Haaga-Helia
ammattikorkeakoulu Oy

Fatman Frame ostotilaus-moduulin tuotteistaminen

Mikko Mattila

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2018

Tiivistelmä

Haaga-Helia
ammattikorkeakoulu Oy

Tekijä(t)
Mikko Mattila

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Fatman Frame ostotilaus-moduulin tuotteistaminen

Sivu- ja liitesivumäärä
26

Toiminnallisen opinnäytetyön tarkoituksena on tuotteistaa Fatman Oy:n Frame-ohjelmiston ostotilaus-moduuli mahdollisimman geneeriseksi. Kyseinen ohjelmisto on toteutettu käyttäen .NET MVC viitekehystä. Muihin projektissa käytettyihin teknologioihin ja tekniikoihin lukeutuu muun muassa C#, JavaScript, jQuery, SQL ja Scrum.

Projektin aloitushetkellä moduuli on hyvin pitkälti räätälöity tietyn yksittäisen asiakkaan käyttöön, jonka myötä se ei sovellu lainkaan muille asiakkaille. Moduuli on vahvasti riippuvainen yrityksen käyttämästä integraatiosovellukseen. Opinnäytetyöprojektin tuloksena ostotilaus-moduulin on tarkoitus olla käytettävissä ilman kyseistä kolmannen osapuolen integraatiosovellusta.

Tuotteistus suoritetaan valmistamalla Fatman Oy:n etukäteen laatimat käyttäjätarinat. Opinnäytetyössä tutustutaan projektissa käytettyihin teknologioihin sekä paneudutaan valittujen käyttäjätarinoiden toteutukseen yksityiskohtaisesti. Opinnäytetyöprojekti suoritettiin syys-marraskuussa vuonna 2018.

Projektin tuloksena Fatman Frame-ohjelmiston ostotilaus-moduulista saatiin tehtyä itsenäinen kokonaisuus, joka soveltuu paremmin myytäväksi useammille asiakkaille. Riippuvuudet integraatiosovelluksesta saatiin poistettua.

Asiasanat
.NET Framework, C#, SQL, JavaScript

Sisällys

Käsitteistö

Moduuli	Itsenäisesti toimiva ohjelmiston osa, jolla on oma tehtävänsä. Frame-sovellus koostuu useista moduuleista, joihin myös itse ostotilaus-moduuli lukeutuu.
SaaS	Peräisin englannin kielen sanoista Software as a Service. Yksinkertaisuudessaan sillä tarkoitetaan sovellusta, joka on käytettävissä internet-selaimen kautta.
Integraatio	Tarkoittaa kahden järjestelmän toisiinsa yhdistämistä. Suoritetun ohjelmistointegraation jälkeen kaksi eri sovellusta voivat keskustella toistensa kanssa sujuvasti. Aiheesta lisää: https://www.integral.fi/yleinen/mika-on-ohjelmistointegraatio/
Olio (Object)	Olio-ohjelmointi perustuu nimensä mukaisiin olioihin. Oliot toimivat ohjelman rakennusosina. Olion voi luoda melkein mistä tahansa ja yhteistä samalle oliolle on sen ominaisuudet (muuttujat) ja toiminnot (metodit).
Luokka (Class)	Oliot muodostetaan luokkien avulla. Esimerkiksi kuvitellaaan luokka nimeltä Kulkuneuvo, jonka pohjalta voidaan muodostaa useampia olioita kuten auto, juna ja laiva. Lisää olioista ja luokista: http://www.uef.fi/web/anja.kareinen/olio-ohjelmointi
Debugger	Tietokoneohjelma, jolla muista ohjelmista voi etsiä virheitä. Usein sisäänrakennettu sovellusympäristöihin, kuten projektissa käytettyyn Visual Studioon.
Funktio (Function)	Itsenäinen lähdekoodin pätkä, joka suorittaa tietyn ennalta määrätyn halutun toiminnon. Esimerkiksi funktioon syötettyjen lukujen erotuksen.
Refaktorointi	Refaktoroinnilla tarkoitetaan lähdekoodin selventämistä ja yksinkertaistamista itse toiminnallisuutta muuttamatta. Lue lisää: https://resources.collab.net/agile-101/code-refactoring

1 Johdanto

Projektin toimeksiantajana toimii yritys nimeltä Fatman Oy. Kyseinen yritys tarjoaa pilvipohjaisia SaaS-palveluita kiinteistönhallintaan. Yritys on toiminut jo vuodesta 1991 alkaen ja sijaitsee tällä hetkellä Espoon Otaniemessä. Työntekijöitä Fatman Oy:lla on noin 40 kappaletta yhdeksästä eri kansallisuudesta. Yrityksen ohjelmistot tarjoavat erilaisia ratkaisuja muun muassa toiminnanohjaukseen, laskutukseen ja raportointiin. Itse olen työskennellyt Fatman Oy:ssä vuoden 2017 tammikuusta asti ohjelmistokehittäjänä. Aluksi harjoittelijana Junior Software Developer-tittelillä ja myöhemmin vakituisena työntekijänä Software Developer-tittelillä.

Itse opinnäytetyöprojekti toteutetaan yrityksen uusimpaan sovellukseen nimeltä Frame. Aikaisemmin sovelluksesta käytettiin nimitystä Fatman Framework. Kyseinen ohjelmisto on tehty käyttäen .NET MVC-viitekehystä. Kaikessa yksinkertaisuudessaan Frame sisältää kiinteistön toiminnanohjauksen kätevässä verkkopohjaisessa paketissa. Ohjelmisto koostuu useista itsenäisistä moduuleista, joita asiakkaille myydään yhdessä ja erikseen. Ostotilaus-moduuli on yksi edellä mainituista. Se on myös yksi sovelluksen vanhimmista sekä vähiten käytetyistä moduuleista. Muihin sovelluksen moduuleihin lukeutuu esimerkiksi palvelupyynnöt ja sopimustenhallinta.

Ostotilaus-moduuli on lähtötilanteessa sidottu vahvasti kolmannen osapuolen integraatiosovellukseen, joka on ainoalla tällä hetkellä moduulia käyttävällä asiakkaalla käytössä. Kyseisestä asiakkaasta käytän jatkossa nimeä yritys X. Fatman Oy ei halua jatkossa myydä moduulia kolmannen osapuolen ohjelmistoon riippuvaisena, joten yritykselle syntyi tarve irroittaa se riippuvuudesta kyseiseen integraatioon. Opinnäytetyön tarkoituksena on irrottaa moduuli integraatiosovelluksesta ja samalla muokata siitä soveltuva mahdollisimman monelle asiakkaalle. Tämä tulisi suorittaa niin, että aikaisemmin moduulia käyttävälle asiakkaalle se pysyisi entisellään. Samanaikaisesti lähdekoodia tulisi myös refaktoroida mahdollisimman paljon mikäli tarvetta sille ilmenee.

Itse opinnäytetyössä perehdyin Frame-ohjelmiston ostotilaus-moduuliin ja Fatman Oy:n luomiin käyttäjätarinoihin. Suunnittelin ja toteutin ratkaisut edellä mainittuihin tarinoihin käyttämällä .NET MVC-tekniikoita. Tarpeen tullen ongelmien ilmaantuessa sain totta kai myös apua yrityksen kokeneemmilta ohjelmistokehittäjiltä ja muilta työntekijöiltä. Moduulin tuotteistamisella tässä kontekstissa tarkoitan siis sen muokkaamista sopivaksi useammille asiakkaille yritys X:n lisäksi. Opinnäytetyön ohjelmointiosuus toteutettiin syys-marraskuussa vuonna 2018. Suurimman osan toteutuksesta tein itse yksin noin kolmessa kahden viikon kehitysjaksossa.

2 Selvitys- ja määritystyö

Seuraavissa kappaleissa käydään läpi projektille oleellisia osia. Ensimmäisessä kappaleessa (2.1 Sovellusympäristö) tutustutaan projektille relevanttiin sovellusympäristöön. Seuraavassa kappaleessa (2.2 Liiketoimintalogiikka) käydään läpi kuinka tutustuin ostotilaus-moduulin liiketoimintalogiikkaan ja käydään se peruspiirteittäin myös läpi. Viimeisessä kappaleessa (2.3 Käyttäjätarinat) käydään yleisellä tasolla läpi minkä tyyppisiä käyttäjätarinoita projekti sisälsi.

2.1 Sovellusympäristö

Fatman Oy:n Frame-ohjelmisto on toteutettu C#-ohjelmointikielellä käyttäen .NET MVC Framework versiota 4.6.1. Sovellus on SaaS- (Software as a Service) pohjainen ja tämän myötä web-selaimen kautta käytettävissä. Sovellus on toteutettu SOA- (Service Oriented Architecture) ajattelumallia käyttäen. SOA:n ideana on jakaa sovellukset useampiin itsenäisiin palveluihin (eng. service), joita on mahdollista käyttää rajapintojen kautta. (Service Architecture 2018)

Sovelluksen DataServices-projekti toimii sen tietokantarajapintana ja se vastaa myös kommunikaatiosta tietokannan kanssa. DataServices vastaa SOA-arkkitehtuurin palvelurajapintaa ja myös se on totetutettu C#-ohjelmointikielellä .NET-sovelluskehystä käyttäen.

Sovellusympäristön tietokantana toimii SQL-tietokantapalvelin. Fatman Oy:n kaikki sovellukset käyttävät samaa palvelinta Frame-ohjelmisto mukaan lukien. Jokaisella asiakkaalla on palvelimella oma tietokantansa, joiden taulurakenteita yritetään pitää mahdollisimman identtisinä. Tähän pientä lisämaustetta tuo integraatiot, joiden tauluja ei jokaiseen kantaan lisätä. Tämän huomasin yritys X:n ja ostotilaus-moduulin kanssa.

