Integraatiokomponentti – Ratkaisu 10Duken järjestelmään liittämiseen

Roope Stenvall

Haaga-Helia ammattikorkeakoulu
Amk-opinnäytetyö
2022
Tradenomin tutkinto

Tiivistelmä

Tekijä(t)
Roope Stenvall

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Integraatiokomponentti – Ratkaisu 10Duken järjestelmän liittämiseen

Sivu- ja liitesivumäärä
43 + 0

Toiminnallisen opinnäytetyön aiheena oli rakentaa integraatiokomponentti 10Duke nimiselle ohjelmistoyritykselle. 10Duken tuote on lisensointiratkaisu, jota markkinoidaan ohjelmistoyrityksille. Lisensointiratkaisu mahdollistaa käyttöoikeuksien myöntämisen 10Duken asiakkaan myymään ohjelmistoon loppuasiakkaalle eli 10Duken asiakkaan asiakkaalle.

Lisensointiratkaisun käyttöönottaminen vaatii, että asiakas yhdistää oman järjestelmänsä siihen, sekä verkkokauppaan, joka voi olla asiakkaan toteuttama tai kolmannen osapuolen tarjoama. Kyseinen yhdistystapa vaatii teknistä osaamista tai kapasiteettia, joka ei ole kaikille asiakkaille mahdollista.

Integraatiokomponentin tarkoituksena on tarjota asiakkaalle vaihtoehtoinen tapa yhdistää 10Duken lisensointiratkaisuun. Integraatiokomponentti olisi valmiiksi yhdistetty lisensointiratkaisuun ja tarjoaisi asiakkaalle helpot rajapinnat yhdistykseen. Integraatiokomponentin toiminta perustuu kolmannen osapuolen maksupalveluntarjoajan lähettämiin tietoihin ostotapahtumista, joiden perusteella lisenssejä myönnetään. Työn laajuudessa vähittäisvaatimuksena komponentin piti luoda loppuasiakkaalle käyttötili 10Duken järjestelmään ja myöntää tilille lisenssi.

Teoriaosuudessa selvitettiin konsepteja integraatiokomponentin teknologisesta ympäristöstä. Lisäksi käytiin läpi digitaaliseen kaupankäyntiin liittyviä asioita, kuten erot maksupalveluntarjoajan ja kauppiastilin välillä. Esiteltiin integraatiokomponentin vaatimat liitoskohteet ja määriteltiin minimitoiminnallisuus. Esiteltiin myös komponentin kehittämistapa. Työn tavoitteiden ja menetelmien jälkeen tehtiin pieni listaus palveluntarjoajista, jotka komponenttiin voisi liittää sen valmistumisen jälkeen. Vertailun jälkeen esiteltiin maksupalveluntarjoaja, joka asiakasvaatimuksesta oli valittu komponenttiin yhdistettäväksi. Tarjoajan osalta esiteltiin komponentin liittämistä varten oleelliset osat, kuten webhook:t, joita komponentti käyttää sisäisessä logiikassaan.

Komponentin kuvauksessa esiteltiin toiminnallisuuden kannalta keskeinen rakenne ja logiikka. Komponentti käsittelee saapuvat webhook:t ja luo niiden pohjalta käyttötilejä ja mahdollistaa lisenssien myöntämisen. Komponentti saavutti sille asetetut minimivaatimukset, jotka todennettiin end-to-end-testin avulla.

Lopuksi pohdittiin jatkokehitystä, opinnäyteprosessia ja omaa oppimista. Tekijä oppi työn aikana paljon Java-verkkosovelluksen suunnittelusta ja luomisesta. Työ toteutettiin 30.9.2021 – 20.1.2022.

Asiasanat
Java, integraatio, prosessointi, lisenssit, ohjelmistokehitys

Sisällys

1 Johdanto 1
1.1 Ympäristö ja toimijat 1
1.2 Opinnäytetyön tarve ja tavoite 1
1.3 Rakennettavan komponentin kuvaus ja tekniikat 2
1.4 Rajaukset 2
1.5 Tekijän tavoitteet 3
1.6 Keskeiset käsitteet 3
2 Teknologinen ympäristö 5
2.1 Sovelluskontit ja Quarkus 5
2.2 Hibernate 5
2.3 Retrofit 6
3 E-Commerce - Verkkokaupankäynti 8
3.1 PCI DSS 8
3.2 Kauppiastili, Maksupalveluntarjoaja ja payment gateway 9
4 Työn taustat ja menetelmät 12
4.1 Projektin tavoite 12
4.2 Projektin vaatimukset 13
4.3 Projektin hallinta 15
5 Sopivien palveluntarjoajien listaus 17
6 FastSpring 19
6.1 API - rajapinta 19
6.2 Webhookit ja tapahtumat 20
7 Integraatiokomponentti 24
7.1 Komponentin rakenne 24
7.2 Käyttäjän liitos 25
7.3 Asiakkaan provisiointi 26
7.3.1 Provision rajapinta 27
7.3.2 Käyttäjätilin luonti – FastSpring ja CustomerMapping-rajapinta 28
7.3.3 Prosessoinnin kulku 29
7.3.4 FastSpring-eventien vastaanotto ja prosessointi 30
7.3.5 FastSpring-event:n käsittely pääpiirteittäin 31
7.3.6 CustomerMapping prosessointi 32
7.4 Tilinhallintaan ja lisensointiin liittäminen 32
7.5 Testaus 34
8 Pohdinta 37
8.1 Tavoitteet 37
8.2 Opinnäytetyö prosessi 38

8.3 Oman oppimisen tarkastelu..39
Lähteet 40

1 Johdanto

Tämän toiminnallisen opinnäytetyön tarkoituksena on luoda integraatiokomponentti 10Duke Software ltd:lle. 10Duke on ohjelmistoyhtiö, joka markkinoi tuotettaan B2B periaatteella. 10Duken myymä tuote on lisensointiratkaisu ohjelmistoyrityksille, jolla he voivat myydä omaan tuotteeseensa tai palveluun käyttölisenssejä omille asiakkaillensa eli loppuasiakkaille.

1.1 Ympäristö ja toimijat

Lisenssejä myönnetään loppuasiakkaille, joten 10Duken asiakkaan pitää yhdistää eli integroida lisensointiratkaisuun, jokin asiakastietoja sisältävä järjestelmänsä. Yleensä tämä on asiakkuuksienhallinta- eli CRM-järjestelmä. Liitos tarvitaan myös verkkokauppaan, joka voi olla 10Duken asiakkaan tai ulkopuolisen maksupalveluntarjoajan toteuttama. Yhdistämiseen käytetty logiikka hajautuu tällöin useampaan kohteeseen ja tällainen yhdistäminen vaatii asiakkaalta teknistä osaamista ja kapasiteettia. 10Duken ja asiakkaan välinen laskutus ei liity edellä mainittuun prosessiin.

Etuna edellä mainitussa toteutuksessa on nopeus ja laajat toteutuksen räätälöintiominaisuudet, kunhan se asiakkuuksienhallintajärjestelmässä on mahdollista. Etuna on myös lisenssien myöntämiseen tarvittavan tiedon saamisen helppous, sen ollessa keskitetysti CRM-järjestelmässä.

Huonoina puolina on vaikeudet virheen lähteen paikantamisessa mahdollisissa virhetilanteissa ja suurimpana haittapuolena on tarvittavan teknisen osaamisen ja kapasiteetin määrä, jota järjestelmien integraatio vaatii ja joillain yrityksillä tarvittavia resursseja ei ole.

1.2 Opinnäytetyön tarve ja tavoite

Opinnäytetyössä tehtävän integraatiokomponentin tekemisen tarpeena on saada kaupallinen etu tarjoamalla vaihtoehtoinen integraatioratkaisu, jonka kaltaista 10Duken kilpailijat eivät projektin tekohetkellä tarjoa. Integraatiokomponentti houkuttelee mahdollisesti pienempiä yrityksiä asiakkaiksi, joilla nykyisen toteutuksen vaatimaa teknistä kapasiteettia integraatiota varten ei ole. Komponenttiprojekti on myös kokeilu ohjelmiston toteuttamisesta Java pohjaista Quarkus-sovelluskehystä käyttäen, josta kerrotaan luvussa 2.

Edellä mainitun tarpeen täyttämiseksi tavoitteena on luoda integraatiologiikan sisältämä komponentti, joka tarjoaa 10Duken asiakkaalle vaihtoehtoisen tavan liittää oma järjestel-

mänsä 10Duken lisensointiratkaisuun. Komponentin tekemisen lisäksi opinnäytetyöprojektin tarkoituksena on selvittää mitä digitaalisen kaupankäynnin maksupalveluntarjoajat ja niihin liittyvät asiat ovat. Tämän lisäksi toteutetaan pieni vertailu 10Duken kannalta kiinnostavista palveluntarjoajista. Lisäksi kuvataan miten komponentti toteuttaa integraatiologiikan liikaa yksityiskohtiin menemättä.

1.3 Rakennettavan komponentin kuvaus ja tekniikat

Lähtötilanteen täysin manuaaliseen integraatiotapaan verrattuna syntyvä integraatiokomponentti tulee olemaan välipiste, joka yhdistää asiakkaan järjestelmän 10Duken lisensointijärjestelmään. Tämä säästää asiakkaan tekemän integraatiotyön määrää, kun pelkästään komponentin rajapintoihin pitää yhdistää, eikä integraatiologiikkaa tarvitse toteuttaa itse. Integraatiologiikka pysyy myös vain yhdessä paikassa.

Komponentin ollessa järjestelmien yhteyspiste on myös sen heikkous, koska järjestelmien välinen viestintä lakkaa, jos komponentti on toimintakyvytön, jonkin virheen tai muun ulkoisen syyn takia. Komponentti on yhdistetty vain tiettyihin maksualustoihin, eikä asiakas voi räätälöidä integraatiologiikkaa toisin kuin ilman komponenttia. Komponenttia voisi kuvata valmiina pakettina, joka on yksinkertaisempi ottaa käyttöön, mutta tarjoaa samalla yksinkertaisemman toteutuksen. Asiakas valitsee omien tarpeidensa mukaan, tarvitseeko monimutkaisemman, mutta itselleen räätälöidyn toteutuksen vai käyttääkö integraatiokomponenttia helpomman käyttöönoton saamiseksi.

Komponenttiin liittyviä töitä on tällä hetkellä asiakaskäytössä oleva lisensointiratkaisu sekä toinen tekeillä oleva lisensointiratkaisu, jonka sisälle komponentti rakennetaan ja tulee käyttämään osittain tämän luokkia ja komponentteja toteutuksessaan. Rakennettava komponentti tehdään Java-ohjelmointikielellä 10Duken tiloissa Linux-pohjaisella käyttöjärjestelmällä. Rakennustyökaluna on käytössä Maven. Yhdistettäväksi maksualustaksi on valittu FastSpring-nimisen yrityksen tarjoama ratkaisu asiakasvaatimuksen mukaisesti. FastSpring:stä kerrotaan tarkemmin luvussa 6.

1.4 Rajaukset

Projektin tarkoituksena ei ole tutkia ja läpikäydä 10Duken liiketoimintaan tai järjestelmiin liittyviä asioita, muuten kuin mitä tehtävän ohjelmiston tarpeellisuuden ja toiminnan ymmärtäminen vaatii. Integraatiokomponentti on vain välikappale ja ei puutu käyttäjätilienhallintaan eikä korvaa olemassa olevaa asiakkaan integrointiprosessia vaan tarjoaa asiakkaille toisen vaihtoehdon oman järjestelmänsä yhdistämisestä 10Duken lisensiointiratkaisuun.

Tähän liittyen komponentti itsessään ei myönnä tai hallitse lisenssejä vaan käsittelee maksupalveluntarjoajan lähettämät tapahtumat ja välittää lisensointijärjestelmälle käsitellyn tiedon. 10Duke ei käsittele asiakkaan ja loppuasiakkaan rahaliikennettä, vaan se tapahtuu edellä mainitun ulkoisen kaupankäyntialustan kautta ja alusta liitetään komponenttiin.

1.5 Tekijän tavoitteet

Projektin tekijän tavoitteena on oppia backend:ssä toimivan kokonaisuuden suunnittelu ja teko, joka toimii yhdistyspisteenä järjestelmien välillä. Tähän liittyen oppimistavoitteena on kehittää osaamista Java-verkkosovellus kokonaisuuden luomiseksi. Tavoitteena on myös syventää osaamista tietokannan käsittelystä ohjelmallisesti Hibernate-kirjastoa käyttäen. Lisäksi tutustuminen sovelluskonttien (containers) käytöstä komponentin rakennukseen ja testaukseen. Hibernate:sta ja sovelluskonteista kerrotaan luvussa 2.

1.6 Keskeiset käsitteet

Asiakas: 10Duken asiakas eli ohjelmistoyritys, joka ostaa 10Dukelta lisensointiratkaisun. Esimerkiksi kuvitteellinen tapaus, jossa Microsoft haluaisi ulkoistaa kauppaamansa ohjelmistonsa lisensoinnin. Tällöin Microsoft tekisi sopimuksen 10Duken kanssa.

Loppuasiakas/ostaja: 10Duken asiakkaan asiakas, joka on yritys tai yksityinen henkilö. Asiakas-esimerkkiin viitaten loppuasiakas olisi Microsoftin asiakas, joka ostaisi esimerkiksi käyttöjärjestelmiä. Loppuasiakas saisi lisenssit käyttöjärjestelmiin 10Duken lisensointiratkaisusta.

B2B: Business-to-Business. Kahden tai useamman yrityksen välistä kaupankäyntiä (Mourya & Gupta 2014, luku 2.2).

Integraatio:
Tämän työn osalta integraatio tarkoittaa järjestelmien liitosta tai yhdistämistä.

CRM-järjestelmä
Customer relationship management-järjestelmä, joka helpottaa asiakassuhteiden hallintaa yrityksessä. Asiakassuhteita pystytään hallitsemaan koko elinkaaren ajan markkinoinnista asiakaspalveluun. (Salesforce s.a.)

Lisensointiratkaisu
10Duken kauppaama tuote, jolla myönnetään ja hallitaan lisenssejä.

Lisenssi

Tämän työn osalta lisenssi on käyttöoikeus ohjelmistoon, joka myönnetään loppuasiakkaalle tämän ostaessa 10Duken asiakkaan tuotetta.

2 Teknologinen ympäristö

Tässä luvussa esitellään oleellisimmat toteutukseen käytettyjen teknologioiden konseptit. Teknologiset vaatimukset tulivat toimeksiantajalta. Sovellus rakennettiin Quarkus-sovelluskehystä käyttäen, joka optimoi Javaa sovelluskontteja varten. Projektin tietokannan hallintaa ohjataan Hibernate-kirjaston avulla ja projektin ollessa verkkosovellus sen HTTP-kutsujen toteutukseen on käytössä Retrofit2-kirjasto. Varsinkin sovelluskontit, Quarkus ja Hibernate ovat hyvin laajoja kokonaisuuksia, joten niiden tarkka läpikäyminen tämän työn laajuuden tai tavoitteen kannalta ei ole tarpeellista. Esittelen konseptit seuraavaksi.

