, että PrestaShop on jo valmiiksi osittain Symfony-pohjainen. PrestaShopista löytyy jo valmiiksi toinen HTTP-pyyntöjen tekemiseen tarkoitettu asiakasohjelma, Guzzle, mutta PrestaShop käyttää siitä yhteensopivuusongelmien takia niin vanhaa versiota, että toisen ratkaisun käyttäminen tuntui mielekkäämmältä.



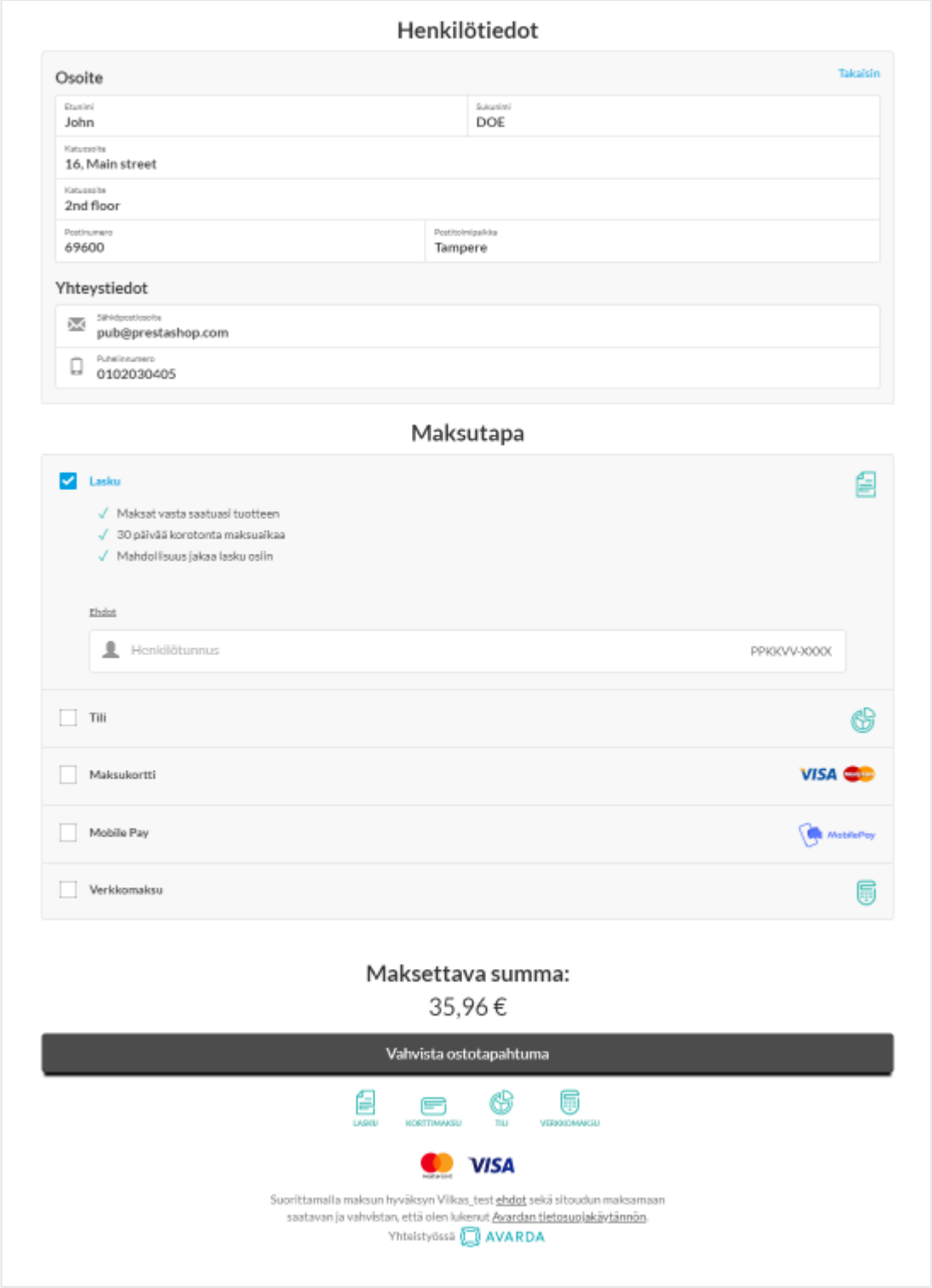
KUVA 1. Asiakasohjelman eli AvardaClient-luokan toiminta

3.3 Maksaminen

Jotta maksumoduuli on valittavissa tilausprosessin aikana, moduulin on rekisteröitävä *paymentOptions*-koukku, jonka tulee palauttaa yhdestä tai useammasta PaymentOption-luokan instanssista koostuva lista (PrestaShop 2020d). Yksi maksumoduuli voi siten rekisteröidä monta maksutapaa yhdellä kerralla.