Integraatiot sovelluksien välillä on yrityksessä toteutettu usein eri tavoin. Viime aikoina erillisten integraatioiden valmistusta on vältelty ja asiakkaita on ohjattu käyttämään Fatman Oy:n REST-ohjelmointirajapintaa. Aikaisemmin niitä on toteutettu muun muassa Java-ohjelmointikieleen perustuvan Mule-viitekehyksen avulla. Mule-sovellukset tyypillisesti hyödyntävät myös aikaisemmin mainittua DataServices-projektia. Mule-projektien lisäksi integraatioita on toteutettu muun muassa Delphi-ohjelmointikielellä.

2.2 Projektissa käyteyt teknologiat ja tekniikat

Seuraavissa alaluvuissa (2.2) tutustutaan projektissa käytettyihin teknologioihin ja tekniikkoihin hiukan tarkemmin. Näistä oleellisimpia ovat C#-ohjelmointikieli, .NET-sovelluskehys, Javascript-ohjelmointikieli ja sen jQuery-kirjasto. Toistaiseksi Fatman Oy ei ole ottanut käyttöönsä uudempia Javascript sovelluskehyksiä kuten Angular tai React. Tutustutaan myös Fatmanin ohjelmistokehitykselle oleellisen Scrum-viitekehyksen perusperiaatteisiin.

2.2.1 C# ja .NET Framework

C# on Microsoftin kehittämä ylemmän tason (eng. high level) ohjelmointikieli. Se on oliopohjainen ja syntaksiltaan melko lähellä esimerkiksi Javaa. Se suunniteltiin alun perin käytettäväksi lähinnä Microsoftin Windows-ympäristössä, mutta nykyään sillä kehitetään ohjelmistoja niin Linuxille, Androidille kuin iOS-järjestelmillekin.

.NET Framework on Microsoftin kehittämä avoimen lähdekoodin ohjelmistokehys. Se on tehty helpottamaan ohjelmistokehitystä tukemillaan ohjelmointikielillä, joihin lukeutuu C#:n lisäksi muun muassa Visual Basic ja F#. Se pitää sisällään suuren FCL-luokkakirjaston (Framework Class Library), joka tarjoaa esimerkiksi uudelleenkäytettäviä luokkia ja rajapintoja. Näitä käyttämällä .NET-ohjelmistokehittäjän ei tarvitse keksiä pyörää uudestaan vaan hän voi keskittyä tarkemmin ohjelmistonsa liiketoimintalogiikkaan. (Microsoft Docs 2018)

Projektissa ohjelmistoympäristönä (eng. IDE, integrated development environment) käytin Microsoftin Visual Studio 2017 Enterprise-versiota. Kyseisen ympäristön kanssa käytin myös suosittua JetBrains Resharper laajennusosaa. Resharper helpottaa .NET-kehittäjän elämää muun muassa erilaisten formatointi-työkalujen ja uusien näppäinyhdistelmien avulla.

C# ja .NET Framework ovat projektissa käytössä palvelinpuolella (eng. back end). Palvelinpuoli vastaa liiketoimintalogiikan toteutumisesta ja tietokannan kanssa kommunikoinnista.

Frame-sovellus on toteutettu käyttäen .NET-viitekehykseen kuuluvalla .NET MVC-komponentilla. MVC toteuttaa nimensä mukaista malli-näkymä-käsittelijä (eng. model-view-controller) ajattelumallia. Siinä malli vastaa tiedon käsittelystä ja sen tallentamisesta. Näkymä määrää miltä käyttöliittymä näyttää ulospäin. Käsittelijä vastaa käyttäjän

komentoihin käyttämällä mallia ja näkymää hyödykseen. Osat ovat riippumattomia toisistaan ja esimerkiksi mallia on mahdollista hyödyntää useissa eri näkymissä.

2.2.2 JavaScript ja jQuery

JavaScript on ylemmän tason ohjelmointikieli. Se muodostaa HTML:n ja CSS:n kanssa WWW:n (World Wide Web) kolme pääkieltä. JavaScript on alunperin luotu selainpohjaiseksi ja tekemään staattisista verkkosivuista dynaamisia ja interaktiivisia. Nykyään JavaScriptin käyttö on mahdollistettu myös palvelinpuolella esimerkiksi Node.js-kirjaston avulla. (Wikipedia 2018)

jQuery on avoimen lähdekoodin JavaScript-kirjasto. Kyseinen kirjasto on lähinnä tehty helpottamaan ja yksinkertaistamaan JavaScriptin käyttöä. Frame-sovelluksessa yleisesti ja ostosopimus-moduulissa sitä käytetään lähinnä Ajax-kutsujen suorittamiseen. Ajax mahdollistaa asynkronisten JavaScript-kutsujen käytön, joka käytännössä tarkoittaa sitä että sivun dataa voidaan päivittää lataamatta kuitenkaan koko sivua uudestaan. (W3Schools 2018)

Javascript ja sen kirjasto jQuery ovat projektissa käytössä selainpuolella (eng. front end). Sovelluksissa selainpuoli vastaa lähinnä ohjelmiston ulkoasusta.

2.2.3 SQL ja relaatiotietokannat

SQL (Structured Query Language) on kyselykieli, jota projektissa käytetään relaatiotietokantaan tehtäviin hakuihin, lisäyksiin ja muutoksiin. Palvelimena Microsoft painotteisessa ympäristössämme toimii Microsoftin oma SQL Server. Relaatiotietokannassa data on tallennettu tauluihin, joiden sisällä se on järjestetty sarakkeisiin. Yksittäiseen sarakkeeseen voi tallentaa vain yhden tietotyypin dataa. Relaatiotietokantojen oleellinen ominaisuus on suhteet taulujen ja sarakkeiden välillä. Tauluihin voidaan lisätä indeksejä hakujen nopeuttamiseksi. Indeksi on tapa etsiä tauluista dataa yhden tai useamman sarakkeen perusteella. (Agiledata 2018)

2.2.4 Git

Projektin versionhallintajärjestelmänä toimi Git. Versionhallintajärjestelmä on järjestelmä, jolla tiedostoihin tehtyjä muutoksia voidaan tallentaa aika ajoin, jotta aikaisempiin versioihin on mahdollista palata myöhemmin. Muutokset tallennetaan ensin paikallisesti omalle tietokoneelle ja myöhemmin siirretään palvelimelle. Palvelimella ne tallennetaan niin sanottuun tietolähteeseen (eng. repository), josta versioihin pääsee käsiksi

samanaikaisesti useampikin ohjelmistokehittäjä. Fatman Oy:n tietolähteen asemaa ajaa GitLab, jolle toinen yleisesti käytetty vaihtoehto on GitHub. Ohjelmistokehityksen parissa versionhallintaa käytetään tietenkin ohjelmiston lähdekoodiin, mutta tarpeen vaatiessa mikä tahansa tiedosto sopii versiohallittavaksi.

Oleellisena ominaisuutena Git sallii työskentelyn haaroissa (eng. branch). Haara on ohjelmistokehittäjän henkilökohtainen versio lähdekoodista. Luotuaan oman haaran lähdekoodista kehittäjän ei tarvitse välittää muiden tekemistä muutoksista samoihin tiedostoihin, vaan hän voi työskennellä rauhassa. Fatman Oy:n kohdalla henkilökohtaiset haarat yhdistetään (eng. merge) yhteiseen kehityshaaraan aina ennen testiympäristön päivitystä kahden viikon välein. Tässä vaiheessa samoihin tiedostoihin tehdyissä muutoksissa saattaa ilmetä konflikteja, jotka täytyy selvittää käsin. Kun konflikteista selvitään, lähdekoodi on valmis siirrettäväksi testipalvelimelle. Myöhemmin ennen tuotantopäivitystä testattu kehityshaara yhdistetään tuotantohaaraan, joka siirretään samaan tapaan tuotantopalvelimelle. (Git SCM 2009).

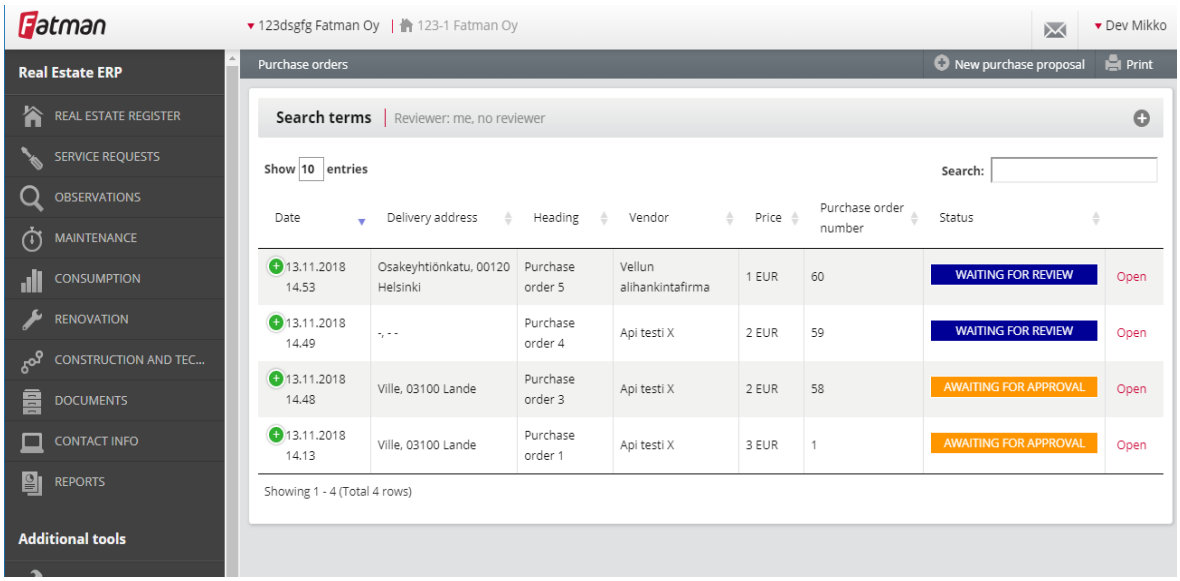
2.2.5 Scrum

Scrum on viitekehys projektinhallintaan. Fatman Oy:ssä ohjelmistopuolella Scrumin ketterää lähestymistapaa ohjelmistokehitykseen seurataan melko tarkasti.
Scrumin perusideana on jakaa projektin valmistus kehitysjaksoihin, joita kutsutaan sprinteiksi. Tyypillisesti sprintit kestävät noin 1-4 viikkoa, Fatman Oy:n tapauksessa sprintit ovat kahden viikon pituisia. Kehitysjaksoille valitaan tietty määrä käyttäjätarinoita työmääräarvioiden ja asiakkaiden toiveiden perusteella. Käyttäjätarinoista lisää kappaleessa 2.4. Scrum sanana ei ole akronyymi vaikka niin voisi olettaakin.

