KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutus

Niko Ainali

OHJELMISTOINTEGRAATION KEHITTÄMINEN B2B-VERKKOKAUPPAAN

Opinnäytetyö
Toukokuu 2021

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Toukokuu 2021
Tietojenkäsittelyn koulutus

Tikkarinne 9
80200 JOENSUU
+358 13 260 600 (vaihde)

Tekijä
Niko Ainali

Nimeke
Ohjelmistointegraation kehittäminen B2B-verkkokauppaan

Toimeksiantaja
Skycode Oy

Tiivistelmä

Tässä opinnäytetyössä kehitettiin ohjelmistointegraatio käyttämällä Netvisor-ohjelmistorajapintaa. Ohjelmistointegraatio tehtiin Skycode Oy:n kehittämän Skyshop -verkkokaupan ja Visma Solutions Oy:n kehittämän Netvisor-taloushallintaohjelmiston välille.

Opinnäytetyössä käydään läpi ohjelmistointegraation kehittämisen kulkua sekä käytettyjä tekniikoita ja niiden ominaisuuksia. Käytettyjä tekniikoita ja työkaluja ovat muun muassa PHP, CakePHP, MySQL ja Composer sekä REST-tyyppinen ohjelmistorajapinta. Opinnäytetyön toiminnallisessa osassa valmistettiin ohjelmistointegraatio toimeksiantajan kriteerien ja tarpeen mukaisesti. Netvisor-ohjelmistointegraatio toteutettiin siten, että sitä voidaan käyttää myös myöhemmin muihin projekteihin.

Kertynyttä osaamista sekä uutta tietoa Netvisor-ohjelmistorajapinnasta on tarkoituksena jakaa Skycodelle ja käyttää tietoja jatkokehitykseen sekä muihin erilaisiin ohjelmistointegraatioihin.

Kieli
Suomi

Sivuja 62

Asiasanat
ohjelmistointegraatio, verkkokauppa, CakePHP, PHP, MySQL

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
May 2021
Business Information Technology

Tikkarinne 9
80200 JOENSUU
FINLAND
+ 358 13 260 600 (switchboard)

Author
Niko Ainali

Title
Developing Software Integration to B2B E-commerce

Commissioned by
Skycode Oy

Abstract

This thesis is about developing a software integration commissioned by Skycode Ltd. Software integration was built between an e-commerce called Skyshop from Skycode Ltd and Netvisor accounting software by Visma Solutions Ltd. The integration was made using a Netvisor API.

The purpose of thesis was to develop an integration in a way so that it is reusable in other projects or programs which need an integration to Netvisor. The development of program was built within needs and the requirements that the commissioner had set.

The thesis reviews the software integration process from planning to actual development and also reviews the techniques and properties of used technologies. The technologies and tools used are PHP, CakePHP, MySQL, Composer and REST-styled API among other things.

The result of this thesis was a plugin for CakePHP-framework, fulfilling the requirements set by the commissioner. Accumulated know-how and new information gathered from Netvisor will be shared within Skycode Ltd and used in projects and programs in the future.

Language

Finnish

Pages 62

Keywords

software integration, e-commerce, CakePHP, PHP, MySQL

Sisältö

1 Johdanto

Opinnäytetyö toteutetaan toimeksiantona Skycode Oy:lle. Skyshop nimekkeellä toimivaan business-to-business eli B2B-verkkokauppaan on tarkoitus kehittää uusi ohjelmistointegraatio. Skycode on vuonna 2009 perustettu vaasalainen ohjelmistojen ja integraatioiden kehittämiseen keskittyvä ohjelmistotalo, joka tarjoaa verkkopohjaisia ratkaisuja asiakkaiden tarpeisiin. (Skycode 2020).

Skyshop on Skycoden kehittämä B2B-verkkokauppa, josta jälleenmyyjät tai asiakkaat voivat tehdä tilauksia helposti. Verkkokauppa integroidaan asiakkaan toiminnanohjaus- tai taloushallintajärjestelmään. (Skycode 2020).

Tämä ohjelmistointegraatio keskittyy Visma solutions Oy:n kehittämään Netvisor taloushallinnon ja toiminnanohjaus järjestelmän integroimiseen, Netvisorin tarjoaman rajapinnan avulla Skyshop-verkkokauppaan. Toimeksiannon tarkoituksena on edistää Skyshopin valmiiden integraatioiden määrää kilpailukyvyn parantamiseksi. Opinnäytetyön ja projektin kehittämisen aikana on tarkoitus avata lukijalle ohjelmistointegraatioiden hyötyjä, niihin käytettäviä tekniikoita sekä integraatiokehityksen kulkua. Toimeksiannon kohteena olevan ohjelmistointegraatioon käytetään pääasiallisesti PHP-ohjelmointikielen pohjalta kehitettyä CakePHP-ohjelmistokehystä sekä REST-tyyppistä rajapintaa XML kutsuilla.

Ohjelmistojen ja täten integraatioiden kysyntä kasvaa jatkuvasti digitalisoitumisen myötä, tuotteet muuttuvat digitaalisiksi palveluiksi ja verkkokauppojen käyttö yleistyy jatkuvasti (Dikisyke 2020; Posti 2020). Digitalisoituminen varsinkin työssäni ohjelmistokehittäjänä heijastuu kasvaneeseen integraatioiden kyselyiden ja tilausten määrään. Ohjelmistointegraatiot tuovat mukanaan hyötyjä, kuten tehokkuutta, vähemmän virheitä sekä nopeampaa myyntiä tai tuottamista (WSO2 Team 2017).

Ohjelmistointegraatioiden myötä esimerkiksi tilaukset siirtyvät automaattisesti taloushallinnan ohjelmistoihin, näistä eteenpäin paketti- ja kuljetuspalveluihin ja

näin vapauttavat resursseja, säästävät aikaa ja poistavat ikävän manuaalisen syöttökenttien täyttämisen ilman ihmisten aiheuttamia virheitä (Shahraki 2013, 94–96). Edellä mainitut toimintojen automatisaatiot, tilausten siirtäminen ohjelmien välillä sekä automaattiset pakettilähetykset tai seurannat ovat toteutettavissa, varmasti toteutettu jo moneen kertaan, mutta ohjelmistointegraatioiden saattaminen toimintaan vaatii aina suuren työmäärän liittyen yrityksen sekä ohjelmistojen rakenteeseen, toimintamalleihin ja prosesseihin. Parhaimmillaan integraatio tarpeet voidaan täyttää jo valmiiksi kehitetyillä integraatioilla, jotka on kehitetty tai sopivat käytettyjen ohjelmien välille.

Integraation, siirrettävän tiedon, palveluiden ja prosessien läpikäyminen todennäköisesti vaatii taasen yhteistyötä monelta eri osapuolelta, jopa monesta organisaatiosta, jotka ylläpitävät mahdollisesti integraation tilaajan dataa, palvelimia tai kumpaakin (Niemi 2019). Tavoitteeni on avata lukijalle ohjelmistointegraatioiden määritelmää, niiden tuomia hyötyjä sekä itse toteutettavan integraation kehityskulkua.

2 Ohjelmistointegraatio

2.1 Mitä ohjelmistointegraatiot ovat?

Ohjelmiston integraatio, toiselta sekä yleisemmältä nimitykseltään ohjelmistointegraatio tarkoittaa kahden tai useamman eri ohjelmiston yhdistämistä toisiinsa. Ohjelmistojen ei tarvitse olla samankaltaisia eikä edes samalla ohjelmistokielellä kirjoitettuja. Ohjelmistointegraatioita voi olla useampia yhtä ohjelmistoa kohden ja tämä luonnollisesti kasvattaa hallittavaa tietoa sekä mahdollisesti integraatioiden toimintojen suoritusajankohtia tai niiden käsittelemien tietojen muutoksia. (Microsoft. 2002a.)

Ohjelmistointegraatioista hyötyvät kaikki, jotka käyttävät resursseja tietojen manuaaliseen kopiointiin paikasta toiseen, hyppivät ohjelmasta toiseen

lisäämässä tietoa, joka on saatu toisesta ohjelmasta tai eivät pysty hallitsemaan suurta määrää tietoa virheettömästi ja tehokkaasti samaan aikaan. Ohjelmistoja saattaa olla pienimmissä tai keskisuurissa yrityksissä vähemmän ja näiden välillä siirtyminen onnistuu suhteellisen hyvin, ajoittaisia virheitä lukuun ottamatta. Suuret yritykset taas saattavat käyttää kymmeniä ohjelmia, introja tai verkkosivustoja päivittäin pyörittämään liiketoimintaansa eteenpäin. ERP-ohjelmistot (Enterprise Resource Planning) pyrkivät korvaamaan ja auttamaan yrityksiä pitämään useat toiminnallisuudet yhdessä paikassa ja yleensä yhdistävätkin tärkeimmät hallinnalliset toiminnot yhteen ohjelmistoon. ERP-ohjelmistot eivät kuitenkaan sisällä valmiina läheskään yhtä paljon toimintoja mitä käyttäjä toivoisi tai tarvitsisi, joten ERP-ohjelmistojen integroiminen onkin yksi halutuimmista ohjelmistointegraatioista. (Hohpe 2003.)

Ohjelmistointegraatioita voidaan toteuttaa rajapintojen tai suorien tietokantayhteyksien kautta. Rajapintojen käyttö integraatiototeutuksissa on lähtökohtaisesti turvallisempi ja nopeampi vaihtoehto, koska ohjelmiston antamat tiedot rajapinnan kautta on valmiiksi määritelty yleensä erilaisten kutsujen alle sekä tieto on saatavilla hallitusti. Virheelliset kutsut tai resurssin käytöt estetään jo pyyntövaiheessa, jolloin ohjelmistojen tietojen virheellinen muutos tai ongelmat pysäytetään jo aloitusvaiheessa. Virheet ovat helposti saatavilla palautuneesta vastauksesta riippuen tietenkin rajapinnasta sekä sen virheenkäsittelystä. Virheilmoitukset ohjaavat ohjelmoijan suoraan ongelman jäljille tai virhe on voitu jo dokumentoida sekä se on avattu itse rajapinnan dokumentaatiossa. (Preibisch 2018.)

Suorassa tietokantayhteydessä ei samanlaisia varmenteita tai virhetapauksia saada välttämättä selville vasta kun loppukäyttäjä huomaa tietojen virheellisyyden tai ohjelmiston toimimattomuuden kautta. Pahimmassa tapauksessa integraation kohteena olevan ohjelmiston tiedot voivat olla virheellisiä näitä käsitellessä ja tämä voi aiheuttaa ohjelmiston kaatumisen tai master-datan eli ydintietojen sekoittumisen väärillä tiedoilla sekä tuo lisäkustannuksia niin integraation tilaajalle kuin toteuttavalle ohjelmistoyritykselle. Virhehallinta ohjelmistossa on pääasiallisesti ohjelmistoissa läsnä ohjelmoijan kirjoittamien testien tai virhehallinnan kautta,

mutta ohjelmoinnin aikana tapahtuvat tahattomat virheet keskittyessä tehtävään esimerkiksi suoraa tietokantayhteyttä käyttäessä ja ilman kolmannen osapuolen virheenhallintaa tuovat projektiin mahdollisia ennakoimattomia ongelmia. (Cater 2002.)

Normaali käyttötapaus voi olla esimerkiksi taloushallinnonohjelmiston integroiminen verkkokauppaan, josta päivitetään, haetaan sekä hallitaan tuotteiden hintoja ja saldoja. Tilaukset siirtyvät verkkokaupasta automaattisesti käytössä olevaan taloushallinnon järjestelmään, eikä näitä tietoja täten tarvitse manuaalisesti hakea verkkokaupasta jokaisen tilauksen yhteydessä.

Integraatiot eivät suinkaan ole aina tilaustöitä, asiakaskohtaista koodia tai pitkiä prosesseja ohjelmistokehittäjän ja organisaation osapuolten välillä, vaan monet toimijat tarjoavat omia tai kolmansien osapuolten kehittämiä valmiita ratkaisuja integraatiotarpeisiin integraatioalustoilla.

Valmiiden ratkaisujen kanssa on tärkeää varmistaa, että integraation tarjoajalla on ratkaisu juuri organisaation käyttämiin ohjelmiin tai tarjoaa hyvän ja pitkäaikaisen ratkaisun esimerkiksi uuden ohjelmiston käyttöönotolla integraation yhteydessä. Integraatioalustan kautta voidaan tehdä omia sääntöjä ja integraatiovaiheita esimerkiksi datan käsittelyyn ja sen muodostamiseen. Integraatioalusta tai ETL-työkalujen käyttäminen ovat hyviä vaihtoehtoja, mikäli projektin ja toimeksiantajan vaatimukset voidaan täyttää valmiin alustan toiminnoilla ja integraatioilla. Parhaimmillaan integraation käyttöönotto integraatioalustan tai tarjoajan kautta ei vaadi muuta kuin ohjelmien tunnusten lisäyksen ohjelmistoon. (Kelley 2020).

ETL eli ”Extract, Transform, Load”-menetelmällä, jossa tietoa luetaan yhdestä tai useammasta ohjelmistosta tai palvelusta, muunnetaan yhtenäiseksi dataksi ja oikeaan formaattiin vastaanottavaa ohjelmistoa varten. Lopuksi lähetetään haluttuun ohjelmistoon tai näkymään käsitelty data. ETL-työkalulla voidaan lukea useasta tietokannasta tai ohjelmistosta tietoa, käsitellä se halutulla tavalla ja lähettää eteenpäin joko suoraan näytettäväksi tai tallennetaan jatkotoimenpiteitä varten ohjelmistoon. (Zhao 2017.)

Opinnäytetyössä on lähdetty rakentamaan ohjelmistointegraatiota itse ilman kolmannen osapuolen ETL-työkaluja tai integraatioalustoja. Integraatio tehdään itse, koska se on tulossa osaksi jo valmista tuotetta, joten tuotettavan koodin omistajuus sekä muokkaus mahdollisuudet tulisi pysyä toimeksiantajalla.

2.2 Mitä hyötyä integraatiot tuovat ohjelmistoon?

Ohjelmistointegraatioiden avulla pyritään vähentämään manuaalisia tehtäviä automatisoimalla tehtäviä tai toimintoja, joita eri käyttäjät suorittavat. Nämä tehtävät sekä toiminnot yleensä käsittävät rutiininomaisia tehtäviä, kuten tietojen siirtäminen ohjelmistosta toiseen. Automatisointi vapauttaa henkilöstöresursseja sekä mahdollisesti vähentää käyttäjien tekemiä virheitä, vapautuneet resurssit voidaan käyttää hyödyllisempiin tehtäviin (Microsoft 2002b).

Kaikkiin ohjelmiin ei välttämättä voida tehdä haluttuja integraatioita ja tämä kannattaakin ottaa huomioon jo ohjelmiston hankinta vaiheessa. Integraatio mahdollisuus ei välttämättä ole ensimmäinen asia, joka tulee mieleen ohjelmistoja hankittaessa, mutta saattaa olla suuri apu esimerkiksi yrityksen kasvuun ja kehitykseen (Puro 2019).

Ohjelmistointegraatioiden hyödyt kuitenkin riippuvat täysin tehtävien tai toimintojen laadusta, niiden manuaalisen läpikäynnin tarpeesta sekä itse yhdistettävistä ohjelmistoista. Manuaalisten tehtävien automatisointi kannattaa aina katsoa tapaus ja käyttökohtaisesti. Tehtävät kuten tietojen päivitys tai vienti järjestelmästä toiseen ovat tavanomaisia aiheita, joiden ympärillä ohjelmistointegraatiot toimivat (Hohpe 2003).

Tehtävien tai toimintojen automatisointia ei voida väittää helpoksi, mikäli integraatio sisältää useamman ohjelmiston, jotka sisältävät esimerkkitapauksena asiakkaan tietoja. Esimerkiksi, asiakas voi verkkokaupassa tarkastella omaa laskutusosoitetta ja muuttaa sitä. Kyseiset laskutustiedot voivat olla kahdessa tai jopa useammassa ohjelmistossa, joihin

tieto pitää päivittää. Helpolta ja normaalilta käyttötapaukselta vaikuttava tehtävä saattaa sisältää monimutkaisen reitin eri ohjelmistoihin, jotta saadaan näihin päivitettyä asiakkaan uudet tiedot. Ohjelmistot joihin tietoja päivitetään saattavat olla hyvinkin erilaisia ja näihin tiedon muuttaminen voi olla hyvinkin monimutkaista riippuen ohjelmistojen liiketoiminnallisesta logiikasta sekä ohjelmiston tietokanta tasoisista riippuvuuksista tai rajoituksista. Parhaimmillaan pieni laskutusosoitteen muuttaminen voi olla siis pitkä ja monimutkainen päivitys eri ohjelmistojen tietojen, riippuvuuksien sekä rajoitteiden määrittelemä reitti. (Hohpe 2003.)

2.3 Tehtävien ja töiden automatisointi integraatioiden avulla

Ohjelmistointegraatioita voidaan toteuttaa usealla eri tavalla, toteutustavan valinta perustuu yleensä täysin integroitavien ohjelmistojen kohteena olevan datan, sijainnin sekä saatavuuden perusteella. Seuraavan luetteloinnin aikana käydään läpi yleisimpiä menettelyitä sekä tarkastellaan niiden soveltuvuutta eri käyttötarkoituksiin.

ETL eli ”Extract, transform, load”-menettely perustuu kolmeen vaiheeseen; datan tiivistämiseen, käsittelyyn sekä uudelleen näyttämiseen tiivistetyssä muodossaan. Esimerkiksi useiden ohjelmistojen data, jota yritys käyttää, tiivistetään uudeksi käsitellyksi ja muokatuksi dataksi, joka säilötään tietokantaan ja asetetaan esille esimerkiksi täysin uuteen ympäristöön tai ohjelmistoon. Tällä menetelmällä saadaan tärkeät tiedot esille esimerkiksi tuotteen varastoinnista tai myynnistä, jota ei normaalisti saataisi esille pelkästään yhdestä yrityksen käyttämästä ohjelmistosta. Menettelyä käytetään yleensä datan varastointiin useammasta ohjelmasta. (Guru99 2020.)