2.1 Sovelluskontit ja Quarkus

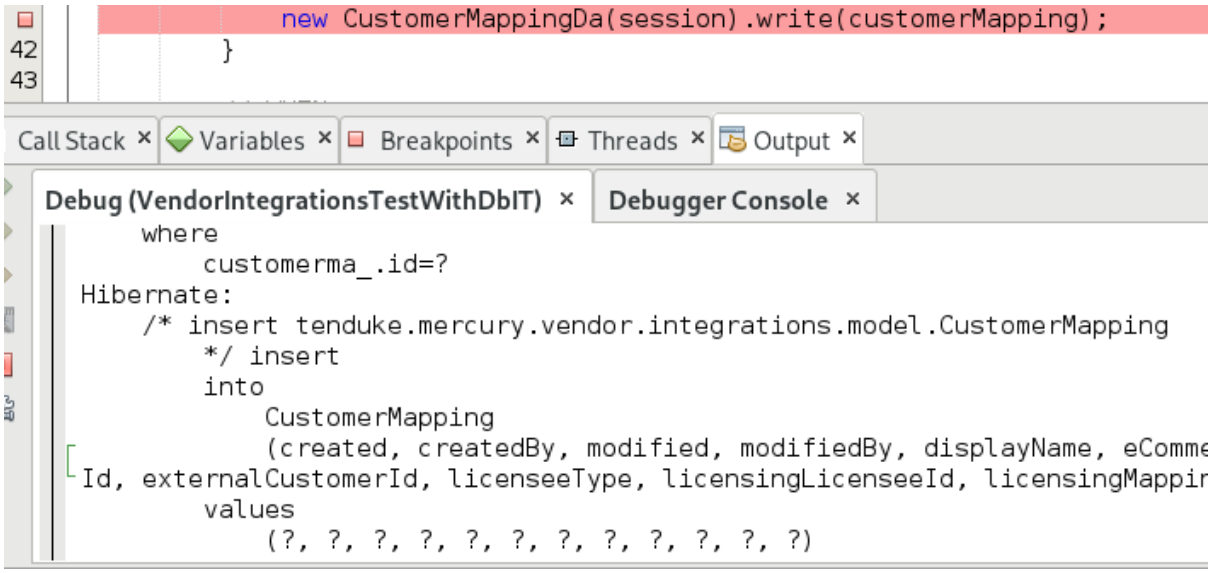
Sovelluskontit ovat virtuaalikoneiden tapaan paketoituja ympäristöjä, jotka sisältävät komponentteja ja ne eristetään muusta järjestelmästä. Pääero virtuaalikoneisiin on laajuudessa ja siirrettävyydessä sovelluskonttien ollessa kooltaan megatavuja virtuaalikoneiden gigatavuihin verrattuna. Sovelluskontteja käytetään usein yhtä tehtävää suorittaviin paketteihin, joita kutsutaan mikropalveluiksi. Sovelluskontit käyttävät jaettua käyttöjärjestelmää, toisin kuin virtuaalikoneet, jotka luovat täysin oman ympäristön käyttöjärjestelmineen. Sovelluskonttien kevyen rakenteen ansiosta niiden siirrettävyys on helppoa perinteisissä- ja pilviympäristöissä. Ne ovat myös sopiva väline useista mikropalveluista koostuvien pilvisovelluksien kehittämiseen. Rajoitteena sovelluskonteille on niiden vaatimus olla yhteensopiva alla olevan käyttöjärjestelmän kanssa. (Redhat 2020a.)

Quarkus on Redhat:n kehittämä sovelluskehys, joka optimoi Javaa sovelluskonttien käyttöön, pyrkimällä pienentämään Java-sovelluksen muistinkäyttöä ja käynnistysaikaa, joka puolestaan pienentää pilvipalvelun kuluja. Quarkus on suunniteltu toimimaan suosittujen Java sovelluskehysten, kirjastojen ja standardien kanssa. (Redhat 2020b.) Quarkus pyrkii saavuttamaan tavoitteen kevyiden ohjelmistojen tuottamisesta prosessoimalla mahdollisimman paljon ajonaikaisista prosesseista rakennusvaiheessa, jotta rakennettu sovellus sisältää vain ajonaikaisesti tarvittavat luokat. Rakennusvaiheessa tarvittavia luokkia ei edes ladata Javan virtuaalikoneeseen. Rakennusvaiheessa Quarkus pyrkii myös alustamaan kaikki sovelluksen käyttämät komponentit, jotta sitä ei tarvitse tehdä sovelluksen käynnistyessä. (Quarkus s.a.)

2.2 Hibernate

Hibernate on Java Persistence API:n ja Java Database Connectivity API:n toteuttava ORM- eli Object Relational Mapping-kirjasto, joka pyrkii ratkaisemaan ongelman tiedon

lataamisessa ja tallentamisessa, joka esiintyy olio- ja relaatiomallisen tiedon välillä (Hibernate s.a; Mihalcea s.a.). Hibernate mahdollistaa Java-olioiden helpon tallentamisen tietokantaan muuntamalla oliot vastaaviksi tietokannan tauluiksi. Tavallisiin tietokannan toimintoihin ei tarvitse itse kirjoittaa SQL:ää vaan Hibernate luo tarvittavat lausekkeet automaattisesti (Kuva 1). SQL on lyhenne sanoista Structured Query Language ja sillä hallitaan relaatiotietokantoja. Vaativampia tietokantakyselyitä varten Hibernate tarjoaa SQL:ään pohjautuvan Hibernate Query Languagen.



Kuva 1. Hibernaten luoma SQL-lause

Hibernatessa keskeisessä roolissa on Session-olio, jota käytetään aina tietokantaa käsiteltäessä. Sessio-oliot rakennetaan käyttäen SessionFactory-luokkaa, jonka luominen on resurssien kannalta kallista, joten niitä suositellaan rakentavan vain yksi jokaista käytettävää tietokantaa kohti. Session-olio jakamista säikeiden välillä ei suositella, koska vaarana on tiedon menetys tai tietokannan hetkellinen jumiutuminen. Toinen suositeltava käytäntö on tehdä tietokannan käsittely Hibernate:n transaktioiden sisällä, jonka voi avata Session-olion kautta. Esimerkiksi transaktion käyttö peruuttaa kaikki sen sisällä tehdyt muutokset, jos jonkin operaation aikana tapahtuu virhe. Transaktion sisällä olevaan tietoon ei vaikuta muiden järjestelmän käyttäjien toiminta. (Ottinger, Linwood & Minter 2022, luku 8.)

2.3 Retrofit

Retrofit on Squaren kehittämä Java-kirjasto, joka yksinkertaistaa JSON tai XML-tiedon lataamista verkkorajapinnasta. Ladattu tieto muunnetaan tavalliseksi Java-olioksi, joka määritellään kutsussa, joiden tekemiseen Retrofit käyttää OkHttp-kirjastoa. (Codepath 2021.) OkHttp on myöskin Squaren kehittämä Java-kirjasto HTTP-kutsujen lähettämiseen ja vastaanottamiseen (Codepath 2020).

Retrofit:ssa kutsuttavat HTTP-operaatiot määritellään Java-interfaceja käyttäen Retrofit:n tarjoamia annotaatioita. Annotaatiot määrittelevät kutsun parametrit ja HTTP-metodin. Kutsun paluuarvo on aina tyypitetty Call-instanssi, jolla määritellään mihin muotoon paluuarvo muunnetaan. (Codepath 2021.) Retrofit tarvitsee myös Java-luokan, joka vastaa tarvittavaa JSON-rakennetta. Lisäksi tarvitaan RetrofitBuilder-instanssi, joka määrittelee muuntajan, jota käytetään kutsun ja vastauksen muuntamiseen. Lisäksi builder:lla rakennetaan instanssi halutusta interface:sta, jolla kutsutaan haluttua operaatiota. Esimerkiksi suorittamalla GET-operaatio, vastauksesta saadaan suoraan Java-olio, jonka voi tallettaa tietokantaan.

3 E-Commerce - Verkkokaupankäynti

E-commerce tai verkkokaupankäynti on hyödykkeiden ostamista, myymistä ja vaihtamista internettiä käyttäen (Mourya & Gupta 2014, luku 1.1). Verkkokauppa-alustojen kehittyminen on mahdollistanut tehokkuuden parantamisen ja kulujen vähentämisen perinteisen fyysisen myynnin ja markkinoinnin sijasta (Shopify s.a.).

Verkossa myymisen aloittamiseksi tulee selvittää mitä maksuvaihtoehtoja tarjoaa asiakkaille. Kortilla maksamisen mahdollistamiseksi tulee noudattaa luottokorttiyritysten määrittelemää säädöstä nimeltä PCI DSS. (Wróbel-Konior 2017.) Tässä luvussa selvitetään lyhyesti verkkokaupankäyntiin liittyviä termejä kuten kauppiastili, maksupalveluntarjoaja ja payment gateway. PCI DSS:n piiriin joutumisen välttämiseksi tämä komponentti ei käsittele 10Duken asiakkaan myynnin aiheuttamaa rahaliikennettä, vaan maksutapahtumat käsittelee ulkoinen maksupalveluntarjoaja.

3.1 PCI DSS

Suuret luottokorttiyhtiöt (Visa, MasterCard, American Express, Discover ja JSB International) perustivat vuonna 2006 PCI SCC:n eli Payment Card Industry Security Standard Counsil:n, joka kehittää säädöksiä ja tukipalveluita maksuliikenteen turvallisuuden parantamiseksi. Jokainen perustajajäsenistä on sisällyttänyt Payment Card Industry Data Security Standard:n eli PCI DSS:n vaatimuksena myönnytysohjelmiinsa. Neuvosto ei kuitenkaan ota kantaa säädöksen noudattamisen valvontaan tai sanktioihin noudattamatta jättämisestä, vaan niistä vastaa itse maksumerkit tai maksunprosessoija. (PCI Security Standards Council s.a.)

Säädös kehitettiin pitämään ja parantamaan kortinhaltijoiden tietojen turvallisuutta ja mahdollistamaan säännöllisen tietoturvan käyttöönoton maailmanlaajuisesti. Säädös koskee kaikkia maksujen prosessointiin liittyviä tahoja mukaan lukien niitä, jotka jollain tapaa käsittelevät kortinhaltijan tietoja tai muita arkaluonteisia tunnistetietoja. Säädösten noudattaminen tarkastetaan vuosittain ja tarkastettavan tulee vahvistaa säädöksen riittävän laaja noudattaminen ennen arviointia. (PCI Security Standards Council 2018, 5–10.) Taulukossa 1 korkean tason katsaus PCI-säädöksen määrittelemistä asioista.

Taulukko 1. Korkean tason katsaus PCI-säädöksestä (PCI Security Standards Council 2018, 5)

Turvallisen verkon ja järjestelmän rakentaminen ja ylläpito	1. Kortinhaltijoiden tietoja turvaavan palomuurin asennus ja ylläpito. 2. Myyjän antamien oletus arvojen käyttökielto turvallisuuteen liittyvissä parametreissa esimerkiksi salasanat.
Kortinhaltijoiden tietosuoja	3. Suojaa kortinhaltijoiden tiedot 4. Julkisessa verkossa lähetettävien kortinhaltijoiden tietojen salaus.
Haavoittuvuuksien hallinta-operaatiot	5. Haittaohjelmilta suojautuminen ja virustorjunnan säännöllinen päivittäminen. 6. Turvallisten ohjelmistojen ja järjestelmien kehitys ja ylläpito.
Vahva pääsynhallinta	7. Kortinhaltijoiden tietoihin pääsyn rajoittaminen. 8. Järjestelmiin pääsyyn pyrkivien tunnistaminen ja vahvistaminen. 9. Fyysisen pääsyn rajoittaminen kortinhaltijoiden tietoon.
Säännöllinen verkkojen seuranta ja testaaminen	10. Kaiken verkossa olevan tiedon pääsyn seuranta ja valvonta. 11. Säännöllinen turvajärjestelmien testaus.
Tietoturvapolitiikan ylläpito	12. Tietoturvapolitiikka henkilökunnalle.

PCI-säädöksen noudattaminen ei ole lailla vaadittua, mutta ollessaan kansainvälisesti käytössä, sen noudattamatta jättäminen aiheuttaa suuria sanktioita. Tietomurtojen ja niiden aiheuttamien lisätarkastusten lisäksi noudattamattomat yritykset joutuvat maksamaan suuria kuukausittaisia sakkoja niin kauan, kunnes ongelma on korjattu. Sakon suuruus riippuu yrityksen maksutapahtumien määrästä ja kuinka monta säädöksen kohtaa on rikottu. (Wróbel-Konior 2017.)

3.2 Kauppiastili, Maksupalveluntarjoaja ja payment gateway

Kauppiastili tai merchant account on pankkitili, jolle raha menee ennen kuin se talletetaan myyjän tilille. Verkkokaupankäynnissä kauppiastilillä on suuri rooli, koska kaikki maksut myytävästä tuotteesta riippumatta, olkoon se fyysinen tuote tai ohjelmisto, talletetaan tälle

yhdelle tilille ja talletetut tuotot maksetaan yleensä kerran viikossa. Kauppiastiliä tarvitaan tapauksissa, joissa maksuja vastaanotetaan pankki- ja luottokorteilta, koska vain kauppiastilit voivat kerätä näitä maksuja. Kauppiastilin saamisen jälkeen kauppiaalle myönnetään uniikki tunnistearvo. (BBCIncorp 2021.)

Maksupalveluntarjoaja tarjoaa kauppiaalle tavan myydä tuotettaan elektronisia maksuvälineitä käyttäen maksukorteista digitaalisiin lompakoihin ja pankkisiirtoihin. Palveluntarjoaja voi olla liitoksissa usean pankin, maksu- ja maksukorttiverkoston kanssa, joka vähentää kauppiaan riippuvaisuutta rahaa hallitseviin instituutioihin, koska palveluntarjoaja hoitaa kyseiset yhteydet. (Ecommerce Platforms s.a.)

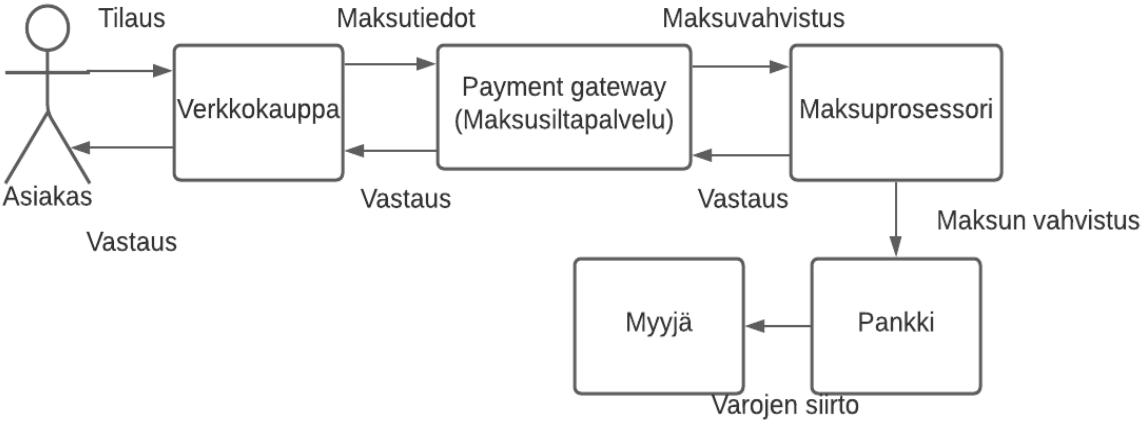
Maksupalveluntarjoajat käyttävät vain yhtä kauppiastiliä yhdellä tunnistearvolla, jolle luodaan uusi alatili uutta kauppiasta lisättäessä (BBCIncorp 2021). Kauppiastiliä pelkästään käytettäessä, jokaiselle myyjälle myönnetään oma yksilökohtainen tili ja ennen sen myöntämistä myyjän taustat tutkitaan tarkemmin kuin kolmannen osapuolen palveluntarjoajaa käytettäessä. Kauppiastilit keskittyvät maksujenprosessointiin, kun taas palveluntarjoajilta voi saada laajemman ratkaisun. (Ecommerce Platforms s.a.)

Payment gatewayn on palvelu, jonka tarkoitus on turvallisesti prosessoida maksuja. Se salaa tietoa ja parantaa käyttökokemusta sulavoittamalla maksamisprosessia. Luotto- ja pankkikorttimaksuihin tarvitaan maksuprosessori ja payment gateway on prosessorin ja verkkokauppasivun välissä. Useissa tapauksissa maksupalveluntarjoaja tarjoaa payment gateway-palveluita. (OroCommerce 2021.)