Kun sprint aloitetaan järjestetään suunnittelupalaveri, johon scrum-ryhmä kokoontuu. Palaverissa päätetään sprintille otettavista käyttäjätarinoista tuoteomistajan (eng. product owner) johdolla. Tuoteomistaja on yleensä henkilö ohjelmistokehitysryhmän ulkopuolelta. Tuoteomistajan roolina tapaamisissa on vastata sovelluksista ja edustaa asiakkaita. Sprintin päätteeksi järjestetään retrospektiivi, jossa käydään läpi miten kulunut sprint sujui ja kuinka tulevasta sprintistä voidaan tehdä entistä parempi näiden huomioiden pohjalta. Palavereita johtaa scrum master. Scrum master on ohjelmistokehitysryhmän jäsen, joka vastaa siitä että ryhmä noudattaa scrum-viitekehyksen periaatteita. (Scrum 2018)

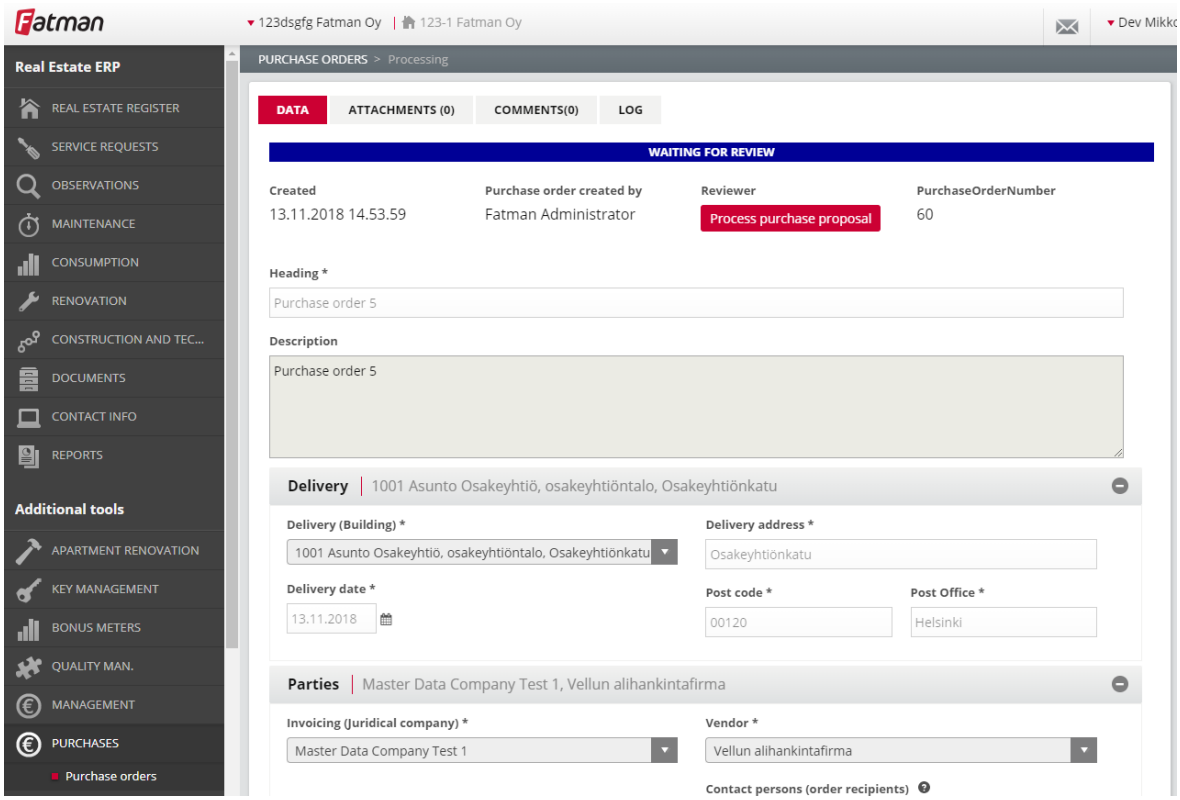
2.3 Ostotilaus-moduulin käyttöliittymä

Käyttöliittymältään ostotilausmoduuli ei ole monimutkainen. Se sisältää listanäkymän, johon ostotilaukset listataan erilaisten käyttäjäseulojen läpi. Seulat määrittelevät muun muassa oikeudet rakennuksiin, joihin ostotilauksia voi osoittaa. Lisää rakennuksista liiketoimintalogiikkakappaleessa (2.4). Listanäkymä sisältää kaikki oleellisimmat tiedot tilauksista näppärästi tiivistettynä. Näihin lukeutuu esimerkiksi ostotilauksen luontiaikaleima, nimi ja tämän hetkinen tila. Lista sisältää myös hakuehdot-valikon, josta ostotilauksia voi seuloa esimerkiksi tilojen perusteella. Listasta yksittäisen ostotilauksen voi avata omaan muokkausnäkymäänsä, jossa yksityiskohtiin päästään käsiksi. Tästä esimerkki myöhemmin kuvassa 2. Listasta pääsee myös ostotilauksen luontinäkymään, joka on melko lähellä aikaisempaa muokkausnäkymää. Aikaisempien näkymien lisäksi erikseen on vielä asetusnäkymä, jossa päästään muokkaamaan esimerkiksi muokkaus- ja luontinäkymien alasvetovalikkojen sisältöä. Asetusnäkymä on näkyvissä vain mikäli käyttäjällä on siihen erillinen oikeus. Kuva 1 havainnollistaa ostotilauslista-näkymän käyttöliittymää.



Kuva 1. Ostotilauslista, hakuehdot suljettuna

Ostotilauslistassa on käytetty jQuery-kirjaston DataTables-liitännäistä. Vihreistä plus-merkeistä yksittäisen ostotilauksen saa laajennettua näyttämään enemmän tietoa itsestään. ”Open”-nappulasta ostotilaus aukeaa omaan näkymäänsä, josta on kuva seuraavalla sivulla. Molemmissa kuvissa vasemmassa laidassa sijaitsee navigointipalkki, josta Frame-sovelluksen eri moduulien välillä on mahdollista liikkua. Frame-sovellus tukee useita eri kieliä ja opinnäytetyön kuvissa päädyin käyttämään englantia.



Kuva 2. Avattu ostotilaus

2.4 Liiketoimintalogiikka

Ensi askeleena projektin toteuttamiseen tutustuin ostotilaus-moduulin liiketoimintalogiikkaan, koska siitä minulla ei juuri ollut aikaisempaa kokemusta. Liiketoimintalogiikka kuvaa periaatteessa sen kuinka ohjelman tulee toimia liiketoiminnan kannalta katsottuna. Oman kokemukseni pohjalta liiketoimintalogiikan hahmottaminen lähtee asiakkaan tarpeen ymmärtämisestä. Tutustuin liiketoimintalogiikkaan ja asiakkaiden tarpeisiin lukemalla aikaisempaa dokumentaatiota aiheesta sekä yksinkertaisesti käymällä moduulia läpi askel askeleelta käyttöliittymän kautta. Vaikka dokumentaatio jätti osittain hieman toivomisen varaa, sai siitä kohtalaisen hyvän kuvan moduulin käyttötarkoituksesta ja asiakkaiden tarpeista. Moduulin toiminnan selvittämiseen sain myös apua aikaisemmin parissa työskenneiltä ohjelmistokehittäjiltä ja projektipäälliköiltä.

Ostotilauksille oleellista on myös käydä läpi Frame-sovelluksen kiinteistörekisteri-moduulin toiminta. Fatman Frame on pääosin tehty kiinteistöhallintaa varten. Tämän myötä yksi sen oleellisimmista moduuleista on kiinteistörekisteri. Kyseisen moduulin tarkoituksena on yritysten kiinteistöihin liittyvän datan ylläpito. Tätä dataa hyödynnetään lähes kaikissa Frame-sovelluksen moduuleissa, ostotilaukset mukaan lukien. Kiinteistötiedot on pääosin jaettu kolmeen tasoon: kiinteistö-, rakennus- ja huonetasoon.

Kiinteistöt sisältävät useita rakennuksia ja rakennukset useita huoneita. Kukin tasoista sisältää tasolle ominaista oleellista tietoa. Esimerkiksi kiinteistöt sekä rakennukset sisältävät molemmat osoitetietoja ja huoneet muun muassa pinta-alatiedon. Kiinteistöjä voidaan myös jaotella erilaisiin ryhmiin ja eri moduuleissa voidaan näin toimia myös kiinteistöryhmätasoilla.

Liiketoimintalogiikkansa osalta ostotilaus-moduuli on loppujen lopuksi melko yksinkertainen. Moduulilla luodaan järjestelmään ostotilauksia, jotka toimivat dokumentaationa ulkoisilta palveluntarjoajilta ostetuille palveluille tai tuotteille. Frame-sovelluksen tapauksessa ostotilaus luodaan ensin järjestelmään ostoehdotuksena, joka kulkee useamman tilan läpi ja lopulta muuttuu ostotilaukseksi. Tilausten pohjalta yritykset voivat suorittaa erilaisia ostoksiaan. Minkään tyypistä automaattista ostoa ei hyväksytyn ostotilauksen pohjalta siis suoriteta, mutta ostotilauksen valmistuessa asiakkaat voivat niitä suorittaa. Ostotilauksia pystyy Frame-sovelluksessa luomaan joko itse omassa moduulissaan tai muutaman muun eri moduulin kautta. Ostotilauksen voi luoda esimerkiksi olemassa olevan palvelupyynnön pohjalta. Tämä tulee tietenkin pitää mielessä moduuliin muutoksia tehdessä.