Suora tietokantayhteys perustuu integroitavien ohjelmien tietokantaan kirjoittamiseen ja lukemiseen. Suoran tietokantayhteyksien käyttäminen vaatii

ohjelmistokehittäjältä tarkkaa tietoa tietokannan taulujen relaatioista, niiden pää- ja vierasavaimista sekä itse datan mallista. Suora tietokantayhteys tekee datan alttiiksi korruptoimiselle. Suoralla tietokantayhteydellä toteutus suoritetaan yleensä ohjelmistoihin, joihin ei ole saatavilla tai mahdollista tehdä rajapintoja. Rajapinnat tarjoavat ohjelmistoon pääsyn tiettyjen pyyntöjen tai tiedon vaihtamisen kautta. Rajapintojen kautta myös yleensä ohjelmistojen liiketoiminnallinen logiikka on käytettävissä ja toteutuu automaattisesti esimerkiksi päivittäessä tai lisätessä resursseja rajapintojen kautta. Rajapinnat yleensä tarjoavat dokumentaation, josta selviää mitä sekä minkä tyyppisiä kutsuja voidaan rajapinnalle lähettää. Rajapinnat ottavat kutsun vastaan, tarkistavat kutsun sisältämät tiedot ja palauttavat pyynnössä olevan asian tiedot, haetun resurssin tai luo uuden resurssin, mikäli kutsu on onnistunut ja sisältää rajapinnan vaatimat autentikointi tiedot. (Microsoft 2003c.)

Melkein jokainen integraatio ympäristö tarvitsee kuitenkin toimiakseen ihmisten tekemiä toimintoja, kuten tietojen validoimista, muuttamista tai tietojen syöttämistä. Käyttäjän tekemät toiminnot voivat olla helppoja ja nopeita, jonka jälkeen integraatio tekee esimerkiksi syötetystä asiasta merkinnän kolmeen eri ohjelmistoon integraation kautta tai hakee näistä hypoteettisesta kolmesta ohjelmistosta kaikki käyttäjän syöttämän hakusanaa vastaavat tiedot. (Microsoft 2003c.)

2.3.1 Automatisoinnin hyödyt ja haitat

Ohjelmistointegraatioiden tuomat hyödyt yleensä mitataan resurssien sekä ajansäästön perusteilla. Keskimääräistä hyötyä integraatiosta rahassa tai ajassa ei voida mitata yleisesti, koska integraatiot eivät ole koskaan samankaltaisia. Esimerkiksi kuitenkin hypoteettinen tilanne, josta voidaan mitata suora hyöty kuvitellusta integraatiosta; työntekijä tekee 40 tuntia viikossa töitä ja häneltä odotetaan, että hän tienaa yritykselle keskimäärin 120 € tunnissa. Mikäli tämä kyseinen henkilö käyttää joka työpäivä tunnin tietojen muokkaukseen tai vientiin ohjelmasta A ohjelmistoon B, vastaavasti integraatio joka tekee muokkauksen ja viennin ohjelmistosta A ohjelmistoon A heti muokkauksen

jälkeen automaattisesti, säästää yhden työntekijän kohdalla viikossa 600€. Mikäli useampi tai kaikki henkilöt tekevät samaa manuaalista tietojen siirtelyä, niin suora säästö ohjelmistointegraation kautta yritykselle tulisi olemaan viikossa *n* * 600 €, jossa *n* = työntekijöiden lukumäärä. Automatisointi ohjelmistointegraation avulla myös yleensä poistaa tahattomat virheet, joita ihmiset tekevät ja täten säästää aikaa sekä resursseja.

Haitat joita ohjelmistointegraatio ja sen tuoma tehtävien automatisointi tuovat ovat pääosin estettävissä jo integraation suunnittelu vaiheessa, suurin riski integraation toteutuksessa ovat epäselvät suhteet ja käyttötapaukset ohjelmistojen kesken. On tärkeää tietää kuka, milloin ja miksi muokkaa tietoa, sekä mihin kaikkiin järjestelmiin tämä tieto vaikuttaa.

Integraation myötä ohjelmistot voivat keskustella keskenään, vaikka niitä ei olisi tähän suunniteltu. Kyseinen keskustelu, tietojen vaihtaminen ja muokkaaminen saattaa johtaa epäluotettavaan dataan ohjelmistojen välillä. Siirrettävän, muokattavan sekä uuden data luonnissa tuleekin ottaa huomioon jo alussa datan mahdolliset eri formaatit, niiden kytkemiset toisiinsa esimerkiksi samalla tunnistavalla tekijällä sekä datan ja kummankin tai kaikkien ohjelmistojen skaalautuvuus, luotettavuus ja saatavuus. Myös automatisoitujen tehtävien tai töiden ajoitusten suunnittelu on tärkeää, mikäli jokin ohjelmisto lisää tai muokkaa käsiteltävää dataa samalla hetkellä kuin sitä muokataan integraation puolella, aiheuttaa se ikäviä virheitä ja epäluotettavaa dataa. Edellä mainitut tekijät voivat esimerkiksi muuttua, kun organisaation tai yrityksen sisällä oleva data muuttuu niin, ettei integraatio enää pysty kontrolloimaan tai käyttämään aiempia yhdistäviä tekijöitä datan käsittelyyn. Integraatiosta vastaavaa yritystä on aina konsultoitava, kun lähdetään tekemään suurempia muutoksia tai datan muoto tai saatavuus vaihtuu. Integraatio on kuitenkin rakennettava sekä suunniteltava niin, ettei integraation julkaisun jälkeen siihen tarvitse tehdä jatkuvasti muutoksia, vaan se pystyy käsittelemään muuttuvaa dataa tiettyyn pisteeseen asti. (Microsoft 2003d.)

2.3.2 Ohjelmistointegraation toimintojen rajaus

Tiedon lähetys ohjelmistosta toiseen, sen muokkaaminen ja esimerkiksi rajapintojen kutsuja käytettäessä on hyvä ottaa huomioon integraation rajaus. Rajaus kannattaa tehdä sovittujen töiden, tehtävien tai tietojen siirron perusteella, mutta varsinkin rajapintoja käytettäessä on hyvä muistaa rakentaa pohjalle skaalautuva ja helposti muokattava pohja. Pohja voi olla esimerkiksi rajapintakutsujen luomiseen tarkoitettu funktio, joka voi kasata valmiiksi kutsua varten osoitteen pyynnön lähettämiseen, hoitaa tunnistautumisen kasauksen sekä muuttaa lähetettävän kutsun oikeaan muotoon. Ennakoimalla ja hyvin suunnittelemalla saadaan hyvä pohja integraatiolle, joka on valmis muutoksiin sekä lisäyksiin, jos niitä vaaditaan. (Microsoft 2003b.)

Ohjelmistointegraatiot voivat kasvaa nopeasti monimutkaisiksi kokonaisuuksiksi, kun toimitaan usean ohjelmiston parissa. Dataformaatit, yhteydet ohjelmistojen välillä, reititys ohjelmistojen välillä, toimintojen ajastus ja muutettava data ovat kaikki elementtejä, jotka levittyvät useisiin eri käyttöjärjestelmiin. Kaiken edellä mainitun hallitsemiseksi on kannattavaa kehittää samalla integraatioon järjestelmän hallintaa, tämä voi olla vähäpätöinen ilmoittamiseen tai varoittamiseen tarkoitettu funktio, joka ilmoittaa mahdollisista vakavista virheistä tai datan puuttumisen ohjelmoijalle esimerkiksi sähköpostiin tai lokitiedostoon. (Hohpe 2003.)

2.4 Rajapinnat

Rajapintoja on olemassa useita erityyppisiä, näitä kuitenkin voidaan käyttää yhdessä jonkun palvelun tai prosessin suorittamiseen. Rajapintoja esimerkiksi mobiilisovelluksissa ja varsinkin sosiaalisen median sovelluksissa käytetään paljon. Esimerkkinä kuvan julkaisu tai lähetys sosiaalisen median alustalle; ensimmäisenä kutsutaan laitteen kamera rajapintaa kuvan ottamiseen, tämän jälkeen käytetään mahdollisesti laitteen sovelluskirjastosta löytyvää kuvanmuokkaus tai katseluohjelman rajapintaa, jonka kautta voidaan esimerkiksi lisätä tekstiä, säätää värejä tai kuvan kokoa. Seuraavaksi otetaan

yhteys sovelluksen palvelimeen, johon mahdollisesti muokattu kuva lähetetään etärajapintaan kutsulla. Onnistuneen kutsun seurauksena palvelimella oleva sovellus julkaisee kuvan näkyviin muille käyttäjille. (Lauret 2019A.)

Yksinkertaisen kuvan julkaisuun siis kerettiin jo käyttää useampaa rajapintaa ja niitä tuleekin käytettyä normaaleissa jokapäiväisissä toiminnoissa esimerkiksi älypuhelinta käyttäessä useasti. Rajapintoja voidaankin nimittää eräänlaisiksi käyttöliittymiksi, joilla voidaan tehdä kutsuja sovelluksiin tai ohjelmistoihin ja täten käyttää niiden toiminnallisuuksia tai ominaisuuksia.

Rajapinnat antavat käyttöliittymän kahden eri järjestelmän, kohteen, sovelluksen tai organisaation välille, jolloin nämä voivat keskustella keskenään. Rajapintoja voidaankin pitää käyttöliittyminä toisilleen, kun käyttäjä selaa ja valitsee ohjelman käyttöliittymästä jo aiemmin esimerkkinä käytetyn kuvan lisäyksen, toimivat rajapinnat samoin keskenään. Rajapinta A lähettää viestin rajapinnalle B, että se on onnistuneesti ottanut kuvan ja sitä halutaan nyt muokata. Rajapinta B ottaa kutsun vastaan ja avaa omat ohjelmalliset toimintonsa, ottaa vastaan kuvan ja sallii sen muokkaamisen, tämän jälkeen rajapinta B kertoo etärajapinnalle, että nyt kuva on muokattu ja se halutaan julkaista. Palvelimella oleva ohjelma hyväksyttää kutsun ja julkaisee kuvan. (Lauret 2019B.)

2.5 Tunnistautuminen rajapintayhteyksissä

Rajapinnat vaativat yleisesti tunnistautumisen ennen kutsun varsinaista käsittelyä. Tunnistautumisen protokollia ja metodeja on monia, tässä yhteydessä käydään läpi vain muutaman tunnetuimpia sekä suosituimpia metodeja.

2.5.1 HTTP Basic

HTTP Basic-tunnistautuminen. Basic tunnistautuminen on haavoittuvainen verrattuna muihin tunnistautumiseen tarkoitettuihin metodeihin, se on kuitenkin suoraviivaisin ja täten helpoin tunnistautumismenetelmä. (RestCase 2019.) Tunnistautuminen aloitetaan, kun käyttäjä lähettää pyynnön salattuun alueeseen, käyttäjältä kysytään käyttäjänimeä sekä salasanaa. Tunnusten syötön jälkeen otetaan käyttäjänimi sekä salasana ja asetetaan näiden väliin kaksoispiste (käyttäjä:salasana).

Muodostettu merkkijono tämän jälkeen kooditetaan Base64 menetelmällä ja lähetetään uusi pyyntö. Base64 koodaus on algoritmi, joka muuntaa minkä vain merkkijonon tai merkin ensin jokaisen merkin binääriluvuksi ja tämän jälkeen muuntaa binääriluvut Base64 merkkitaulukon mukaisesti omaksi merkikseen.

Esimerkkinä ”`käyttäjä:salasana`” merkkijonosta tulee Base64 koodauksen jälkeen: a8OkeXR0w6Rqw6Q6c2FsYXNhbmEK –merkkijono. Kyseinen algoritmi ei missään tapauksessa sovellu herkkäluontoisten tietojen kryptaukseen eikä sitä tulisi sellaiseen käyttää. (Base64Guru 2020.)

Mikäli tunnus sekä salasana vastasivat sallittuja kirjautumistunnuksia, päästetään käyttäjä eteenpäin. Ongelmana tässä tavassa on, että Base64 koodattu merkkijono on helppo purkaa ja selvittää lähetetyt tunnukset. Basic tunnistautumista ei siis kannata käyttää semmoisenaan, mikäli sivustoa ei ole suojattu TLS/SSL-protokollalla. (Sayer 2002.)

SSL on protokolla, joka validoi verkkosivun tai yrityksen sertifikaateilla. Jokainen sertifikaatti sisältää yksityisen sekä julkisen avaimen, jonka avulla voidaan varmentaa, että sivusto on se, joksi itseään väittää. Sertifikaatin luonnin jälkeen voidaan yksityisavaimella allekirjoittaa useita digitaalisia dokumentteja kuten verkkosivuja. (SSL.com 2021).

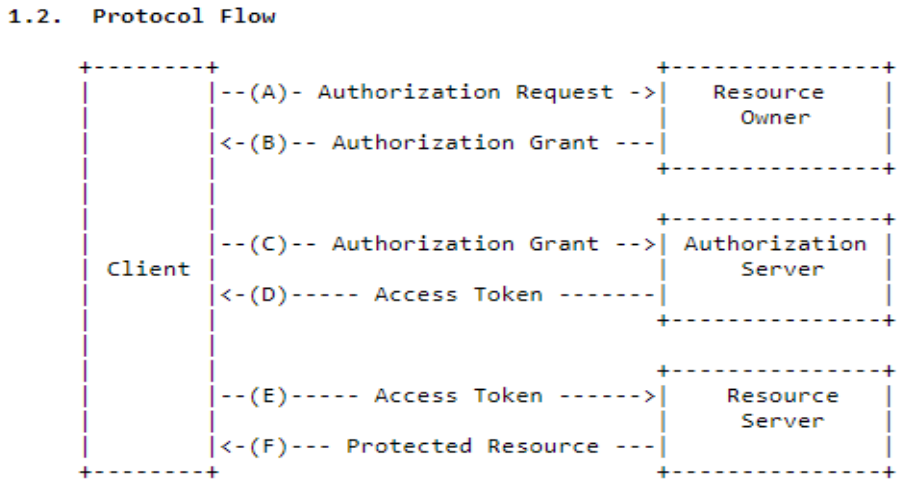
2.5.2 API-avaimet

API-avaimet, eli käyttäjä kohtaiset rajapintakäyttöön luotu ja annettu salausavain. Rajapintatunnistautuminen hoidetaan joko asettamalla annettu avain suoraan rajapintaan lähettävään kutsuun, pyynnön otsikkoon tai evästeenä kutsun mukaan. API-avainten on tarkoitus olla salaisia, jonka vain pääteohjelma sekä palvelin tietävät. API-avainten käyttö on yleistä ohjelmistoissa ja sovelluksissa, joissa on lukuisia käyttäjiä. (Swagger 2020.) Esimerkkinä Googlen omistava ja kehittämä Google Maps palvelu, jolla voidaan hakea paikkoja, reittejä sekä osoitteita. (Google 2020.)

API-avainten käytöllä voidaan myös tarkastella lähetettyjen pyyntöjen sekä toimintojen määrää ja laatua sekä kohdentaa tämä tiettyyn käyttäjään avaimien avulla, käyttäjillä voi olla myös useita API-avaimia samalla käyttäjällä. (Microservice API Patterns 2020).

2.5.3 OAuth 2.0

OAuth 2.0 on erittäin yleinen protokolla käyttöoikeuksien todentamiseen. OAuth 2.0 määrittää neljää roolia, joiden avulla muodostetaan kirjautumiseen tarvittu käyttöoikeus (Access Token).



Kuvio 1. OAuth 2.0 protokollan mukainen tunnistautuminen (IETF 2020.)

Kuviosta 1 voidaan nähdä protokollan toimintamalli. Neljästä roolista ensimmäinen resurssin omistaja, joka omistaa palvelun tai tunnistautumiskäytännön. Resurssin omistaja voidaan kuvata henkilönä, joka on yleinen käyttötapaus. Toisena roolina on resurssi palvelin, kyseinen palvelin sisältää suojatun resurssin tai palvelun. Resurssipalvelin voi hyväksyä ja vastata suojattua resurssia vasten tehtyihin pyyntöihin. Kolmantena roolina toimii sovellus, ohjelmisto tai mikä vain resurssin omistajan puolesta pyynnön lähettävä asia. Tätä resurssia voidaan nimittää itse OAuth-asiakkaaksi, se voidaan myös kuvata sovelluksena, joka lähettää rajapintapyyntöjä. Viimeisenä on itse palvelin, joka myöntää käyttäjälle onnistuneen tunnistautumisen yhteydessä pääsyavaimen. (Oracle 2020.)

2.5.4 OpenID Connect

OpenID Connect on rakennettu OAuth 2.0 protokollan päälle, mutta lisää vain sen päälle tunnistautumistietoja säilyvän kehyksen, jossa tunnistautumistiedot muutetaan uniikiksi ID-tunnukseksi. OAuth tunnistautumispalvelin, joka tukee OpenID Connect metodia, palauttaa ID-tunnuksen pääsyavaimen kanssa. ID-tunnus on JWT-tekniikalla (JSON web token) muodostettu tunnus, joka siirtää tunnistautuneen käyttäjän tiedot tunnistautumispalvelimelta pyynnön lähettävän sovelluksen käytettäväksi. OpenID Connect mahdollistaa yhden tunnuksen käyttämisen useammassa palvelussa, esimerkkinä kirjautuminen Google-tunnuksien avulla palveluun tai sivustolle, johon tiliä tai tunnuksia ei ole vielä luotu. Edellisessä esimerkkitapauksessa Google toimii autentikoivana osapuolena, joka tarjoaa OpenID tunnuksen kirjautumiselle ja myöntää pääsyn jo valmiiksi olemassa olevilla tunnuksilla luotetun osapuolen palveluun tai sivustolle. (Siriwardena 2014.)

2.5.5 Rajapinnan käyttö

Rajapinta protokollia on olemassa useampi. Yleisimpiä näistä ovat REST sekä SOAP protokollan mukaiset rajapinnat. REST ja SOAP protokollat ovat

kummatkin verkkopohjaisia protokollia. Verkkopohjaisilla protokollilla on yhteisenä tekijänä sekä vahvuutena niiden turvallisuus tunnistus protokollien kautta. Vahvuuksia ovat myös palautussanomien ja kyselyiden talletus välimuistiin sekä kyselyiden kuorman tasapainottaminen, ettei palvelin ylikuormittuisi kyselyiden määrästä.

REST eli Representational State Transfer tekee tietojen hakemisesta resurssi tyyppistä. REST-protokollaa käytetään HTTP-rajapinnan mukaisesti, jolloin käytössä on CRUD-komennot, eli Create, Retrieve, Update, Delete. Lisäksi voidaan käyttää esimerkiksi PATCH metodia, kun halutaan muokata resurssin yhtä kenttää. Käyttäjä voi REST protokollan mukaan pyytää rajapinnalta esimerkiksi resurssia ”`käyttäjä`” jonka tunniste on 4, tällöin rajapinta palauttaa käyttäjä 4 resurssin, joka sisältää kaikki kyseisen käyttäjän tiedot.