Maksutavan toteuttamiseen on monta käytäntöä, mutta tämän moduulin kanssa päädyttiin aika yksinkertaiseen ratkaisuun. Kun käyttäjä valitsee maksutavakseen Avardan, hänet ohjataan uuteen käsittelijään. Käsittelijä muodostaa tilauksen tiedot Avardan tarvitsemassa muodossa ja lähettää ne rajapinnalle asiakasohjelman kautta. Rajapinta vastaa uudella maksutunnuksella, jolla tämä maksutapahtuma tunnistetaan jatkossa, mikäli tilaustiedot ovat kelvolliset. Tämän jälkeen käsittelijä näyttää sivun, johon avataan Avardan maksulomake (kuva 2). Maksulomaketta alustettaessa sille kerrotaan maksutunnus ja URL-osoite johon käyttäjä tulee ohjata maksuprosessin jälkeen.

Käyttäjä kulkee Avardan maksuprosessin läpi, ja maksaa tilauksen haluamallaan maksutavalla. Maksamisen jälkeen Avarda palauttaa käyttäjän moduulin aiemmin määrittelemään osoitteeseen.



KUVA 2. Avardan maksulomake

3.4 Maksun validointi

Maksutapahtumasta palatessaan käyttäjä ohjautuu validointikäsittelijään, jonka URL-osoite annettiin Avardan maksulomakkeelle alustuksen yhteydessä. Validointikäsittelijä käy kysymässä Avardan rajapinnalta maksun tilan maksutunnuksen perusteella. Mikäli maksu oli onnistunut, PrestaShop luo ostoskorista uuden tilauksen ja maksun tiedot tallennetaan tietokantaan. Tämän jälkeen käyttäjä ohjataan tilausvahvistussivulle. Mikäli maksu ei onnistunut, käyttäjälle näytetään virhesivu, josta voi palata takaisin kassasivulle ja yrittää maksamista uudelleen tai käyttää toista maksutapaa tilauksen viimeistelyyn.

Validointi on tehty mahdollisimman tietoturvalliseksi, koska moduuli käsittelee rahaliikennettä. Avardan rajapinnan palauttamaan tietoon luottaminen arveluttaa, koska rajapinta ei palauta mitään, jolla voitaisiin varmistaa, että rajapintakutsuun vastaava osapuoli on varmasti Avarda eli välissä ei ole kolmatta osapuolta, kuten mies välissä -hyökkäyksessä. Hyökkäyksen mahdollisuutta voidaan pitää kuitenkin vähäisenä, koska Avardan rajapinta käyttää HTTPS-protokollaa. HTTPS suojaa mies välissä -hyökkäystä vastaan (Google 2020).

3.5 Tilauksen viimeistely

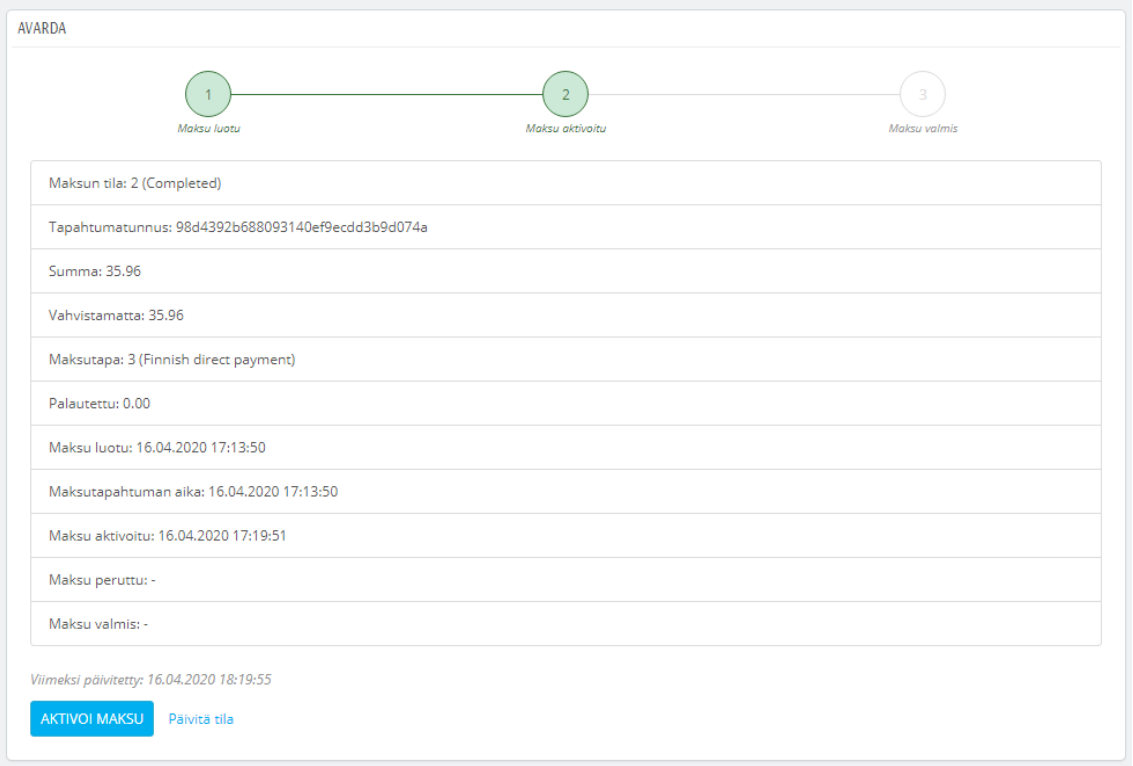
Kauppiaan näkökulmasta Avardan maksuprosessiin kuuluu oleellisesti myös tilausten viimeistely. Tämä tarkoittaa sitä, että rajapinnalle pitää lähettää erillinen kutsu, kun tilaus on lähetetty asiakkaalle. Vasta tilauksen viimeistelyn jälkeen Avarda siirtää rahat kauppiaan tilille.