Hyvinä puolina maksupalveluntarjoajaa käytettäessä on nopea käyttöönotto (BBCIncorp 2021). Sekä valmiin kokonaisuuden saaminen sisäänrakennetuin työkaluin, kuten laskutusta tai raportteja varten (Ecommerce Platforms s.a.). PCI DSS:n ehdot täyttävän Payment gateway:n käyttäminen ei täysin poista kauppiaan vastuuta ehtojen täyttämisestä. Näiden ehtojen täyttämisen vahvistamiseen tarjotaan itse täytettäviä lomakkeita. (Mai 8.6.2017.)

E-commerce-kauppiaita koskevat kolme lomaketta, SAQ A, SAQ A-EP, SAQ D. A-tasolla kauppias on ulkoistanut kaiken maksukortteihin liittyvän palveluntarjoajalle. A-EP-tasolla kauppias tarjoaa verkkosivullaan maksulomakkeen, joka lähetetään palveluntarjoajalle. D-tasolla kauppias ei täytä A ja A-EP:n kriteerejä. Yleensä palveluntarjoajat tarjoavat lomakkeiden ehdot täyttävät tavat, jolla hoitaa korttitietojen hallinnan ja täyttää sitä vastaavan lomakkeen. (Mai 8.6.2017.)

Haittapuolina palveluntarjoajaa käytettäessä pelkkään kauppiastiliin verrattuna on mahdollinen heikkous tuen saamisessa esimerkiksi maksuliikennettä koskevissa asioissa. Kysymyksiä varten voi löytyä erillisiä resursseja, mutta suoraa apua voi olla vaikea saada. Kauppiastiliä käytettäessä tuen saaminen on usein helppoa, koska yksi kauppiastili on suunniteltu yhtä asiakasta kohden toisin kuin palveluntarjoajan usean asiakkaan järjestelmä. (Ecommerce Platforms s.a.)



Kuva 2. Maksuprosessorin ja gatewayn suhde (mukaillen OroCommerce 2021)

Kuvassa 2 asiakas tekee tilauksen verkkokaupan sivulla. Payment gateway salaa tiedon ja lähettää sen maksuprosessorille, joka tarkistaa, että asiakkaan syöttämät tiedot ovat oikein. Vahvistuksen jälkeen maksuprosessori lähettää tiedon pankille, jossa tapahtuma hyväksytään tai hylätään. Hyväksytyssä tapauksessa prosessori pyytää varojen siirtoa ja lähettää vastauksen payment gateway:lle, joka puolestaan lähettää tapahtuman tiedot verkkokaupan sivulle. Lopuksi maksuprosessori siirtää varat pankille, jossa kauppiaalla on tili. (OroCommerce 2021.)

4 Työn taustat ja menetelmät

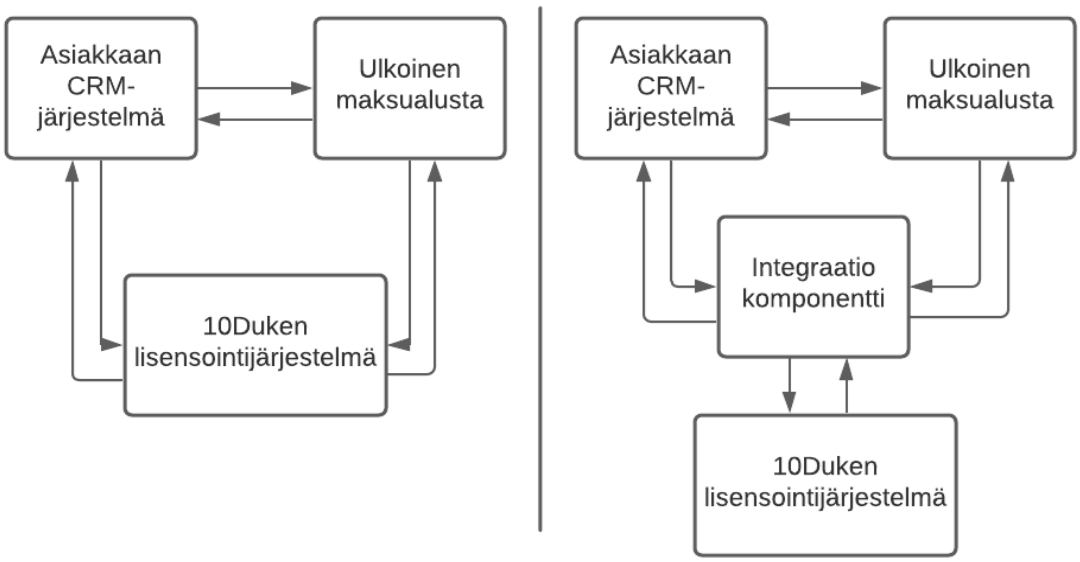
Työn kohdeorganisaatio on 10Duke, joka on ohjelmistokehitysyritys, jonka päätuote on lisensointiratkaisu ohjelmistoyrityksille. 10Duken asiakkaat kauppaavat omaa tuotettaan yrityksille tai yksityishenkilöille ja tarvitsevat tavan myöntää käyttöoikeudet eli lisenssin tuotteeseensa. Jotta lisensointiratkaisun saa käyttöön asiakkaan pitää liittää siihen oma asiakkuuksienhallintajärjestelmänsä tai vastaava, johon tallentuu ostajan tiedot, joita käytetään lisenssien myöntämiseen. Järjestelmien liittämiseen liittyy myös jokin yleensä kolmannen osapuolen tarjoama maksualusta, jonka avulla asiakas myy tuotettaan.

Edellä mainittu integraatiotapa mahdollistaa käytännössä minkä tahansa järjestelmän liittämisen lisensointiratkaisuun ja tuloksena on nopeasti toimiva lisensointi järjestelmien ollessa suoraan liitoksissa toisiinsa. Kyseisen tapa on teknisesti vaativa ja aikaa vievä prosessi, mikä on sen heikkous. Joillain pienemmillä yrityksillä ei ole tarvittavia resursseja järjestelmien välisen liitoksen muodostamiseen. 10Dukella oli tarve vaihtoehtoisen ratkaisun tarjoamiseen järjestelmien liittämiseksi, jolla pienempiä yrityksiä saadaan houkuteltua asiakkaiksi.

4.1 Projektin tavoite

Tämän työn tuloksena syntyvän komponentin tarkoituksena oli toimia integraatiopisteenä lisenssejä myyvän tahon sekä lisenssinhallintaohjelmiston välillä. Tämä mahdollistaa kyseessä olevien komponenttien toiminnan osana toimivaa monitahoista ohjelmistotoimitusta. Tämän tarkoituksena on mahdollistaa saumaton ohjelmistolisenssien myynti. Työssä syntyvä komponentti tulisi toimimaan vaihtoehtoisena tapana manuaalisen järjestelmien liittämisen rinnalla, joista asiakas voisi valita omien tarpeidensa mukaisesti.

Valmistuessaan komponentti olisi valmiiksi yhdistetty 10Duken lisensointiratkaisuun ja johonkin ennalta valittuun maksualustaan. Tämä yksinkertaistaa 10Duken asiakkaan integraatiotyötä, kun yhdistäminen vaaditaan vain komponentin rajapintoihin ja integraatiologiikka on jo valmiiksi toteutettu. Ulkoisen maksualustan kautta komponentti saa tapahtumatiedot tilien luomiseen ja tilauksiin liittyvissä tapauksissa. Kuvassa 3 ylätason kuvaus integraatiotapojen erosta.



Kuva 3. Perinteinen integraatio vasemmalla. Komponentin kanssa oikealla

4.2 Projektin vaatimukset

Projekti alkoi projektisuunnitelman tekemisellä, jossa esitettiin projektin vaatimukset. Maksualustaksi, johon integraatiokomponentti yhdistettiin, valittiin FastSpring:n tarjoama ratkaisu asiakasvaatimuksen mukaisesti. Projektissa esitellään FastSpring:n olennainen toiminta luvussa 6. Yhtenä projektin vaatimuksena oli tuottaa pieni listaus eri palveluntarjoajien järjestelmistä, johon integraatiokomponentti voitaisiin yhdistää sen jälkeen, kun FastSpring:n täysi yhdistys on suoritettu. Vertailu oli kuitenkin viimeisenä prioriteettilistalla, koska ensin piti saada komponentti rakennettua työn vähittäisvaatimusten mukaisesti.

Komponentti tarvitsi erilaisia liitoksia toimiakseen järjestelmien yhteyspisteenä (Taulukko 2). Liitokseen tarvittiin FastSpring:n järjestelmä, joka hoitaa maksuliikenteen ja välittää tiedon käyttäjätilin luonnista ja tilauksen tekemisestä webhook:n avulla. Webhook:ja lähetetään järjestelmästä jonkin tapahtuman seurauksena ja sisältävät oleellisen tiedon tapahtumaan liittyen. Liitokset tarvittiin myös lisensointiratkaisuun, joka koostuu 10Duken tilinhallintajärjestelmästä ja lisensointijärjestelmästä. Liitos tilinhallintaan tarvittiin loppuasiakkaan käyttäjäprofiilin luomiseksi ja 10Duken lisensointijärjestelmään lisenssien myöntämiseksi ja hallitsemiseksi. Aluksi suunnitelmassa oli yhdistää komponentti toiseen vielä rakenteilla olevaan lisensointijärjestelmään, mutta toinen liitos jätettiin työn ulkopuolelle ajan puutteen takia.

Taulukko 2. Komponentin liitoskohteet

Liitoksen kohde	Miksi?
FastSpring	Saadaan tieto maksupalvelussa luodusta tilistä ja tilauksen tekemisestä.
10Duken tilinhallinta	Mahdollistaa käyttäjä- tai yritystilin luomisen 10Duken järjestelmään. Kyseiselle tilille tehdään lisenssinhallintaoperaatioita.
10Duken lisensointijärjestelmä	Mahdollistaa lisenssienhallinnan tilinhallinnasta löytyvälle tilille.

Vähittäisvaatimuksina komponentilla tuli olla toiminnallisuus käynnistää FastSpring:n lähettämistä webhook:sta riippumaton tilinluontiprosessi rajapinnan kautta, koska tilinluomisen pitäisi onnistua esimerkiksi asiakkaan järjestelmän kautta. Tätä kautta komponentin pitäisi pystyä luomaan tili 10Duken tilinhallintaan ja FastSpring:n järjestelmään.

FastSpring:n järjestelmään liittäminen tarkoittaa, että komponentti käyttää lisenssinhallintakutsujen rakentamiseen ja yhtenä polkuna tilinluomisen aloittamiseen FastSpring:ltä tulevaa webhook tietoa. Komponentin toiminnan kannalta FastSpring:n lähettämät webhook:t laitettiin prioriteettijärjestykseen (Taulukko 3). Tämän opinnäytetyön laajuudessa FastSpring:n webhook:en käsittelyn kannalta olennainen toiminnallisuus saavutetaan prioriteetilla 1. Tämä tarkoittaa, että komponentti pystyisi luomaan käyttäjätilejä 10Duken tilinhallintajärjestelmään ja tekemään lisenssien aktivointikutsuja lisensointijärjestelmään.

Edellä mainitun toiminnallisuuden toteutuminen luo tarvittavan pohjan jäljellä olevien toimintojen toteuttamiselle, jotka on merkitty prioriteetilla 2. Prioriteetilla 1 merkityt toiminnot olivat minimivaatimus ja loput toteutettaisiin, jos aika riittää. 3 merkityt toiminnot tarkoittavat, että niiden lisääminen on epävarmaa, tuetun toiminnallisuuden tai toteutustavan takia. Esimerkiksi käyttäjätilin sähköpostin muuttaminen aiheuttaisi ketjureaktion, jolloin kaikki kyseisen tilin lisenssit pitää päivittää.

Taulukko 3. Olennaiset FastSpring:n webhook:t

Webhook:n nimi	Kuvaus	Prioriteetti
Account created	Loppuasiakkaalle on syntynyt tili FastSpring:n järjestelmään.	1
Order completed	Loppuasiakas on ostanut tuotteen. Voi sisältää kertaostoksia ja tilauksia.	1
Subscription activated	Loppuasiakas on aktivoinut tilauksen.	1
Subscription updated	Tilauksen tietoja on päivitetty	2

Subscription deactivated	Tilaus on poistettu käytöstä.	2
Subscription charge completed	Seuraava tilauskausi on maksettu.	3
Account updated	Loppuasiakkaan tiliä on päivitetty.	3

Esimerkki käyttötapauksesta, joka komponentilta vaaditaan. Asiakasyritys X saa uuden asiakkaan A, joka valitsee FastSpring:n verkkokauppasivulta haluamansa tuotteen ja maksaa ostoksen. Oston yhteydessä A:lle syntyy tili FastSpring:iin. Tilin luomisesta ja ostotapahtumasta FastSpring lähettää webhook:t integraatiokomponentille. Komponentti ottaa webhook-event:t vastaan ja käynnistää asiakkaan luontiprosessin. Komponentti luo webhook:n mukana tulleen tiedon avulla tilin A:lle 10Duken identiteetinhallintajärjestelmään. Tilin luomisen jälkeen komponentin pitää onnistuneesti kutsua lisensointijärjestelmää ja myöntää A:lle lisenssi X:n tuotteeseen tilauksesta syntyneen webhook-tiedon perusteella.

4.3 Projektin hallinta

Projektisuunnitelmassa oli määritelty aikataulu ja projektille asetettiin ohjausryhmä, johon kuuluivat itseni lisäksi ohjaaja koululta ja 10Duken edustaja. Ohjausryhmän kanssa pidettiin ohjauskokouksia kuukauden välein ensimmäisen ollessa kuukauden päässä aloituskokouksesta. Tarvittaessa epävirallisia ohjauskokouksia voitaisiin järjestää jonkun ohjausryhmän jäsenen toiveesta. Ohjauskokouksissa pohdittiin edistymistä ja keskusteltiin seuraavan raportointikauden sisällöstä.

Aikataulullisesti suunniteltiin, että raportti valmistuisi kolmanneksen joka raportointikaudella. Ensimmäisen kolmanneksen raportointikauteen kuului johdannon kirjoittaminen ja aineiston kerääminen. Komponentin edistymiselle ei määritelty erityisiä päivämääriä vaan sen toteuttaminen tapahtui yhtä kestoisesti koko projektin läpi. Raportin komponenttia kuvaavaa osaa täydennettiin, kun jokin komponentin osa valmistui.

Ennen projektin aloittamista projektin riskinä tunnistettiin komponentin etenemisen mahdollinen hidastuminen puutteellisen ymmärryksen vuoksi tekeillä olevasta osuudesta tai käytetystä teknologiasta, koska suurin osa projektissa käytetyistä teknologioista oli itselleni vähemmän tunnettuja. En myöskään ollut rakentanut vastaavaa kokonaisuutta aikaisemmin. Itse komponentin teknisestä toteutuksesta ei ollut selvää suunnitelmaa, joten kehittäminen tapahtui pienissä osissa iteroiden.

Komponentin kehittämisessä käytettiin 10Duken GitLab versionhallintaympäristöä. GitLabiin lisättiin ongelma eli issue ja siihen liittyvä liitospyyntö eli merge request tai MR. Tehdyt lisäykset katselmoi kollega, joka tuntee kehitettävän projektin ja teki tarvittaessa muutosehdotuksia. Mahdolliset muutokset lisättiin ja katselmointi tehtiin uusiksi. Kun MR on saatu hyväksyttävään tilaan, se liitettiin master-haaraan ja siirryttiin seuraavaan ongelmaan.

5 Sopivien palveluntarjoajien listaus

Tämän luvun tarkoitus on selvittää pieni määrä 10Duken kriteerejä vastaavia palveluntarjoajia, joiden integraatiokomponenttiin liittämistä voidaan harkita tulevaisuudessa, kun komponentin toiminnallisuus on varmistettu FastSpring:n kanssa. Taulukossa 4 esitelty listaus on hyvin pintapuolinen, joten tarjoajien täysi sopivuus komponentin rakenteen ja toimintojen kannalta pitää varmistaa.