REST-rajapinta voi myös palauttaa resursseja useammassa muodossa, kuten XML tai JSON. (Patni 2017a.)

```
<message>
    <to>kenelle@osoite.fi</to>
    <from>keneltä@osoite.fi</from>
    <text>
        Tämä on xml tiedosto.
    </text>
</message>
```

Kuvio 2. Esimerkki XML-tiedoston oikeasta rakenteesta.

```
<message>
    <to>kenelle@osoite.fi</to>
    <from>keneltä@osoite.fi</from>
    <text>
        Tämä on xml tiedosto.
</message>
    </text>
```

Kuvio 3. Esimerkki XML-tiedoston väärästä rakenteesta.

XML eli eXtensible Markup Language on tekstipohjainen merkintäkieli, joka on standardi datan siirrossa verkossa. XML-kielessä lähetettävä data identifioidaan

käyttämällä elementtejä. Elementtien väliin voidaan asettaa tietoa, jonka itse elementit identifioivat. XML-tiedoston rakenne tulee aina olla sisäkkäinen. Esimerkki oikean mallisesta rakenteesta voidaan nähdä kuviosta 2 sekä virheellinen rakenne kuviosta 3.

```
<message to=kenelle@osoite.fi from=keneltä@osoite.fi>
    <text>
        Tämä on xml tiedosto.
    </text>
</message>
```

Kuvio 4. Esimerkki XML-tiedoston attribuuteista.

XML-tiedoston elementit voivat olla täysin itse määriteltyjä, mutta kun dataa käytetään useammassa ohjelmistossa, pitää elementtien tarkoitus olla kummallakin tiedossa ja standardoitu siten, että dataa voidaan lähettää ja lukea. XML-tiedostojen elementit voi olla myös attribuutteja (ks. Kuvio 4), jolloin lisätiedot syötetään suoraan elementtien sisään attribuuttina. (Patni 2017b.)

```
{
    "firstname": "Niko",
    "lastname": "Ainali",
    "subject": "Opinnäytetyö",
}
```

Kuvio 5. Esimerkki JSON rakenteesta.

JSON eli JavaScript Object Notation on kevyt tekstipohjainen avoimen standardin tiedostomuoto, jota on helppo lukea. JSON-tiedostomuoto on kehitetty alun perin helppolukuiseksi ja siten että myös ihminen pystyy lukemaan sitä suoraan. Kuviosta 5 nähdään esimerkki JSON-rakenteesta. Standardi on tehty Javascript-kielen pohjalta sekä se on ohjelmointikielistä riippumaton. JSON-rakenne on helppo ja suoraviivainen, vaikkapa verrattuna edelliseen XML-rakenteeseen. Rakenne on suoraviivainen ja yleensä avain – arvo pari on ainoa asia mitä esitetään. JSON-tiedostomuotoa käytetään monesti rajapintojen datan välityksessä. Huonoja puolia JSON-tiedostomuodossa on sen yksinkertaisuus, datasta on vaikea lukea minkä laatuista dataa ollaan

käsittelemässä, koska JSON-tiedostoon ei voida laittaa mukaan kommentteja. (Json.org 2021.)

SOAP eli Simple Object Access Protocol on viestipohjainen protokolla, jossa rajapinnan käyttö tapahtuu XML-tiedostoilla. SOAP protokolla tukeutuu vahvasti XML-standardiin, jossa lähettävä tieto kasataan dokumenttiin XML-kielen mukaisesti. (Tutorialspoint 2021.)

XML-viestinnän kohteena voi olla mitä vain, esimerkiksi ostotilaus, kysely verkkokaupan tuotteen varastotilanteesta tai tiedot tilauksista aikaväliltä pp.kk.vvvv – pp.kk.vvvv. XML-viestintä ei ole sidoksissa mihinkään ohjelmistoon, käyttöjärjestelmään tai ohjelmointikieleen, joten sitä voidaan käyttää järjestelmästä riippumatta. Ainoana vaatimuksena on, että lähetetty XML-sanoma on samassa muodossa ja tiedoilla kuin vastaanottava ohjelmisto on sen määritellyt. SOAP-viestintä muodostuu kirjekuoresta(Envelope), johon tarvittavat tiedot annetaan. SOAP-kehyksessä voidaan valinnaisesti käyttää SOAP-ylätunnistetta, jonka sisässä voidaan määritellä esimerkiksi kutsun käsittelytapa, joka sisältää tunnistautumisen, lähetys ja reititystietoja tai suoritettavan kyselyn tapahtuman laadun. Lisäksi kehyksessä tulee olla myös rakenne, joka sisältää XML-tiedoston. (Tidwell, Kulchenko, Snell 2001.)

```
<s:Envelope
xmlns:s="http://www.w3.org/2001/06/soap-envelope">
    <s:Header>
        <m:transaction xmlns:m="soap-transaction" s:mustUnderstand="true">
            <transactionID>1111</transactionID>
        </m:transaction>
    </s:Header>
    <s:Body>
        <n:ostoTilaus xmlns:n="urn:tilausPalvelu">
            <from>
                <person><Esemeralda Esimerkki</person>
                <customerId>1234</customerId>
            </from>
            <order>
                <quantity>1</quantity>
                <item>esimerkki tuote</item>
                <price>199,99</price>
                <currency>euro</currency>
            </order>
        </n:ostoTilaus>
    </s:Body>
</s:Envelope>
```

Kuvio 6. Esimerkki SOAP-kehyksen rakenteesta.

Kuviosta 6 nähdään ostotilauksen lähetys SOAP-kehyksellä. Vastaanottava ohjelmisto joka osaa lukea SOAP-kutsuja, pystyy lukemaan tiedoston määrittelemän ostotilauksen ja tekemään siitä tilauksen kohteena olevaan järjestelmään. Riveiltä 10–21 luetaan tiedot tilaukselle, jossa tilaajana Esmeralda Esimerkki, asiakasnumerolla 1234 tilaa yhden kappaleen esimerkki tuotetta hintaan 199,99 euroa.

2.6 Suorat tietokantayhteydet

Vanhemmat ohjelmistot eivät välttämättä tarjoa laisinkaan rajapintoja ohjelmisto integraatioihin, joko näiden toteutus ei ole teknisesti mahdollista tai se on liian kallista verrattuna ohjelmiston uudelleenrakentamiseen. Syitä voi olla monia, mutta näihinkin ohjelmiin integraation tekeminen on yleensä mahdollista.

Suoralla tietokantayhteydellä otetaan yhteys ohjelmiston relaatiotietokantaan, johon ohjelmistoon syötetty tieto tallennetaan. Tällä metodilla on kuitenkin riskinsä, kun puhutaan suorista yhteyksistä tietokantaan. Suorissa tietokantaintegraatioissa on suurempi mahdollisuus syöttää tai päivittää väärää tietoa väärään paikkaan, poistaa tai jopa hajottaa tietokannan sisältämät tiedot. Ymmärrys projektin luonteesta, käsiteltävästä datasta, vaatimuksista ja datan muokkaamisesta tai päivittämisestä on oltava kunnossa kummallakin osapuolella, niin asiakkaalla kuin toteuttavalla kehittäjällä tai muilla mahdollisilla kolmansilla osapuolilla. Mikäli asiakas ei ole esimerkiksi osannut kertoa tarpeeksi tarkkaan tarvettaan ja toteuttava osapuoli on lähtenyt tekemään integraatiota näillä tiedoilla, on riskinä aiemmin mainittu datan korruptoituminen tai väärät tiedot asiakkaan ohjelmistossa. (Galin 2018.)

Suora yhteys antaa pääsyn kaikkeen dataan, mutta ei itsessään sisällä tai noudata logiikkaa, joka taas alkuperäisessä ohjelmassa on suunniteltu ja testattu noudattamaan tarkkaan tiettyä logiikkaa. Relaatiot eri taulujen välillä joudutaan tutkimaan tarkkaan ja ottamaan huomioon, mikäli tietoa syötetään ohjelmallisesti tietokantaan integraatiosta käsin.

Esimerkkinä ”Testaaja”-käyttäjän tiettyjen tietojen muokkaaminen ”käyttäjät”-taulussa voi vaatia muokkaamisen myös tauluissa ”käyttäjä_myynnit”, ”käyttäjä_oikeudet” sekä ”käyttäjä_suunnitelmat”, mikäli nämä sisältäisivät vierasavain tiedon ”käyttäjät”-taulusta. Vierasavaimilla voidaan luoda relaatioita tietueiden välillä ja pystytään määrittämään niiden pääasiallinen omistaja tai tekijä (DeBarros 2018).

Esimerkkinä vierasavainten käytöstä seuraava skenaario: Käyttäjä tunnisteella 12 luo uuden suunnitelman, hänen tietoja vastaan tarkistetaan onko hänellä tarvittavia oikeuksia luoda uusi suunnitelma, jos oikeudet on luodaan ”käyttäjä_suunnitelmat”-tauluun uusi tietue jossa vierasavaimena ”käyttäjä”-taulusta käyttäjän tunniste, joka tässä tapauksessa 12. Äskeisen tapahtuman johdosta voidaan nyt tarkistaa käyttäjän 12 alla olevat suunnitelmat ja esimerkiksi antaa vain kyseisen käyttäjän muokata suunnitelmaa. Mikäli vierasavainta ei kyseiseen suunnitelma tietueeseen jostain syystä tulisi tai sitä ei olisi otettu huomioon integraation tuomissa tiedoissa, ei kukaan pystyisi muokkaaman kyseistä tietuetta.

Suorissa relaatiotietokanta yhteyksissä käytetään SQL-komentoja. SQL eli Structured Query Language on IBM:n kehittämä kieli, jolla relaatiotietokantoihin voidaan tehdä erilaisia toimintoja, kuten lisäyksiä, poistoja tai päivityksiä. Erinäisten SQL-komentojen suorittamiseen tarvitaan käyttäjätunnus sekä salasana, jotka on luotu kyseistä kyselyjä vastaanottavaa tietokantaa vasten. Käyttäjätunnuksilla on tietty valtuutus, jonka mukaan kyseinen käyttäjä voi tehdä erilaisia kyselyitä. Käyttäjä, jolla on esimerkiksi valtuudet käyttää INSERT, SELECT ja UPDATE komentoja, nämä sallivat käyttäjän lukea, syöttää tai päivittää rivejä sekä kenttiä tietokantataulussa. Valtuuksilla voidaan myös myöntää tietokantoihin erikseen oikeuksia, mikäli tietokantaympäristössä näitä on useampi. (Beaulieu 2020.)

3 Projekti ja toimeksianto

3.1 Toimeksiantajan vaatimukset

Projektin toimeksiantajana toimii Skycode Oy. Skycode Oy halusi lisätä ohjelmistointegraatioiden määrää heidän kehittämäänsä Skyshop-nimikkeellä toimivaan business to business verkkokauppaan kilpailukyvyn parantamiseksi. Ohjelmistointegraation toteutus haluttiin tehtävän Netvisor taloushallinnon ohjelmistosta. Netvisor on Visma Solution Oy:n kehittämä laaja taloushallinnon ohjelmisto, jonka asiakkaina on pk-yrityksiä (pieni ja keskisuuria yrityksiä) Suomessa. Netvisor tarjoaa rajapinnan, jonka kautta ohjelmistointegraatio tehdään.

Toimeksiantajan vaatimuksena on kehittää Skyshop-verkkokauppaan integraatio Netvisor-taloushallinnan ohjelmistosta. Pääosassa ovat tuotteiden ja asiakkaiden vieminen verkkokauppaan sekä tilausten luominen verkkokaupasta Netvisor-ohjelmistoon rajapinnan kautta.

Toimeksiantajan yhtenä vaatimuksen on, että tuotteiden saldot, hinnat, alennukset sekä arvonlisäverojen tulisi päivittyä verkkokauppaan Netvisorista. Edellä mainittujen päivityksien tulisi tapahtua tunnin välein, jolloin tarkastetaan saldojen, hintojen, alennusten sekä mahdollisten arvonlisäverojen muutokset. Päivitysten tarkistuksen tulisi olla myös käynnistettävissä manuaalisesti. Arvonlisäveron muuttuminen Netvisor taloushallinnan järjestelmässä on harvinaisempi tapahtuma, mutta myös mahdollinen. Asiakkaiden tiedot tuodaan Netvisorista verkkokauppaan, tiedot käsittävät asiakkuuden yritystiedot, kuten toimipaikan, osoitteen, asiakasryhmän sekä y-tunnuksen. Asiakastietueisiin tuodaan Netvisorista mukana myös mahdolliset kontaktit, joille voidaan esimerkiksi toimittaa mahdolliset tilaus ja lähetysvahvistukset.

Asiakasryhmittäiset alennukset tulisi tuoda ja ottaa myös huomioon, mikäli sellaisia on asetettu käyttäjän Netvisor ohjelmistoon. Asiakastiedoista ei

automaattisesti luoda lähetettäviä tunnuksia, vaan tämä tapahtuu verkkokaupan hallintapaneelin kautta. Verkkokaupan pääkäyttäjä voi itse valita kenellä on oikeus tehdä tilauksia verkkokaupasta ja mitä tuotteita he näkevät. Myyntitilaukset Netvisoriin tulisi luoda automaattisesti asiakkaan tilauksen yhteydessä, tuotetiedot, määrät ja hinnat kerätä riveittäin sekä ottaa huomioon mahdolliset alennukset. Asiakastiedot lisätään myyntitilaukseen ja lähetetään rajapinnan kautta Netvisor-ohjelmaan.

3.2 Projektin suunnittelu

Projektin suunnittelu aloitettiin kartoittamalla toimeksiantajan kanssa tarpeelliset toiminnot listaksi ja jaoteltiin näistä pääprioriteetit. Toimintoja käytiin läpi useampaan kertaan, jotta saatiin esille tärkeimmät toiminnot ja tehtävät integraatiossa. Tuote, asiakas ja myyntitilaus tiedot ovat pääroolissa verkkokaupan toteutuksessa ja näihin osa-alueisiin keskityttiin eniten suunnittelupalavereiden aikana. Netvisor-rajapinnan kautta ovat kaikki edellä mainitut resurssit saatavilla sekä suodatettavissa useilla eri tiedoilla kuten päivämäärät, jolloin resurssit on luotu. Itse tiedon hakeminen rajapinnan kautta ja sen tuominen ohjelmistoon ei ole suurin työ, vaan sen sovittaminen sekä jaottelu käytettäväksi jo valmiiseen verkkokauppaan sekä sen tietokantatauluihin. Suunnittelu vaiheessa tuli jo selväksi, että kyseisen integraation aikana tulisi luoda taulut ”`netvisor_`”-etuliitteellä, koska edelliset kauppaa varten luodut tietokantataulut eivät täysin taivu Netvisorista tulevien tietojen käsittelyyn ja tallentamiseen.

Kauppaan tulee tallentaa ja hakea sellaista dataa tilauksista, että niistä voidaan rajapinnan kautta muodostaa tilaus, jossa asiakkaan tilaamat tuotteet, tiedot sekä toivomukset laskutus ja toimitusosoitteista. Rajapinnan kautta käytettävissä olevaa logiikkaa pyritään hyödyntämään mahdollisimman paljon, pääohjelmana tulisi pysyä Netvisor ja tämän kautta hallita niin asiakkuuksia, kuin tuotteiden hintoja tai muita tietoja.

4 Ohjelmistointegraation toteutus

Tässä luvussa tarkastelemme, kuinka integraatiota lähdetään toteuttamaan. Toteutuksen aikana käydään läpi tarvittavia työkaluja sekä tekniikoita sekä näiden yhdistämistä toimivaksi ohjelmistointegraatioksi. Integraatio toteutetaan kahden jo olemassa olevan ohjelmiston välille. Skycoden kehittämään Skyshop-verkkokauppaan saadaan tehtyä muutoksia ja lisäyksiä tarpeen mukaan, mutta rajapintayhteydellä olevaan Netvisor-taloushallintajärjestelmään ei voida tehdä muutoksia, vaan käyttää jo olemassa olevia rajapintakutsuja ja resursseja. Rajapintojen käyttäminen integraatioissa on yksi toteutuksen kulmakivistä ja tätä käydään läpi tarkemmin omissa alaluvuissaan.

4.1 Tarvittavat tekniikat sekä työkalut

Projektia varten tarvitaan lukuisia eri tekniikoita sekä työkaluja. Seuraavassa osiossa käydään läpi tarvittavat tekniikat ja työkalut sekä mihin tarkoitukseen niitä on käytetty projektissa

Skyshop on rakennettu CakePHP-ohjelmistokehyksen päälle, CakePHP on MVC-arkkitehtuurinen kehys PHP-ohjelmointikielen päälle. (Cake Software Foundation. 2020a.) CakePHP tarjoaa paljon hyödyllisiä ominaisuuksia ja nopeuttaa ohjelmistokehitystä huomattavasti.
PHP on ohjelmointikieli, joka sopii erinomaisesti web-kehitykseen. Toiminnot PHP-kielessä ajetaan aina suoritusvaiheessa, eli esimerkiksi sivulautauksen yhteydessä (The PHP Group 2020a). PHP-pakettien hallintaan käytetään Composer nimistä ohjelmaa. Composer on paketinhallinta PHP-ohjelmointikielellä tehtyihin projekteihin. Composerilla hallitaan ohjelmistopaketteja ja ne voidaan helposti päivittää uusimpaan versioon tai pitää ne halutussa käyttäjän määrittelemässä versiossa (Composer. 2021a).
Datan tallentamiseen ja käyttämiseen käytetään MySQL-tietokantaa. MySQL on relaatiotietokanta, joka tarjoaa nopean ja turvallisen tietokannan projektiin (Kromann 2018a). Versionhallintaan käytetään Git ohjelmistoa. Git on ilmainen

versionhallintajärjestelmä ja sitä käytetään projektissa muutoksien tarkasteluun, versiointiin sekä tuotanto- ja kehityspuolen koodin säilömiseen (Daityari 2020a).

4.1.1 CakePHP

CakePHP on ohjelmistokehys, joka on rakennettu PHP-kielen ympärille. Ohjelmistokehyksellä tarkoitetaan rakennetta, joka tukee, lisää tai auttaa toiminnallisuutta alkuperäiseen rakenteeseen (WhatIs 2020). CakePHP seuraa MVC eli Model, View, Controller (Malli, Näkymä, Kontrolleri) tyyppistä ohjelmistoarkkitehtuuria, joka erottelee käyttöliittymän ja sisäiset tietueet sekä datan, jolla tietoa esitetään ja hyväksytetään käyttäjällä. Model eli malli kuvaa miten järjestelmän tietoa tallennetaan, käsitellään ja ylläpidetään. View eli näkymä määrittelee mitä tietoa esitetään käyttöliittymässä sekä miltä käyttöliittymän ulkoasu näyttää. Controller eli kontrolleri tai ohjain vastanottaa tietoa ja käskyjä käyttäjältä ja vaihtaa tai muuttaa näkymää käyttäjän käskyjen mukaisesti. CakePHP MVC pohjainen lähestymistapa tekee ohjelmistosta modulaarisen, ylläpidettävän ja tarjoaa nopean kehitysympäristön. Erittelyn avulla voidaan tehdä muutoksia haluttuun ohjelmiston kohtaan ilman, että sillä on vaikutusta muihin ohjelmiston osiin (Cake Software Foundation 2020).