Ostotilaus tulee osoittaa jollekin asiakkaan rakennuksista. Rakennuksen pohjalta voidaan määrittää kyseiselle ostoehdotukselle toimitusosoite. Tämän lisäksi ehdotukselle vaaditaan sekä laskuttaja että toimittaja. Laskuttaja toimii ostotilauksen maksajana ja toimittaja kuvaa miltä yritykseltä itse tuotteen tai palvelun tilaus tehdään. Toimittajalle lisätään myös yhteyshenkilön sähköpostiosoite, johon voi ottaa yhteyttä esimerkiksi ongelmatilanteissa. Rakennuksia, laskuttajia sekä toimittajia seulotaan käyttäjätunnuksille annettujen seulojen pohjalta. Kaikilla käyttäjillä ei siis ole oikeuksia tehdä tilauksia kaikille rakennuksille, laskuttajille tai toimittajille. Asiakkaiden admin-käyttäjät määrittelevät seulat muille käyttäjille Frame-sovelluksen seula-moduulin kautta.

Jokaiseen ostoehdotukseen tulee myös lisätä vähintään yksi hintarivi, joka määrittää tilattavan tuotteen tai palvelun. Hintariville tulee lisätä kappalemäärä ja hinta. Ennen moduulin uudistamista hintarivi valittiin valmiiden vaihtoehtojen joukosta kolmen alasvetovalikon avulla. Tutustutaan hintariveihin tehtäviin muutoksiin lisää kappaleessa 3.1.

Edellä mainittujen lisäksi ostotilauksiin voi lisätä liitetiedostoja tai kommentteja. Liitetiedostona voi esimerkiksi olla valokuva korjausta vaativasta osasta. Kommentteihin voi lisätä mahdollisia ehdotusta koskevia lisätietoja. Kommentit sisältävät myös oleellisen aikaleiman.

Monimutkaisuutta moduuliin aiheuttaa roolien määrittämät oikeustasot. Rooleja on kolme: toimittaja, tarkastaja ja hyväksyjä. Roolit liitetään Frame-sovelluksen käyttäjätunnuksiin. Roolin anto tuo käyttäjälle lisäoikeuksia, ei rajaa niitä pois. Toimittajan on ainoastaan mahdollista luoda uusia ja selata tehtyjä ostotilauksia. Tarkastajalla on toimittajan oikeuksien lisäksi mahdollisuus muokata ehdotusten ja tilausten tietoja sekä tehdä roolille sallittuja tilasiirtoja. Tarkastajalla on myös oikeus asettaa itsensä ostoehdotuksen käsittelijäksi. Käsittelijä ei ole oma roolinsa vaan ostoehdotuksen tieto, joka rajoittaa sitä ettei kuka tahansa muokkaa ostoehdotusta sen käsittelyyn oton jälkeen. Tarkastaja voi edellisten lisäksi palauttaa ehdotuksen takaisin toimittajalle. Hyväksyjä voi niin ikään myös luoda uusia ja selata tehtyjä ostotilauksia. Hyväksyjä voi myös tehdä roolille sallittuja tilasiirtoja sekä palauttaa ostoehdotuksen takaisin tarkastajalle. Roolien tilasiirto-oikeudet on mahdollista määritellä tietokantataulujen avulla asiakaskohtaisesti.

Lähtötilanteessa ostotilauksella on neljä tilaa. Uusi ehdotus siirtyy ensimmäisenä tilaan avoin. Avoin-tilassa tarkastaja-roolin käyttäjä voi asettaa itsensä ehdotuksen käsittelijäksi. Käsittelijä tarkastaa ehdotuksen tiedot ja tekee niihin tarpeen vaatiessa muutoksia. Tarkastuksen jälkeen käsittelijä siirtää ehdotuksen odottaa hyväksyntää-tilaan ja valitsee vähintään yhden hyväksyjän ehdotukselle. Odottaa hyväksyntää-tilassa ehdotukselle valitut hyväksyjät voivat siirtää ehdotuksen hyväksytty-tilaan. Hyväksytty on ostotilauksen viimeinen tila. Siinä ehdotus muuttuu tilaukseksi ja sen jälkeen osto on turvallista suorittaa. Jokaisesta edellä mainitusta tilasta ostoehdotus on myös mahdollista siirtää peruutettu-tilaan. Peruutettu-tilassa ehdotus lukitaan ja siihen ei voi enää tehdä muutoksia kommentteja lukuunottamatta. Jokaisen tilamuutoksen välillä voidaan vaatia pakollinen kommentti kuvaamaan esimerkiksi peruutuksen syitä. Tilamuutoksien pohjalta voidaan myös lähettää tilamuutos-sähköposteja käyttäjille, joita muutokset eniten koskettavat. Niin sähköpostit kuin pakolliset kommentit ovat asetettavissa myös tietokannan kautta.

Moduuli sisältää myös lokin, johon osa tilauksiin tehdyistä muutoksista tallentuu aikaleiman kera. Näihin muutoksiin lukeutuu muun muassa tilamuutokset kommentteineen, muutokset tarkastajissa tai hyväksyjissä ja tilausten pohjalta automaattisesti lähetetyt sähköpostit.

2.5 Vaatimusmäärittely

Projektin tavoitteena on tehdä Frame-sovelluksen ostotilaus-moduulista itsenäinen kokonaisuus, jonka toiminta ei riipu kolmannen osapuolen sovelluksista. Projektin valmistuttua moduulin tulisi olla paremmin myytävissä itsenäisenä pakettina muillekin asiakkaille yritys X:n lisäksi. Riippuvuus kolmannen osapuolen integraatiosovelluksesta näkyy lähinnä käyttöliittymässä ylimääräisinä kenttinä ja kyseisen sovelluksen vaatimina ominaisuuksina. Tarkoituksena on muun muassa poistaa riippuvuus integraatioon liittyvistä tietokantatauluista. Itse Fatman Oy:n päässä toteutettuun osaan integraatiosovelluksesta ei tarvitse projektin tiimoilta tutustua sen kummemmin, koska moduulin tulee säilyä ennallaan yritykselle X.

2.6 Käyttäjätarinat

Liiketoimintalogiikkaan tutustuttuani sekä vaatimusmäärittelyn myötä on aika selvittää mistä Fatman Oy:n valmiiksi luomissa käyttäjätarinoissa oli kyse. Käyttäjätarinoilla tarkoitetaan yleisesti asiakkailta tai projektipäälliköiltä tulleita vaatimuksia, joita sovelluksen tulee toteuttaa. Käyttäjätarinat eivät ota kantaa teknisiin ratkaisuihin vaan niistä pitää päättää ohjelmistokehittäjien kesken. Käyttäjätarinoiden hallinnointiin yrityksellä on käytössä siihen tarkoitettu verkkopohjainen sovellus JIRA.

Fatman Oy:n käyttäjätarinoissa on tyypillisesti lyhyt kuvaus toiminnallisuuden nykytilanteesta, käyttäjän tarpeesta ja tilanteesta, johon halutaan toiminnallisuuden valmistumisen myötä päätyä. Yleensä projektipäälliköt valmistavat käyttäjätarinat asiakkaiden toiveiden pohjalta. Luonnollisesti huomioon otetaan onko asiakkaan toivoma toiminnallisuus järkevä ollenkaan ja kannattaako sitä ylipäätään ohjelmistoon lisätä. Toiminnallisuuden lisäämisestä päätetään yleensä ennen kuin siitä kerrotaan itse ohjelmistokehittäjille. Poikkeuksia tähän muodostavat tapaukset, joissa toiminnallisuuden valmistamisen tekniset vaatimukset aiheuttaisivat mahdollisia ongelmia muualla ohjelmistossa tai ovat muuten ylitsepääsemättömiä.

Käyttäjätarinat sisältävät yleensä tarkat hyväksymisperusteet (eng. acceptance criteria). Kun ohjelmistokehittäjän toiminnallisuus täyttää hyväksymisperusteiden vaatimukset, voidaan tarina siirtää eteenpäin testattavaksi ja katselmoitavaksi. Testauksesta vastaavat dedikoidut testaajat. Lisää projektin tarkastus- ja testausprosesseista kappaleessa 4.

Fatman Oy:n ohjelmistokehitysryhmä seuraa ketterää Scrum-viitekehystä projektinhallintaan. Viitekehyksen prosessin myötä käyttäjätarinat päätyvät valmistuttuaan ohjelmistokehittäjille tutkittavaksi. Yksittäisten käyttäjätarinoiden teknisiä vaatimuksia tutkitaan pienellä ohjelmistokehittäjien joukolla ja itse tarinalle annetaan karkea työmääräarvio. Fatman Oy:lle tyypillisten käyttäjätarinoiden työmääräarviot vaihtelevat keskimäärin yhdestä viiteen päivään. Työmääräarvioiden ja sprintin osallistujamäärän pohjalta voidaan arvioida kuinka paljon yhden kehitysjakson aikana käyttäjätarinoita on mahdollista valmistaa. Sprintille otettavista käyttäjätarinoista päätetään sen suunnittelupalaverissa.

Ostotilaus-moduulin kehittämistä koskevia käyttäjätarinoita oli etukäteen luotu yhteensä 11 kappaletta. Suurin osa kyseisistä käyttäjätarinoista oltiin tehty jo vuoden 2017 alkupuolella, jonka takia niiden paikkansapitävyys tuli tarkastaa uudelleen. Kävin tarinat läpi eniten moduulista tietävän projektipäällikön kanssa ja hänen kanssaan saimme ne päivitettyä ajan tasalle. Perusideat tarinoiden takana eivät olleet muuttuneet melkein kahden vuoden aikana, mutta pieniä parannusehdotuksia niihin ilmeni keskusteluiden myötä.