10Duken asiakkaiden ollessa ohjelmistoyrityksiä palveluntarjoajia etsittäessä keskitytään toimijoihin, joiden tuote sopii ohjelmistokaupankäyntiin. Lisensoinnin kanssa vaadittu toiminnallisuus on uusiutuvien tilausten tukeminen asiakkaiden siirtyessä enemmän tilauspohjaiseen toimintamalliin.

Integraatiokomponentin toiminnan perustuessa palveluntarjoajan lähettämiin webhook:hin, palveluntarjoajan tulee tietysti tukea niiden lähettämistä. Vaatimuksena on myös mahdollisuus tilin luomiseen loppuasiakkaalle, koska tätä tietoa tarvitaan lisenssienhallintaan. Tarvitaan myös hyvä dokumentaatio integrointia varten, sekä testaamiseksi tarjoajalla tulisi olla sen mahdollistama ympäristö

Maksutapoina on vähintään oltava isojen korttiyhtiöiden maksukortit, jonka seurauksena myös PCI DSS:n noudattaminen täyttyy. Integraatiokomponentin ei tarvitse tietää maksuprosessista mitään, kunhan se saa tarvittavat käyttäjätili- ja tilaustiedot webhook:n kautta. Tämä tarkoittaa, että palveluntarjoajan järjestelmään voi olla liitettynä ulkopuolinen payment gateway, joka prosessoi korttimaksut.

Taulukko 4. Pieni listaus palveluntarjoajista

	Stripe	Chargebee
Uusiutuvien maksujen tuki	Kyllä	Kyllä
Webhook tuki	Kyllä	Kyllä
Loppuasiakkaan tili	Kyllä	Kyllä
Dokumentaatio kehittäjille	Kyllä	Kyllä
Tuki luottokorteille	Kyllä	Kolmannen osapuolen kautta
Testiympäristö	Kyllä	Kyllä

Stripe on maksupalveluntarjoaja, jonka kanssa ei tarvitse erillistä kauppiastiliä tai payment gateway:ta. Stripe:n tarjontaan kuuluu myös uusiutuvien tilausten tukeminen, joten niitä

varten ei tarvitse kolmannen osapuolen ratkaisua. (Stripe Support s.a.) Stripe tukee monia maksutapoja luottokorttien lisäksi ja tarjoaa kattavan dokumentaation (Stripe s.a.a.). Sen keskeisenä osana on loppuasiakkaan tili, jonka tietoja käytetään tilausten ja kertaostojen veloitukseen (Stripe s.a.b.). Stripe:n voi konfiguroida lähettämään webhook:a monen tyyppisistä event:stä, kuten customer subscription activated ja account created (Stripe s.a.c.; Stripe s.a.d.). Integraation testausta varten tarjotaan omat rajapinnat (Stripe s.a.e.).

Chargebee on laskutussovellus, joka on erikoistunut uusiutuvien maksujen tarjoamiseen ja hallintaan (Chargebee s.a.a). Chargebee ei itsessään tarjoa maksujen prosessointia, vaan siihen pitää liittää kolmannen osapuolen payment gateway, kuten Stripe. Payment gateway:n lisäksi myyjän pitää luoda kauppiastili, jos sitä ei saa payment gateway:n kautta. (Chargebee s.a.b.) Kuvassa 2 oli esitelty maksunprosessoinnin kulku, jossa ostotapahtuma kulki verkkokaupasta payment gateway:n läpi. Kuvaan 2 verrattuna Chargebee:n palvelu sijaitsee verkkokauppasivun ja payment gateway:n välissä. Itse maksusivu voi sijaita myyjän tai Chargebee:n palvelussa ja Chargebee välittää maksutiedon payment gateway:lle. (Chargebee s.a.c.)

Kuten Stripe:lla, Chargebee:llä on käyttäjätilit loppuasiakkaille, jotka sisältävät loppuasiakkaiden tietoja. Tilin voi luoda rajapinnan ja verkkokäyttöliittymän kautta tai se luodaan automaattisesti tilauksen yhteydessä. (Chargebee s.a.d.) Yhteneväisyyksiä esiintyy myös event ja webhook asioissa. Esimerkiksi jos tilauksiin tai asiakkaisiin liittyen tapahtuu muutoksia, ne rekisteröidään event:nä ja niistä voidaan lähettää webhook:ja. Webhook:n lähetystä voi myös testata. (Chargebee s.a.e.; Chargebee s.a.f.)

Ensivaikutelmaltaan molemmat edellä esitetyt palveluntarjoajat ovat hyviä vaihtoehtoja integraatiokomponentin kannalta. Molemmat tarjoavat komponentin toiminnan kannalta oleellisia toimintoja, kuten webhook:t ja toistuvat tilaukset. Chargebee ei suoraan mahdollista maksukorttimaksujen prosessointia, mutta tarjoaa monta vaihtoehtoa payment gateway:n valintaan (Chargebee s.a.b.).

6 FastSpring

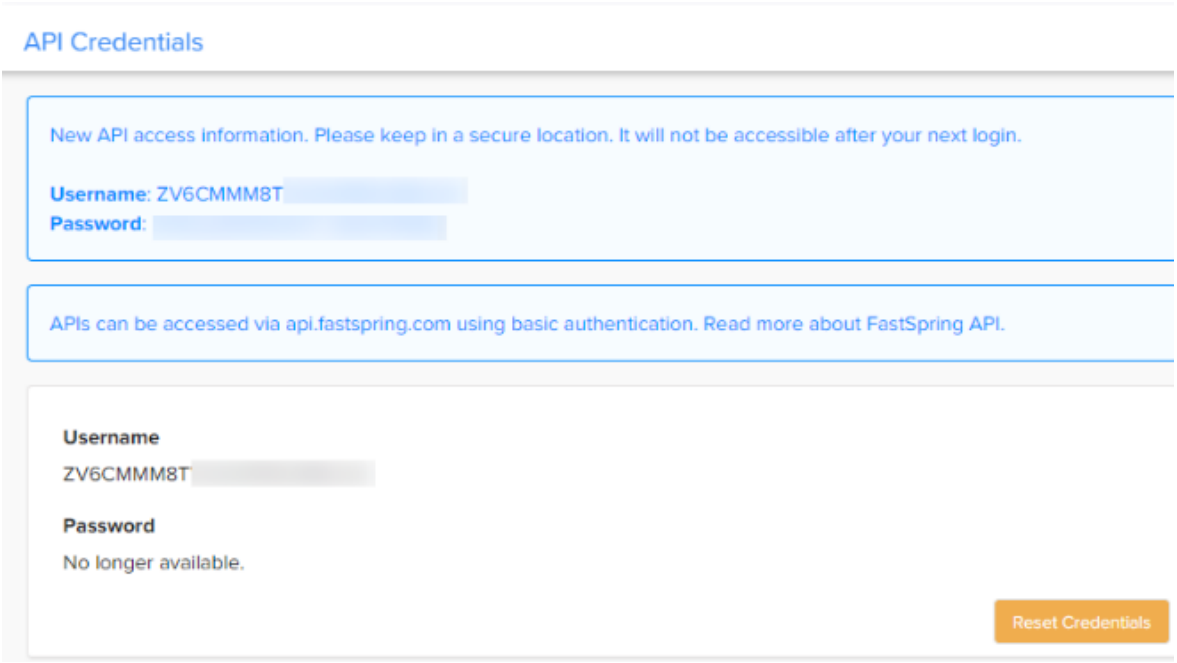
FastSpring:n tuote on täyden palvelun verkkokaupparatkaisu ohjelmistoyrityksille. FastSpringin tuote tukee montaa maksutapaa kattaen myös siirtokulut, valuuttamuunnokset ja verotuksen hallinnan. Palvelun avulla voi myös tarjota tilauspohjaista myyntiä uusiutuvilla maksuilla kertaostojen sijasta. Palvelussa voi rakentaa oman kauppasivun ja se tarjoaa integraatioita kolmansien osapuolien kuten Google Analytics:n kanssa. FastSpringin ratkaisu noudattaa myös PCI-standardeja ja GDPR-säädöksiä. (FastSpring s.a.a.)

FastSpring tarjoaa kauppasivulle laajat määrittely- ja testausominaisuudet, jotta kaupankäynnistä saisi tarpeenmukaisen hyödyn. FastSpring tarjoaa myös kauppiaan sivulle upotetun ostoikkunan, joka mahdollistaa loppuasiakkaalle myymisen kauppiaan omilta sivuilta. (Dwyer 2020.) Integraatiokomponenttiin liittyen 10Duken asiakas luo tilin ja kauppasivun FastSpring:iin ja liittää tiliin integraatiokomponentin, jotta tapahtumatiedot päätyvät komponentille prosessoitaviksi loppuasiakkaan tili- ja lisenssitoimia varten.

6.1 API - rajapinta

FastSpring tarjoaa rajapinnan, jonka avulla verkkokauppaa voi hallita ohjelmallisesti. Hallinta tapahtuu tavallisilla HTTP-metodeilla, jotka FastSpring:n tapauksessa pääosin ovat: GET, POST ja DELETE. Rajapintakutsuilla lähetettävän ja vastaanotettavan JSON-tiedon rakenne on samankaltaista, kuin mitä käytetään myöhemmin esiteltävissä webhook:ssa. Webhook:sta poiketen rajapintaa käytettäessä käyttäjä on aina viestinnän aloittaja. (FastSpring s.a.b.)

Ennen rajapinnan käyttämistä pitää luoda valtuutustiedot (credentials) omalle kaupalle, jotta kutsuja voi tehdä. Valtuutustiedot koostuvat käyttäjänimestä ja salasanasta ja ne luodaan ”API Credentials”-välilehdellä, kun FastSpring App:iin on kirjauduttu sisään. Ennen luontia sivulla on ”Create”-painike, jonka painamisen jälkeen käyttäjänimi ja salasana näytetään sivulla (Kuva 4). Tunnukset suositellaan tallentamaan, koska salasana on näkyvissä vain seuraavaan sisäänkrjautumiseen asti. Valtuutustiedot voi nollata, mutta kaikki niihin liitetyt toiminnot lakkaavat toimimasta, koska kutsut käyttävät vielä vanhaa valtuutusta. Rikkoutuneiden kutsujen toiminnallisuuden pystyy palauttamaan päivittämällä valtuutustiedot kyseisistä kutsuista. (FastSpring s.a.b; FastSpring s.a.c.)



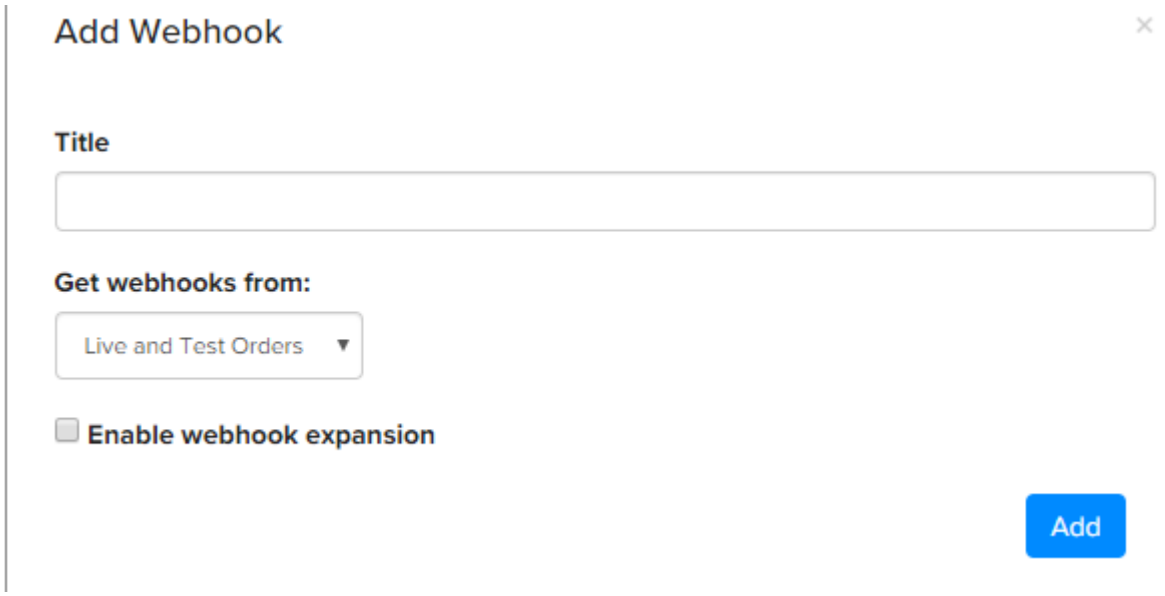
Kuva 4. FastSpring valtuutustiedot ”Create”-napin painalluksen jälkeen (FastSpring s.a.c.)

Integraatiokomponentin kannalta olennaiset rajapinnan päätepisteet ovat ”/accounts” ja ”/events”. accounts-päätepiste mahdollistaa FastSpring-tilien hakemisen, päivittämisen ja tilin luomisen, joka on tärkeä osa integraatiokomponentin toimintaa, koska se mahdollistaa käyttötapauksen, jossa loppuasiakkaan rekisteröityminen ei tapahdu FastSpring:n välityksellä. Events-päätepisteestä voi kutsua prosessoimattomiksi jääneitä webhook-tapahtumia.

6.2 Webhookit ja tapahtumat

FastSpring:n webhook:ja on kahdenlaisia. Selain pohjaisissa webhook:ssa käyttäjän selaimessa suoritetaan JavaScript-funktio, joka käsittelee käyttäjän tilauksen, jollain tavalla ja aktivoi tapahtumia tilauksen tiedon perusteella. Palvelin webhook:ssa FastSpring lähettää JSON-tietoa ulkoisiin verkko-osoitteisiin tai päätepisteisiin, joiden perusteella voi päivittää tietokantoja ja suorittaa muita tapahtumia. (FastSpring s.a.d.) Palvelin webhookit ovat integraatiokomponentin kannalta olennaisia.

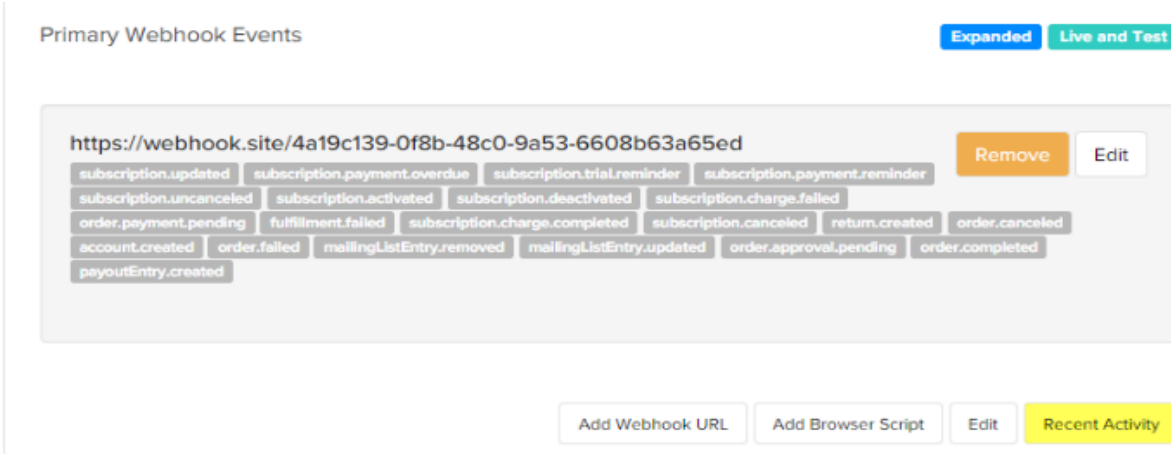
Webhook:t voi ottaa käyttöön FastSpring App:n integrations-välilehdeltä, joka luo webhook-tilan, johon webhook:ja voi lisätä. Kuvassa 5 näkyy uuden webhook:n lisääminen. Webhook:n voi nimetä erottamisen helpottamiseksi, jos webhook:ja on useampi. Samalla voi määritellä mistä tilauksista webhook-tapahtumia lähetetään. Vakiona valittuna on ”Live”- ja testitilaukset, mutta halutessaan voi valita jommankumman. Viimeisenä voi valita käyttääkö webhook:n laajennusta, jossa lähetetty JSON-kuorma sisältää enemmän tietoa. (FastSpring s.a.d.)