CakePHP sisältää myös paljon hyödyllisiä ominaisuuksia kuten koodin generointi työkalun, jolla voidaan luoda koodipohjia helposti komentoriviltä, eikä perusohjelmiston pohjatoimintoja, kuten lisäys, poisto tai muokkaus funktiota tarvitse itse kirjoittaa. CakePHP:n ”`bake`”-komennolla voidaan esimerkiksi luoda kaikki tarvittava valmiiksi tai erotella kontrolleri, näkymä tai malli tasolla, johon tiedot otetaan valitusta tietokannan taulusta.

```
root@Sintti:/var/www/html/sintti# php bin/cake.php bake all Testi
Bake All
---------------------------------------------------------------------------
One moment while associations are detected.

Baking table class for Testi...

Creating file /var/www/html/sintti/src/Model/Table/TestiTable.php
Wrote `/var/www/html/sintti/src/Model/Table/TestiTable.php`

Baking entity class for Testi...
```

Kuvio 7. Esimerkki CakePHP bake komennosta.

Kuviosta 7 nähdään ”bake”-komennon toiminto, kun sille annetaan argumentiksi ”all” sekä taulun nimi valinnaksi. ”bake all” luo kaikki tarvittavat osat kuten mallin, kontrollerin, näkymät sekä testit. ”Bake”-komennolla voidaan myös luoda yksittäisiä osia antamalla ”bake”-komennolle parametriksi haluttu asia, joka halutaan luoda sekä taulun nimi. Esimerkiksi käyttäjätaulun kontrollerin luominen onnistuu ”bake controller users”-käskyllä.

```
Available Commands:

Bake:
 - bake all
 - bake behavior
 - bake cell
 - bake command
 - bake component
 - bake controller
 - bake controller all
 - bake fixture
 - bake fixture all
 - bake form
 - bake helper
 - bake mailer
 - bake middleware
 - bake model
 - bake model all
 - bake plugin
 - bake shell
 - bake shell_helper
 - bake task
 - bake template
 - bake template all
 - bake test

To run a command, type `cake command_name [args|options]`
To get help on a specific command, type `cake command_name --help`
```

Kuvio 8. Saatavilla olevat ”bake”-komennot.

Kuviosta 8 nähdään kaikki saatavilla olevat ”bake”-komennot, joita voidaan käyttää esimerkiksi komponenttien, testien sekä lisäosien luomiseen. ”bake”-komennolla voidaan automatisoida tarvittavien osioiden luominen ja täten nopeuttaa ohjelman kehittämistä.

4.1.2 PHP

PHP eli rekursiivisesti PHP: Hypertext Preprocessor on avoimen lähdekoodin ohjelmointikieli, joka on kehitetty erityisesti web-kehitykseen ja sitä voidaan sisällyttää HTML-kielen sekaan. PHP-kielellä toteutetut komennot ajetaan ohjelman suoritusvaiheessa, kuten sivulatauksen yhteydessä. PHP-kieltä voidaan myös käyttää komentoriviltä suoritettavien tiedostojen ajamiseen. PHP ei ole riippuvainen käyttöjärjestelmästä ja sitä voidaan käyttää kaikissa yleisimmissä käyttöjärjestelmissä. PHP käyttää useita ohjelmointiparadigmoja, OOP eli olio-ohjelmointia, proseduraalista ohjelmointia tai näiden sekoitusta näin halutessaan. PHP yksi tärkeimmistä ominaisuuksista on laaja yhteensopivuus erilaisiin tietokantoihin, ominaisuuksiin sisältyy myös eri protokollien käyttö sekä tekstin käsittely. PHP-kielessä käytetään muuttujan merkkinä dollarimerkkiä (`$`) (The PHP Group 2020b).

4.1.3 Composer

Skyshop eli verkkokauppa on rakennettu CakePHP-sovelluskehyksen päälle. CakePHP tarvitsee paketinhallintaan Composer nimisen paketinhallintajärjestelmän, joka mahdollistaa ohjelmistopakettien ja lisäosien asentamiseen sekä päivittämiseen. Composer lähtökohtaisesti hallitsee paketteja projektin kansiotasolla, mutta sitä voidaan käyttää myös muissa projekteissa käyttämällä ”`global`”-komentoa. Composer tarvitsee alleen PHP 5.3.2 versiota uudemman PHP-asennuksen. Composer tarkastelee projektin ”`composer.json`”-tiedostoa, johon on listattuna projektiin tarvittavat paketit sekä asetukset mistä projektin tiedostoja luetaan (Composer. 2021b).

Kuvio 9. Esimerkki composer.json tiedostosta ja sen sisällöstä.

Composer lukee kansion sisältä tiedoston ja asentaa siinä annetut paketit tiedoston parametrien mukaisesti. Kuviosta 9 voidaan nähdä, kuinka vaadittuina paketteina on listattu PHP, joka on sama tai yli version 5.6 (>=5.6). CakePHP-paketti, josta voidaan asentaa uusin versio, kunhan pysytään 3.8 version sisällä (3.8.*). Pakettien versioiden asennuksia voidaan myös lukita suoraan ennalta tiettyyn versioon, kuten vaikkapa pakettiversioon CakePHP 3.8.1, jolloin versio pysyy aina samana, eikä päivity.

Antamalla komennon ”`composer install`” lukee Composer tarvittavat paketit projektiin ”`composer.json`”-tiedostosta ja lähtee hakemaan paketteja niiden virallisilta palvelimilta. Paketit ladataan ja asennetaan yksi kerrallaan. Haettavien pakettien versioita säätelee aiemmin mainittu ”`composer.json`”-tiedosto, johon voidaan määrittää versio numero.

4.1.4 MySQL

MySQL on relaatiotietokanta, joka on joustava valinta tietokannaksi. MySQL tukee 14 eri käyttöjärjestelmää, tarjoaa rajapintoja suosituimmille ohjelmointikielille kuten Java, C, C++ ja PHP. MySQL sisältää 35 eri merkistöä, joilla voidaan kontrolloida dataa, kuinka sitä lajitellaan ja säilötään tauluihin. MySQL on nopea ja turvallinen vaihtoehto tietokannalle. Tietokantakäyttäjillä voidaan esimerkiksi säädellä mihin tietokantaan voidaan ottaa yhteys, suorittaa

kyselyitä, muokata tai poistaa dataa. (Kromann, F. 2018b.)

4.1.5 Git

Git on ilmainen ja avoimen lähdekoodin versionhallintajärjestelmä, jolla voidaan päivittää ja muuttaa tiedostoja sekä palata aikaisempiin versioihin, mikäli niitä halutaan tarkastella tai käyttää uudelleen (Daityari 2020b).

Git-versionhallintajärjestelmää voidaan käyttää joko komentoriviltä tai ohjelmistona. Ohjelmistoja on useita ja ne ovat pääasiassa ilmaisia käyttää, ilmaiseksi käytettäviä ohjelmistoja ovat muun muassa SourceTree, GitHub Desktop sekä TortoiseGit. Ohjelmistot tuovat graafisen käyttöliittymän Git-versionhallintajärjestelmän käyttöön ja sisältävät erilaisia toimintoja versionhallinnan käytön helpotukseksi. Git sisältää itsessään graafisen käyttöliittymän, jonka saa näkyviin ”`git gui`”-komennolla, eikä vaadi lisäksi muita asennuksia (Git-scm 2020).

Komentoriviltä ohjelmistovarastoa haettaessa käytetään komentoa ”`git clone https://xxxx@bitbucket.com/työtila/repositorio.git`”, jossa ”`xxxx`” on käyttäjänimi, ”`bitbucket.com`” on webpohjainen palveluntarjoaja, jossa voidaan ylläpitää repositoryja (ohjelmavarastoja). ”`työtila`” on työtila, jonka alla voi olla useampia repositoryja ja ”`repositorio.git`” on itse repository. Kyseisessä komennossa haetaan repository bitbucket.com palvelusta. Onnistuneen kopioinnin seurauksena tuo versionhallintajärjestelmä kansion, joka sisältää repositoryn tiedostot. Git-järjestelmä tuo myös ”`.git`”-kansion. Kansio ”`.git`” sisältää tarvittavat tiedot projektin ja tiedostojen versionhallinnasta, tiedot aiemmista muutoksista sekä osoitteen johon muutokset tulee lähettää.

Git-versionhallintajärjestelmällä voidaan ylläpitää ohjelmasta eri haaroja, tarkoittaen että julkisena ja valmiina ohjelmistona pidettyä versiota pidetään haarassa ”`master`” tai ”`main`” riippuen palveluntarjoajasta. Muissa haaroissa, kuten yleisimmin ”`developement`” tai ”`dev`” haarassa pidetään työstettävää ja

kehitettävää versiota. Haaroja voidaan myös käyttää eri ohjelmistoversioiden ylläpitämiseen haaroilla, kuten ”ohjelma1.0” tai ”ohjelma2.0”. Haaroja voi olla useampia, eikä niille suoranaisesti ole olemassa maksimi määrää, palveluntarjoaja voi kuitenkin rajoittaa haarojen lukumäärää.

Versiota voidaan yhdistää versionhallintajärjestelmän ”merge”-käskyllä eli yhdistys toiminnolla, joka pyrkii automaattisesti yhdistämään tiedoston uuden version vanhaan. Esimerkiksi jos tiedoston riviltä 32 on korjattu kirjoitusvirhe ”eplooginen”, sanalla ”epälooginen” kehitysversiossa, pyrkii versionhallintajärjestelmän merge-toiminto havaitsemaan muutoksen ja korvaamaan master-haarasta ”eplooginen” sanan uudella versiolla.

Yhdistäminen voi epäonnistua ja aiheuttaa konfliktin tiedostojen välille. Näissä tapauksissa yhdistäminen ei tapahdu ennen kuin käyttäjä on korjannut konfliktin aiheuttaneen tiedoston tai rivin. Mikäli yhdistäminen onnistuu, päivitetään valittu kohde annetulla datalla. Esimerkkinä ”git merge develop” komennon antaminen master-haarassa pyrkii yhdistämään develop-haaran uudet, poistetut tai muuttuneet tiedot master-haaran tietoihin. Tietoja voidaan viedä versionhallintaan syöttämällä muutokseen vietävät tiedostot indeksiin joko yksitellen ”git add esimerkki.txt” komennolla tai kaikki tiedostot, joissa on havaittu muutos ”git add .” komennolla. Kun tiedot on viety indeksiin, voidaan ne siirtää vietäväksi versionhallintaan ”git commit”-käskyllä, siirron yhteydessä voidaan kirjoittaa kuvaus muutoksista antamalla komento ”git commit -m ”Poistettu esimerkki2 virheellisen tiedon vuoksi” ”. Siirtämisen ja siirtoviestin jälkeen voidaan tiedostot lähettää versionhallintaan komennolla ”git push”. Ennen lähettämistä on kuitenkin hyvä tarkastaa, että omassa lokaaliympäristössä on uusin versio tiedostoista. Versionhallinnasta voidaan hakea uusimmat muutokset komennolla ”git pull”. Useampi käyttäjä voi työskennellä samaa haaraa tai repositorya vasten ja tallentaa versionhallintaan tekemänsä lokaalit muutokset. (Git-scm. 2020).

4.2 Verkkokauppan alustus ja projektin aloitus

Projekti aloitetaan hakemalla verkkokaupan pohja Bitbucket repositorysta. Tämä tapahtuu joko käyttämällä Git-versionhallintajärjestelemää tai lataamalla suoraan Bitbucket palvelusta. Verkkokauppapohjan lataamisen jälkeen luodaan uusi repository ja pusketaan sama pohjan sinne. Tällä toimella saadaan tehtyä ensimmäinen versionhallinta projektiin.

Verkkokauppaa lähdetään työstämään tyhjälle palvelimelle, johon on asennettu LAMP-ympäristö eli Linuxista, Apachesta, MySQL:stä sekä PHP:sta muodostuva palveluarkkitehtuuri. Tarvittavien edellä mainittujen tekniikoiden, komponenttien asennuksen sekä lataamisen jälkeen voidaan pystyttää verkkokaupan pohja. Verkkokaupan pohja siirretään palvelimelle ”`skyshop`”-kansion alle. Kyseinen verkkokaupan pohja sisältää valmiiksi hallinnan, josta voidaan lisätä käyttäjiä, tuotteita, kategorioita, tiedostoja, näytettäviä sivuja sekä tarjouksia. Logiikat tuotteiden, omien käyttäjätietojen sekä tuotelistausten näyttämiseen myös sisältyivät verkkokaupan pohjaan. Pääasiallisesti verkkokauppa olisi valmis käytettäväksi, mikäli toiminnanohjausjärjestelmänä olisi Visman kehittämä Nova tai Visma.net ohjelmisto eikä verkkokauppaan haluttaisi mitään muutoksia ulkoasuun.

Opinnäytetyön kohteena on kuitenkin Netvisor-integraatio ja verkkokaupan ulkoasua ja rakennetta halutaan hieman muokata, joten kehitystyö on aloitettava ainakin Netvisorin kohdalta täysin alusta. Suurinta osaa verkkokaupan logiikasta sekä taulumäärityksistä pystytään käyttämään uusiksi, mutta avuksi kuitenkin tarvitaan lisätauluja Netvisor-integraatiota varten. Ensimmäisenä työnä onkin tarkastella Netvisor-rajapinnan palauttamaa dataa Netvisor-rajapinta dokumentaation kautta sekä sen pohjalta aloittaa tarvittavien lisätaulujen ja kenttien suunnittelu.

4.3 Netvisor-integraatioon lisäys ohjelmistoon

Aiemmin toteutetut Nova- ja VismaNET-integraatiot on lisätty verkkokauppaan lisäosana. CakePHP mahdollistaa omien lisäosien luonnin samalla tavalla kuin aiemmin esimerkein esitetty mallin tai kontrollerin luonti. Lisäosa voidaan luoda komentoriviltä käskyllä ”`php bin/cake.php bake plugin Netvisor`”, tämä muodostaa uuden kansion nimellä ”`Netvisor`”-projektin ”`plugin`”-kansion sisään.

```
root@      :/var/www/html/      # php bin/cake.php bake plugin Netvisor
Plugin Name: Netvisor
Plugin Directory: /var/www/html/      ./plugins/Netvisor
------------------------------------------------------------------------------
Look okay? (y/n/q)
[y] > y
Generating .gitignore file...

Creating file /var/www/html/      ./plugins/Netvisor/.gitignore
Wrote `/var/www/html/      ./plugins/Netvisor/.gitignore`
Generating README.md file...

Creating file /var/www/html/      /plugins/Netvisor/README.md
Wrote `/var/www/html/      /plugins/Netvisor/README.md`
Generating composer.json file...

Creating file /var/www/html/      'plugins/Netvisor/composer.json
Wrote `/var/www/html/      /plugins/Netvisor/composer.json`
Generating phpunit.xml.dist file...

Creating file /var/www/html/      /plugins/Netvisor/phpunit.xml.dist
Wrote `/var/www/html/      ./plugins/Netvisor/phpunit.xml.dist`
Generating src/Controller/AppController.php file...

Creating file /var/www/html/      /plugins/Netvisor/src/Controller/AppController
.php
Wrote `/var/www/html/      ./plugins/Netvisor/src/Controller/AppController.php`
Generating src/Plugin.php file...
```

Kuvio 10. Lisäosan generointi bake-komennolla.

Kuten kuviosta 10 nähdään, luo CakePHP automaattisesti tarvittavat tiedostot kansion sisään, lisäosissa on oma App-kontrolleri. App-kontrolleria käytetään pohjana muille kontrollereille, jotka ovat Netvisor-lisäosan sisällä. App-kontrolleri on siis yliluokka muille kontrollereille. App-kontrollerin sisällä on hyvä olla metodit, joita käytetään lapsikontrollerissa, sillä yliluokan metodit periytyvät aina lapselle ja ovat täten käytettävissä myös lapsikontrollereissa.

Lisäosan hyötynä on sen uudelleenkäytettävyys ja kutsuminen pääasiassa missä vain ohjelmiston osassa joko kontrolleri- tai mallitasolla. Lisäosaa voidaan jatkossa myös käyttää muissakin projekteissa, koska CakePHP-lisäosa sisältää itsessään kaiken tarvittavan. Luotu lisäosa voidaan aktivoida käyttöön

lisäämällä rivi ”`Application.php`”-tiedostoon, joka määrittelee mitä lisäosia, reitityksiä, väliohjelmistoja tai komentorivikäskyjä, joita ohjelma voi käyttää.

Lisäosien lisääminen ohjelmistoon onnistuu lisäämällä rivi ”`$this->addPlugin(”xxx”);`”, jossa xxx on lisäosan nimi. Lisäosia voidaan myös lisätä helposti komentoriviltä antamalla komento ”`plugin load LisäOsanNimi`”.

Komentorivillä suoritettava käsky lisää samaan tiedostoon saman ”`this->addPlugin(”xxx”);`” rivin ohjelmiston bootstrap metodiin. Bootstrap metodi sijaitsee Application.php tiedostossa ja se sisältää konfiguraatio tiedot.

Bootstrap on termi, jollaa tarkoitetaan itsestään suoriutuvaa ohjelmaa, joka suorittaa tarvittavat komennot, jotka esimerkiksi hakevat käyttäjätietoja tai lataa tarvittavat paketit käyttöön ilman käyttäjän syötettä (TechTerms 2020).

Ennen jokaista pyyntöä tarkistetaan tarvittavat tiedot bootstrap.php tiedostosta sekä lisäksi bootstrap-metodista, josta voidaan tässä tapauksessa ladata tai poistaa käytöstä lisäosia. Lisäosa on nyt ladattu käyttöön ja siihen voidaan alkaa lisäämään malleja, näkymiä, komentojonotiedostoja, kontrollereita sekä muita integraatioon tarvittavia asioita.

4.3.1 Netvisor-rajapintayhteys

Netvisor rajapintayhteyden käyttö edellyttää Netvisor-tilin. Netvisor-testitilinä toimi Skycode Oy:lle myönnetty testitili integraatioita varten ISV-kumppaniohjelman kautta. Netvisor-tililtä saadaan API-tunnukset ohjelmistorajapintaa varten.

Rajapinta on REST-mallin mukainen ja kommunikointi tapahtuu HTTP-pyynnöillä. Tietoa tuotaessa tulee aineistosta luoda XML-sanoma, joka lähetetään HTTP-pyynnön kanssa Netvisoriin. Tunnistautuminen ja tietoturva on toteutettu rajapinnassa salatun yhteyden eli HTTPS kautta sekä rajapinnalle

lähetettävien tunnistautumistietojen salaaminen tarkistussumman avulla, jolloin niitä ei voida muodostaa pyynnöstä takaisinpäin (Visma Solutions Oy 2020).