Moduulissa tilausten viimeistely on toteutettu taustahallinnan tilausta koskevalle sivulle. Viimeistelykutsu lähtee rajapinnalle automaattisesti, kun tilaus merkitään lähetetyksi, mutta tämän saa halutessaan pois moduulin asetuksista. Viimeistelykutsun voi lähettää myös manuaalisesti erillisestä napista.

Oikean koukun löytäminen maksujen automaattista viimeistelyä varten oli haasteellista. Aluksi loogiselta valinnalta tuntui *actionOrderStatusPostUpdate*-koukku,

joka nimensä mukaisesti ajetaan tilauksen tilan muuttamisen jälkeen. Normaaliolosuhteissa tämä koukku oli ihan toimiva ratkaisu, mutta poikkeustilanteet, kuten aikakatkaisut rajapintapyyntöjä tehtäessä, johtivat siihen, että PrestaShopin oma koodi, joka on vastuussa koukun ajamisesta, keskeytyi kesken suorituksen. Tämä johti muun muassa siihen, että tilauksen tilan muuttuessa muutos ei välittynyt käyttöliittymälle, jolloin käyttöliittymä näytti tilan väärin.

Ongelman kiertämiseksi viimeistelyssä käytetään *actionOrderHistoryAddAfter*-koukkua, joka ajetaan sen jälkeen, kun tilauksen tilahistoriaan lisätään uusi kirjaus. Tämä tapahtuu käytännössä viimeisenä asiana tilaa vaihdettaessa, joten poikkeustilanteet eivät enää aiheuta sivuvaikutuksia.



KUVA 3. Avardan ”toimintopaneeli” tilauksien hallintasivulla

3.6 Tuotepalautukset

Maksujen viimeistelyn kaltaisesti myös tuotepalautukset on moduulissa vahvasti sidottu PrestaShopin omiin prosesseihin. Kun kauppias tekee palautuksia hallintaliittymässä, moduuli laskee palautettavan rahasumman ja tekee rajapintakutsut taustalla automaattisesti.

Tuotepalautuksia toteuttaessa koukkujen kanssa oli kuitenkin vielä huomattavasti enemmän ongelmia kuin maksujen viimeistelyn kanssa. PrestaShopin dokumentaatiosta löytyvän koukkulistauksen perusteella loogisin valinta koukuksi, jonka avulla tuotepalautustoiminnallisuus olisi voitu toteuttaa oli *actionProductCancel*. Kehityksen aikana koodia lukemalla kuitenkin ilmeni, että kyseinen koukku ajetaan palautuksia tehtäessä kerran jokaista tuoteriviä kohden. Nämä iteraatiot eivät tiedä toisistaan mitään, esimerkiksi että monesko iteraatio on menossa. Tämän takia koukku ei ollut kelvollinen tuotepalautusten toteuttamiseen, sillä tavoitteena oli se, että palautuksesta tehdään yksi rajapintakutsu, joka sisältää kaikki palautukseen liittyvät tiedot.

Toinen potentiaalinen koukku oli myös nimensä perusteella *actionOrderReturn*, mutta kokeilun perusteella se ei liity mitenkään taustahallinnan palautuksiin, vaan se ajetaan, kun asiakas tekee kaupan puolella palautuspyynnön. Kun tälläkään koukulla ei saatu haluttua tulosta, etsintä oli kohdistettava suoraan PrestaShopin ohjelmakoodiin.

Palautusta vastaavaa koodia tiedostossa AdminOrdersController.php tutkimalla löytyi vielä yksi potentiaalinen koukku, *actionOrderSlipAdd*, joka ajetaan silloin, kun tilaukselle luodaan tilauslipuke palautuksen yhteydessä. Ongelmaksi tämän koukun kanssa muodostui se, että koukkua ei välttämättä aina ajeta, koska vaikka tilauslipuke luodaan aina tehtäessä osittaispalautuksia, sen luominen on jostain syystä valinnaista muita palautuksia tehtäessä. Yritin ratkaista tämän pakottamalla valinnan JavaScriptin avulla, jolloin valintaruutu olisi aina valittuna, ja siten myös koukku tulisi aina ajetuksi. Tämä oli teknisesti ihan toimiva ratkaisu, mutta käyttäjän valinnan pakottamisessa on tietysti myös omat ongelmansa, joten tämäkään ratkaisu ei ollut kelvollinen.