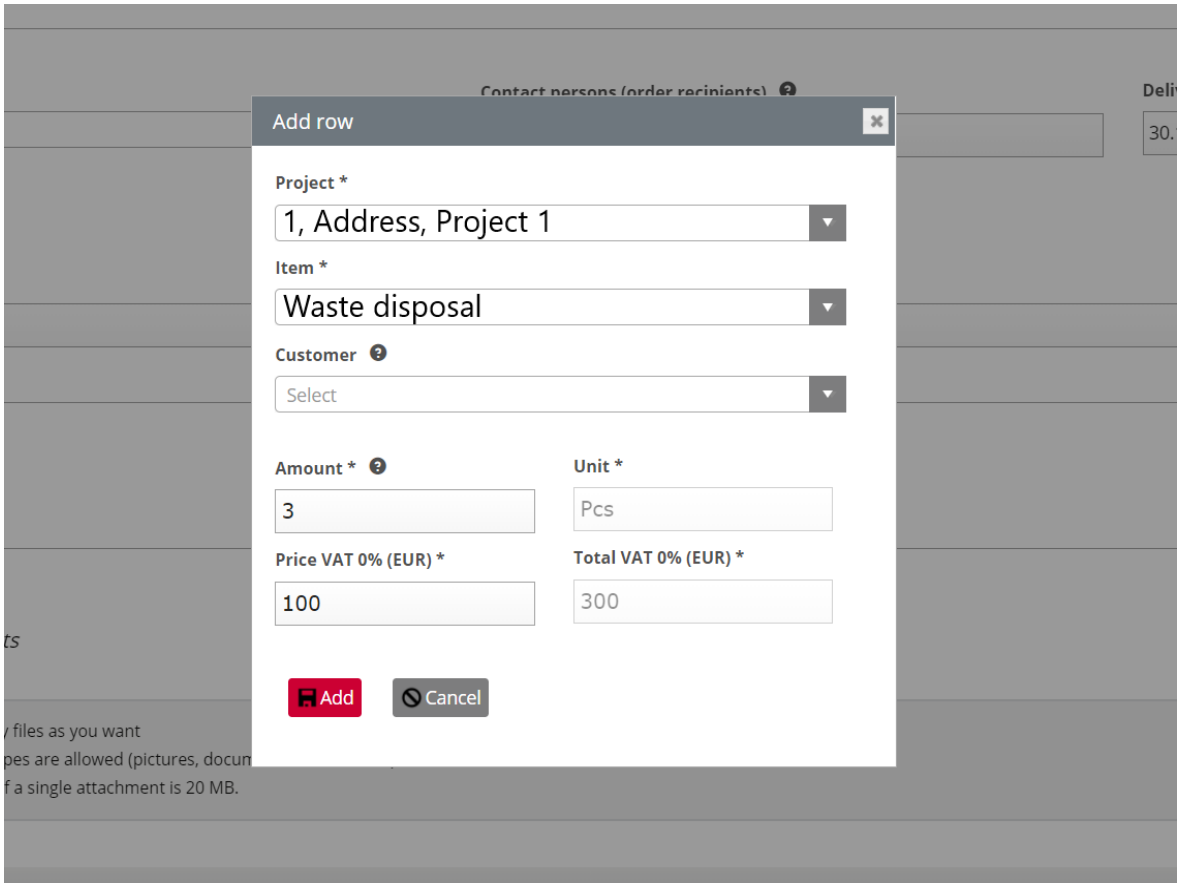
Käyttäjätarinoiden selkiydyttyä oli aika suunnitella missä järjestyksessä ne olisi järkevintä suorittaa. Suoritusjärjestyksen suunnittelua helpotti se, että tarinoiden välille syntyi selkeitä linkkejä liiketoimintalogiikan myötä. Osaa tarinoista ei pystyisi valmistamaan ennen kuin ensimmäisiä niistä on suoritettu. Tämän myötä järjestyksestä sai suunniteltua kohtuu loogisen.

3 Toteutus

Seuraavissa kappaleissa käydään läpi kolmen valitun käyttäjätarinan toteutus pääpiirteittäin. Kappaleissa pohjustetaan käyttäjätarinat ja avataan niiden toteutusta tutustumalla niin lähdekoodiin kuin muihinkin vaadittuihin muutoksiin. Jokaisen käyttäjätarinan läpi käymistä en koe oleellisena. Osa käyttäjätarinoista vaati enemmän työtä, osa vähemmän. Seuraavat käyttäjätarinat antavat hyvän kuvan projektista.

3.1 Hintarivit-käyttäjätarina

Ensimmäisenä tutustutaan ostotilausten hintariveihin tehtäviin muutoksiin. Käyttäjätarina kuvaa nykytilanteen, tavoitellun tilanteen sekä hyväksymisperusteet. Ostoehdotuksen luoja haluaa mahdollisuuden antaa hintariville nimen itse eikä valita sitä alasvetovalikkojen pohjalta. Kuva hintarivien lisäys-dialogin nykytilanteesta liitteenä alapuolella. Aikaisemmissa käyttötarinoissa ostotilausprosessista halutaan kokonaan poistaa projekti (liitteessä project), valmiit tuotteet (liittessä item) ja asiakkaat (liitteessä customer).



Kuva 3. Hintarivin lisäys käyttöliittymässä, lähtötilanne

Valmiit dialogin kautta lisätyt hintarivit listataan ostotilauksen alla HTML taulu-elementissä. Kyseisessä taulussa on hyödynnetty jQuery-kirjaston DataTables-liitännäistä.

Heti lähtötilanteessa havaitaan selkeä tarve asetukselle, jolla saadaan hintarivien lisäys säilymään ennallaan yritys X:n tapauksessa. Tähän sopii ratkaisuna tietokantapohjainen konfiguraatio, jolla osa toiminnallisuudesta saadaan piiloon asiakkailta yritys X lukuunottamatta. Frame-sovelluksessa konfiguraatiot on tallennettuna omaan tietokantatauluunsa, josta ne ladataan välimuistiin (eng. cache). Sovellus tukee useita eri konfiguraatiotyyppejä, mutta tässä tapauksessa asetuksemme arvo on tyyppiä boolean. Asetus siis joko on päällä tai ei ole. Samaa asetusta käytän lähes joka käyttäjätarinassa, jotta yritys X:n ominaisuudet voidaan piilottaa kätevästi vain yhtä asetusta käyttäen.

```
INSERT INTO Configs (Section, ParameterName, Parametervalue, ValueType)
VALUES ('PurchaseOrders', 'XIntegrationEnabled', 1, 'Boolean')
```

Kuva 4. SQL-komento konfiguraation luomiselle

Kun konfiguraatio on saatu luotua, tutustuaan itse dialogin näkymään (eng. view). Dialogi on toteutettu omaan ”partial view”-näkymäänsä. Kyseisten näkymien ideana on se, että niitä voidaan sisällyttää toisten näkymien sisään ja näin tarpeen vaatiessa käyttää helposti muuallakin sovelluksessa. Jos hintarivejä pitäisi pystyä lisätä ostotilauksiin toisen moduulin kautta, näkymän lisääminen muualle on suhteellisen helppoa. s

MVC-arkkitehtuurin myötä näkymä toteuttaa jotain mallia (eng. model). Tässä tapauksessa toteutetaan hintariveille luotua mallia. Kyseiseen malliin lisätään uusi boolean-tyypin attribuutti vastaamaan aikasemmin luotua konfiguraatiota, jotta sen arvoon on mahdollista päästä käsiksi itse näkymässä.

```
public bool XIntegrationEnabled => Configurations.GetBoolConfig("Purchase
Orders", "XIntegrationEnabled");
```

Kuva 5. Lähdekoodia malli-luokasta.

Seuraavaksi näkymään tarvitaan ehtolause konfiguraatiolle, jolla dialogista saadaan piilotettua siitä pois halutut alasvetovalikot projekteille, tuotteille ja asiakkaille. Alasvetovalikot korvataan mallissa jo valmiina olleella merkkijono-tyyppisellä (string) ”Name”-kentällä. Kentän viereen luodaan myös nimi ja tila validointivirhe viestille. Näiden toteutus hoidetaan käyttämällä .NET MVC:n Html-komponenttia. Alasvetovalikkojen

piilottamisen lisäksi yksikkö-kenttä (eng. unit) halutaan säilyttää yritys X:lle disabloituna, mutta muille asiakkaille se halutaan muokattavaksi. Seuraavaksi vilkaisu lähdekoodiin.