Rajapintayhteyden tunnistautumiseen sekä pyyntöihin aloitetaan kasaamaan funktioita, joiden summana on onnistunut pyyntö Netvisor-rajapintaan. Yhteydelle luodaan uusi tiedosto nimeltä ”NetvisorConnection.php”, Netvisor-lisäosan alle Model-kansioon. Luotu tiedosto saa jatkossa toimia mallina kaikille kutsuille, riippumatta siitä missä kutsua tarvitaan. Mallin tulee kerätä tarvittavat tiedot kutsulle, lähettää tarvittaessa xml-pohjainen kysely ja palauttaa vastaus tuloksineen.

Netvisorin ohjelmistointegraatio dokumentissa on valmiita funktioita PHP-kielellä kirjoitettuna, kuten mac-osoitteen laskenta ja lähetettävien otsikoiden kasaus. Näillä valmiilla funktioilla saadaan jo hyvä pohja tunnistautumiseen, mutta kutsujen lähetys helposti tämän mallin kautta vaatii lisäksi itse kyselyn lähetys metodin sekä tarvittavien tietojen luominen kyselyyn mukaan tunnistautumisen onnistumiseksi.

Tunnistautuminen tarvitseekin seuraavat asiat: Otsikkotiedot, jotka sisältävät lähettäjän, Netvisor tilin asiakastunnuksen, Netvisorin kumppaniavaimen, aikaleiman, kielen, organisaation tunnisteen, vuorovaikutustunnisteen sekä MAC-koodin. Edellä mainituista tiedoista kasataan otsakkeet tunnistautumispyyntöön. Lisäksi tarvitaan REST-mallin mukaisesti pyynnön laatu, eli joko GET POST, PUT tai DELETE-komento. Tunnistetiedot talletetaan käytettäväksi `settings.json` tiedostoon, joka sisältää muitakin arkaluontoisia tietoja, kuten tietokannan nimen, käyttäjätunnuksen, salasanan ja osoitteen. Konfiguraatiotiedosto settings.json ladataan käyttöön `bootstrap` tiedostossa komennolla ”`Configure::load(”settings”);`”. Konfiguraatio tiedostoon lisätään uusi listaus Netvisorin rajapinnan tarvitsemista tiedoista ”`.json`” muodossa.

Kuvio 11. Esimerkki konfiguraatio tiedostoon lisättävästä tiedosta.

Kuviosta 11 nähdään, kuinka tarvittavat tunnistetiedot listataan `settings.json` tiedostoon. Näitä tietoja voidaan nyt hakea Netvisor yhteysmalliin ”`Configure::read();`”-funktiolla. Funktio lukee konfiguraatiotiedostosta haetun arvon. Esimerkiksi kuvion 10 ”`NetvisorTest`” listauksen indeksin ”`ApiLanguage`”-arvo saadaan käytettäväksi seuraavasti: ”`Configure::read(″Netvisor.ApiLanguage″)`”, samalla tavalla myös muut indeksin arvot voidaan hakea käyttöön tunnistautumista varten.

4.3.2 Tietojen hakeminen ja vienti Netvisor rajapinnan kautta

NetvisorConnection-mallin sisään aletaan rakentamaan tarvittavia funktiota, joita jo aiemmin käsiteltiin. MAC-koodin, kutsussa lähetettävien otsikoiden ja vuorovaikutustunnisteen luonnit hoidetaan erikseen funktiolla. Pyynnön rakentaminen aloitetaan luomalla ”`doRequest`”-funktio, jota voidaan kutsua lataamalla NetvisorConnection-malli kontrollerissa tai komentojonotiedostossa, josta kutsu halutaan lähettää. ”`doRequest`”-funktio tarvitsee parametrikseen resurssin, tarkemmin tietyn rajapinnan resurssin.

Esimerkiksi asiakaslistan nouto Netvisorista käyttää ”`Customerlist.nv`”-resurssia ja ottaa sisään ”`keyword`”, ”`changedsince`” tai ”`customercodelist`”-parametrit. Kyseisillä parametreilla voidaan määritellä haettavat asiakkaat hakusanan, muutospäivämäärän tai asiakkuuskoodien perusteella. Tietyn parametrin käyttäminen voi jättää muita parametreja

huomiomatta, esimerkiksi mikäli ”keyword”-parametria käytetään kutsussa, ei ”changedsince”-parametria oteta laisinkaan huomioon.

Parametrit syötetään kysymysmerkin(?) jälkeen seuraavasti: ”customerlist.nv?keyword=asiakas”, jossa ”customerlist.nv” on resurssi, ”keyword”-parametri ja ”asiakas” parametrin arvo, jolla listaus suodatetaan. Resurssi sekä parametrit syötetään ”doRequest”-funktiolle ”$url” muuttujaan. Resurssitieto syötetään funktiolle merkkijonona.

”doRequest”-funktion seuraava vaadittu parametri on ”$method”. Parametri sisältää kutsussa käytettävän HTTP-pyyntöjen mukaisen metodin; GET, POST, PUT tai DELETE. Pääasiassa käytettävät metodit ohjelmassa ovat GET sekä POST – metodit.

GET kun halutaan hakea resursseja ja POST kun niitä halutaan luoda tai muokata. Metodi syötetään merkkijonona.

```
$doc = new \DOMDocument('1.0', 'utf-8');
$doc->preserveWhiteSpace = false;
$doc->formatOutput = true;
$root = $doc->createElement('Root');
$root = $doc->appendChild($root);

$contentItemFirst = $doc->createElement('customer');
$contentItemFirst = $root->appendChild($contentItemFirst);
//Start of customer basic information
$contentItemCustomerBase = $doc->createElement('customerbaseinformation');
$contentItemCustomerBase = $contentItemFirst->appendChild($contentItemCustomerBase);
//Asiakastietoelementtien luonti
////...

$contentItem = $doc->createElement('invoicinglanguage');
$contentItem = $contentItemAdditional->appendChild($contentItem);
$text = $doc->createTextNode('FI');
$text = $contentItem->appendChild($text);

$dataToSend = $doc->saveXML($doc->documentElement);
//Lähetetään asiakastieto Netvisorin rajapintaan
$createNetvisorOrder = $this->connection->doRequest('customer.nv?Method=add', 'POST', $dataToSend);
```

Kuvio 12. Lyhennetty esimerkki asiakasdatan kasaamisesta XML-tiedostoon sekä sen lähettämisestä funktiolle.

Kolmas parametri ”doRequest()”-funktiolle on ”$data”. Tämä parametri sisältää lähetettävän tiedon XML-muodossa josta esimerkki kuviossa 12. Esimerkiksi Netvisorin resurssi ”customer.nv” mahdollistaa asiakkaan lisäyksen Netvisor järjestelmään. Lisäykseen vaaditaan asiakkaasta tiedot XML-

muodossa. Asiakkaan tiedot kasataan XML-dokumenttiin ja lähetetään pyynnön mukana. XML-tiedoston sisältö vaihtelee resurssin mukaan.

4.3.3 Tietojen kasaus XML-tiedostoon

DOM eli Document Object Model on esitystapa dokumenteille, jonka olioita voidaan hakea, luoda ja muokata. (Whatwg. 2020.) DOM:in avulla luodaan dokumenttiin ”`Root`”-elementti, eli juuri.

```
<?xml version="1.0"?>
<root> //taso 1
    <customer> //taso 2
        <customerbaseinformation> //taso 3
            <externalidentifier>123456-7</externalidentifier> //taso 4
            <name>Esimerkki Asiakas</name>
            <streetaddress>Test 1</streetaddress>
            <additionaladdressline>PL 1</additionaladdressline>
            <city>Vaasa</city>
            <postnumber>65100</postnumber>
            <country type="ISO-3166">FI</country>
            <email>test@test.fi</email>
            <isactive>1</isactive>
        </customerbaseinformation>
    </customer>
</root>
```

Kuvio 13. Esimerkki kasatusta asiakasdatasta DOM:in avulla, sekä elementtien tasoista.

Dokumentin juuri on taso yksi (1) dokumentissa. Juureen lähdetään tasoittain kasaamaan elementtejä Netvisor-rajapintakuvauksen ohjeilla. ”`Customer`” eli asiakas elementti on taso kaksi (2), tämän alle kasataan luotavan asiakkaan erityyppiset tiedot. Asiakkaan perustiedot kuten yrityksennimi, y-tunnus, osoite sekä asiakaskoodi tulevat elementin ”`customerbaseinformation`” alle, tämä taso on dokumentissa taso kolme (3), joten perustietoihin kuuluvat asiat syötetään ”`customerbaseinformation`”-elementin alle tasolle neljä (4). Kuviosta 13 voidaan nähdä kuinka elementit rakentuvat eri tasolle ja muodostavat toimivan XML-rakenteen

4.3.4 Tietojen lähetys Netvisor rajapintaan

Haluttujen parametrien syötön jälkeen, lähtee NetvisorConnection-mallin ”`doRequest`”-funktio kasaamaan tarvittavaa dataa pyynnön lähettämiseen. Ensimmäisenä käsitellään syötetty resurssiparametri, resurssiparametri lisätään konfiguraatiotiedostosta löytyvän Netvisor-rajapinta osoitteen perään. Lopputuloksena saadaan osoite, johon pyyntö lähetään.

Esimerkiksi: ”`https://isvapi.netvisor.fi/customerlist.nv?keyword=test`”. Kyseisessä osoitteessa, ”`https://isvapi.netvisor.fi`” on konfiguraatiotiedostosta ”`Configure::read(”Netvisor.url”)`”-funktiolla haettu osoite, johon kaikki tämän projektin kutsut lähetetään. ”`customerlist.nv`” on resurssi, josta tietoa haetaan. Kyseisellä ”`customerlist.nv`”-resurssilla haetaan listausta asiakkaista. ”`?keyword=test`” on parametri, jolla voidaan tarkentaa asiakaslistan hakua palauttamaan vain asiakkaat, joiden nimi, asiakasnumero, y-tunnus tai lisänimi sisältävät sanan ”`test`”.

Osoitekäsittelyn jälkeen tarkistetaan, onko ”`$data`” muuttujaan asetettu dataa. Mikäli dataa on tullut mukana, asetetaan ”`$content`” nimiseen muuttujaan ”`$data`” muuttujan tiedot. Mikäli dataa ei ole muuttujassa ”`$data`”, annetaan ”`$content`” muuttujalle arvoksi yksi välilyönti. ”`$content`” muuttujaa käytetään pyynnön lähetyksessä ilmentämään pääasiassa POST tyyppisissä lähetyksissä luotavaa, muokattavaa tai poistettavaa resurssia. Netvisor-rajapinta ei hyväksy pyyntöä mikäli ”`content`” eli sisältö on täysin tyhjä. Tästä syystä sisällön sisältävä ”`$content`”-muuttuja alustetaan GET-pyynnöissä yhdellä välilyönnillä.

4.3.5 Tunnistautumisotsikoiden luonti kyselylle

Osoitteen ja sisällön käsittelyn jälkeen on aloitettava kasaamaan tunnistautumiseen tarkoitettuja otsikoita lähetettävään kutsuun. Ensimmäisenä

tehdään funktio nimeltä ”generateRandomString()”. Kyseistä funktiota käytetään satunnaisten merkkijonojen luomiseen. Satunnaista merkkijonoa tarvitaan useammassa kohdassa tunnistautumisvaiheessa.

”generateRandomString()”-funktioon määritellään muuttujaan halutun merkkijonon pituus. Pituus tunnistautumista varten tulee olemaan 20 merkkiä. Muuttujaan ”$characters” määritellään käytettävät merkit satunnaista merkkijonoa varten. Muuttujan sisällä olevat merkit ovat seuraavat: ”0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ”.

Muuttujasta ”$characters” lasketaan pituus ”strlen($characters)”-funktiolla ja tallennetaan muuttujaan ”$characterLength”. Luodaan silmukka, joka käydään läpi syötetyn pituuden mukaan (esimerkiksi 20). Jokaisessa silmukan kierroksessa lisätään ”$randomString” muuttujaan yksi satunnainen merkki ”$characters”-muuttujassa määritellystä merkkilistasta. Satunnainen merkki arvotaan seuraavalla funktiolla: ”rand(0,$characterLength -1)”.

Funktio arpoo merkin 0 ja laskettujen merkkien väliltä ($characterLength - 1) numeron, jolla haetaan merkistöstä numeron paikalla oleva kirjain. Esimerkiksi $character[17] palauttaisi merkin ”h”. Silmukan läpikäynnin jälkeen lisätään vielä perään päivämäärä sekä aika muodossa vuosi, kuukausi, päivä, tunnit, minuutit sekunnit. Lopputulos voi olla esimerkiksi merkkijono ”8kfEC23SD445LS21a0FC2020-12-03 19:32:22”. ”generateRandomString()”-funktio palauttaa generoidun satunnaisen merkkijonon.

Tunnistautumisotsikoita varten tarvitaan satunnainen merkkijono sekä valmis osoite pyyntö varten. Tunnistautumisen otsikot kasataan ”getAuthenticationHeaders()”-funktiossa, joka ottaa parametrikseen generoidun satunnaismerkkijonon sekä osoitteen pyynnölle. ”getAuthenticationHeaders()” -funktio palauttaa merkkijonon, joka sisältää täydennetyt tiedot seuraavan kuvan mukaisesti.

```php
/**
 * Luodaan tarvittavat headerit NetVisor pyynnön tekoa varten
 * @param  string $generatedStr Lähetyksen tunnus
 * @param  string $url          Taustajärjestelmän endpointin osoite
 * @return string               Headerit merkkijonona
 */
private function getAuthenticationHeaders($generatedStr, $url)
{
    return "Content-Type: text/plain" . "\r\n" .
        "X-Netvisor-Authentication-Sender: ". Configure::read('Netvisor.authSender') . "\r\n" .
        "X-Netvisor-Authentication-CustomerId: " . Configure::read('Netvisor.customerId') . "\r\n" .
        "X-Netvisor-Authentication-PartnerId: " . Configure::read('Netvisor.partnerId') . "\r\n" .
        "X-Netvisor-Authentication-Timestamp: " . date('Y-m-d H:i:s') . "\r\n" .
        "X-Netvisor-Interface-Language: ". Configure::read('Netvisor.ApiLanguage') . "\r\n" .
        "X-Netvisor-Organisation-ID: ". Configure::read('Netvisor.Y-ID') . "\r\n" .
        "X-Netvisor-Authentication-TransactionId: " . $generatedStr . "\r\n" .
        "X-Netvisor-Authentication-MAC: " . $this->getMAC($generatedStr, $url) . "\r\n" .
        "X-Netvisor-Authentication-MACHashCalculationAlgorithm: SHA256" . "\r\n";
}
```

Kuvio 14. getAuthenticationHeaders-funktion palauttaman merkkijonon luonti.

Kuvion 14 mukaisesti täytetään vaaditut tunnistautumisotsikot funktion sisällä. Otsikoiden arvoksi haetaan konfiguraatiotiedostosta Netvisor tietoja kuten pyynnön lähettäjä (`Netvisor.authSender`), asiakaskoodi (`Netvisor.customerId`), sekä lukuisia muita kuvio 13 esiintyviä arvoja.

Lisäksi tarvitaan päivämäärä ja aika, aiemmin generoitu merkkijono sekä MAC-arvo. MAC-arvo joudutaan luomaan jokaiselle pyynnölle uudelleen. MAC-arvo luodaan omassa funktiossa nimeltä ”`getMAC()`”. ”`getMAC`”-funktio ottaa parametrikseen luodun satunnaisen merkkijonon sekä osoitteen ”`$url`”-muuttujasta, jotka syötettiin ”`doRequest`”-funktiossa ”`getAuthenticationHeaders`”-funktiolle. getMAC()-funktio palauttaa SHA256-salatun merkkijonon, jolle on syötetty parametreina konfiguraatiotiedostosta löytyviä tunnistautumistietoja, kuten tunnistautumisen lähettäjä, asiakaskoodi, sekä rajapinta-avaimet.

SHA256-salaus on NSA (National Security Agency) kehittämä salausmenetelmä sähköisesti lähetettävälle datalle. SHA256-salaus mahdollistaa viestin vastaanottajalle viestin muuttumattomuuden matkalla kohteeseen. SHA-salaus luo merkkijonon, jonka avulla voidaan tarkastaa ja vertailla dataa sekä onko se saapunut muuttumattomana vastaanottajalle. Mikäli generoitu SHA-merkkijono ei vastaa alkuperäisestä datasta generoitua SHA-merkkijonoa, on sitä muokattu (National Institute of Standards and Technology 2015).

MAC-arvo luodaan Netvisor-ohjelmistorajapinta ohjeen mukaisesti ja koodi on melkein suoraan kopioitu ohjeistuksesta (ks. Kuvio 15). Uudelleen käyttämällä jaettua koodia säästetään aikaa, eikä ole tarvetta itse lähteä kasaamaan MAC-arvon luontia.

```php
/**
 * Luodaan sha256 tiiviste NetVisor-pyyntöä varten
 * @param  string $generatedStr Lähetyksen tunnus
 * @param  string $url          Taustajärjestelmän endpointin osoite
 * @return string               Tiiviste/tarkistussumma
 */
private function getMAC($generatedStr, $url)
{
    $parameters = [
        $url,
        Configure::read('Netvisor.authSender'),
        Configure::read('Netvisor.customerId'),
        date('Y-m-d H:i:s'),
        Configure::read('Netvisor.ApiLanguage'),
        Configure::read('Netvisor.Y-ID'),
        $generatedStr,
        Configure::read('Netvisor.apiUserKey'),
        Configure::read('Netvisor.partnerKey')
    ];

    return hash('sha256', implode('&', $parameters));
}
```

Kuvio 15. getMAC-funktio, sen toiminnallisuus ja palautuksessa käytetty sha256-salaus.

MAC-arvo sekä otsakkeet palautetaan takaisin ”`doRequest`”-funktiolle ja niiden avulla lähetetään pyyntö Netvisor-rajapintaan. Rajapinta palauttaa vastauksen XML-muodossa ja riippuen kutsusta myös pyydetyn toiminnon vastaus, kuten uuden tilauksen luonti tai tuotetietojen päivitys. Vastaus tulee XML-muodossa ja sisältää juuren jälkeen tason ”`Status`” joka sisältää pyynnön onnistuessa vastauksen ”`OK`” tai virheen merkkinä ”`FAILED`” sekä virheviestin epäonnistumisen syystä.