Ratkaisu ongelmaan löytyi lopulta liettualaisen Invertus-yrityksen avulla, josta tuli kouluttaja pitämään toimeksiantajayritykseen koulutusta PrestaShop-kehityksestä. Tutkimme palautusten toimintaa PhpStormin virheenjäljittimen (engl. debugger) avulla, jolloin koodia voitiin ajaa askel askeleelta nähdäksemme tarkemmin mitä se tekee. Virheenjäljittimen avulla löytyi koukku nimeltä *actionDispatcher*, joka testauksen perusteella ajetaan luotettavasti aina, kun tehdään palautuksia. *actionDispatcher*-koukkua ajava Dispatcher-luokka on vastuussa URL-uudelleenohjauksista (PrestaShop 2016), mikä tarkoittaa sitä, että koukku ajetaan palautusten lisäksi aivan jokaisella sivunlatauksella. Suorituskykyyn on tämän takia kiinnitetty erityistä huomiota, moduulin koodissa on vahvat tarkistukset, että mahdollisimman vähän koukkuun rekisteröityä koodia ajetaan, paitsi jos ollaan tekemässä palautuksia.

Tuotepalautuksissa oli muitakin ongelmia kuin pelkkä oikean koukun löytäminen. Jostain tuntemattomasta syystä PrestaShop ei pidä mitään lukua siitä, paljonko rahaa on palautettu asiakkaalle, mikä aiheuttaa ongelmia aiemmin mainitun maksujen viimeistelyn kanssa. Avarda haluaa, että viimeistelyjä tehtäessä kutsun mukana lähetetään kaikki tilaukseen liittyvät tuoterivit listattuna, ja näiden rivien summan pitää olla sama kuin tilauksen loppusumma. Jos meillä ei ole tietoa aiemmin palautetun rahan määrästä, se ei voi olla sama. Tähän oli kuitenkin helppo ratkaisu, eli moduuli pitää itse kirjaa palautuksista. Palautussumma oli loogista sijoittaa tietokantaan maksutietojen perään, jotka jo tallennetaan tilausta luotaessa, koska ne näytetään tilaussivulla, ja on suorituskykyisempää käydä kysymässä ne omasta tietokannasta kuin ulkoiselta rajapinnalta.

Alennuskupongit toivat myös oman merkittävän haasteensa tuotepalautuksien toteuttamiseen. Tuotepalautusten palautettava summa oli laskettava käsin (käytännössä POST-parametreistä), koska kun käytetään niin ”geneeristä” koukkua kuin *actionDispatcher*, se ei tietysti saa parametreinä mitään suoraan palautukseen liittyvää. Itse laskutoimitus ei ollut sen suurempi ongelma, mutta laskutoimituksessa käytettävien arvojen löytäminen oli. Kaikki arvot täytyi hakea käsin lähdekoodin seasta, joka oli huonosti dokumentoitu. Koodissa oli myös selkeitä taikanumeroita, joiden merkitys täytyi selvittää oikean lopputuloksen saamiseksi.

3.7 Käännökset ja lokalisointi

PrestaShop toi versiossa 1.7.6 vanhan käännösjärjestelmän rinnalle uuden järjestelmän (PrestaShop 2020b), ja täten kehityksen aikana pitikin päättää aika nopeasti kumpaa järjestelmää moduulin tulisi käyttää, koska molemmilla järjestelmillä on omat tapansa merkkijonojen kääntämiselle. Molemmissa järjestelmissä on omat hyvät ja huonot puolensa, mutta tulevaisuutta ajatellen päätin käyttää moduulissa uutta järjestelmää, vaikka se tarkoittikin sitä, että moduulia ei voi käyttää aiemmalla versiolla kuin PrestaShop 1.7.6.

Uudessa käännösjärjestelmässä ei ole vielä mahdollisuutta viedä tai tuoda (engl. import / export) tehtyjä käännöksiä, joten käännöksien sisällyttämiseen moduuliin oli keksittävä jokin oma ratkaisu. Vein tekemäni käännökset JSON-muodossa suoraan tietokannasta, ja tallensin ne JSON-tiedostoina moduulin sisälle. Näin käännökset saatiin sisällytettyä moduuliin, mutta PrestaShop ei osaa käyttää niitä JSON-muodossa. Ratkaistakseni tämän ongelman kirjoitin metodin, joka käy läpi kaikki moduulin translations-kansiossa olevat JSON-tiedostot, ja lisää niistä löytyvät käännökset tietokantaan, jolloin PrestaShop voi käyttää niitä. Metodi hoitaa samalla myös sen, että käännösten kielikenttään tulee oikeat tunnisteet (koska jokaisella PrestaShop-kaupalla voi olla joka kielelle omat tunnisteensa). Metodi ajetaan automaattisesti moduulin asennuksen yhteydessä.

Ongelmaton ratkaisu tämä ei ole, sillä esimerkiksi, jos kauppaan lisätään uusi kieli, moduulin käännökset on tuotava tietokantaan uudestaan, koska moduulin asennushetkellä voidaan ainoastaan tuoda käännökset kielille, jotka on jo asennettu kauppaan. Tämän helpottamiseksi moduulin hallintasivulla on työkalu, jolla käännösten uudelleentuonti on helppo tehdä. Käännösten uudelleentuonti kuitenkin hävittää kaikki mahdolliset kauppiaan itse tekemät muutokset käännöksiin, mikä pakottaa kauppiaan pitämään kirjaa muutoksistaan jossain moduulin ulkopuolella. Käännöstiedostot täytyy myös päivittää manuaalisesti aina, kun moduuliin tulee päivitysten mukana uusia merkkijonoja, mikä teettää turhaa, ylimääräistä työtä kehittäjälle.