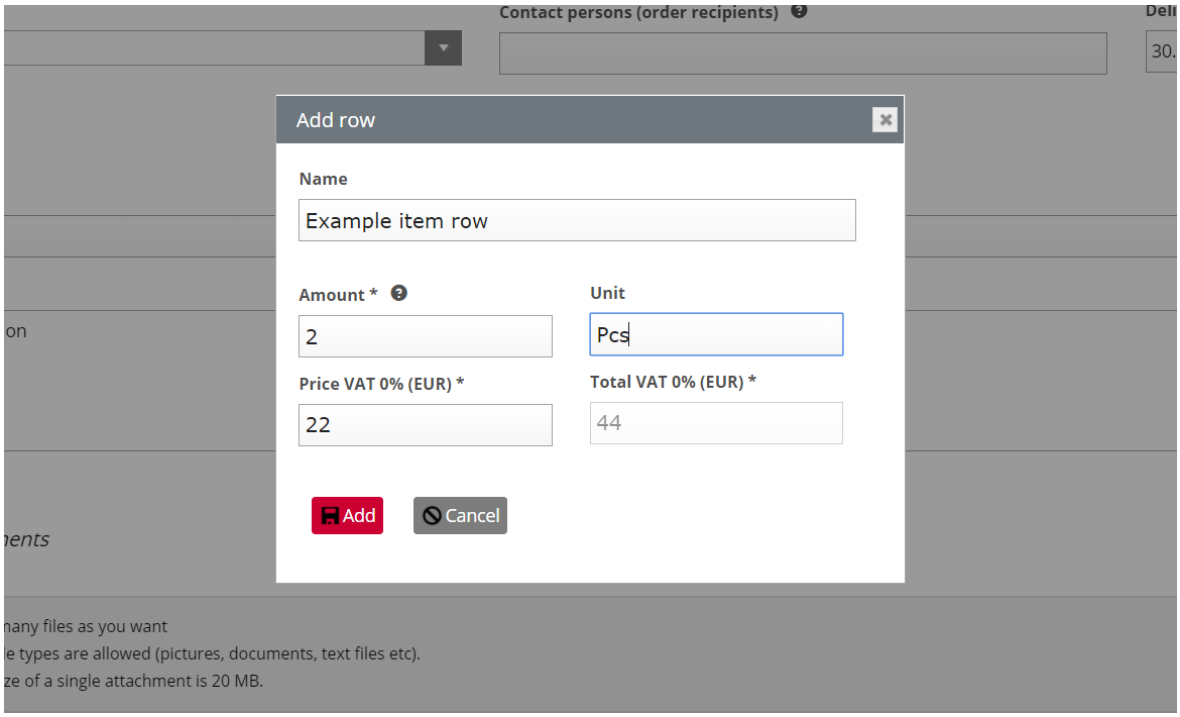
```
@if (Model.XIntegrationEnabled) // Ehtolause
{
<div class="form-row">
    @Html.LabelFor(m => m.ProjectID)
    <div class="select-input">
        @Html.DropDownListFor(m => m.ProjectID,Enumerable.Empty<SelectLis
tItem>())
    </div>
    @Html.ValidationMessageFor(m => m.ProjectID)
</div>

// Tuotteet jne.

}
else
{
<div class="form-row">
    @Html.LabelFor(m => m.Name)
    @Html.TextBoxFor(m => m.Name)
    @Html.ValidationMessageFor(m => m.Name)
</div>
}

@if (Model.XIntegrationEnabled)
{
    @Html.TextBoxFor(m => m.Unit, new { @disabled = "disabled" })
}
else
{
    @Html.TextBoxFor(m => m.Unit)
}
```

Kuva 6. Hintarivin lisäys-dialogin lähdekoodia. Kyseessä näkymä-luokka (eng. view)

Javascript-puolelle muutoksia tarvittiin myös muutamia. Lähinnä siihen kuinka nimi näytetään lisäämisen jälkeen DataTables-liitännäisen taulussa. Aikaisemmin nimi muodostettiin kuvan 3. kolmen alasvetovalikon perusteella. Muutosten jälkeen alasvetovalikot on piilotettu ja nimi tulee suoraan sille varatusta Name-kentästä. Konfiguraation haku on tallennettu näkymään hidden-kentäksi arvolle ”x-integration-enabled”, jolla sen voi helposti jQueryn avulla noutaa JavaScript-puolella.

```
var xIntegrationEnabled = $("#x-integration-enabled").val() === "True";
var itemName;
if (xIntegrationEnabled) {
    itemName = purchaseItemID + " " + purchaseItemName;
    if (customerId) {
        itemName = itemName + " (" + customerName + ")";
    }
} else {
    itemName = $("#AddPriceRow_Name").val();
}
```

Kuva 7. Nimen muodostaminen näkymän taulua varten JavaScriptillä

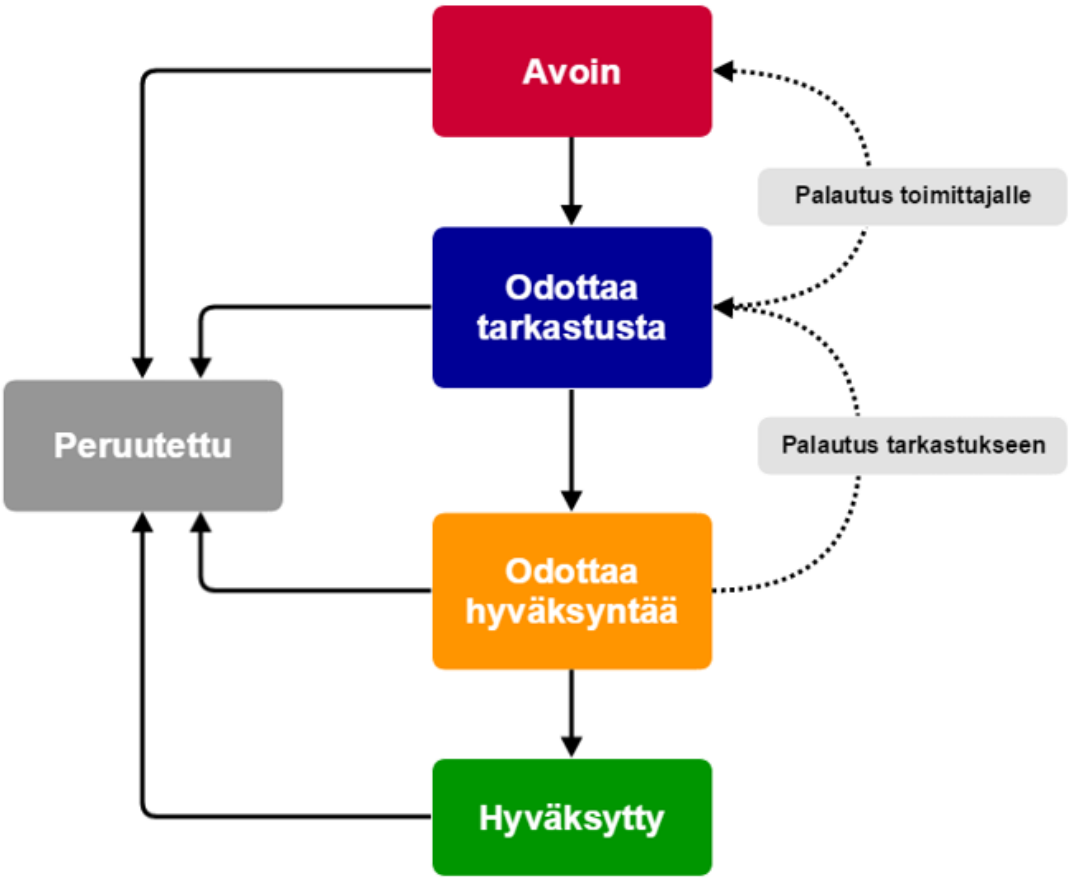


Kuva 8. Hintarivin lisäys käyttöliittymässä, lopputulos

Edellä käyty hintarivit-käyttäjätarina oli melko yksinkertainen. Ratkaisuun päätyminen oli näin ollen helppoa. Lopputuloksen piilotetut alasvetovalikot ja uusi Name-kenttä havainnollistettu kuvassa 8. Uskon että käyttäjätarinan lopputulos olisi ollut melko samanlainen riippumatta siitä kenelle Fatman Oy:n ohjelmistokehittäjistä se oltaisiin annettu. Saman tyyppisiä konfiguraatioita on käytetty yrityksen ohjelmistoissa laajasti ja tapoja tehtävän ratkaisuun ei lopulta ole kovin montaa.

3.2 Uusi tila-käyttäjätarina

Seuraavaksi käydään läpi kuinka uuden tilan lisääminen itse ostotilaus prosessiin onnistui. Käyttäjätarina kuvaa jälleen lyhyesti nykytilanteen, tavoitellun tilanteen sekä hyväksymisperusteet sille. Tässä tapauksesa ostotilauksen tarkastaja roolille halutaan mahdollisuus palauttaa luotu ehdotus takaisin käyttäjälle, joka alunperin loi ehdotuksen. Tarkastaja haluaa tämän mahdollisuuden, jotta hän voisi pyytää lisätietoja tai muita korjauksia ehdotukseen sen tekijältä. Tarinan myötä halutaan prosessiin yksi uusi tila: ”Odottaa tarkastusta”. Käyttäjätarinassa on liitteenä kuva uudesta tilojen kulkujärjestyksestä. Hyväksymisperusteiden mukaan uuden ”odottaa tarkastusta”-tilan tulisi korvata ”avoin”-tila tilana, johon ostoehdotus ensimmäiseksi siirtyy. Prosessin ensimmäinen tila ei liitteen kuvasta ilmene.



Kuva 9. Tilamuutoskaavio

Lisävaatimuksina käyttäjätarinalla on, että ehdotuksen palattua ”Avoin”-tilaan ”Odottaa tarkastus”-tilasta, ehdotuksen alkuperäisen luojan tulisi olla ainut käyttäjä, joka pystyy muokkaamaan ehdotusta ja sen statusta. Palauttaminen vaatii myös pakollisen kommentin, joka tallennetaan ostotilauksen kommenttiosioon aikaleiman kera.

Liikkeelle lähdin tarkastamalla kuinka tilat ovat tallennettu tietokantaan. Kantaan on määritetty omiin tauluihinsa itse tilat (Status), tilojen välillä liikkuminen (StatusConfig) ja oikeudet tilojen muuttamiseen (StatusPermissions). Ensi askeleina lisäsin uuden ”Odottaa tarkastusta”-tilan Status-tietokantatauluun ja loin sille enumeraation lähdekoodiin. Enumeraatioilla tarkoitetaan tyyppiä, jonka arvot ovat tiedossa jo etukäteen. Lisäsin uuden tyypin ”WaitingForReview".

```
public enum PurchaseOrderStatus
{
    Open = 0,                   // Avoin
    Awaiting = 1,               // Odottaa hyväksyntää
    Approved = 2,               // Hyväksytty
    Received = 3,               // Vastaanotettu - irrelevantti
    Invoiced = 4,               // Laskutettu - irrelevantti
    Canceled = 5,               // Peruutettu
    WaitingForReview = 6        // Odottaa tarkastusta
}
```

Kuva 10. Enumeraatio-luokan lähdekoodia.