NetvisorConnection-mallin avulla voidaan nyt tehdä kyselyitä Netvisor-rajapintaan. ”`doRequest`”-funktio myös palauttaa kyselyn vastauksen rajapinnasta. Funktiota voidaan kutsua mistä vain kontrollerista, mallista tai komentorivisuoritteesta(Shell) lisäämällä käsittelyssä olevaan tiedostoon rivi: ”`use Netvisor\Model\NetvisorConnection;`”. Kyseisen rivin

lisääminen mahdollistaa mallin julkisten funktioiden kutsumisen muista tiedostoista.

NetvisorConnection-mallin ”`doRequest();`” funktiota voidaan kutsua esimerkiksi alustamalla NetvisorConnection-malli muuttujaan seuraavasti: ”`$this->connection = new NetvisorConnection();`”. Alustamisen jälkeen voidaan ”`$this->connection`” muuttujalla kutsua ”`doRequest`”-funktiota esimerkiksi: ”`$this->connection->doRequest(productlist.nv,”GET”);` ”. Funktiolle syötetään kappaleen alussa käydyt parametrit halutun haettavan tai luotavan resurssin tietojen mukaisesti.

4.4 Tuotteiden hakeminen ja päivittäminen rajapinnan avulla

Kutsujen toiminnallisuuden rakentamisen ja testaamisen jälkeen lähdettiin hakemaan rajapinnan kautta Netvisorissa olevat tuotteet. Tuotteiden hakuun ja päivityksiin luotiin uusi Shell-tiedosto sekä uusi tietokantataulu tuotteille.

Shell-tiedostot ovat komentoriviltä ajettavia tiedostoja, joilla voidaan esimerkiksi hakea, lähettää tai muokata tietoa. Tuotteiden tapauksessa, Shell-suoritteella hoidetaan tuotteiden hakemisen, tuotteiden tietojen-, varastosaldojen- ja hintapäivitysten toimintoja.

4.4.1 Tuotteiden hakeminen ja lisääminen tietokantaan

Tuotetaulua lähdettiin rakentamaan tutkimalla Netvisorin tuotekyselyn palauttamat tiedot, tilauksen luonnin yhteydessä tarvittavat tiedot sekä kaupan toimintaan tarvittavat tiedot ja yhdisteltiin näitä, jotta saataisiin kaikki tarvittava tieto kasattua tuote tauluun.

Tuote tauluun tehtiin lisäyksiä migraationa CakePHP-työkalujen avustuksella. Itse tuote taulu sisältää jo kaupalle tarpeelliset kentät, mutta kyseistä Netvisor-integraatioita varten on lisättävä kenttiä. CakePHP mahdollistaa migraatioiden

ylläpitämisen Migrations-lisäosalla, joka hallinnoi migraatiotiedostojen tietojen lisäyksen(CREATE), muokkauksen(CHANGE/UPDATE) tai pudotuksen(DROP) taulusta. Uusi migraatio tiedosto saadaan luotua antamalla komentoriville komento ”`php bin/cake.php migrations create addFieldsForProductTable -p Netvisor`”. Kohta ”`-p Netvisor`” komennon lopussa ohjaa luomaan migraatiotiedoston Netvisor-lisäosan alle.

CakePHP noudattaa CamelCase nimeämiskäytäntöä, joten mallit, kontrollerit, funktiot ja myös muuttujat ohjataan kirjoittamaan CamelCase kirjoitusasua noudattaen, tässä tapauksessa migraation nimi luodaan myös käyttäen CamelCase-nimeämistä.

CamelCase nimeämisessä jokaisen erillisen sanan ensimmäinen kirjain on iso kirjain, kuitenkin ensimmäisen sanan kirjain jätetään pieneksi. CamelCasen käyttäminen ohjelmoinnin parissa on yleistä, koska yleensä tiedostot tai funktiot eivät voi sisältää välilyöntejä, täten CamelCase luo parempaa luettavuutta koodiin (TechTerms 2020). Esimerkkinä merkkijono ”`uusilaskunluontifunktio`” on helpompi lukea CamelCase-tyylillä ”`uusiLaskunLuontiFunktio`”.

```
13      public function change()
14      {
15          $table = $this->table('products');
16          $table->addColumn('price', 'decimal', [
17              'default' => 0,
18              'limit' => 15.4,
19              'null' => true,
20              'after' => 'tags',
21          ]);
22          $table->addColumn('is_new', 'integer', [
23              'default' => 0,
24              'limit' => 1,
25              'null' => true,
26              'after' => 'price',
27          ]);
28          $table->update();
29      }
30  }
```

Kuvio 16. Esimerkki migraatiotiedostosta.

Migraatiotiedostolla voidaan lisätä, muuttaa tai poistaa kenttiä taulusta. Kuviossa 16 nähdään kaksi lisäystä tauluun ”addColumn”-funktion avulla. ”addColumn”-funktio ottaa sisäänsä kolme parametria, ensimmäisenä kentän nimi, toisena tyyppi ja kolmantena lista arvoista sekä lisävalinnoista, kuten ””after” => ”tags”” joka määrittää minkä kentän jälkeen uusi kenttä lisätään.

Migraatiotiedoston kirjoittamisen jälkeen ajetaan komentorivillä komento: ”php bin/cake.php migrations migrate -p Netvisor”, joka päivittää migraatiotiedoston sisällön tietokantaan. Tarvittavien lisäysten jälkeen lähdettiin luomaan Shell-tiedostoa, jonka avulla pidetään verkkokaupan tuotetiedot ajan tasalla.

”GetProductsShell”-tiedoston rakentaminen aloitettiin tyhjästä tiedostosta, jolle annettiin nimiavaruus ”namespace App\Shell”. Seuraavaksi otettiin käyttöön Shell-luokka ” Cake\Console\Shell”, jonka avulla voidaan ajaa ja käyttää tiedostoa komentorivin kautta. Kontrolleri ja Shell tiedostot voivat sisältää ”initialize”-funktion, jonka sisällä olevalla käskyllä ”parent::initialize()” aliluokka perii yliluokan toiminnallisuuden olio-ohjelmoinnin mukaisesti.

”initialize”-funktioon voidaan lisätä esimerkiksi tarvittavia malleja, kuten tässä tapauksessa ”Products”-, sekä ”ProductVariations” -mallit. Mallin lataus voidaan alustaa muuttujaan sen käyttämisen helpottamiseksi, esimerkiksi ”$this->Products = $this->loadModel(”Products”)”.

Muuttujaan alustuksen jälkeen, ”$this->Products” vastaa mallin kutsuihin ja sitä voidaan käyttää esimerkiksi tuotetaulun tietojen hakemiseen seuraavasti: ”$this->Products->find(”all”);”. Edeltävällä rivillä saadaan haettua tuotteet objektiin tai listaan, joka sisältää avain (keyField) ja arvo (valueField) parit kaikista taulun tietueista.

Ennen tuotteiden hakemista Netvisorin-rajapinnasta, on kuitenkin tietokantataulu tyhjä. Tuotteet lähdettiin hakemaan aikaisemmin luodun

NetvisorConnection-mallin avulla käyttämällä ”`doRequest`”- funktiota. NetvisorConnection-malli alustetaan ”`initialize`” funktioon käytettäväksi muuttujaan ”`$this->connection = new NetvisorConnection()`”.

Muuttujaa ”`$this->connection`” voidaan nyt käyttää pyyntöjen tekemiseen tiedoston sisällä. Ensimmäisenä lähdettiin hakemaan kaikki tuotteet Netvisorin-rajapinnasta käyttämällä yhteysmuuttujaa seuraavasti: ”`$netvisorProductList = $this->connection->doRequest(”productlist.nv”, ”GET”);`”.

Onnistuneen pyynnön seurauksena palautuu muuttujaan ”`$netvisorProductList`” lista tuotteista, jotka löytyvät Netvisor-ohjelmistosta. Netvisor-rajapinta palauttaa pyyntöjen vastaukset xml-muodossa. Datan helpomman käsittelyn vuoksi, vaihdetaan vastauksen tietojen koodaus ensin JSON-tyyppiseksi PHP:n ”`json_encode`”-funktion avulla, jonka jälkeen muutetaan JSON-tyyppinen tieto PHP:n muuttujaksi ”`json_decode($muuttuja, true)`”-funktion avulla ja tässä tapauksessa assosiaatiotauluksi (Associative array).

Tuotelistauksen saamisen jälkeen lähdetään vertailemaan, löytyykö tuote jo valmiiksi verkkokaupan tietokannasta. Tietokannasta haetaan kaikki tuotteet, mutta niistä otetaan vain avain – arvo pari siten, että avaimena toimii tuotteen SKU-koodi ja arvona tuotteen Netvisor-tunniste.

Lista saadaan haettua antamalla taulusta haun ” `$this->ProductVariations->find(”all”)->toArray();` ” parametrin sijaan ” `$this->ProductVariations->find(”list”, [”keyField” => ”sku”, ”valueField” => ”product_id”])->toArray();`”, jossa palautusparametri ”`all`” vaihdetaan ”`list`” tyyppiseksi ja sille määritellään avain – arvo tiedot. ”`->toArray()`”-funktio lopussa palauttaa suoraan assosiaatiotaulun, ilman sitä palauttaa CakePHP-sovelluskehys oletuksena objektin.

Edellisten toimien tuloksena on nyt lista Netvisorista löytyvistä tuotteista, sekä verkkokaupan tietokannan tuotteista. Näitä vertailemalla saadaan selville, tuleeko uusia tuotteita lisätä verkkokauppaan. Tuotelisäykselle annettiin myös lisäehtoja, kuten onko tuote aktiivinen Netvisorissa tai onko se myynnissä. Myös tuotteen Netvisor-kategorian perusteella voidaan suodattaa tuotteita pois, etteivät ne ilmesty kauppaan.

Tuotteen lisäyksen kriteerien täytyttyä (aktiivinen tuote, myynnissä, eikä ole kategorioissa x, y tai z), lähdetään lisäämään tuote tietokantaan. Lisättävistä tuotteista kerätään lista ja käsitellään sitten listanmukaiset tuotteet eteenpäin.

```
//Go trought products
foreach ($productInformation as $arrayKey => $product) {
    $base = $product['ProductBaseInformation'];
    //Some products may not have description, in these cases its presented as empty array and needs to be converted to string
    if (is_array($base['Description'])){
        $base['Description'] = '';
    }

    //Create new empty product entity and patch it with product information from netvisor
    $newProduct = $this->Products->newEntity();
    $productInfo = [
        'title' => $base['Name'],
        'body' =>  $this->getDescripition($base['Description'], true),
        'status' => 'private',
        'price' => str_replace(',', '.', $base['UnitPrice']),
        'is_new' => 0,
    ];
    $newProduct = $this->Products->patchEntity($newProduct, $productInfo);
    $newProduct = $this->Products->save($newProduct);
    $this->out('Added product: ' . $base['Name']);

    //Add variation based on created product, product id and productinformation from netvisor needed
    $this->addVariation($newProduct->id, $product);
}
```

Kuvio 17. Esimerkki tuotteen lisäyksestä tietokantaan.

Tuotetauluun lisätään vähäisesti tietoja itse tuotteesta, pääasiassa tuotteen nimi, näkyvyys verkkokaupassa, hinta sekä tuotteen kuvaus. Tuotevariaatio tauluun ”`ProductVariations`” lisätään tarkemmat tiedot tuotteesta, kuten saldo, verokanta, koko ja muita tietoja mitä tuotteesta saadaan.

CakePHP mahdollistaa tietojen tallennuksen sen sisäänrakennetun ORM-käsittelyn kautta. Kuviosta 17 voidaan havaita, kuinka Products-mallista luodaan uusi entiteetti ja alustetaan se ”`newProduct`”-muuttujaan. Muuttuja sisältää nyt mallin Product-taulun sisältämästä rakenteesta, jossa kentät ovat tyhjiä. ”`productInfo`”-muuttujaan taas lähdetään alustamaan Product-taulun mukaisiin kenttiin tiedot, joita halutaan tallettaa. Seuraavana alustetaan tyhjä entiteetti ”`newProduct`”, ”`productInfo`”-muuttujan tiedoilla. Alustettu

entiteetti tallennetaan tietokantaan funktiolla ”`$this->Products->save($newProduct)`”.

Onnistuneen tuotetallennuksen jälkeen Products-tauluun, lähdetään viemään tarkemmat tiedot tuotteesta ProductVariations-tauluun, tämä tapahtuu funktion ”`addVariation`” kautta, joka saa parametrikseen luodun tuotteen tietokantatunnisteen, sekä tuotetiedot. Tarkempia tietoja tuotteesta tarvitaan niin loppukäyttäjälle käyttöliittymään esille, kuin luotaviin tilauksiin.

Tiedot käsittävät esimerkiksi tuotteen hinnan, tuotekoodit, Netvisorin sisäisen tunnisteen sekä varastosaldon. Samalla kun tuotteen tarkempia tietoja haetaan ja käsitellään, katsotaan myös Netvisorista saapuneesta datasta, onko tuotteella kategorioita. Mikäli kategoria löytyy, lisätään se verkkokauppaan, mikäli ei ole aiemmin esiintynyt sekä liitetään tuote kategoriaan.

Skyshop-verkkokauppa käsittelee ja listaa tuotteet automaattisesti kategorioittain, joten kategorian tarkastamisella ja lisäyksellä saadaan pienellä toimella automatisoitua loppukäyttäjälle näkyvät tuoteryhmät tai kategoriat. ”`GetProductsShell`”-tiedoston perusreitti on nyt määritelty ja se hakee sekä lisää uusia tuotteita verkkokauppaan eri yllä mainittujen määritysten mukaisesti.

4.4.2 Tuotteiden varastosaldojen päivittäminen

Tuotteiden tietoja on kuitenkin päivitettävä, jotta tieto Netvisorista kohtaisi kaupan tuotteiden kanssa niin hinnoiltaan kuin varastosaldoiltaan. Päädyin rakentamaan samaan tiedostoon lisäfunktioita, joilla voidaan päivittää tuotetietoja. Lisäfunktion ajaminen erikseen, ilman että käydään aina uusia tuotteita läpi. Erottelu tuo myös vaihtoehdon eri toimintojen eriaikaiseen suorittamiseen. Esimerkiksi tuotesaldoja tai hintoja päivitetään useammin kuin tuotteen muita tietoja.

Erottelu aloitetaan lisäämällä funktio ”`getOptionParser()`” tiedoston alkuun. Kyseisellä funktiolla voidaan lisätä valintoja sille, mitä funktiota halutaan saman

tiedoston sisällä lähteä ajamaan, valinta tapahtuu syöttämällä halutun funktion parametri konsoliin suorittamisen yhteydessä. Valintoja voidaan lisätä syöttämällä yllä mainittuun funktioon seuraava pätkä: ”`$parser->addOption(”updateStock”,array(”short”=>”o”,”help”=>”Only update product stock.”));`”.

Valinnan lisääminen ottaa parametrikseen funktion nimen ”`updateStock`”, konsolissa annettavan parametrin ”`short`” sekä aputekstin, joka esiintyy jos käyttäjä antaa komennon kanssa ”`–help`”-parametrin (Cake Software Foundation. 2021). Lisäfunktioiden rakentaminen samaan tiedostoon tuntui luontevalta, koska tiedosto sisälsi valmiiksi mallit sekä yhteydet uusien päivitys funktioiden toteutukseen.

Ensimmäiseksi otin käsittelyyn tuotteiden varastosaldojen päivityksen. Tarkat varastosaldot saadaan Netvisor-rajapinnan ”`inventorybywarehouse.nv`”-kutsulla. Kutsu ottaa parametrikseen tuotteiden Netvisor-avaimet listana ja palauttaa tuotekohtaisesti kaikki tuotetta varastoivien varastojen saatavuudet. Varastosaldojen päivitys siis alkaakin ensin luomalla uusi funktio, ”`updateStock`”. Funktio aloitetaan hakemalla verkkokaupan tietokannasta kaikkien tuotteiden Netvisor-avaimet ”`$this->ProductVariations->find(”list”, [”keyField” => ”id”, ”valueField” => ”external_id”])->where([”external_id IS NOT” => null])->toArray();`”-haulla.

Haku palauttaa listan kaikista tuotteista, jossa avain on verkkokaupan tietokannan id ja arvo on tuotteen Netvisor-avain. Lista puretaan ja muutetaan yhdeksi merkkijonoksi PHP:n implode-funktiolla: ”`$products=implode(”,”,$databaseproducts);`”. Implode-funktio yhdistää listan arvot yhdeksi merkkijonoksi ja erottelee ne annetulla parametrilla, joka tässä tapauksessa on pilkku (PHP-Group. 2021a). Luotu merkkijono syötetään parametrina Netvisor-rajapintaan lähtevään hakuun ja vastauksena saadaan lista, joka käydään läpi ja tehdään oma uusi lista. Avaimina toimivat tuotteen Netvisor-avaimet ja arvona varaston ilmoittama tuotekohtainen tuotesaatavuus. Tämän listan avulla vertaillaan tietokannasta

löytyvää tietoa, sekä uutta tietoa varastoilta, jos summat eroavat käytetään varaston ilmoittamaa määrää ja muutetaan se tietokantaan tuotteelle.

4.4.3 Tuotteiden tietojen päivittäminen

Tuotteiden tietojen päivitystä varten luodaan myös oma funktio samaan tiedostoon. Tuotteiden tiedoista päivitetään hintaa, koko- ja lisätietoja sekä EAN-tietoja. Tuotepäivityksille lisättiin oma valinta ”`getOptionParser`”-funktion alle, jotta myös tätä voitaisiin ajaa yksittäisenä komentona.

Tuotepäivitys tapahtuu hakemalla ensin tietokannasta Netvisor-avaimet, sekä tuotteen id. Netvisor-avaimista muodostetaan lista, jota käytetään rajapinnassa parametrina. Tuotetiedot Netvisor-rajapinnan kautta saatiin ”`getproduct.nv`”-resurssilla, rajapinta palauttaa vastauksena listan tuotteista, sekä niiden sisältämästä tiedosta. Tietokannasta löytyvää tietoa, sekä rajapinnan palauttamaa tietoa vertaillaan keskenään käymällä tuotteet läpi yksi kerrallaan ja päivitetään tietokantaan, mikäli Netvisorista tuleva tieto eroaa tietokannan tiedoista. Esimerkiksi, tuotteella voi olla lisätieto ”*Huippuhyvä tuote, takuu 3v!*” ja tämä on päivitetty Netvisoriin tekstiksi ”*Huippuhyvä tuote, takuu 5v!*”. Tuotepäivitys-funktio tarkastelee esimerkiksi 30-minuutin välein tuotteiden tietoja ja edellisessä esimerkissä päivittäisi tietokantaan uuden lisätieto tekstin, joka sitten näkyy verkkokaupassa loppukäyttäjälle.