3.8 Tunnuksien hallinta

Monesta muusta maksunvälittäjästä poiketen Avardan rajapintaan ei voi lähettää tietoa siitä, millä rahayksiköllä tilaus on tehty, vaan Avarda antaa kauppiaalle omat rajapintatunnukset jokaista kaupassa käytettävää rahayksikköä kohden. Tämän takia moduuliin piti kehittää toiminnallisuus, jolla kauppias voi luoda eri rajapintatunnukset jokaiselle käytetylle valuutalle, ja hallita niitä.

Toiminnallisuuden kehittäminen käsin olisi vaatinut aika paljon työtä, kun ottaa huomioon CRUD-toimintojen ja käyttöliittymän toteuttamisen. PrestaShop kuitenkin onneksi tarjoaa valmiin Active Record -mallin kaltaisen ObjectModel-luokan, jonka avulla nämä on helppo toteuttaa.

Active Record -malli on arkkitehtuurimalli, jossa oliot vastaavat vahvasti rakenteeltaan tietokannan rakennetta: jokaiselle tietokantataulun sarakkeelle on luokassa oma jäsenmuuttuja. Olio vastaa tällöin yhtä riviä tietokannassa. Jokainen olio on vastuussa tiedon tallentamisesta tietokantaan ja myös sen hakemisesta (Fowler 2003, 160).

Rajapintatunnuksille luotiin ObjectModel-luokasta periytyvä AvardaCredential-luokka (eli malli), johon määriteltiin staattisella *definition*-listamuuttujalla seuraavat kentät:

- Rahayksikön tunniste
- Sivustokoodi (käytännössä käyttäjätunnus)
- Sivustosalasana
- Ympäristö (ovatko tunnukset tuotanto- vai testausympäristöön)

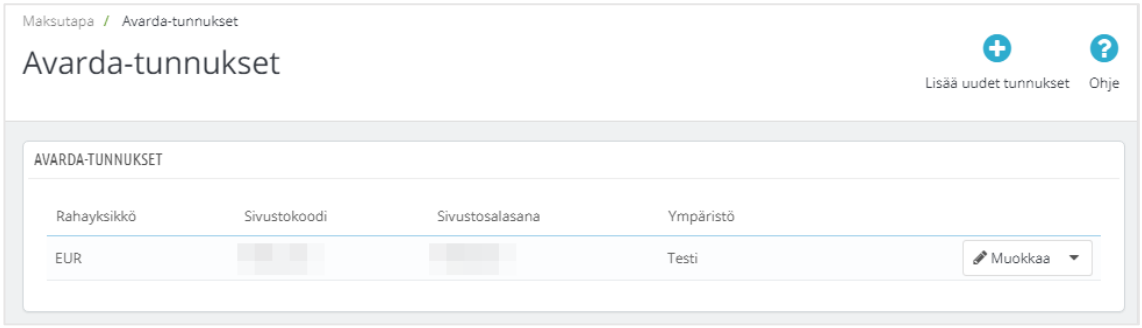
Hallintasivua varten täytyi tehdä uusi käsittelijä. Käsittelijän saa helposti toimimaan mallin kanssa kertomalla luokan ja tietokantataulun nimet muuttujien avulla. Käsittelijän saa listaamaan tietokannasta löytyvät rajapintatunnukset *fields_list*-muuttujan avulla, jolla käytännössä määritellään listan rakenne (kuva 4). (Urbanovich 2016.)

```
$this->fields_list = [
    "id_currency" => [
        "title"           => $this->trans( id: "Currency", [], domain: "Modules.Avarda.Settings"),
        "type"            => "text",
        "callback"        => "displayCurrencyCode",
        "callback_object" => $this
    ],
    "site_code" => [
        "title" => $this->trans( id: "Site Code", [], domain: "Modules.Avarda.Settings"),
        "type"  => "text"
    ],
    "site_password" => [
        "title" => $this->trans( id: "Site Password", [], domain: "Modules.Avarda.Settings"),
        "type"  => "text"
    ],
    "live" => [
        "title"           => $this->trans( id: "Environment", [], domain: "Modules.Avarda.Settings"),
        "type"            => "text",
        "callback"        => "displayLiveStatus",
        "callback_object" => $this
    ]
];
```

KUVA 4. Rajapintatunnuslistauksen rakenteen määrittely *fields_list*-muuttujan avulla

Lisäys-, poisto- ja muokkaustoiminnallisuudet sai aktiiviseksi myös melko yksinkertaisesti. Lisäystä ja muokkausta varten käytettävä lomake määritellään hyvin samalla tavalla kuin alkioiden listaus, käyttämällä *fields_form*-listamuuttujaa. ”Lisää” ja ”Poista” -napit saa näkyviin *addRowAction*-funktiolla, jolle annetaan parametrinä toivottu toiminto (*edit* tai *delete*) (Serny 2014, 102).