Kuva 5. Webhookin määrittely (FastSpring s.a.d.)

Tavallisesti webhook-tapahtuma sisältää vain kyseisen tapatuman tarvitseman tiedon esimerkiksi tilauksen yhteydessä vain tilaajan FastSpring tilin id-arvon. Webhook:n laajennuksen ollessa käytössä tapahtuman mukana lähetetään koko tili-olio, joka sisältää esimerkiksi käyttäjän nimen ja sähköpostin. Toteutuksesta riippuen webhook:n laajennus voi tehdä tarpeettomaksi valita jokin tietty webhook-tapahtuma, kuten esimerkiksi tilin luomisesta syntyvä tapahtuma, koska tieto saapuu jo toisen tapahtuman mukana. (FastSpring s.a.d.)

Kuvassa 6 on palvelin webhook konfiguraatio verkko-osoitteella ja valituilla tapahtumilla. Kyseiseen webhook:n on valittu tiedon laajennus ja se on toiminnassa live- ja testitilauksissa.



Kuva 6. Esimerkki palvelin-webhookista (FastSpring s.a.d.)

Jokainen tapahtuma sisältää yhteisiä kenttiä, mutta kenttien sisältö vaihtuu merkityn tyypin mukaisesti. Esimerkiksi subscription.activated-tyypin tapahtuma sisältää eri tietoa kuin subscription.deactivated-tyypin tapahtuma. Tyyppikohtainen tieto sijaitsee ”data”-kentän sisällä. Tunnistekentän lisäksi tapahtuma sisältää luontihetken, prosessoinnintilan ja tiedon onko tapahtuma aito- vai testitapaus (Kuva 7). (FastSpring s.a.d.)

```
{
    "id": "jazYJQw5RSWVR474tU2Obw",
    "live": true,
    "processed": false,
    "type": "subscription.activated"
    "created": 1426560444800,
    "data": {
        .... See webhook payload exampl
    }
}

{
    "id": "VOe5PQx-T4S6t8yS_ziYeA",
    "live": true,
    "processed": false,
    "type": "subscription.deactivated"
    "created": 1426560444900,
    "data": {
        .... See webhook payload exampl
    }
}
```

Kuva 7. Kaksi eri tyypin webhook-tapahtumaa (FastSpring s.a.d.)

```
{
  "id":"TBODFuwPRX2OiZj3u0MlQw",
  "account":"TBODFuwPRX2OiZj3u0MlQw",
  "contact":{
    "first":"John",
    "last":"Doe",
    "email":"ne1@all.com",
    "company":null,
    "phone":"1-805-409-9008"
  },
  "language":"en",
  "country":"US",
  "lookup":{
    "global":"lc3PZF_mSIiwV2ZUvcoZSQ",
    "custom" : "8675309"
    "url":"https://yourexamplestore.onfastspring.com/account"
  }
}
```

Kuva 8. Esimerkki tapahtumakohtaisesta tiedosta (FastSpring s.a.e.)

Kuvassa 8 on esimerkki tapahtumakohtaisesta tiedosta, joka voisi sijaita account.created-tapahtuman ”data”-kentässä. Se sisältää luodun FastSpring tilin id-arvon, rekisteröityneen käyttäjän tietoja ja parametreja tilin hakemiseen FastSpring rajapinnan kautta.

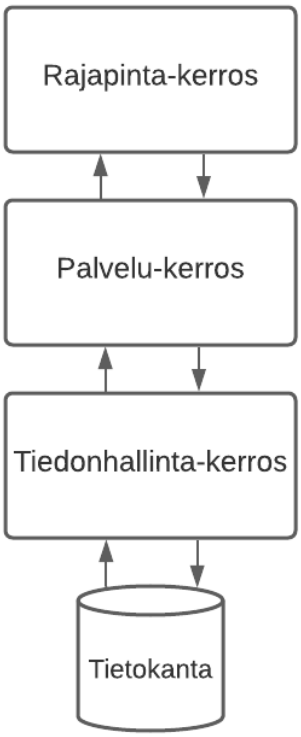
Webhook-tapausten käsittelyn jälkeen palautetaan HTTP-koodi 200 tai 202, joka merkitsee tapaukset prosessoiduksi riippuen annetusta koodista. 200 koodi merkitsee kaikki saapuneet tapaukset prosessoiduksi, kun taas 202 koodilla voi valita mitkä tapahtumat merkitään. Koodin 202 lisäksi annetaan prosessoiduksi merkittävän yksittäisen tapahtuman id-arvo aina uudelle riville. (FastSpring s.a.d.)

Prosessoiduksi merkitseminen on oleellista, koska FastSpring lähettää prosessoimattomat tapahtumat uudelleen 10 minuutin välein 24 tunnin ajan, kunnes tapahtuma merkitään prosessoiduksi. 24 tunnin jälkeen FastSpring ei enää lähetä tapahtumaa uudelleen, mutta prosessoimattomat tapahtumat voi kysyä manuaalisesti kutsumalla ”/events/unprocessed” päätepistettä FastSpring:n rajapinnasta. Manuaalisesti haetut tapahtumat suositellaan merkitsemään prosessoiduksi, jotta samaa tapahtumaa ei lähetetä uudestaan rajapintaa kutsumalla. (FastSpring s.a.d.)

Tapahtumaa ei luoda ja lähetetä, jos sitä ei ole lisätty webhook:iin mukaan. Tämän seurauksena sitä ei myöskään voi saada events-rajapinnan kautta edes silloin, kun tapahtuma lisättäisiin webhook:iin jälkikäteen. Esimerkiksi maanantaina tehtyä tilausta ei voi hakea events-rajapinnan kautta, vaikka order.completed-tapahtuma lisättäisiin tiistaina. Sen voi kuitenkin hakea orders-rajapinnan kautta, koska se ei ole riippuvainen webhook-tapahtumista. (FastSpring s.a.f.)

7 Integraatiokomponentti

Tässä luvussa käydään läpi komponentin rakenne ja keskeisimmät osat sovelluslogiikkaa sovelluksen toiminnan ymmärtämiseksi. Komponentin rakenne noudattaa yleistä Java-verkkosovelluksen rakennetta, jossa sovelluksen osat ovat jaettu rajapinta-, palvelu-, tiedonhallinta- ja tietokantakerroksiin (Kuva 9). Kyseisessä rakenteessa kerros on liitoksissa vain suoraan alla olevaan kerrokseen. Tämä esimerkiksi mahdollistaa muutoksien tekemisen tiedonhallintakerrokseen ilman, että rajapintakerros menee rikki. Komponentissa ei ole käyttöliittymää.



Kuva 9. Komponentin rakenne

7.1 Komponentin rakenne

Rajapintakerros sisältää kaikki ulospäin näkyvät päätepisteet. Rajapinnat on toteutettu Retrofit2-kirjaston avulla ja ne vastaanottavat ja lähettävät JSON-muotoista tietoa. Retrofit:n muuntaa vastaanotetun JSON-tiedon REST-olioksi, joka ohjataan palvelukerrokselle. Samankaltaisesti palvelukerrokselta saatu REST-olio muunnetaan JSON-muotoon ja lopulta palautetaan kutsujalle. REST-oliot ovat tavallisia Java-olioita, joita ei ole tarkoitus tallentaa tietokantaan.

Yleisesti palvelukerros tarjoaa tavan luoda, lukea, poistaa, päivittää ja listata haluttua kohdetta. Palvelukerros muuntaa REST-oliot tietokantaan tallennettaviksi entiteeteiksi ja kutsuu tiedonhallintakerrosta. Yleisesti REST-olioiden rakenne on sama kuin tietokantaentiteeteillä pois lukien kentät, joilla on vain järjestelmän sisäisen toiminnan kannalta merkitystä. Joissain tapauksissa sisään tuleva ja lähetettävä REST-olio poikkeaa toisistaan, kuten tapauksessa, joissa saapuvassa REST-oliossa on salasana-kenttä, mutta ulospäin lähetettävässä REST-oliossa ei, koska salasanaa ei haluta lähettää takaisin verkon yli.

Tietokantana projektissa on käytössä PostgreSQL-relaatiotietokanta, jota tiedonhallintakerros hallitsee Hibernate-kirjaston avulla. Jokaiselle entiteettiluokalle luodaan oma tietokantaan pääsyä hallitseva luokka. Turhan koodin toistamisen välttämiseksi näille luokille on tehty geneerinen kantaluokka, joka toteuttaa tavalliset tietokannan operaatiot eli luonti, kirjoitus, päivitys ja poisto. Entiteetin pääsynhallinta-luokkiin toteutetaan operaatioita, joita vain kyseinen entiteetti tarvitsee. Esimerkiksi listaus tietyn tila-arvon omaavista entiteeteistä.

Integraatiokomponentti on multi-tenant järjestelmä eli yhdellä sovelluspalvelimella ajettavalla sovellusinstanssilla käsitellään useampaa asiakasta. Jokaisella 10Duke asiakkaalle luodaan oma Tenant-entiteetti ja siihen liittyvä konfiguraatioentiteetti. Konfiguraatio-entiteetti sisältää asiakkaan integroimiseen liittyvää tietoa, kuten käytetty lisensointijärjestelmä, maksupalveluntarjoaja ja käytettävä tunnistustapa. Saapuvat event:t ja luodut CustomerMapping-oliot ovat kuitenkin tenant-kohtaisia ja niiden prosessointia kuvataan omassa osiossa.

7.2 Käyttäjän liitos

Yksi keskeinen osa integraatiokomponentin toimintaa on CustomerMapping-olio, joka toimii liitoksena järjestelmien välillä, koska käsitellyssä tilassa kyseinen olio sisältää rekisteröidyn käyttäjän käyttäjätilien id-arvot maksupalveluntarjoajan ja 10Duken tilinhallintajärjestelmässä. Näitä id-arvoja käytetään lisenssien hallinnassa kyseiselle käyttäjätilille. Mapping-olio sisältää lisensoitavan käyttäjän tyypin, joka määrittää onko tämä yritys- vai yksityistili. Olio sisältää kaksi tilaa kuvaavaa kenttää (status), jotka seuraavat käyttäjän tilin luonnin vaiheita maksupalveluntarjoajan ja 10Duken järjestelmissä. Tila-kenttiä käsittelemään, päivittämään ja tilin luomiseen on tehty oma CustomerMapping-prosessori.

Taulukossa 5 kuvaillaan mahdolliset tilat, joissa CustomerMapping-olio voi olla. Mapping operaation ollessa kesken tilana on pending. Onnistuneessa operaatiossa tila on complete. Jos käyttäjä haluaa keskeyttää provisioinnin ennen kuin tila on valmis, niin tilaksi

asetetaan canceled. Disabled tilaa käytetään, kun operaatio halutaan jättää huomioimatta esimerkiksi, asiakas haluaa vain yhdistyksen lisensointiin ilman FastSpring:ä.

Epäonnistumista kuvaa kolme tila-arvoa, jotka myös määrittelevät yritetäänkö prosessointia uudestaan. Failed, Partial Fail ja Terminal Fail. Failed-tilassa virhe on käynyt, mutta käsittelyä voi yrittää uudelleen. Partial Fail-tilassa operaatio on osittain epäonnistunut. Esimerkiksi organisaation luonti identiteettijärjestelmään vaatii myös käyttäjän luomisen, joka tehdään ennen organisaation luontia. Jos käyttäjän luonti onnistuu mutta organisaation luonti ei tilaksi asetetaan Partial Fail. Terminal Fail-tila asetetaan tapauksissa, joissa luontioperaatio ei voi missään tapauksessa onnistua, kuten pakollisen tiedon puuttuessa.

Taulukko 5. CustomerMapping-tilat

Status	Tarkoitus
Pending	Vaihe aloitettu ja/tai kesken. Esimerkiksi asiakkaan provisioinnissa status alustetaan pending-tilaan.
Complete	Operaatio valmis. Kyseinen liitos on suoritettu onnistuneesti.
Failed	Operaatio epäonnistunut jostain syystä. Esimerkiksi epäonnistunut HTTP-kutsu.
Partial Fail	Operaatio on osittain epäonnistunut.
Terminal Fail	Operaatio on epäonnistunut eikä sitä voi yrittää uudelleen.
Canceled	Operaatio on keskeytetty käyttäjän toimesta.
Disabled	Operaatio poistettu käytöstä. Mapping-prosessori jättää huomioimatta.

CustomerMapping-prosessori tekee ajastetusti CustomerMapping tietokantakyselyitä. Se valitsee prosessoitavaksi vanhimmat pending-tilassa olevat CustomerMapping-oliot pienen määrän kerrallaan. Tilin luontiprosessi käynnistetään, jos sitä vastaava tila on pending-tilassa. Esimerkiksi 10Duken tilinhallinnan tilan ollessa pending, käynnistetään rekisteröintiprosessi 10Duken järjestelmään. Jos operaatio epäonnistuu, tila asetetaan failed-tilaan.

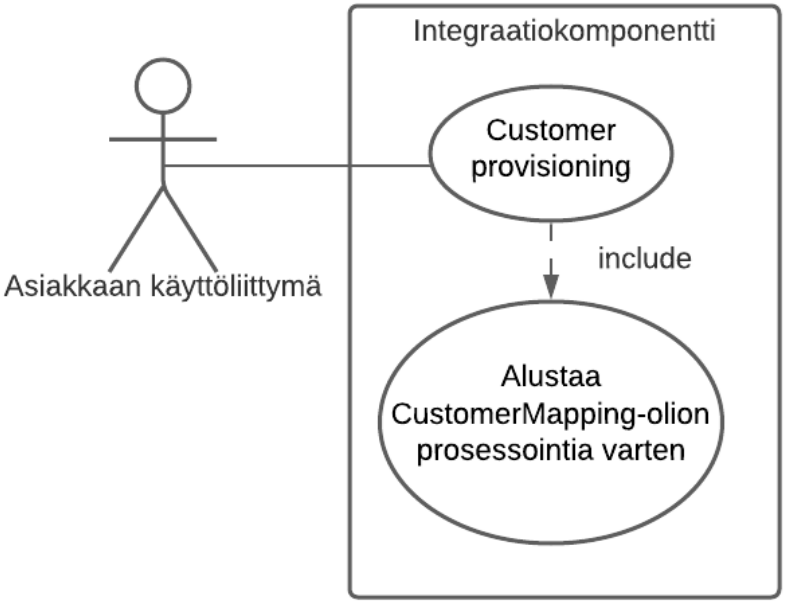
7.3 Asiakkaan provisiointi

Asiakkaan provisiointiin eli käyttäjätilien luomiseen tarjotaan kolme tapaa. Ensimmäinen tapa on kutsua CustomerProvisioning-rajapintaa asiakkaan oman käyttöliittymän kautta. Toinen tapa on tilin luonti FastSpring:ssä, jossa FastSpring tarjoaa tilin luonnin rajapinnan kautta tai automaattisesti tuotteen oston yhteydessä. Kolmas tapa on kutsua suoraan

CustomerMapping rajapinnan Create-toimintoa ja tätä käytetään tapauksessa, jossa asiakkaalla on jo tili FastSpring:n ja 10Duken järjestelmässä. Vaaditut tiedot käyttäjän luomiseksi ovat etu- ja sukunimi, sähköpostiosoite ja maa.