4.5 Asiakastietojen tuominen verkkokauppaan

Verkkokauppa on suunniteltu palvelemaan yritysasiakkaita, joten käytettävät asiakastiedot voidaan tuoda suoraan Netvisorista. Netvisor-rajapinta mahdollistaa asiakkaiden hakemisen Netvisorista, jotta tiedot voidaan tuoda suoraan verkkokaupan tietokantaan. Asiakashakua varten luodaan uusi Shell-tiedosto ”`getCustomerShell.php`”. Tiedostoon alustetaan 4.4.1 luvun mukaisesti Netvisor-yhteys sekä ”`netvisor_customers`”-taulun malli.

Tiedostoon rakennetaan asiakkuuksien hakeminen. Netvisor-rajapinta palauttaa metodilla ”customerlist.nv” listan asiakkaista. Asiakas tiedoissa palautuu mm. asiakkaan nimi, y-tunnus, Netvisor-avain ja asiakasryhmä. Palautunutta listaa hyödyntäen lähdetään hakemaan yksi kerrallaan tarkempia asiakastietoja. Tarkempia asiakastietoja haetaan silmukalla, jossa asiakaslistauksen tiedoista käytetään Netvisor-avainta.

Netvisor-avainta käyttämällä saadaan haettua Netvisor-rajapinnasta tarkat asiakastiedot ”getcustomer.nv”-resurssilla. Pyyntö rajapintaan palauttaa tarkemmat asiakastiedot asiakkaasta, mikäli semmoinen löytyy annetulla Netvisor-avaimella. Asiakastiedoista otetaan talteen perustiedot, kuten yrityksen nimi, apunimet, osoitetiedot, puhelinnumero sekä sähköposti. Rajapinnan palautuneesta tiedosta otetaan myös talteen asiakkaan toimitustiedot sekä mahdolliset kontaktitiedot.

Jokaiselle haetulle asiakkaalle luodaan myös käyttäjä verkkokauppaan. Käyttäjänimenä toimii sähköposti asiakkaan tiedoista ja lyhyt ensikirjautumiseen tarkoitettu salasana generoidaan. Käyttäjänimeä ja salasanaa ei alun perin toimitettu asiakkaalle automatisoidusti sähköpostin välityksellä integraation kehityksen aikana, vaan tämä lisättiin jatkokehityksessä.

Asiakkaat voivat muokata tai lisätä perustietojaan, kuten toimitusosoitetta tai nimeään verkkokaupassa, näitä tietoja ei kuitenkaan viedä Netvisoriin. Verkkokauppa mahdollistaa erillisen toimitusosoitteen käytön, esimerkiksi yrityksille, jotka tarvitsevat tilatut tuotteet väliaikaisesti toiseen osoitteeseen tai esimerkiksi varastoon. Toimitusosoitteen vaihto tilausta tehdessä on kertaluontoinen, eikä sitä käytetä uudestaan tuleviin tilauksiin.

4.6 Verkkokauppa tilausten siirto Netvisoriin

B2B-tyylinen verkkokauppa ei tarvitse tässä tapauksessa maksamiseen tarkoitettuja reitityksiä tai lisäosia. Laskutus hoidetaan tilausten perusteella, jolloin, laskut luodaan Netvisor-ohjelmistossa manuaalisesti, joten rajapinnan

kautta lähetetään vain tilaukset. Laskut voidaan myös luoda rajapinnan kautta, mutta tämä ei ollut toimeksiantajan vaatimuksena. Tilaukset Netvisor-integraation tapauksessa muodostetaan asiakkaan tilaamien tuotteiden ja tietojen mukaisesti. Tilaukset siirtyvät tilatessa Netvisoriin käsittelyn jälkeen. Tilaukset tulee muuntaa XML-muotoon, jotta ne voidaan lähettää Netvisor-rajapintaan. Tilauksen muodostaminen XML-muotoon vaatiikin hieman enemmän työtä verrattuna esimerkiksi aiemmin suoritettuihin asiakas tai tuotetietojen hakuun.

4.6.1 Tilauksen luominen rajapintaan

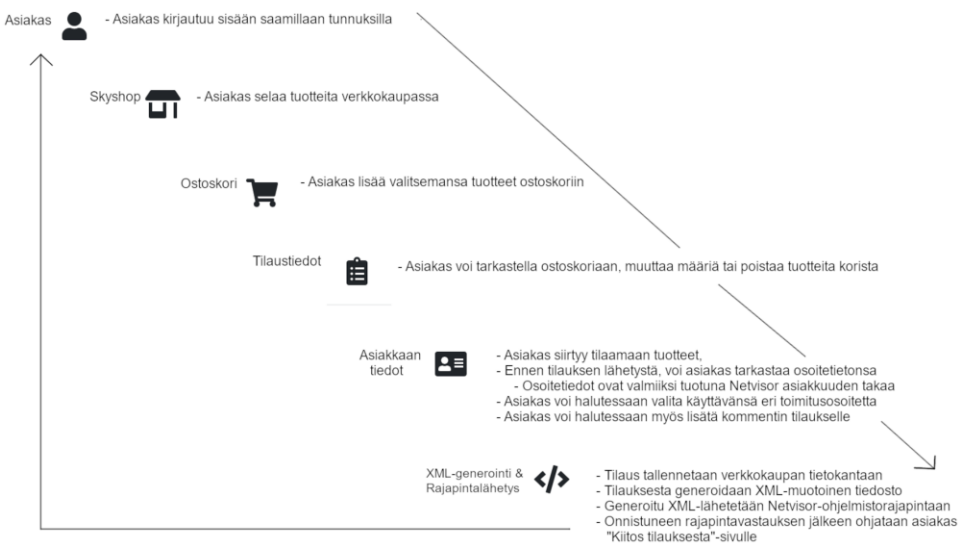
Havainnollistava esimerkki tilauksen luomisen kulusta: Verkkokaupan asiakas on päättänyt tilata tuotetta X 20kpl, tuotetta Y 10kpl sekä tuotetta Z 50kpl. Asiakas voi tarkastaa tilauksensa ostoskorista, jossa listataan tuotteet, joita asiakas on ostoskoriin lisännyt. Ostokorin tuotteet näytetään riveittäin. Riveiltä löytyy tieto tuotteen nimestä, verottomasta kappalehinnasta, verollisesta kappalehinnasta, määrästä sekä rivin kokonaissummasta. Kun asiakas on lisännyt kaikki tilattavat tuotteet ostoskoriin, voi hän jatkaa tilaus vaiheeseen.

Ennen tilauksen jättämistä, voi asiakas vielä tarkastaa omat perustietonsa, sekä valita haluaako tilauksen eri toimitusosoitteeseen kuin Netvisorista tuotuun toimitusosoitteeseensa. Asiakas voi myös lisätä tilaukselleen ylimääräisen kommentin, johon voidaan antaa tarvittavaa lisätietoa esimerkiksi tilauksesta.

Asiakas tarkastaa tietonsa ja painaa ”Lähetä tilaus”-nappia, ostoskorissa olevat tuotteet tallennetaan verkkokaupan tietokantaan riveittäin OrderRows-tauluun, näistä saadaan myöhemmin tilaukselle tilausrivit. Asiakaskohtaiset tilaustiedot myös tallennetaan, kuten asiakkaan nimi, sisäinen asiakas id (tunniste), laskutusosoite, toimitusosoite ja muita tilaukselle tarpeellisia tietoja. Edellä mainitut tiedot tallennetaan Orders-tauluun omaksi rivikseen. Tietojen tallettamisen jälkeen, lähdetään kasaamaan Orders-mallissa tilausta XML-muotoon.

Rajapinta sisältää toiminnon tilauksen tuomiselle resurssilla ”`saleinvoices.nv`” ja parametrilla ”`add`”. XML-muotoinen tiedosto pitää kuitenkin olla Netvisor-ohjelmistorajapinnan mukainen ja sisältää vähintään rajapintakuvauksessa määritellyt pakolliset tiedot. Pakollisia tietoja ovat tilaukselle seuraavat: tilauksen päiväys, tilauksen loppusumma, tilauksen tila, tilauksen tehnyt asiakas sekä linkitystieto (Netvisor-avain, asiakaskoodi tai organisaatioiden välisen tiedonsiirto tunnuksen (OVT-tunnus)), maksuehdon netto- tai eräpäivä, vähintään yksi tilausrivi, joka sisältää tuotteen linkitystiedon (Netvisor-avain), tuotteen nimen, tuotteen yksikköhinnan, tuotteen alv-prosentin ja tuotekoodin sekä kappalemäärän (Netvisor 2021).

Edellä mainittuja tietokantaan tallennettuja tietoja käytetään XML-tiedoston luomiseen. XML-tiedoston luonti tapahtuu PHP:n DOM-lisäosalla. DOM-lisäosalla voidaan luoda ja muokata esimerkiksi XML-dokumentteja (PHP-Group 2021b).



Kuvio 18. Esimerkki prosessista, kun asiakas tekee tilauksen järjestelemään.

XML-tiedosto lähetetään Netvisor-ohjelmistorajapintaan ja onnistuneen rajapintakutsun seurauksena, ohjataan asiakas ”Kiitos tilauksesta”-sivulle. Virheen sattuessa, on XML-tiedosto kuitenkin tallessa ja se voidaan siirtää manuaalisesti Netvisoriin, tai rajapintakutsujen kautta. Kuviosta 18 voidaan

nähdä esimerkki tilausreitistä.

4.6.2 Tilauksen muuttaminen XML-muotoon

XML-dokumentin rakennus aloitetaan luomalla uusi DOM-dokumentti ”`$doc = new \DOMDocument('1.0', 'utf-8');`”. Dokumentin luonnissa annetaan asetukseksi XML-versio sekä koodaustyyppi. Yllä olevassa esimerkissä käytetään xml-versiota 1.0 sekä UTF-8 koodaustapaa, joka mahdollistaa esimerkiksi erikoismerkkien ja ääkkösten käytön.

UTF-8 kuuluu Unicode-standardiin joka mahdollistaa suuren määrän käytettäviä merkkejä tai kirjaimia eri kielistä. Unicode-standardi ei ole kuitenkaan fontti tai ohjelmisto vaan merkistön koodaukseen avuksi luotu käytäntö. (Unicode, Inc. 2021).

XML-dokumenttiin voidaan myös määritellä muita asetuksia luonnin aikana, kuten ”`$doc->preserveWhiteSpace = false;`”, jossa annetaan DOM-dokumentille asetukseksi, että se ei välitä laisinkaan tyhjistä tilamerkeistä dokumentissa. Asetusten jälkeen lähdetään kasaamaan XML-tiedoston sisältöä, tiedostolle tulee ensinnä antaa juuritaso (root), jonka sisälle tulee luoda kaikki lähetettävät elementit. Netvisorin rajapintakuvaus kertoo elementtien järjestyksen, tarvittavat attribuutit sekä elementtien ilmentymien tarpeellisuuden.

Taso	Elementti	Muoto ja pituus	Ilmentymiä	Kuvaus	Esimerkki
Root	root	Aggregaatti	1		
1	salesinvoice	Aggregaatti	1		
2	salesinvoicenumber	Kokonaisluku, max. 15 merkkiä	0...1	Laskunumero, jos ei anneta, Netvisor hakee uuden automaattisesti. Ei anneta luonnokselle.	123456
2	salesinvoicedate	Päivämäärä	1	Laskun päiväys (myös arvopäivä)	2019-12-31
Attr.	format	Merkkijono	1	Laskun päivämäärän muoto, aina *ansi*	ansi
2	salesinvoiceeventdate	Päivämäärä	0...1	Laskun kirjauspäivä. Tälle päivälle muodostetaan kirjanpidon tosite.	2020-01-01
Attr.	format	Merkkijono	1	Laskun kirjauspäivän muoto, aina *ansi*	ansi

Kuvio 19. Ote Netvisor-ohjelmistorajapinnan ohjeesta XML-tiedoston luontiin.

Kuviosta 19 nähdään, kuinka tilaustuonnin XML-tiedosto lähdetään rakentamaan. Luodun juuritason jälkeen tarvitaan ensimmäisenä ”`salesinvoice`” elementti, tämä on pakollinen elementti ja se on tasolla yksi. Kaikki tieto sekä elementit, jotka asetetaan tämän ”`salesinvoice`”-elementin sisälle käsitellään siten, että ne kuuluvat tälle yhdelle tilaukselle. Tieto kuitenkin tulee olla ohjelmistorajapinnan ohjeiden mukaista ja oikeassa muodossa.

Elementtejä lähdetään luomaan ohjeiden mukaan, uusi taso sekä elementti saadaan luotua seuraavasti: ”`$contentItemFirst = $doc->createElement('salesinvoice');`”. Ensin luodaan muuttujaan uusi elementti nimeltä ”`salesinvoice`”, tämän jälkeen liitetään luotu elementti joko jo luodun elementin alle tai samalle tasolle. Koska ainoa elementti tällä hetkellä on juuri-elementti, lisätään luotu ”`salesinvoice`” elementti juuren alle seuraavasti: ”`$contentItemFirst = $root->appendChild($contentItemFirst);`”, jossa ”`$root`”-muuttuja on aiemmin luotu juuri-elementti ja ”$contentItemFirst”-muuttuja on liitettävä ”`salesinvoice`”-elementti. Elementtien luomisesta nähdään esimerkki kuviossa 20.

```
3   $doc = new \DOMDocument('1.0', 'utf-8');
4   $doc->preserveWhiteSpace = false;
5   $doc->formatOutput = true;
6
7   $root = $doc->createElement('root');
8   $root = $doc->appendChild($root);
9
10  $contentItemFirst = $doc->createElement('salesinvoice');
11  $contentItemFirst = $root->appendChild($contentItemFirst);
12
13
14  $contentItem = $doc->createElement('salesinvoicenumber');
15  $contentItem = $contentItemFirst->appendChild($contentItem);
16  $text = $doc->createTextNode('Testi teksti');
17  $text = $contentItem->appendChild($text);
```

Kuvio 20. XML-tiedoston kasaaminen DOM-funktioiden avulla.

Luodun elementin sisään lähdetään samalla tavalla kasaamaan tarvittavia elementtejä, sekä niille arvoja tai attribuutteja. Attribuutti saadaan lisättyä elementille seuraavasti: ”`$contentItem->setAttribute('format', 'ansi');`”, jossa ”`setAttribute`”-funktion sisällä oleva ”`format`” on attribuutin avain ja ”`ansi`” taas attribuutin arvo. Mikäli elementin sisään halutaan lisätä jokin arvo, tapahtuu se seuraavasti: ”`$text = $doc->createTextNode('testi teksti');`”. ”`createTextNode`”-funktio valmistelee annetun arvon ja ”`appendChild`”-funktio lisää annetun arvon elementin sisään, esimerkiksi: ”`$text = $contentItem->appendChild($text);`”.