StatusConfig-tauluun määritetään tilojen välillä liikkuminen. Taulussa on kentät nykyiselle ja seuraavalle tilalle sekä bit-tyypin kentät sille vaatiiko tilanvaihdos kommenttia, sähköpostin lähetystä tai JavaScriptillä toteutettua vahvistus-dialogia. Nykyisen tilan arvona käytetään tässä tapauksessa enumeraatioon tallennettuakin arvoa 6 kun liikutaan ”Odottaa tarkastusta”-tilasta. Tilamuutoskaavion pohjalta uusia rivejä tauluun tulee siis lisätä neljä, koska uudesta ”Odottaa tarkastusta”-tilasta on päästävä ”Avoin”-, ”Odottaa hyväksyntää”- ja ”Peruutettu”-tiloihin ja ”Avoin”-tilasta on päästävä uuteen ”Odottaa tarkastusta”-tilaan. StatusPermissions-tauluun täytyy tallentaa StatusConfig-taulusta Id sekä roolin määrittävä id. ”Odottaa tarkastusta”-tilan tapauksessa vain tarkastaja roolin omaavan käyttäjän tulee pystyä aikaisemmin kuvattuihin tilasiirtoihin.

```
INSERT INTO StatusConfig
(PurchaseOrderStatusID, NextStatusID, CommentRequired, SendStatusEmail, ConfirmRe-
quired)
VALUES (6, 1, 0, 1, 0), (6, 0, 1, 0, 0), (0, 6, 0, 0, 0), (6, 5, 1, 0, 1)
```

Kuva 11. SQL-komento tila muutosten asettamiseen.

Yllä kuvattujen tietokantamuutosten myötä tila on mahdollista nähdä käyttöliittymän kautta ja siihen voi tarkastaja myös ostoehdotuksen siirtää tilasta ”Avoin”. Rivien lisääminen tietokantaan kävi helposti, koska kentät oli nimetty hyvin intuitiivisesti. Seuraavaksi täytyy lähteä muuttamaan lähdekoodia hieman enemmän ja hyödyntää jo aikaisemmin luotua enumeraatiota.

Ostotilaus-moduuli on hyvin suunniteltu etukäteen, joten uuden tilan lisääminen prosessiin ei ole kovin hankalaa. Valmiit funktiot löytyvät esimerkiksi tilojen perusteella ostotilauslistan suodattamiseen ja tilojen vaihtamiseen. Näiden hyödyntäminen periaatteessa vaati vain enumeraation lisäämistä jo niihin olemassa oleviin paikkoihin, joissa tilaa halutaan käyttää. Käytännössä ostotilauksen näkymämalli- ja sähköposti-ilmoituksia lähettävään-luokkaan tuli lisätä muutamia ehtolauseita uuden tilan erityisvaatimuksille.

Ostotilauksissakin käytetty näkymämalli-luokka (eng. View model) on malli-luokka, joka on tarkoitettu käytettäväksi suoraan näkymässä. Tavallista mallia käytetään enemmän tietokannassa datan tallentamiseen ja näkymämallia siihen miten mallin data näytetään itse käyttöliittymän näkymässä. Esimerkiksi malliin ja näin myös tietokantaan voidaan tallentaa käyttäjälle etu- ja sukunimi erikseen ja näkymämallissa voidaan samoja arvoja käyttää yhdessä yhtenä nimi arvona. (C# Corner 2018)

Erityisvaatimuksena uudelle tilalle oli esimerkiksi se, että sen ollessa ”Odottaa tarkastusta” vain käyttäjä, joka on asettanut itsensä ostoehdotuksen käsittelijäksi pystyy vaihtamaan ehdotuksen tilaa. Tilojen vaihtamisen estoa varten ostotilaus-luokalla oli jo olemassa ”ChangeStatusDisabled”-ominaisuus, jonka arvo tulee muuttaa todeksi kun aikaisemmin mainitut ehdot täyttyvät. Voimme käyttää hyödyksi Frame-sovelluksen SessionTool-kirjastoa, jonka avulla saamme selvitettyä tämänhetkisen sisäänkirjautuneen käyttäjän tunnistearvon. Tunnistearvoa voi verrata ostoehdotukseen tallennettuun käsittelijän vastaavaan tunnisteeseen. Ehtolause havainnollistettu kuvassa 12.

```
// Only handler should be able to edit status changes from
// 'waiting for review '
if (PurchaseOrderInstance.DN_PurchaseOrderStatus_ID ==
    (int) Enumerations.PurchaseOrderStatus.WaitingForReview
    && PurchaseOrderInstance.HandlerUserID != SessionTool.GetUser().UserI
D)
{
    PurchaseOrderInstance.ChangeStatusDisabled = true;
}
```

Kuva 12. Ehtolause näkymämalli-luokassa

Kokonaisuudessaan tämäkin käyttäjätarina oli melko suoraviivainen. Heti kun tilojen tietokantarakenne selvisi, uuden tilan lisäys kävi helposti. Tietokantalisäysten jälkeen vaadittiin muutamia muutoksia edellä mainittuun näkymämalliin. Suurin osa halutuista toiminnallisuuksisa, kuten tila muutosten disablointi, oli jo olemassa näkymämallissa, joten tehtäväkseni jäi vain käyttää jo olemassa olevia ominaisuuksia oikeissa paikoissa.

3.3 Ostotilausnumero-käyttäjätarina

Seuraavana esimerkkinä tutustutaan ostotilausnumeroon liittyvään käyttäjätarinaan. Käyttäjätarinassa tarkastaja-roolin omaava henkilö haluaa jokaiselle ostotilaukselle uniikin numeron, jotta hän voi tunnistaa tilaukset toisistaan, sekä hakea niitä ostotilauslistasta kyseisen tunnistenumeron perusteella.

Lähtötilanteessa ostotilaukselle tulee uniikki numero integraatiosovelluksen kautta. Kyseinen numero saadaan suoritettaessa tietokantahaku ostotilaus-tietokantatauluun ja liittämällä se integraatiotauluun, jossa integraationumero sijaitsee, käyttämällä SQL:n ”LEFT JOIN”-toimintoa. Haluttuna numerona ei ole mahdollista käyttää ostotilaus-tietokantataulun uniikkia tunniste-kenttää (eng. auto increment), koska tulevat asiakkaat haluvat mahdollisesti tuoda aikaisemmista järjestelmistään vastaavia ostotilauksia Frame-sovellukseen.

Sovellus vaatii siis uuden tietokantakentän uniikille numerolle. Vaikka puhutaan numeroista päädyn lisäämään tietokantaan NVARCHAR-tyypin kentän ”PurchaseOrderNumber”, johon soveltuu numeroiden lisäksi niin kirjaimet kuin erikoismerkitkin. Päädyn ratkaisuun, koska vastaavan tyyppisissä tilanteissa muualla Frame-sovelluksessa on toimittu samaan tapaan. On hyvin mahdollista, että kun asiakas tuo omia ostotilauksiaan Fatman Oy:n järjestelmään, heidän vastaava tilausnumeronsa sisältää kirjaimia.

Lähdekoodiin muutoksia vaaditaan DataServices-puolelle, jossa tietokantakyselyt suoritetaan. Jälleen hyödynnetään kappaleessa 3.1 luotua konfiguraatiota, jotta yritys X:n toiminnallisuus pysyisi ennallaan. Tietokantakyselyssä integraatiotaullun tehtävä ”LEFT JOIN”-piilotetaan konfiguraatiolla ja korvataan juuri lisätyllä ”PurchaseOrderNumber”-kentällä. Sekä DataServices-projektin että Frame-projektin ostotilausmalliin lisätään uusi ”string”-tyypin kenttä säilyttämään uusi tunniste.

Siirrytään käsittelijä-luokkaan, jossa ostotilauksen malli alustetaan tallennusta varten. Käsittelijässä DataServices- ja Frame-ostotilausmallit kartoitetaan (eng. map) vastaamaan toisiaan, jotta kyseisen olion tallennus onnistuisi DataServices-puolella mutkattomasti. Kartoitusta tehtäessä kyseiseen malliin on alustettava muutama välttämätön tieto: luontiaika, ehdotuksen luoneen käyttäjän tunniste sekä nyt lisättävä uusi kenttä ”PurchaseOrderNumber”. Ehdotuksen luoneen käyttäjän hakemiseen käytetään Frame-sovelluksen SessionTool-kirjastoa. Ostotilausmallin tunnistearvon (purchaseOrder.ID) ollessa 0, kyseessä on uusi ostoehdotus. Alustus havainnollistetu kuvassa 13.

```
if (purchaseOrder.ID == 0)
{
    //Set required data for new purchase orders
    purchaseOrder.CreatedTime = DateTime.Now;
    purchaseOrder.CreatedByUserID = SessionTool.GetUser().UserID;
    purchaseOrder.PurchaseOrderNumber = GeneratePurchaseOrderNumberString
();
}
```

Kuva 13. Lähdekoodia käsittelijä-luokasta

Käyttäjätarinan mukaan uuteen kenttään tallennettava tieto on generoitava uniikiksi, joten luon sitä varten uuden funktion”GeneratePurchaseOrderNumberString”. Funktion lähdekoodi kuvattuna kuvassa 14.

```
private string GeneratePurchaseOrderNumberString()
{
        var latestId = _purchasesService.GetLatestPurchaseOrderId(
                SessionTool.GetSessionID());
       var newPurchaseOrderNumber = latestId + 1;
       return newPurchaseOrderNumber.ToString();
}
```

Kuva 14. GeneratePurchaseOrderNumberString-funktio käsittelijä-luokassa.