7.3.1 Provision rajapinta

Provision-rajapintaa (kuva 10) käytettäessä loppuasiakkaalla ei ole 10Duke- tai FastSpring-tiliä. Loppuasiakas syöttää tarvittavat tiedot 10Duken asiakkaan järjestelmään, joka on yhdistetty integraatiokomponenttiin. Asiakkaan järjestelmä kutsuu 10Duken CustomerProvisioning-rajapintaa loppuasiakkaan syöttämillä tiedoilla.



Kuva 10. Provision rajapinnan kutsuminen

Loppuasiakkaan rekisteröityessä suoraan provisioning-rajapinnan kautta integraatiokomponentti luo CustomerOnboardingData-olion ja tallentaa rajapinnan kautta saapuneen tiedon payload-nimiseen kenttään muuttamalla saapuneen JSON-datan Jackson ObjectMapper:a käyttäen String-muotoiseksi. String-muodossa tallentaminen mahdollistaa yksinkertaisen entiteetin luonnin tietokantaan, eikä kantaan tarvitse rakentaa monimutkaista oliorakennetta. Payload:iin tallennettua tietoa käytetään rekisteröintiprosessin edetessä lukemalla payload-kentästä tarvittavaa dataa.

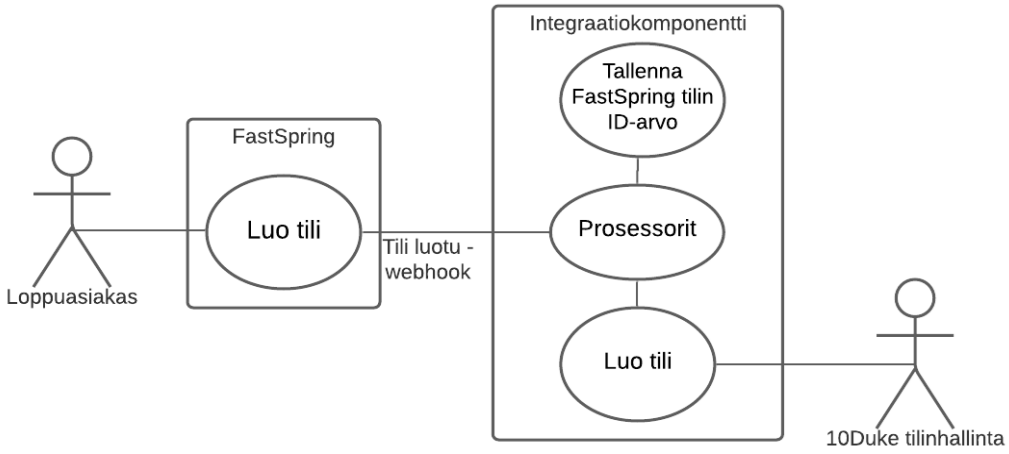
Payload-kentän sisältämä tieto poistetaan, kun sitä ei enää tarvita, koska GDPR-säädöksen artikla 17 mukaan yksilöllä on oikeus omien tietojensa poistamiseen, kun organisaatiolla ei ole sille enää tarvetta (asetus luonnollisten henkilöiden suojelusta henkilötietojen

käsittelyssä sekä näiden tietojen vapaasta liikkuvuudesta ja direktiivin 95/46/EY kumoamisesta 2016/679/EU).

CustomerProvisioning-vaiheessa luodaan myös CustomerMapping-olio, jolla on yksi-yhteen relaatio luodun CustomerOnboardingData-olion kanssa. Provisiointi vaiheessa CustomerMapping sisältää vain e-commerce järjestelmän ja 10Duken järjestelmän tila-arvot ja ne alustetaan pending-tilaan. Tilin luominen aloitetaan ja loput tiedot päivitetään CustomerMapping-prosessoinnin yhteydessä.

7.3.2 Käyttäjätilin luonti – FastSpring ja CustomerMapping-rajapinta

Loppuasiakas luo käyttäjätilin FastSpring:ssä (Kuva 11). Tilin luonnista syntyy tapahtuma, josta lähtee account created-webhook integraatiokomponenttiin.



Kuva 11. Käyttäjätilin luonti FastSpring webhook:n avulla

Integraatiokomponentti ottaa vastaan FastSpring:n lähettämät webhook:t, joka on tässä tapauksessa create account. Komponentti käsittelee saapuneen webhook:n sisällön ja aloittaa tilin luontiprosessin, josta kerrotaan lisää prosessoinnin kuvauksessa. Toinen mahdollinen tilin luomiseen liittyvä polku, jossa prosessori on mukana, on tilanne, jossa loppuasiakas ostaa tuotteen myyjän sivustolta ja ostajalla ei ole olemassa olevaa tiliä FastSpring:llä. FastSpring luo tilin oston yhteydessä ja siitä syntyy samanlainen account created-tapahtuma, kuin edellä.

Kolmas tapa alustaa käyttäjä lisensointia varten tapahtuu tilanteessa, jossa rekisteröitävällä käyttäjällä on jo olemassa olevat tilit FastSpring:ssä ja 10Duken tilinhallinnassa.

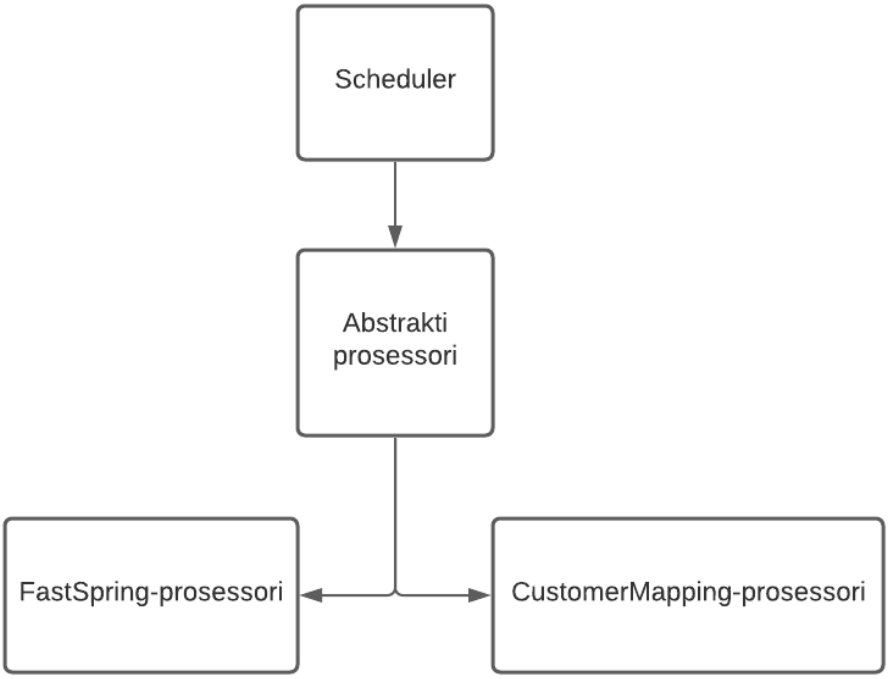
Tässä tapauksessa voidaan suoraan kutsua CustomerMapping-rajapinnan create-toimintoa, johon syötetään käyttäjän id-arvot, jotka löytyvät FastSpring:ltä ja 10Duken tilinhallinnasta. Tällä tavalla 10Dukelta löytyvä käyttäjä on yhdistetty vastaavaan FastSpring-tiliin.

7.3.3 Prosessoinnin kulku

Järjestelmän vastaanottaessa ulkopuolelta tulevia event:ä, sen toimimiseksi piti suunnitella toiminnallisuus kyseisten event:n käsittelemiseksi. Ratkaisuksi kehitettiin ajastetusti toimiva scheduler-luokka, abstrakti event-prosessori ja siitä perivät implementaatioluokat FastSpringEvent-olioiden ja CustomerMapping-olioiden prosessointiin. Kahteen varsinaiseen prosessoriin päädyttiin, koska tunnistettiin järjestelmässä olevan sisältä ja ulkoa tulevia tapahtumia ja prosessoinnin on oltava sujuvaa. Esimerkiksi lisenssin myöntäminen olemassa olevalle käyttäjälle ei saisi hidastua sen takia, koska prosessointijonossa on monta prosessoimatonta CustomerMapping-oliota.

Scheduler-luokka kutsuu ajastetusti FastSpringEvent-prosessoria sekä CustomerMapping-prosessoria. Kutsu kulkee AbstractEvent-prosessorin kautta, joka valitsee ennalta määrätyn kokoisen listan tenant:a kerrallaan (Kuva 12). Valinta suoritetaan seuranta-olion avulla, jolla on yksi-yhteen relaatio Tenant-olioon ja sillä seurataan tenant:lle kuuluvien eventien yleistä tilaa ja sitä päivitetään prosessoinnin tuloksen mukaisesti. Kyseisellä entiteetillä on boolean-arvo, joka ilmaisee, onko tenant:lla uusia event:ä ja lista järjestetään kyseisen boolean-arvon ja event:n saapumisajan mukaan vanhimmat event:t ensimmäisenä.

Prosessointi suunniteltiin siten, että missään vaiheessa prosessoitavan event:n tila ei jää välitilaan. Tämän toiminnallisuuden mahdollistaa riittävät tilaa kuvaavat kentät, jotka määrittelevät prosessoinnin logiikkaa. Yksittäisten event:n prosessointi tapahtuu Hibernate:n transaktion sisällä. Transaktion sisällä tietokanta asetetaan takaisin siihen tilaan, jossa se oli event:n prosessoinnin alkaessa esimerkiksi tietokannan yhteysvirheen sattuessa (Ottinger, Linwood & Minter 2022, luku 8). Mahdollisia virhetilanteita ja pullonkauloja seurataan lokitiedon avulla. Lokitietoa kerätään, jokaisen webhook-event:n saapuessa, sekä prosessoinnin aikana.



Kuva 12. Ylätason kuvaus prosessoinnin kulusta

7.3.4 FastSpring-eventien vastaanotto ja prosessointi

FastSpring:n lähettämien webhook event:n vastaanottamista varten rakennettiin REST-rajapinta ja -palvelu. FastSpring:n webhook konfiguroinnissa voi määrittää mihin osoitteeseen webhook:t lähetetään. Komponentin rajapintakerroksen osoitepolku lisätään kyseiseen kenttään. Webhook-event:ä varten tehtiin FastSpringEvent-olio, joka sisältää luonti- ja tilatiedot, sekä kentän johon webhook:n sisältö tallennetaan. Ennen tallennusta REST-palvelu tarkastaa saapuneen lähetyksen varmistaen sen olevan JSON-muodossa. FastSpring mahdollistaa useamman event:n sisältyvän samaan webhook:iin. Jos niin on tapahtunut, vastaanottava palvelu käy lähetyksen läpi ja luo jokaiselle event:lle oman FastSpringEvent-olion. Samalla kun luodut FastSpringEvent-oliot tallennetaan tietokantaan odottamaan käsittelyä, päivitetään seuranta-olio ilmoittamaan prosessorille, että uusia event:ä on saapunut.

FastSpring-event:n prosessointiin tehtiin FastSpringEvent-prosessori. FastSpringEvent-oliot sisältävät prosessoinnin tilaa kuvaavan kentän. Prosessointia odottavat event:t ovat Pending-tilassa ja onnistuneesti valmistuessaan muuttuvat Processed-tilaan. Lisäksi vaihtoehtoina on Ignored- ja Skipped-tilat. Prosessori hakee ennalta määrätyn kokoisen listan rajatun ikäisiä Pending- ja Skipped-tilassa olevia event:ä, jotta muiden tenant:n event:t eivät jää prosessoimatta, jos yhdellä tenant:lla on erittäin suuri määrä prosessoimatta olevia event:ä. Prosessoitavien event:n ikä on rajattu, jotta uudet event:t pääsevät prosessointiin

eikä jono tukkiudu samoista loputtomasti epäonnistuvista event:stä. Skipped- ja Ignored-tiloihin asettamista kuvataan seuraavaksi.

Pääasiallinen ero Skipped- ja Ignored-tilojen välillä on se, että Skipped-tilassa olevia event:ä voidaan yrittää uudelleen. Esimerkiksi event asetetaan Skipped-tilaan siinä tapauksessa, jos prosessoitavana oleva event käsittää jonkin tuotteen tilauksen, mutta tilauksen tekevän asiakkaan tiliä ei ole vielä luotu 10Duken järjestelmään. Tällainen tilanne on hyvin mahdollista, koska FastSpring:n lähettämässä webhook:ssa voi olla monta eri event:ä satunnaisessa järjestyksessä.

Event asetetaan Ignored-tilaan pääasiallisesti kahdessa tapauksessa. Ensimmäisessä tapauksessa prosessoinnin aikaisissa tarkistuksissa tai itse prosessoinnissa ilmenee virhe, joka estää prosessoinnin onnistumisen myös jatkossa. Esimerkiksi prosessointiin tarvittavan tiedon puuttuessa. Toinen tapaus liittyy FastSpring:n toimintaan, jossa tilauksen ostamisesta lähetetään order completed- ja subscription activated-webhook event:t. Prosessointia ei haluta suorittaa kahdesti, joten prosessointi suoritetaan vain subscription activated-eventille ja order completed-event asetetaan Ignored-tilaan. Tämä tilanne koskee vain tapauksia, jossa tilaus sisältää pelkästään subscription-eventejä. Edellä mainittua käytäntöä ei sovelleta tilanteisiin, jossa order completed-eventissä on myös kertaostoksena maksettavia tuotteita.

7.3.5 FastSpring-event:n käsittely pääpiirteittäin

Pääasiassa FastSpring-event:t lukeutuvat kahteen kategoriaan tilaus- ja käyttäjätili-event:t. Tilauksiin liittyvissä tapauksissa FastSpringEvent-olion String-muotoinen payload-kenttä muunnetaan Map-rakenteeksi ja sen sisältämistä tiedoista rakennetaan olio, joka sisältää tarvittavat tiedot lisenssin myöntämiseksi. Rakennettu olio annetaan lisenssikutsun rakentavalle ja lähettävälle palvelulle.

FastSpring:n mahdollistamat käyttäjätiliin liittyvät event:t ovat account created ja account updated. Näitä event:ä varten luotiin oma AccountEventHandler-luokka, jonka tarkoituksena account created-tyypin event:ssä on luoda FastSpringEvent-olion sisältämän tiedon perusteella CustomerOnboardingInfo-olio ja alustaa CustomerMapping-olion FastSpring tilin id-arvo ja merkata FastSpring-tila valmiiksi ja 10Duke-tila odottamaan prosessointia. Account updated-event:n käsittelyä ei toteuteta opinnäytetyön laajuudessa.

7.3.6 CustomerMapping prosessointi

CustomerMapping-olioiden prosessointiin rakennettiin oma prosessori, jonka tarkoituksena on luoda käyttäjätili FastSpring:iin ja 10Duken tilinhallintaan riippuen CustomerMapping-olion tilakentistä. Kuten FastSpringEvent-prosessori myös CustomerMapping-prosessori hakee vain tietyn kokoisen listan prosessoitavia olioita, jotta suoritusaika pysyy kohtuullisena ja prosessointi voi jatkua muillekin tenant:lle.

CustomerMapping-prosessori käyttää käyttäjätilin luomiseen CustomerProvisioning-vaiheessa tai FastSpring:n account created-event:n seurauksena luotuja CustomerOnboardingInfo- ja CustomerMapping-olioita. Asiakas vaatimuksen mukaisesti tilin luominen 10Duken tilinhallintaan tuli saada valmiiksi ensimmäisenä. Tässä tapauksessa CustomerOnboardingInfo-olion kenttien avulla rakennetaan tilin luomiseen vaadittavat käyttäjä- ja organisaatio-oliot, jotka annetaan tilinhallintaa kutsuvalle palvelulle, josta puhutaan luvussa ”Tilinhallintaan ja lisensointiin liittäminen”.