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Root>
   <SalesInvoice>
        <SalesInvoiceNetvisorKey>12345</SalesInvoiceNetvisorKey>
        <SalesInvoiceNumber>000012345</SalesInvoiceNumber>
        <SalesInvoiceDate format="ansi">2021-02-02</SalesInvoiceDate>
        <SalesInvoiceEventDate format="ansi"/>
        <SalesInvoiceValueDate format="ansi"/>
        <SalesInvoiceDeliveryDate format="ansi">2021-02-02</SalesInvoiceDeliveryDate>
        <SalesInvoiceDueDate format="ansi">2021-02-02</SalesInvoiceDueDate>
        <SalesInvoiceReferencenumber>10000213342</SalesInvoiceReferencenumber>
        <SalesInvoiceAmount iso4217currencycode="EUR" currencyrate="1">110,05</SalesInvoiceAmount>
        <SellerIdentifier type="name">Myyjä Martti</SellerIdentifier>
        <InvoiceLines>
            <InvoiceLine>
                <SalesInvoiceProductLine>
                    <NetvisorKey>1234</NetvisorKey>
                    <productidentifier type="customer">testituote</productidentifier>
                    <ProductName>TestiTuote</ProductName>
                    <ProductNetvisorKey>001</ProductNetvisorKey>
                    <ProductUnitPrice>20,95</ProductUnitPrice>
                    <ProductPurchasePrice>10</ProductPurchasePrice>
                    <ProductVatPercentage vatcode="">24</ProductVatPercentage>
                    <SalesInvoiceProductLineQuantity>1</SalesInvoiceProductLineQuantity>
                    <SalesInvoiceProductLineDiscountPercentage/>
                    <SalesInvoiceProductLineFreeText/>
                    <SalesInvoiceProductLineVatSum>6,23</SalesInvoiceProductLineVatSum>
                    <SalesInvoiceProductLineSum>27,32</SalesInvoiceProductLineSum>
                    <SalesInvoiceProductLineInventoryID>1</SalesInvoiceProductLineInventoryID>
                    <SalesInvoiceProductLineInventoryName>Varasto 1</SalesInvoiceProductLineInventoryName>
                </SalesInvoiceProductLine>
            </InvoiceLine>
        </InvoiceLines>
   </SalesInvoice>
</Root>
```

Kuvio 21. Esimerkki luodusta XML-tiedostosta.

Haluttujen elementtien ja tietojen lisäyksen jälkeen, tallennetaan XML-tiedosto, jolloin se voidaan lukea XML-muodossa. Tallennettu XML-tiedosto säilytetään palvelimella, jotta virhetilanteissa siitä voidaan katsoa tilauksen tiedot ja lähettää se tarvittaessa uudelleen. Kuviosta 21 nähdään valmis XML-tiedosto, joka on luotu käyttäen DOM-dokumentin luomiseen tarkoitettuja funktioita.

4.6.3 Tilauksen lähetys rajapintaan

Tilausdatasta luotu XML-tiedosto lähetetään Netvisoriin, XML-tiedoston tulee olla oikein jäsennelty ja sisältää tarvittavat tiedot, jotta tilaus voidaan luoda Netvisor järjestelmään. Luotu XML-tiedosto lähetetään Netvisoriin ohjelmistorajapintaa pitkin resurssilla ”`saleinvoices.nv`” ja parametrilla ”`add`”.

Tilauksen lisäys on POST-tyyppinen, joten rakennettua lähetysfunktiota käytetään nyt seuraavasti: ”`$this->connection->doRequest('salesinvoice.nv?Method=add,'POST',$createdXML);`

”, jossa käytetään Netvisorin rajapintaresurssia ”salesinvoice.nv”, metodia ”add”, viedään POST-tyyppisenä sekä lähetetään kutsussa mukana luotu XML-tiedosto.

Netvisor rajapinta vastaa hyväksyttiinkö lähetetty XML. Mikäli XML-tiedostossa havaittiin virheitä, kuten väärä elementtien järjestys, puuttuva attribuutti tai virheellinen arvo, palauttaa rajapinta XML-tiedostona virheviestin, joka sisältää tietoa virheestä. Rajapintavastauksen tieto voidaan lukea palautuvasta XML-datasta kohdasta ”Status”, joka onnistuneen lähetyksen yhteydessä palauttaa ”OK” tai virheellisen lähetyksen yhteydessä ”ERROR” sekä virhetekstin virheellisestä osiosta.

4.7 Automatisointi Cron-ajastuspalvelulla

Shell-suoritteet voidaan asettaa käynnistettäväksi Cron-ajastuspalvelun avulla, joka löytyy valmiina suosituimmista unix-pohjaisissa käyttöjärjestelmissä. Cron-ajastuspalvelulla voidaan esimerkiksi määrittää tuotteiden haku suorittumaan puolentunnin välein.

Cron-ajastuspalveluun voidaan lisätä suoritteita ja komentoja ajettavaksi konsolissa komennolla ”crontab -e”. Komento avaa crontab tiedoston, joka sisältää suoritettavat komennot seuraavassa muodossa ”* * * * * /var/www/html/loppupolku/php bin/cake.php suoritettava_tehtävä”.

Komennossa ”*” merkitsee vasemmalta oikealle minuutteja, tunteja, päiviä, kuukausia, viikonpäiviä. Esimerkiksi ”*/30 * * * * /var/www/html/loppupolku/php bin/cake.php suoritettava_tehtävä” syötettynä crontab-tiedostoon, pyörittäisi nyt ”suoritettava_tehtävä”-komentoa joka kolmaskymmenes minuutti, joka päivä. Cron-ajastuspalvelulla hoidetaan integraatiossa esimerkiksi uusien tuotteiden haku, tuotepäivitykset sekä uusien asiakkaiden tarkastus.

5 Tulokset

Tämän opinnäytetyön tuloksena oli tarkoitus parantaa ja kehittää toimeksiantajan kilpailukykyä verkkokauppa-, ja integraatiomarkkinoilla kehittämällä uudelleenkäytettävä Netvisor-integraatio hyödyntäen Netvisorin ohjelmistorajapintaa.

Toimeksiantajan vaatimuksena oli ohjelmistointegraatio käyttäen Netvisor-talousohjelmiston rajapintaa kilpailukyvyn parantamiseksi. Ohjelmistointegraation tavoitteina oli saada verkkokauppaan normaalit toiminnot toimimaan Netvisor-integraation kanssa. Näitä olivat tilausten vienti, asiakastietojen haku sekä päivitys, tuotetietojen haku sekä päivitys ja integraation uudelleenkäytettävyys.

Opinnäytetyön aikana toteutettiin toimeksiantajan vaatimusten mukainen ohjelmistointegraatio, joka kykenee asiakas-, tuote- ja tilaustietojen käsittelyyn tarpeiden mukaisesti ja automatisoidusti. Kyseinen ohjelmistointegraatio rakennettiin lisäosaksi CakePHP-ohjelmistokehyksen mukaisesti, jotta sen uudelleen käyttäminen olisi mahdollisimman sujuvaa ja helppoa.

Lisätyt tietokantataulut ja niiden tiedot, luotiin käyttäen CakePHP-migraatio ominaisuutta ja täten myös tarvittavat taulut saadaan seuraavaankin projektiin mukaan ja integraatio voidaan asentaa semmoisenaan toimintaan CakePHP-ympäristöön, kunhan CakePHP-ohjelmistokehyksen versio on 3.x.

Lopulliset tavoitteet ja ominaisuudet, jotka haluttiin integraatioon, muodostuivat opinnäytetyön aikana toteutetuissa suunnittelupalavereissa sekä ohjelmistokehityksen aikana. Runsaan suunnittelun, katselmuksien, jatkoideoinnin sekä toiveiden tuloksena saatiin ohjelmistointegraatiota muokattua siten, että se palvelee mahdollisimman hyvin toimeksiantajaa sekä hänen tavoitteitaan.

Asiakkaat siirtyivät Netvisorista helposti Skyshop-verkkokauppaan, mutta alun perin tunnusten luontia ja toimittamista ei ollut toimeksiantaja tai minä sen enempää miettinyt. Ratkaisuksi löytyi kuitenkin kehityksen aikana helppo ratkaisu: lähetetään tunnukset sähköpostitse niille asiakkaille, joilta sähköpostiosoite löytyi. (ks. 4.5)

Opinnäytetyön sekä integraation kehityksen aikana, heräsi useita hyviä jatkokehitysideoita, kuten myyntitilausten, laskujen ja ostotilausten automatisoinnista. Opinnäytetyön aikana kuitenkin keskityttiin verkkokauppaan sopivaan ohjelmistointegraatioon, pitäen kuitenkin sen uudelleenkäyttö mahdollisuudet mielessä. Uudelleenkäytettäviä kohteita ei nimetty ennalta, joten integraatio rakennettiin lisäosana, jonka voi ottaa käyttöön tulevissa projekteissa semmoisenaan tai muokattuna. Ainoana rajoittavana tekijänä on CakePHP-ohjelmistokehys, joka on kuitenkin toimeksiantajalla suurimmassa osassa projekteista käytössä. Opinnäytetyössä valmistetun integraation jälkeen, on lähdetty yhdessä toimeksiantajan sekä asiakkaitten kanssa jatkokehittämään integraatiota ja lisäämään siihen ominaisuuksia sekä automatisointia eri osa-alueille.

Jatkokehityskohteita ovat olleet esimerkiksi, automaattinen laskujen luonti tilauksista Netvisorin rajapinnan kautta ja asiakaskohtaisesti useamman toimipaikan tuominen Netvisorista sekä niiden käyttö yhden verkkokaupan asiakastilin alta. Laskujen luonti tapahtuu Netvisor-ohjelmistorajapinnan avulla samalla resurssilla ja XML-tiedostolla kuin tilausten luonti. (ks. 4.6.2) Ainoana erona on XML-tiedoston ’`InvoiceType`’-kenttä, jolle annetaan arvoksi ”`invoice`” sekä lisäkenttiä, joilla saadaan laskulle tarvittavat tiedot, kuten eräpäivä tai laskutusosoite.

Opinnäytetyön toteutuksen myötä, toimeksiantaja sai ohjelmistointegraation, joka voidaan yhdistää Skyshop-verkkokauppaan. Toteutuksen myötä potentiaalisen Netvisoria käyttävän asiakkaan verkkokaupan käyttöönoton kustannukset laskivat huomattavasti, jos verrataan täysin alusta alkaen tehtyyn ja kustomoituun integraatioon. Toimeksiantajan omat kustannukset kyseisen Netvisor-integraation tekemiseen myös pienentyivät, koska tuloksena oli valmis

uudelleenkäytettävä lisäosa, jota voidaan käyttää suurimmassa osassa projekteista, joissa käytössä Netvisor sekä CakePHP-ohjelmistokehys.

Lisäosasta voidaan kuitenkin käyttää myös osioita koodipohjasta, vaikka sitä ei käytettäisi CakePHP-ohjelmistokehyksessä. Tiedostot lisäosassa sisältävät esimerkiksi valmiita XML-tiedoston rakentamiseen tarkoitettuja funktioita, joita tarvitaan tilausten viemiseen sekä muokkaamiseen ja ovat täten uudelleen käytettävissä, mikäli pohjana kuitenkin pysyy PHP-ohjelmointikieli.

6 Pohdinta

Toteutuksen aikana huomasin, ettei esimerkiksi rajapintakuvaus ollut ihan täydellinen. Usean rajapinnasta löytyvän resurssin kohdalla huomasin testaavani eri kombinaatioita ja tapauksia, jotta voidaan mahdollisimman hyvin ottaa huomioon virhetilanteet ja käsitellä ne niin, ettei verkkokaupan käyttäjälle aiheudu tästä lisävaivaa, esimerkiksi tilauksen uudelleen kasausta. Varalle tehtyjä toimia, joita kehityksen aikana muodostui, olivat esimerkiksi valmiiden XML-muotoisten tilauksien tallennus palvelimelle, niiden uudelleenkäyttöä ja tarkastelua varten ongelmatilanteissa. (ks. 4.6.2)

Rajapintaa testatessa ja käyttäessä huomasin myös asian, jota ei ollut otettu suunnittelussa huomioon, kuten rajapinnan kyselyiden maksimirajat. Mikäli kyselyitä lähetettiin suuria määriä kerralla peräkkäin, saattoi rajapinta vastata virheellä. Syynä oli esiasetettu raja, jonka määrästä ei ole tietoa. Ongelma korjattiin siten, että esimerkiksi tuote- tai asiakastietoja haettaessa, haetaan ennemmin tarvittavien asiakkaitten tai tuotteiden tarkemmat tiedot massana, eikä yksitellen.

Tietojen massana hakemisessakin oli kuitenkin rajansa, 500kpl kerralla ja tähänkin tuli pariin otteeseen törmättyä, kunnes katsoin tarkemmin rajapintakuvausta ja sain pilkottua suuremmat määrät alle 500kpl listoihin.

Ohjelmistointegraation suunnittelun, tekemisen, opiskelun aikana ja testatessa kartutin itselleni tietoa kyseisestä ohjelmistorajapinnasta. Nykyään toimin toimeksiantajan yrityksessä epävirallisena ”Netvisor-rajapinta”-tietopankkina. Ohjelmistointegraatio toi mukanaan myös paljon haasteita, esimerkiksi XML-tiedoston luomisessa ja lähetyksessä. XML-tiedosto sisälsi paljon attribuutteja, vaati oikean järjestyksen. Virheellisen XML-tiedoston siirtäminen Netvisor rajapintaan ei välttämättä palauttanut virheviestissään mitään viittausta siihen, missä ongelma piilee. Työn aikana kulutin paljon aikaa, kun vertailin luomaani XML-tiedostoa ja rajapintakuvausta. Kentät saattoivat olla väärässä järjestyksessä, tai esimerkiksi sisälsivät desimaaleja väärällä erottimella. Lopulta kuitenkin saatiin tulokseksi toimiva tilauksen luonti rajapintaa pitkin.

Saman tyylisiä ohjelmistointegraatioita on tullut toteutettua aikaisemminkin päivätöissä toimeksiantajalla. Opinnäytetyön aikana kuitenkin pääsin pintaa syvemmälle ja aikaisempaa tarkempaa tietoa sain eri komponenttien, tekniikoiden sekä protokollien käytöstä ja mikä niiden oikeaoppinen suhde on toisiinsa, kun syvennyin teoriaan.

Netvisor-rajapintaa vasten olin myös aikaisemmin tehnyt vähäisempiä, pienempiä integraatioita, mutta myös tähän yksittäiseen ohjelmistorajapintaan sain laajemman ja tarkemman kuvan, sekä osaan nyt paremmin suunnitella sekä hahmotella isoa kokonaisuutta projekteissa.

Ohjelmistointegraation valmistamista tuki toimeksiantajan työntekijöiden tarjoama tekninen apu ja aiempi kokemus. Opinnäytetyön alussa ja ohjelmistointegraation tekemisen aikana pidetyt suunnitelmapalaverit auttoivat hahmottelemaan rakennetta sekä lopputuotetta. Suunnitelmat toivat lisävarmuutta työstämiseen ja mahdollistivat integraation onnistuneen toteutuksen.

Lähteet

Beaulieu, A. 2020. Learning SQL, 3rd Edition. O”Reilly Media, Inc.

Cake Software Foundation, Inc. 2020. Intro. https://book.CakePHP.org/3/en/intro.html 20.10.2020

Cake Software Foundation, Inc. 2021. Intro. https://book.CakePHP.org/3/en/console-and-shells/option-parsers.html 18.03.2021

Cater, K. Chalmers, A. Ledda, P. 2002. Selective quality rendering by exploiting human inattentional blindness: looking but not seeing. https://dl.acm.org/doi/10.1145/585740.585744

Composer. 2021a. Documentation. https://getcomposer.org/doc/00-intro.md

Composer. 2021b. Documentation. https://getcomposer.org/doc/00-intro.md

Daityari, S. 2020a. Jump Start Git, 2nd Edition. Chapter 1: Introduction. O”Reilly Media, Inc.

Daityari, S. 2020b. Jump Start Git, 2nd Edition. Chapter 1: Introduction. O”Reilly Media, Inc.

DeBarros, A. 2018. Practical SQL. 6. Joining Tables In a Relational Database. No Starch Press.

Digisyke. 2020. Mitä Digitalisoida ja miksi? https://projects.tuni.fi/digisyke/miksi-digitalisoida/

Galin, D. 2018. Software Quality. 1.4 Software Errors, Faults and Failures. Wiley-IEEE Computer Society Press.

Git-scm. 2020. Documentation https://git-scm.com/docs 06.11.2020.

Google. 2020. Google Maps Platform. https://developers.google.com/maps/documentation/javascript/get-api-key 07.11.2020.

Guru99. 2020. Data Warehousing Tutorial – ETL Process. https://www.guru99.com/etl-extract-load-process.html 19.4.2020.

Hohpe, G. Woolf, B. 2003. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Chapter 1 – The need for integration. Addison-Wesley Professional.

Hohpe, G. Woolf, B. 2003. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Chapter - Solving integration problems using patterns. Addison-Wesley Professional.

IETF. 2012. The OAuth 2.0 Authorization Framework. https://tools.ietf.org/html/rfc6749#section-1.2. 08.10.2020

Json.org. 2021. Introducing JSON. https://www.json.org/

Kelley, D. OneIo. 2020. The Now and Future of Integration Platforms and iPaaS. https://blog.oneio.cloud/the-future-of-integration-platforms-and-ipaas 29.11.2020.

Kromann, F. 2018a. Beginning PHP and MySQL: From Novice to Professional. 22. Introducing MySQL. APress.

Kromann, F. 2018b. Beginning PHP and MySQL: From Novice to Professional. 22. Introducing MySQL. APress.

Lauret, A. 2019. The Design of Web APIs. Part 1: Fundamentals of API design. Chapter 1: What is API design?. Manning Publications.

National Institute of Standards and Technology. Federal information processing standards publication. 2015. Secure Hash Standard (SHS). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf 03.12.2020.

Microservice API Patterns. 2020. Pattern: API Key. https://microservice-api-patterns.org/patterns/quality/qualityManagementAndGovernance/APIKey 11.05.2020.

Microsoft Corporation. 2003a. Guidelines for Application Integration. What Is Application Integration. https://flylib.com/books/en/2.896.1/ 26.04.2020.

Microsoft Corporation. 2003b. Guidelines for Application Integration. Defining Your Application Integration Enviroment. https://flylib.com/books/en/2.896.1/ 26.04.2020.

Microsoft Corporation. 2003c. Guidelines for Application Integration Important considerations for application integration. https://flylib.com/books/en/2.896.1/ 26.04.2020.

Microsoft Corporation. 2003d. Guidelines for Application Integration. https://flylib.com/books/en/2.896.1/ 26.04.2020.

Niemi, P. 2019. Näin johdat integraatioprojektia onnistuneesti – 5 askelta. Blogikirjoitus. https://www.itewiki.fi/p/nain-johdat-integraatioprojektia-onnistuneesti-5-askelta 30.03.2020.

Oracle. 2020. Understanding Identity Concepts. Overview of oAuth Roles https://docs.oracle.com/en/cloud/get-started/subscriptions-cloud/ocuid/overview-oauth-roles.html . 11.05.2020.

Patni, S. 2017a. Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS. Fundamentals of RESTful APIs. Apress.

Patni, S. 2017b. Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS. Introduction – XML, JSON. Apress.

PHP-Group. 2021a. Manual – Implode. https://www.php.net/manual/en/function.implode.php

PHP-Group. 2021b. Manual – Document Object Model. https://www.php.net/manual/en/book.dom.php

Posti Group Oyj. 2020. Suuri verkkokauppa-tutkimus 2020. https://minun.posti.fi/hubfs/Tutkimukset/Suuri-verkkokauppatutkimus-2020_Posti.pdf

Preibisch, S. 2018. Api Developement: A Practical Guide for Business Implementation Success. Chapter 6: Api Implementation Details. https://learning.oreilly.com/library/view/api-development-a/9781484241400/html/466048_1_En_1_Chapter.xhtml

Puro, J. 2019. Kun edessä on ohjelmistojen hankinta, integraatiota kannattaa miettiä hyvissä ajoin. Blogikirjoitus. https://www.itewiki.fi/blog/2019/02/kun-edessa-on-ohjelmiston-hankinta-integraatioita-kannattaa-miettia-hyvissa-ajoin/ 14.4.2020.

Sayer, M. Aggarwal, A. Totty, B. Gourley, D. Reddy, S. 2002. http: The Definitive Guide. 12.2 Basic Authentication. O”Reilly Media, Inc.

Shahraki, A. Nikmaram, M. 2013. Human errors in computer related abuses. http://www.jatit.org/volumes/Vol47No1/12Vol47No1.pdf 15.4.2020.

Siriwardena, P. 2014. Advanced API Security: Securing APIs with Oauth 2.0, OpenID Connect, JWS, and JWE. Chapter 12: OpenID Connect. Apress.

Skycode Oy. 2020. Ohjelmistokehitys. https://skycode.fi/ohjelmistokehitys/ 17.4.2020.

Skycode Oy. 2020. Skyshop - B2B verkkokauppa. https://skycode.fi/skyshop/

SSL.com. 2021. What is SSL?. https://www.ssl.com/faqs/faq-what-is-ssl/
Swagger. API Keys. https://swagger.io/docs/specification/authentication/api-keys/ 9.5.2020.

TechTerms. 2020. Bootstrap. https://techterms.com/definition/bootstrap 07.11.2020.

TechTerms. 2020. CamelCase. https://techterms.com/definition/camelcase 30.01.2020.

Tidwell, D. Kulchenko, P. Snell, J. 2001. Programming Web Services with SOAP. O"Reilly media, Inc.

Tutorialspoint. 2021. What is SOAP? https://www.tutorialspoint.com/soap/what_is_soap.htm

Unicode, Inc. 2021. Basic Questions. https://www.unicode.org/faq/basic_q.html#2

Visma Solutions Oy. 2020. Ohjelmistorajapinta. https://support.netvisor.fi/fi/support/solutions/articles/77000466610-ohjelmistorajapinta 08.11.2020.

WhatIs. 2020. Framework. https://whatis.techtarget.com/definition/framework

Whatwg. 2020. DOM Living Standard. https://dom.spec.whatwg.org/ 29.11.2020.

WSO2 Team. 2017. Six business benefits of integration. https://wso2.com/library/articles/2017/02/six-business-benefits-of-integration/ 15.4.2020.

Zhao, S. Experian. 2017. What is ETL?(Extract, Transform, Load). https://www.edq.com/blog/what-is-etl-extract-transform-load/ 29.11.2020.