Kaikkea PrestaShop ei kuitenkaan tee valmiiksi, sillä esimerkiksi täytyy tarkistaa itse, löytyvätkö käyttäjän antamat tunnukset jo tietokannasta. Toteutin käsittelijään myös muutamia omia käyttäjäkokemusta parantavia ominaisuuksia, kuten tunnusten toimivuuden tarkistus lisäämisen jälkeen, ja erillisen napin tunnusten lisäämistä varten. Toteutin myös funktiot listauksen tunnisteiden selkokielisten nimien hakemiselle, jolloin listassa lukee esimerkiksi tunnisteen ”1” sijasta ”EUR”, mikäli kyseinen tunniste vastaa euroa tietokannassa. Hallintasivua esitellään kuvassa 5.



KUVA 5. Rajapintatunnuksien listaus-/hallintasivu (tunnukset peitetty)

4 JATKUVA INTEGRAATIO

4.1 Testaaminen

Toimeksiantajayrityksen suosituksesta kirjoitin yksinkertaiset testit moduulin rajapinta-asiakasohjelmalle käyttäen PHPUnit-sovellustestauskehystä. Kirjoittamani testit ovat käytännössä integraatiotestejä, jotka testaavat sitä, osaavatko asiakasohjelman metodit kommunikoida Avardan rajapinnan kanssa oikein.

Moduuliin on tulevaisuudessa tarkoitus kirjoittaa huomattavasti kattavammatkin testit, kuten yksikkötestit, ja myös muihin osiin kuin pelkästään asiakasohjelmaan. Toteuttaminen on kuitenkin tässä vaiheessa vielä hyvin epäselvää, sillä PrestaShop-moduulin kunnollinen testaaminen vaatisi esimerkiksi automaattisen testikaupan luomisen, ja oikeiden tilausten ja asiakkaiden kaltaista dataa. Näiden toteuttaminen ei ainakaan nykytietojeni mukaan ole yksinkertaista. Myös oikean käyttäjän liikkeiden simulointi käyttöliittymässä olisi hyödyllistä, esimerkiksi käyttämällä Googlen kehittämää Puppeteer-rajapintaa, jonka avulla voi komentaa oikeaa Chrome-selainta JavaScriptin avulla.

4.2 Testaamisen ja julkaisujen automatisointi

Toimeksiantajayrityksellä on jo käytössään testaamista varten Travis CI-palvelu, ja sitä käytettiin myös tämän moduulin kanssa. Travis CI on jatkuva integraatio -järjestelmä (Travis CI 2019), jolla voi automaattisesti testata ja kääntää (engl. build) ohjelmistoja, ja se on integroitu suoraan GitHubiin, missä myös projektin koodi sijaitsee. Travis CI -palvelua ohjeistetaan lisäämällä projektiin travis.yml-tiedosto, jossa kerrotaan mitä palvelun tulee tehdä (Travis CI 2020).

Palvelua käytettiin aluksi vain testien ajamiseen automaattisesti aina, kun projektin koodikantaan tehtiin muutoksia. Moduulin lähestyessä tuotantokäyttöä heräsi kuitenkin tarve myös julkaisujen automatisoinnille. PrestaShopin valmiiden moduulipakettien luominen käsin on työlästä, sillä kansiorakenteen täytyy olla tietyn-

lainen, paketista täytyy löytyä kaikki riippuvuudet, mutta ei mahdollisesti vaarallisia ohjelmistokehyksiä, joita tarvitaan vain kehityksessä, ja jokaisesta kansiosta täytyy löytyä index.php-tiedosto tietoturvan takia, jotta mahdollinen huonosti konfiguroitu web-palvelin ei anna kansioista tiedostolistausta.

Ohjeistin Traviksen konfiguraatiotiedoston avulla tekemään testaamisen lisäksi kaiken tarvittavan julkaisujen automatisoimiseksi. Mikäli koodikantaan tuleva vetopyyntö (engl. pull request) yhdistetään (engl. merge), ja vetopyyntö on nimetty tietyllä tavalla vahinkojulkaisujen estämiseksi, Travis on ohjeistettu luomaan zip-tiedoston, joka sisältää projektin koodin tuotantoriippuvuuksien kanssa. Jokaiseen kansioon lisätään index.php-tiedosto, ja paketista poistetaan ylimääräiset tiedostot. Lopuksi valmis paketti julkaistaan ohjelmavaraston (engl. repository) julkaisut-osiossa, josta kaupan ylläpitäjä voi ladata moduulin uuden version.

Julkaisuprosessin testaaminen oli hankalaa ja hidasta, sillä nähdäkseni konfiguraatiotiedoston muutoksien vaikutuksen, koodista täytyi aina luoda uusi haara (engl. branch) ja yhdistää se päähaaraan, jotta Travis yrittäisi tehdä uuden julkaisun. Lopputuloksena oli kuitenkin toimiva automaattinen julkaisuprosessi, joka säästää merkittävästi aikaa aina, kun koodikantaan tehdään muutoksia ja ne halutaan saada julkaistua kauppojen ylläpitäjille.