FastSpring:iin tilin luomista varten tulee toteuttaa oma palvelu, jolle rakennetaan ja annetaan tarvittavat kentät sisältävä olio, joka lähettää HTTP POST-pyynnön FastSpring:n CreateAccount-rajapintaan. Kyseistä rajapintaa kutsuttaessa tarvitaan käyttäjän etu- ja sukunimi, sähköposti ja maa, jotka löytyvät CustomerOnboardingInfo-oliolta, joka käsittelyssä olevaan CustomerMapping-olioon on liitetty. Kutsun onnistuessa, saadaan vastauksena luodun FastSpring-tilin id-arvo, joka tallennetaan CustomerMapping-olioon ja FastSpring-tila merkataan valmiiksi. Tästä FastSpring-tilin luonnista syntyy account created-tapahtuma, joka saapuu FastSpringEvent-prosessorin käsiteltäväksi. Jos tili löytyy järjestelmästä niin kuin sen pitäisi, tapahtuma jätetään huomioimatta.

7.4 Tilinhallintaan ja lisensointiin liittäminen

10Duken tilinhallintaan yhdistäminen tapahtuu REST-rajapintojen kautta. Tarpeena siihen yhdistämiseen on luoda uusi käyttäjä- tai organisaatioprofiili, joille käyttäjän ostaman tuotteen lisenssejä lopulta myönnetään. Avuksi REST-kutsujen rakentamiseen käytetään Retrofit2 Java-kirjastoa. Retrofit pystyy muuntamaan tavallisia Java-luokkia JSON-muotoon, joten tämä huomioiden luotiin kaksi Java-luokkaa, jotka sisältävät 10Duken tilinhallinnan rajapinnan määrittelemät kentät käyttäjäprofiilin luomiseksi. Itse kutsun tekevä luokka on yksinkertainen. Se ottaa argumenttina tarvittavan tiedon sisältävän Java-olion, rakentaa auktorisointi HTTP-otsikon, alustaa Retrofit-instanssin, lähettää kutsun ja käsittelee palautuvan vastauksen.

Tilin luovaa luokkaa on testattu manuaalisesti määrittelemällä Java-olio, joka sisältää etunimen, sukunimen ja sähköpostin. Luotu testiolio annetaan testi-instanssille ja saadun

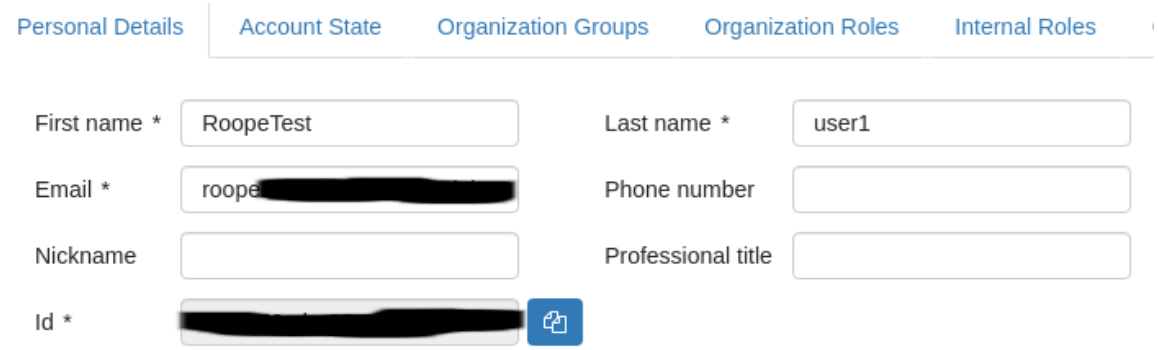
vastauksen tila tarkistetaan. Pseudokoodina kirjoitetut ”Tulos”-luokka ja ”käyttäjäOlio”-instanssi antavat virheen (Kuva 13).

```
// WHEN
Tulos tulos = instance.setupUser(käyttäjäOlio);

// THEN
Assert.assertTrue(tulos.isSuccess());
```

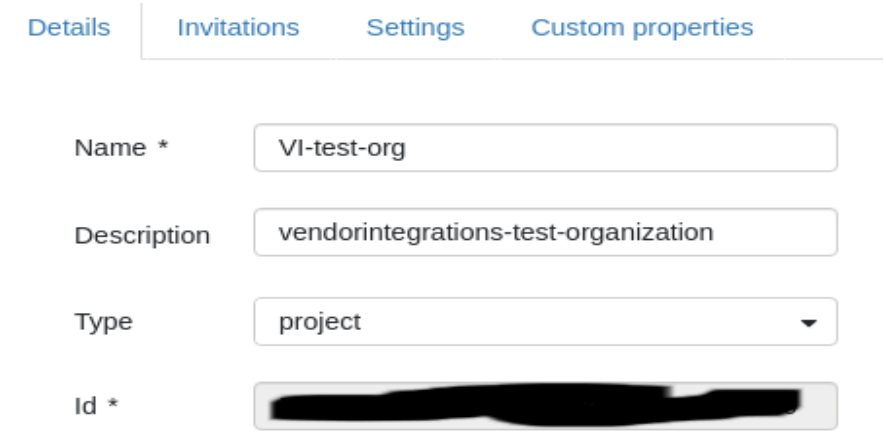
Kuva 13. Testikutsu käyttäjän luomiseksi

Kuvassa 14 verkossa olevaan käyttäjänhallinnan testiympäristöön syntynyt käyttäjä edellä olevan testin seurauksena. Kuvasta piilotettu Id-arvo ja sähköposti.



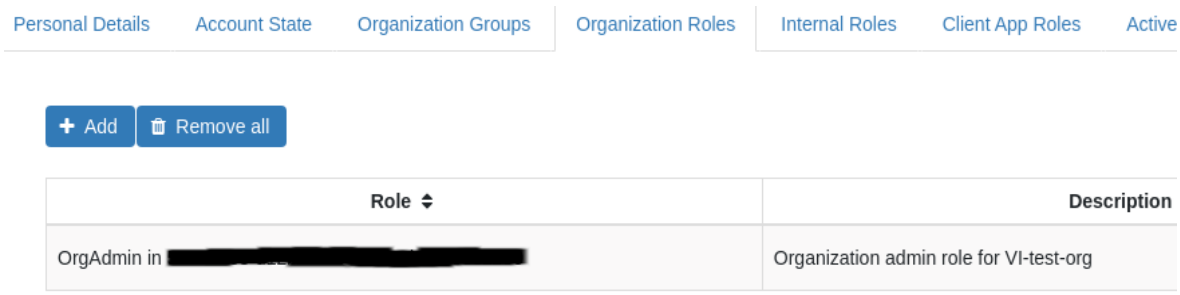
Kuva 14. Testiympäristöön syntynyt käyttäjä

Kuvassa 15 testiympäristöön on luotu organisaatio samalla tavalla kuin käyttäjä edellisessä tapauksessa. Id arvo piilotettuna. Organisaation luonnin yhteydessä voidaan määrittää henkilö, joka toimii ensimmäisenä organisaation hallitsijana (admin).



Kuva 15. Testiympäristöön luotu organisaatio

Edellisessä testitapauksessa luotu käyttäjä on liitetty organisaation luontikutsuun ja organisaatio on asetettu luodun käyttäjän organisaatiorooleihin (Kuva 16).



Kuva 16. Testikäyttäjälle ilmestynyt admin-rooli luotuun testiorganisaatioon

Jotta rekisteröity asiakas pääsee kirjautumaan luodulle tilille, komponentti kutsuu automaattisesti 10Duken tilinhallinnan rajapintaa salasanan palauttamiseen käyttäjän luomisen jälkeen. Salasanan palauttamiskutsu tarvitsee vain sähköpostin, jonka käyttäjä syöttää rekisteröintivaiheessa. Salasanan palauttamisoperaatio lähettää annettuun sähköpostiin salasanan palautuslinkin.

10Duken lisensointijärjestelmään liittämistä ei käydä läpi yksityiskohtaisesti teknisellä tasolla. Yksinkertaisesti lisensointijärjestelmää varten tehtiin oma luokka, kuten tilinhallintaan yhdistettäessä ja kyseiselle luokalle annetaan FastSpringEvent-prosessoinnissa luotu olio, joka sisältää lisensointiin tarvittavat tiedot. Liitoksen tekemisessä oleellista oli jakaa lisensointikutsun rakentaminen lisensoitavan käyttäjän tyypin mukaan, joka on joko yksityis- tai organisaatiotyyppiä. Lisenssinhallinta yksityis- ja organisaatiotileillä poikkeaa toisistaan, joten edellä mainittu jako on tarpeellinen. Lisenssinhallintaan perehtyminen ei kuitenkaan kuulu tämän projektin laajuuteen, eikä sillä ole komponentille vaikutusta muuten kuin edellä mainitussa tapauksessa.

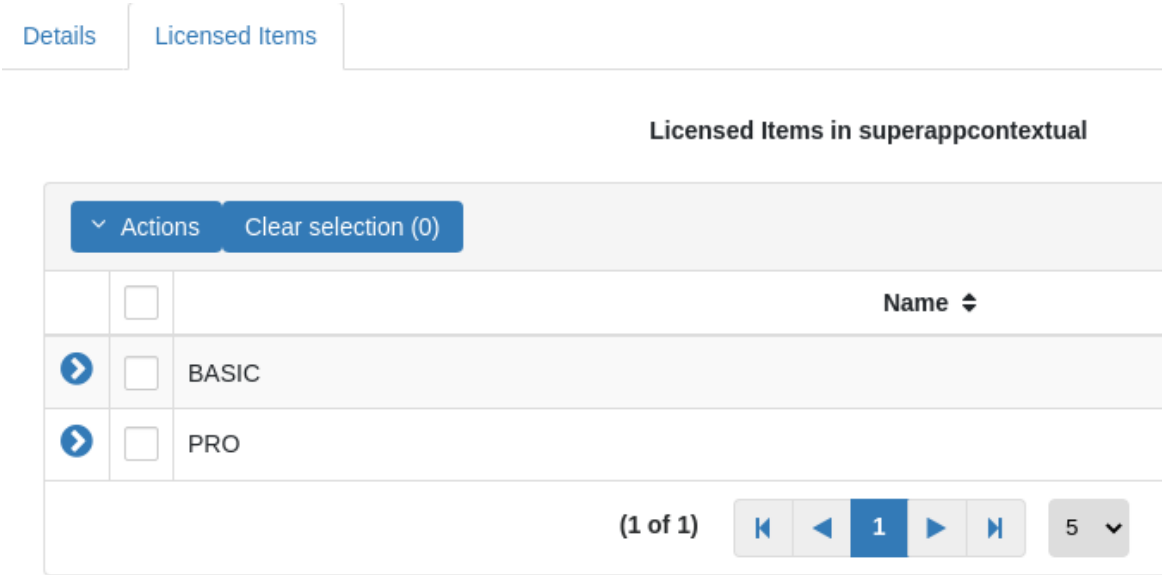
7.5 Testaus

Komponentin toimintojen testaus on suoritettu yksikkö- ja integraatiotesteillä. Yksikkötesteillä testataan kerrallaan yhden luokan metodin toimivuus onnistuneessa tapauksessa ja virhetilanteissa. Esimerkiksi metodi, joka rakentaa OnboardingInfo-oliosta 10Duken tilinhallintaan lähetettävän tiedon. Tämän rakentamisen onnistuminen, sekä virhetilanteessa halutulla tavalla epäonnistuminen varmistetaan syötettyä testitietoa muokkaamalla.

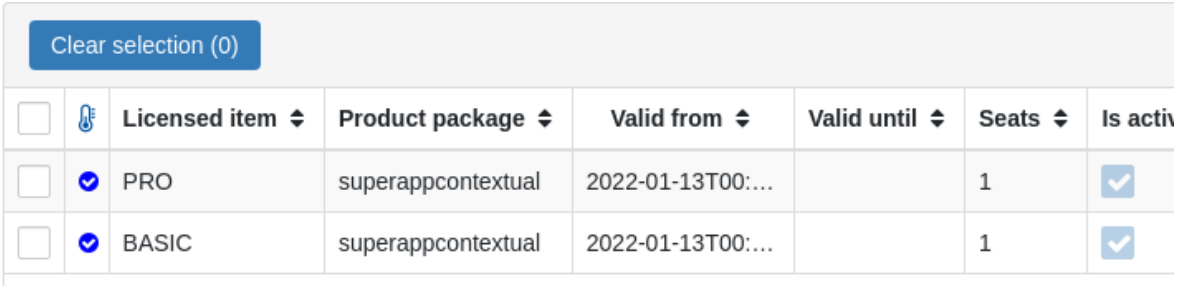
Integraatiotesteissä on käytössä testContainers-kirjasto ja Quarkus:n testi ja testiprofiili toiminnallisuudet. QuarkusTest-annotaatio merkitsee sitä, että koko Quarkus-ympäristö käynnistetään testiä varten, jotta saadaan sovelluksen aito toiminnallisuus käyttöön toimivine REST-päätepisteineen. Testiprofiililla voi määrittää millainen testiympäristö käynnistetään. Näiden avulla testikäyttöön saadaan toimiva tietokantaympäristö, jotta tietokantaa

tarvitsevat operaatiot saadaan testattua. Omassa testiympäristössään operaatioita voidaan testata turvallisesti ilman vaaraa tuotannossa olevaan tietokantaympäristöön. Esimerkiksi FastSpringEvent-prosessorin prosessointimetodin testaus account created webhook-eventin saapuessa. Testissä prosessori-instanssille annetaan itserakennettu account created-tyypin FastSpringEvent, jonka lopputuloksena odotetaan, että annetun FastSpringEvent:n tila on muuttunut Pending:stä Processed:ksi ja tietokantaan on luotu CustomerMapping-olio Pending-tilassa.

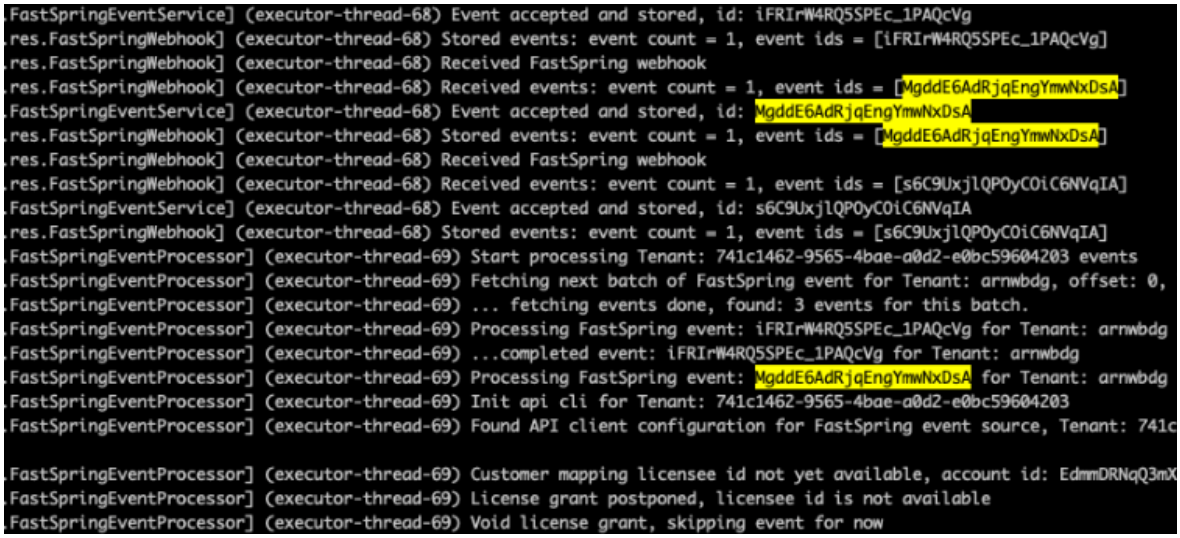
Komponentin vähittäisvaatimusten määrittelemä toiminnallisuus testattiin end-to-end-testillä. Viimeisin komponentin kehitysversio asennettiin 10Duken AWS:n pilviympäristöön. 10Duke:lle luotiin testikauppasivu FastSpring:n järjestelmään, johon komponentin webhook:a vastaanottavan rajapinnan osoite liitettiin. Lisensointijärjestelmään luotiin testiä varten tuotepaketti, johon testikäyttäjälle myönnetään lisenssi (Kuva 17). FastSpring:n testikauppasivulla valittiin ostettava tuote, syötettiin testiostajan tiedot ja painettiin osta-näppäintä. Lopputuloksena tilinhallintaan syntyi käyttäjä testiostajan tiedoilla ja kyseiselle käyttäjälle lisensoiduista tuotteista löytyi testiä varten luotu tuotepaketti (Kuva 18). Testin aikana syntynyttä loki tietoa, jossa vastaanotetaan webhook:ja ja aloitetaan prosessointia (Kuva 19).