5 TIEDOSSA OLEVAT ONGELMAT

5.1 Ohjelmointivirheet

Yritin moduulin kehityksen aikana korjata kaikki ilmenneet ohjelmointivirheet, ja testata moduulia tarkasti löytääkseni niitä. Isoja ohjelmointivirheitä ei ole tiedossa, mutta törmäsin kehityksen aikana mielenkiintoisiin virheisiin, joita en koskaan saanut toistettua. Näistä ensimmäisessä tapauksessa keskeytin sivunlatauksen kesken maksun viimeistelyn (tilauksen tekemisen jälkeen), joka johti siihen, että maksua ei enää löytynyt Avardan rajapinnasta maksutunnisteen perusteella. Toisessa tapauksessa oli kyseessä normaali tilauksen tekeminen, mutta Avardan rajapinta väitti asiakkaan ”vahvistamatta” olevan saldon olevan kaksinkertainen siihen nähden, mitä sen piti olla rajapintakutsuun nähden. Molemmat näistä ovat vakavia ongelmia, jotka tuotannossa tapahtuessa aiheuttaisivat kauppiaalle lisäselviteltävää, mutta niitä on tietysti mahdotonta selvittää tarkemmin ennen kuin ne tapahtuvat uudestaan.

Tulevaisuuden virheenselvittelyä varten moduulin lokitusta täytyy parantaa. Nykyinen toteutus tallentaa kaikkien asiakasohjelman rajapintakutsujen datan tietokantaan PrestaShopin oman lokitusluokan avulla. Tämä auttaa jo huomattavasti virheiden selvittämisessä, mutta lokiviestien yhteyttä toisiinsa joutuu päättelemään aikaleiman perusteella, koska viesteissä ei ole mitään millä tunnistaa mihin tapahtumaan ne liittyvät. Tulevaisuudessa täytyy siis kehittää jokin sessiotunniste, joka liitetään lokiviestiin, ja jolla voi tunnistaa samaan tilaukseen liittyvät viestit.

Lokitukseenkin olisi hyvä kehittää jokin fiksumpi, ulkoinen järjestelmä, joka tukee automaattista lokien seurantaa, jolloin virhetilanteisiin voitaisiin reagoida nopeasti. Tällainen järjestelmä on kuitenkin tämän maksumoduulin kattavuuden ulkopuolella, ja olisi parempi toteuttaa erillisenä toiminnallisuutena, jotta sillä voi seurata muitakin moduuleja.

5.2 Tulevat muutokset PrestaShopiin

Tässä opinnäytetyössä aiemmin käsitelty AdminOrdersController.php-tiedosto, jonka pohjalta muun muassa koko tämän moduulin tuotepalautuslogiikka on kirjoitettu, on PrestaShopin tulevassa minor-julkaisussa (versio 1.7.7) poistettu, koska tilauksia käsittelevä sivu on uudelleenkirjoitettu täysin Symfony-pohjaiseksi (PrestaShop 2020c). Tämä tarkoittaa sitä, että myös moduulista täytyy todennäköisesti uudelleen kirjoittaa osia, kun PrestaShopin uusi versio julkaistaan, ja minun samalla opiskella Symfonyn toimintaa lisää.

Moduulia testattiin kehitysvaiheessa myös PrestaShopin kehitysversiolla (käytännössä 1.7.7, mutta tilauksia käsittelevä sivu ei ollut vielä Symfony-pohjainen), ja tuotepalautuksia tehtäessä rahasummissa oli parin sentin eroja tilauksen loppusumman ja tuotteiden yhteenlasketun summan välillä, jolloin Avardan rajapinta ei hyväksynyt niitä. Vielä ei ole varmuutta, onko tämä jonkinlainen ohjelmointivirhe PrestaShopin puolella, vai onko koko laskentalogiikkaa muutettu. Mikäli on, logiikkaa täytyy muuttaa myös moduulissa.

5.3 Virheviestit ja käännökset

Iso ongelma moduulin käytettävyyden kanssa on se, että Avardan rajapinnalta tulevat virheviestit ovat englanninkielisiä, ja ne näytetään asiakkaalle suoraan englanniksi, vaikka käyttäjän kieli olisi joku muu, kuten suomi. Myös kauppiaan taustahallinnassa näkemät Avardalta tulevat virheviestit ovat kaikki englanniksi. Käyttäjä ei tietysti normaalitilanteessa tule näkemään virheviestejä, mutta Avardan rajapinnalla on joitain oudolta tuntuvia vaatimuksia (kuten se, että sähköpostiosoitteen maksimipituus on 60 merkkiä, vaikka sähköpostiosoitteet voivat standardin mukaan olla 254 merkkiä pitkiä (Sayers 2009)), jotka johtavat virhetilanteisiin, jos ne eivät täyty. Virheviestit olisi siis hyvä saada käännettyä, mutta siihen ei tällä hetkellä ole mitään yksinkertaista tapaa. Virheviestit tulevat sellaisenaan rajapinnasta ilman muita tunnisteita, joten todennäköisesti ainoa tapa kääntää niitä koodissa olisi ylläpitää manuaalisesti listaa kaikista mahdollisista virheviesteistä ja niiden käännöksistä, mikä olisi työläs ja huono ratkaisu.

Myös moduulin suomenkielisissä käännöksissä on parannettavan varaa. Tällä hetkellä esimerkiksi ”aktivointi”-sanalla on käännöksissä monta merkitystä. Rajapinnan käyttämille termeille ei ole mitään suoria, sopivia käännöksiä, mutta käännöksien pitäisi silti olla selkeämpiä ja toisistaan erottuvia, jolloin kauppiaalle ei tule käännöksien takia epäselvyyttä siitä, mitä häneltä vaaditaan. Käännöksien selkeyttämisen on suunniteltu tapahtuvan heti seuraavassa päivityksessä.

6 POHDINTA

Projektin tuloksena syntyi PrestaShop-moduuli, joka mahdollistaa asiakkaille Avardan maksupalveluiden käytön PrestaShop-verkkokaupassa. Moduuli julkaistaan PrestaShopin lisäosakaupassa, ja toimeksiantajayritys voi myös tarjota sitä omille asiakkailleen.

Projektin toteuttamiseen meni enemmän aikaa kuin projektin alussa oletin, mutta olen tyytyväinen tuloksena syntyneeseen moduuliin. Oma osaamiseni PrestaShopin osalta kehittyi valtavasti, mutta myös yleisesti web-kehityksen osalta. Jatkuva integraatio -prosessi on toimiva ja testattu, ja sitä tullaan varmasti jatkossa käyttämään myös muiden PrestaShop-moduulien rakentamisessa.

Olen tyytyväinen myös tekemiini ratkaisuihin, enkä välttämättä tekisi mitään toisin. Asiakkaalle voisi olla selkeämpää, mikäli maksulomake avautuisi kassasivulle eikä omalle sivulleen, mutta kassasivun rakenteen takia se on paljon toimivampi omalla sivullaan. Ohjelmakoodi on mielestäni kelvollisen siistiä, vaikka aina on parantamisen varaa ja jotkin koukkujen sisällä olevat osiot kasvoivat aika pitkiksi, mutta näiden osioiden ohjelmakoodia voi parannella samalla, kun tekee muita muutoksia. Tietoturvallisuudenkaan osalta ei ole tiedossa puutteita, ja tietoturvallisuuden parantaminen vaatisi muutoksia myös rajapinnan puolelta.

Moduulin kehitystyö jatkuu edelleen, ja aiemmin esiteltyjen ongelmakohtien osalta on paljonkin työtä tiedossa. Avardalta on tulossa uusi versio heidän rajapinnastansa, joten PrestaShopiin tulevien muutoksien lisäksi myös varsinaista integraatiota käsitteleviä osia koodista joutunee tulevaisuudessa kirjoittamaan uudelleen.

LÄHTEET

Borowicz, P. 2019. PrestaShop In 2019 And Beyond, Part 1: The Current Architecture. Luettu 30.4.2020. https://build.prestashop.com/news/prestashop-in-2019-and-beyond-part-1-current-architecture/

Fowler, M. 2003. Patterns of Enterprise Application Architecture. 17. painos. Boston: Addison-Wesley.

Google. 2020. Secure your site with HTTPS. Luettu 17.5.2020. https://support.google.com/webmasters/answer/6073543?hl=en

Jash Technologie. 2016. The Dispatcher-. Luettu 27.3.2020. http://doc.prestashop.com/pages/viewpage.action?pageId=51184592

Nguyen, B. 2004. Linux Dictionary. Luettu 27.4.2020. http://www.tldp.org/LDP/Linux-Dictionary/html/

PrestaShop. 2020a. Hooks. Luettu 27.4.2020. https://devdocs.prestashop.com/1.7/modules/concepts/hooks/

PrestaShop. 2020b. Module Translation. Luettu 16.4.2020. https://devdocs.prestashop.com/1.7/modules/creation/module-translation/

PrestaShop. 2020c. Notable changes in PrestaShop 1.7.7. Luettu 23.5.2020. https://devdocs.prestashop.com/1.7/modules/core-updates/1.7.7/

PrestaShop. 2020d. Payment modules. Luettu 3.4.2020. https://devdocs.prestashop.com/1.7/modules/payment/

Sayers, D. 2009. RFC Errata, Erratum ID 1690, RFC 3696. http://www.rfc-editor.org/errata_search.php?rfc=3696&eid=1690

Serny, F. 2014. PrestaShop Module Development. Birmingham: Packt Publishing.

Travis CI. 2019. README.md. Luettu 21.5.2020. https://github.com/travis-ci/travis-ci/blob/master/README.md

Travis CI. 2020. Travis CI Tutorial. Luettu 21.5.2020. https://docs.travis-ci.com/user/tutorial/

Urbanovich, D. 2016. PrestaShop MVC. Part 1: Creating a Model. Luettu 20.4.2020. https://belvg.com/blog/prestashop-mvc-part-1-creating-a-model.html