Kuva 17. Testiä varten luotu tuotepaketti



Kuva 18. Käyttäjälle ilmestynyt tuotepaketti



Kuva 19. Lokitietoa end-to-end-testistä

8 Pohdinta

Tässä luvussa tarkastelen, täyttikö työ sille asetetut tavoitteet ja käyn läpi ajatuksia jatkokehityksen kannalta. Lisäksi pohdin opinnäytetyöprosessia ja omaa oppimistani työn aikana.

8.1 Tavoitteet

Integraatiokomponentti saavutti sille asetetut tavoitteet. Se käsitteli onnistuneesti sille määritellyt FastSpring:n lähettämät webhook-event:t ja loi niiden pohjalta käyttäjän 10Duken tilinhallintaan ja aktivoi luodulle käyttäjälle lisenssin. Toiminnallisuus vahvistettiin end-to-end-testin avulla. Valitettavasti aika ei riittänyt täydentävien toimintojen lisäämiseen, mutta projektin tavoite komponentin kannalta täytettiin. Asiakkaan toiveiden mukaisesti keskittyminen oli FastSpring:n kautta tapahtuvan integroinnin kehittämisessä, joten komponentti ei pystynyt luomaan käyttäjätiliä FastSpring:n järjestelmään.
Jatkokehitys jatkuu puuttuvien webhook toimien lisäämisellä, jotta asiakas saa integraatiokomponentin omaan testikäyttöönsä. Webhook toiminnallisuuden lisäämisen jälkeen toteutetaan palvelu FastSpring-tilin luomista varten. Valmistuneen toiminnallisuuden ansiosta lopun toiminnallisuuden lisääminen on yksinkertaista, koska prosessointirakenne on toteutettu ja testattu.

End-to-end-testauksen aikana esiintyi lokituksessa paranneltavaa. Komponentti kirjoitti turhaan lokia tapauksissa, joissa ei oikeasti tapahtunut mitään. Tämä paljasti, sen että komponentti yritti turhan tiheään tahtiin prosessoida skipped-tilassa olevaa event:ä. Prosessointilogiikan optimointi on myös jatkokehityksen aiheena. Täydet puutteet komponentin toiminnassa paljastuvat, sen päästessä asiakkaan testiympäristöön, pyyntöjen määrän lisääntyessä. Samaten komponentin luoma arvo selviää vasta tuotantoympäristössä.

Palveluntarjoajien listaus jäi ajanpuutteen takia valitettavan lyhyeksi ja melko suppeaksi. Mielestäni kovinkaan suuren listauksen tekeminen ei olisi ollut muutenkaan tarpeellista työn laajuuden puitteissa, vaikka aika olisi siihen riittänytkin. Koen että itse listaukseen kerättyjen tietojen etsiminen oli hyödyllistä, koska sen aikana tuli luettua kyseisien järjestelmien dokumentaatiota kohtuullisesti. Tätä lukukokemusta voi käyttää hyödyksi jatkossa integraatiokomponentin laajentuessa. Koen myös, että palveluntarjoajien järjestelmien vertailusta saisi oikein hyvän tutkimuksen tehtyä tulevaisuudessa.

8.2 Opinnäytetyö prosessi

Vaikein asia ja suurin stressin lähde koko opinnäytetyön aikana oli keksiä aiheeseen sopiva tietoperusta, joka sitoutuisi hyvin yhteen integraatiokomponentin kanssa. Vaatimukset työssä käytettyihin teknologioihin tuli toimeksiantajalta, joten valitsin niistä olennaisimmat, jotka olivat myös itselleni vieraat ja pyrin kirjoittamaan niiden konsepteista. Alkuperäisen suunnitelman mukaan olisin kertonut teknologioiden konseptin lisäksi, myös siitä miten ne saataisiin käyttöön, mutta se karsiutui pois ollessaan tekninen yksityiskohta, eikä se ollut komponentin toiminnan kannalta olennaista.

E-Commerce kappale kärsi kaikista eniten työhön sitoutumisen vähyydestä. Minusta oli vaikea keksiä integraatiokomponenttiin täysin liittyvää teoriaa E-commerce:n tai maksupalveluntarjoajaan liittyen. Päädyin kirjoittamaan asioita, joita itse oisin halunnut lukea ja koin edes jonkin verran oleelliseksi työn kannalta. Maksupalveluntarjoajat kuitenkin liittyvät komponenttiin ja halusin kertoa hieman niiden ympäristöön liittyvistä vaihtoehdoista. Koen, että PCI DSS:ään perehtymisestä voi olla hyötyä jatkossa omalla urallani. Pienimuotoinen palveluntarjoajien vertailu auttoi mielestäni sitomaan E-Commerce lukua hieman paremmin työhön, vaikka se jäikin suppeaksi.

Pieni selvitys maksuliikenteeseen ja sen alan toimijoihin oli mielenkiintoista, mutta laajuuden puolesta jäi pintaraapaisuksi monimutkaisesta ympäristöstä. Se oli myös vaikea sitoa työhön, koska komponentin tehtävänä ei ollut toteuttaa omaa maksujenprosessointia. Yhtenä jatkotutkimusajatuksena olisi selvittää, mitä pitää ottaa huomioon itse toteutetun maksusovelluksen tekemisessä.

Yleisesti työn kannalta oli vaikeaa miettiä, mikä sisältö on olennaista. Ensimmäisten päivien aikana ajattelin komponentin suhteen liian yksityiskohtaista kuvailua, joka ulottui luokkatasolle asti. Yllätyin kuitenkin positiivisesti, kuinka hyvin komponentin toiminnallisuuden kuvaaminen ylätasolla riitti. Olennaisen sisällön löytäminen tietoperustaan oli myös hankalaa. Alussa olin suunnitellut, että tietoperusta olisi valmis muutaman viikon jälkeen aloituksesta. Kävi kuitenkin niin päin, että komponenttiluku oli jo suurimmaksi osaksi valmis ennen, kaiken sisällön löytämistä tietoperustaan. Komponentista kirjoittaessani keksin asioita, joita voisi käydä läpi tietoperustassa.

Komponentin ja muun työn edistyminen oli aikataulullisesti hyvin vaikea arvioida ja merkittyjä aikarajoja tuli ylitettyä useasti. Arvioin tämän johtuneen integraatiokomponentin teknisen monimutkaisuuden takia, verrattuna omaan osaamistasooni. Monesti arvioin, jonkun osan tekemiseen vaadittavan keston olevan enintään muutaman päivän. Usein todellinen kesto kasvoi viikoksi tai kahdeksi. Välillä keston kasvaminen johtui oman ymmärryksen

puutteesta ja välillä havaittiin lisävaatimuksia tai jonkun osan muokkaamistarve halutun toiminnallisuuden saavuttamiseksi. Lisäksi epävarmuus teoriaosuuteen kirjoitetusta sisällöstä vaikeutti aikataulussa pysymistä.

8.3 Oman oppimisen tarkastelu

Oman oppimisen kannalta olen hyvin tyytyväinen. Ennen opinnäytetyön aloittamista olin työpaikallani toteuttanut jonkinlaista koodin refaktorointia ja laatinut yksikkö- ja integraatiotestejä, joihin liittyi tietokannan hallintaa. Minulta puuttui kuitenkin ymmärrys kokonaiskuvasta ja miksi joku asia oli toteutettu juuri niin kuin se oli. Integraatiokomponentin rakentuessa sopivan kokoisissa osissa, viimein havaitsin ja ymmärsin yhteydet eri osien välillä ja miten ne liittyvät toisiinsa, joko suorasti tai epäsuorasti.

Oma osaamiseni tietokannan hallinnasta kehittyi, sekä opin muodostamaan ja hyödyntämään relaatioita eri entiteettien välillä. Lisäksi toteutin ensimmäistä kertaa REST-rajapintoja, joiden avulla tietokantaa pystyi hallitsemaan, joka täytti tavoitteen verkkosovellus-kokonaisuuden rakentamisesta. Kehityin myös testaamisen kannalta luodessani yksikkö- ja integraatiotestejä komponentin osille.

Integraatiokomponentin rakentaminen auttoi minua ymmärtämään paremmin, mitä vastaavan kokonaisuuden tekemisessä pitää ottaa huomioon. Lisäksi opin ymmärtämään 10Duken järjestelmän toimintaa paremmin, vaikkakin täydemmän ymmärryksen saavuttaminen vaatii vielä paljon työtä ja opettelua. Myös sovelluskehityksen näkökulmasta. Komponentin tekeminen paljasti, kuinka paljon asioita en vielä ymmärrä sovelluskehitykseen liittyen. Jatkuva oppiminen on kuitenkin tämän alan hienouksia, jonka otan vastaan mielelläni.

Lähteet

BBCIncorp 2021. Merchant Account vs Payment Service Provider vs Payment Gateway. Luettavissa: https://bbcincorp.com/resources/merchant-account-psp-payment-gateway. Luettu: 2.12.2021.

Chargebee s.a.a. Frequently Asked Questions. Luettavissa: https://www.chargebee.com/faq/. Luettu: 19.1.2022.

Chargebee s.a.b. Payment Gateways. Luettavissa: https://www.chargebee.com/docs/2.0/gateway_settings.html. Luettu: 19.1.2022.

Chargebee s.a.c. Cards. Luettavissa: https://www.chargebee.com/docs/2.0/cards.html. Luettu: 19.1.2022.

Chargebee s.a.d. Managing Customers. Luettavissa: https://www.chargebee.com/docs/2.0/customers.html. Luettu: 19.1.2022.

Chargebee s.a.e. Events and Webhooks. Luettavissa: https://www.chargebee.com/docs/2.0/events_and_webhooks.html. Luettu: 19.1.2022.

Chargebee s.a.f. Webhook Settings. Luettavissa: https://www.chargebee.com/docs/2.0/webhook_settings.html. Luettu: 19.1.2022.

Codepath 2020. Using OkHttp. Luettavissa: https://guides.codepath.com/android/Using-OkHttp. Luettu: 17.11.2021.

Codepath 2021. Consuming APIs with Retrofit. Luettavissa: https://guides.codepath.com/android/Consuming-APIs-with-Retrofit. Luettu: 17.11.2021.

Dwyer, B. 2020. The Complete FastSpring Review. Luettavissa: https://www.cardfellow.com/blog/fastspring-review/. Luettu: 26.11.2021.

Ecommerce Platforms s.a. What is a Payment Service Provider or PSP? Luettavissa: https://ecommerce-platforms.com/glossary/payment-service-provider. Luettu: 7.12.2021.

Euroopan parlamentin ja neuvoston asetus luonnollisten henkilöiden suojelusta henkilötietojen käsittelyssä sekä näiden tietojen vapaasta liikkuvuudesta ja direktiivin 95/46/EY kumoamisesta (yleinen tietosuoja-asetus) (EU) 2016/679. Annettu 27.4.2016.

FastSpring s.a.a. Introducing FastSpring’s Full-Service Ecommerce Platform. Luettavissa: https://fastspring.com/product-overview/. Luettu: 22.11.2021.

FastSpring s.a.b. FastSpring API. Luettavissa: https://fastspring.com/docs/fastspring-api. Luettu: 22.11.2021.

FastSpring s.a.c. Integrations. Luettavissa: https://fastspring.com/docs/integrations#api. Luettu: 23.11.2021.

FastSpring s.a.d. Webhooks Overview. Luettavissa: https://fastspring.com/docs/webhooks. Luettu: 24.11.2021.

FastSpring s.a.e. account.created. Luettavissa: https://fastspring.com/docs/account-created/#example. Luettu: 24.11.2021.

FastSpring s.a.f. /events. Luettavissa: https://fastspring.com/docs/events/. Luettu: 26.11.2021.

FastSpring s.a.g. Customize Your Web Storefront. Luettavissa: https://fastspring.com/docs/customize-your-web-storefront/. Luettu: 26.11.2021.

Hibernate s.a. What is Object/Relational Mapping? Luettavissa: https://hibernate.org/orm/what-is-an-orm/. Luettu: 11.1.2022.

Mai, T. 8.6.2017. PCI DSS COMPLIANCE WITH PAYMENT GATEWAY: GUIDE FOR ECOMMERCE BUSINESSES. Luettavissa: https://magenest.com/en/pci-dss-compliance-payment-gateway/. Luettu: 19.1.2022.

Mihalcea, V. s.a. Why and when you should use JPA. Luettavissa: https://vladmihalcea.com/why-and-when-use-jpa/. Luettu: 10.1.2022.

Mourya, S.K. & Gupta, S. 2014. E-Commerce. Alpha Science International. E-kirja. Luettu: 14.1.2022.

OroCommerce 2021. Payment Gateways and Processing & Their Role in eCommerce. Luettavissa: https://oroinc.com/b2b-ecommerce/blog/gateway-payment/. Luettu: 26.10.2021.

Ottinger, J.B., Linwood, J. & Minter, D. 2022. Beginning Hibernate 6. Apress. E-kirja. Luettu: 11.1.2022.

PCI Security Standards Council s.a. About Us. Luettavissa: https://www.pcisecuritystandards.org/about_us/ Luettu: 7.12.2021.

PCI Security Standards Council 2018. Payment Card Industry (PCI) Data Security Standard Requirements and Security Assessment Procedures. Luettavissa: https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf. Luettu: 7.12.2021.

Quarkus s.a. Container First. Luettavissa: https://quarkus.io/container-first/. Luettu: 11.1.2022.

Redhat 2020a. Containers vs Vms. Luettavissa: https://www.redhat.com/en/topics/containers/containers-vs-vms. Luettu: 10.1.2022.

Redhat 2020b. What is Quarkus? Luettavissa: https://www.redhat.com/en/topics/cloud-native-apps/what-is-quarkus. Luettu: 10.1.2022.

Salesforce s.a. CRM 101: What is CRM? https://www.salesforce.com/crm/what-is-crm/ Luettu: 18.1.2022.

Shopify s.a. Ecommerce. https://www.shopify.com/encyclopedia/what-is-ecommerce. Luettu: 8.12.2021.

Stripe s.a.a. Learn about payment methods. Luettavissa: https://stripe.com/docs/payments/payment-methods/overview. Luettu: 19.1.2022.

Stripe s.a.b. Customers. Luettavissa: https://stripe.com/docs/billing/customer. Luettu: 19.1.2022.

Stripe s.a.c. Use incoming webhooks to get real-time updates. Luettavissa: https://stripe.com/docs/webhooks. Luettu: 19.1.2022.

Stripe s.a.d. Types of events. Luettavissa: https://stripe.com/docs/api/events/types. Luettu: 19.1.2022.

Stripe s.a.e. Test your integration. Luettavissa: https://stripe.com/docs/testing. Luettu: 19.1.2022.

Stripe Support s.a. Merchant accounts, gateways, and Stripe. Luettavissa: https://support.stripe.com/questions/merchant-accounts-gateways-and-stripe. Luettu: 19.1.2022.

Wróbel-Konior, S. 2017. Do I Need To Be PCI Compliant? Luettavissa: https://securionpay.com/blog/do-i-need-to-be-pci-compliant/. Luettu: 8.12.2021.