Funktio, jota päädyn käyttämään on yksinkertainen. Se hakee kannasta viimeisimmän lisätyn ostotilauksen tunnistekentän ja lisää siihen numeron 1 ja palauttaa tämän merkkijonona (eng. string). Viimeisimmän tunnistekentän arvo haetaan DataServices-funktiota kutsuen. Kutsu kulkee IPurchasesService-rajapinnan (eng. interface) läpi itse PurchasesService-luokkaan. Rajapinnan osuus havainnollistettuna kuvassa 15.

```
public int GetLatestPurchaseOrderId(string sessionId)
{
    try
    {
        var database = new CallerData(sessionId).Database;
        return PurchasesFunctions.GetLatestPurchaseOrderId(database);
    }
    catch (Exception e)
    {
        PurchasesErrorHandler.handleError(e, sessionId);
        return 0;
    }
}
```

Kuva 15. IPurchasesService-luokan osuus

PurchasesService-luokan kautta päädytään PurchasesFunctions-luokkaan mikäli sinne selviydytään ilman virheitä (eng. exception). Vasta PurchasesFunctions-luokassa suoritetaan tietokantakysely helposti hyödyntäen kolmannen osapuolen PetaPoco-kirjastoa. Tämä nähtävissä kuvassa 16.

```
public static int GetLatestPurchaseOrderId(Database database)
{
    var countSql = Sql.Builder.Append(@"SELECT COUNT(id) FROM PurchaseOrd
ers");
    var purchaseOrderCount = database.Single<int>(countSql);

    if (purchaseOrderCount == 0)
    {
        return 0;
    }

    var sql = Sql.Builder.Append(@"SELECT TOP 1 id FROM PurchaseOrders OR
DER BY id DESC");
    return database.Single<int>(sql);
}
```

Kuva 16. PurchasesFunctions-luokan lähdekoodia.

Tietokantakyselyä tehtäessä täytyy tietenkin ottaa huomioon tilanne, jossa ostotilauksia ei vielä kannassa ole. Kyseisessä tilanteessa PetaPoco-kirjasto muodostaa virheen. Ratkaisuna tähän päädyn ensin tarkistamaan SQL:n ”COUNT”-toimintoa käyttäen onko tietokantataulussa rivejä ollenkaan. Jos näin on, palautetaan numeroarvo 0 potentiaalisen virheen sijaan. Käsittelijä-luokkaan palatessa arvoon lisätään numero 1.

Edellä läpikäyty käyttäjätarina oli yksinkertainen ja siihen toteutin myös yksinkertaisen ratkaisun. Luodessani tietokantakentästä merkkijono-kentän valmistauduin myös hieman tulevaan. Jos jatkossa uuteen tunnisteeseen halutaan sisällyttää myös kirjaimia tai muita merkkejä jonkin asiakkaan toimesta, vaaditaan ”GeneratePurchaseOrderNumberString”-funktioon konfiguraatio kyseiselle erikoistapaukselle. Työmääräarvion mukaisesti käyttäjätarinan suorittamiseen ei kulunut aikaa juuri paria tuntia enempää. Sen valmistuttua siirsin haaran versionhallintapalvelimelle ja jätin JIRA-järjestelmään tarkat ohjeet testaajalle toiminnallisuuden testaamiseksi. Toiminnallisuus läpäisi testauksen sekä katselmoinnin sujuvasti. Testausprosessia kuvailen tarkemmin vielä omassa kappaleessaan 4.

3.4 Yhteenveto

Käyttäjätarinoiden monipuolisuudesta johtuen yhtä isoa ratkaisua ostotilaus-moduulin paranteluun ei ollut. Keskeiseksi teemaksi nousi kuitenkin tietokantapohjaiset asetukset, joilla ominaisuuksia piilotettiin näkyvistä.

Moduulia muokatessa käyttäjätarinoiden ulkopuolelta löytyi myös paljon tekemistä. Lähtötilanteessa annoin yleisen kehitystietokannan käyttäjälle oikeudet itsenäiseen ostotilaus-moduuliin. Pelkästään moduulin pääsivuna toimivan ostotilauslistan avaaminen aiheutti .NET-kehyksen virhesivun aukeamisen. Sama virhesivu tuli hyvin tutuksi ympäri moduulia. Yritys X:n testitietokannassa moduuli toimi hienosti, mutta integraatio-tietokantataulujen puuttuessa muista tietokannoista, joutui asiaa asiaa selvittämään. Pelkästään se, että kehitystietokannassa sain luotua uuden ostoehdotuksen saman tapaan kuin yritys X:n vastaavassa, vaati muutaman päivän työn. Työ oli pitkälti Visual Studion debuggerin kanssa työskentelyä. Sovelsin paljon aikaisemmin kappaleessa 3.1 esiteltyä konfiguraatiota muun muassa tietokantaan suoritettavien SQL-kyselyiden kanssa. Tutkin kyselyitä ja piilotin konfiguraation taakse esimerkiksi SQL:n ”JOIN”-toimintoja, mikäli liitettävä taulu liittyi ainoastaan yritys X:n integraatiosovellukseen. Tapauksissa, joissa tauluja kyselyissä lähdetaulu yhdistettiin integraation tietokantatauluun ja kyselyn vastaanottama data oli tarpeellista myös integraatio kontekstin ulkopuolella, loin joko kokonaan uusia tietokantatauluja tai vain sarakkeita jo valmiisiin tauluihin. Uusi sarake tuli tarpeen esimerkiksi kun hintariveille piti lisätä valuutta, joka aikaisemmin tuli integraatiosovellukseen liittyvästä taulusta.

Monet ratkaisuista käyttäjätarinoihin löytyivät joko käyttämällä jo aiemmin yrityksessä työskennellessä saatuja oppeja tai etsimällä lähdekoodista kuinka samankaltaisia ongelmia on ratkottu sovelluksen muissa moduuleissa. Frame-sovellus on hyvin laaja, joten hieman lähdekoodia tukimalla ratkaisuja on mahdollista löytää lähes mihin tahansa. Ratkaisuun voi myös tietenkin kysyä apua kollegoilta.

Valmiiden käyttäjätarinoiden lisäksi refaktorointia en lopulta juuri tehnyt. Se mitä projektin tiimoilta tein, rajoittui lähinnä muuttujien uudelleen nimeämiseen ja toistuvien lähdekoodin pätkien siirtämiseen omiin funktioihinsa. Jatkossa refaktoroinnille on varmasti varaa.

4 Testaus

Fatman Oy:ssä toiminnallisuudet testataan Scrum-viitekehykselle tyypilliseen tapaan. Kun ominaisuus ensiksi valmistuu omassa versionhallintahaarassaan, se siirtyy seuraavaksi testaajien testattavaksi paikallisessa haarassa. Testaajien vastuulle jää itse toiminnallisuuden testaaminen. Testaajat eivät ota lainkaan kantaa lähdekoodiin, eivätkä he sitä välttämättä ymmärräkään samalla tasolla ohjelmistokehittäjien kanssa. Testaajat käyvät toiminnallisuuden läpi joko käsin käyttöliittymän kautta tai käyttäen automaatiotestejä. Ostotilaus-moduulin testauksen kohdalla automaatiotestejä ei ennalta ole olemassa ja sen myötä moduulin testaus oli manuaalista. Jokainen käyttäjätarina testataan erikseen niin että haluttu toiminnallisuus todella toimii. Samalla myös tarkistetaan paikat, joihin toiminnallisuuden vaatimat muutokset mahdollisesti vaikuttavat. Mikäli testatessa ilmaantuu ongelmia, viallinen haara palautetaan takaisin alkuperäiselle ohjelmistokehittäjälle.

Ensimmäisen testauskierroksen läpäistessään toiminnallisuus siirtyy toisen ohjelmistokehittäjän katselmoitavaksi. Katselmoiva ohjelmistokehittäjä ei ole saanut osallistua ominaisuuden kehitykseen, jotta suoritettava katselmointi olisi mahdollisimman puolueeton. Katselmoija tutustuu lähdekoodiin ja varmistaa, että se ei sisällä ohjelmointivirheitä. Mikäli virheitä ilmaantuu, ominaisuus palautetaan takaisin sen tekijälle. Katselmoija ottaa myös kantaa lähdekoodin yksityiskohtiin ja sen puhtauteen. Puhtaasta lähdekoodista puhuttaessa tarkoitetaan lähdekoodia, jossa esimerkiksi muuttujien ja funktioiden nimeäminen on järkevää ja jossa funktiot suorittavat vain yhden asian ollen samanaikaisesti myös järkevän pituisia. (Clean Code 2008)

Mikäli käyttäjätarina läpäisee katselmoinnin se siiretään JIRA:ssa takaisin alkuperäiselle ohjelmistokehittäjälle. Kehittäjän vastuulle jää siirtää toiminnallisuus omasta haarastaan haaraan, joka päivitetään demopalvelimelle kahden viikon välein. Kun toiminnallisuus on ollut demopalvelimella viikon ajan, se siirretään tuotantopalvelimelle asiakkaiden käyttöön. Kuluvan viikon aikana demoa testataan testaajien toimesta, jotta voidaan varmistua siitä että tuotantoon siirtyessään uudet aikaisemmin demolle lisätyt toiminnallisuudet eivät tule aiheuttamaan siellä mitään ongelmia. Välillä inhimillisiä virheitä sattuu, mutta ostotilaus-moduulin kanssa pahimmilta virheiltä säästyttiin.

Projektin testausvaiheessa sain myös ohjelmistokehittäjänä auttaa testaajia. Toisinaan liiketoimintalogiikka ei käynyt suoraan selväksi käyttäjätarinoista. Näin oli esimerkiksi uuden ”Odottaa tarkastusta”-tilan lisäyksen kanssa sillä se sisälsi melko paljon muutoksia. Toiminnallisuuden selvittäminen testaajalle vaati yhteistyötä testaajan työpisteellä.

Kävimme yhdessä läpi käyttäjätarinan hyväksymisperusteet, jotta olisimme samoilla linjoilla asian kanssa. Tämän jälkeen seurasin kuinka testaaja suoritti toiminnallisuuden testaamisen ja annoin tarpeen vaatiessa neuvoja.

5 Pohdinta

Opinnäytetyöprojekti onnistui mielestäni melko hyvin. Kaiken kaikkiaan tekemieni muutosten jälkeen koen ostotilaus-moduulin sopivan paremmin useammille asiakkaille. Riippuvuus yritys X:n käyttämästä integraatiosovelluksesta on projektin myötä poistettu. Uskon valmistumisen tuottavan varmasti selkeää rahallista arvoa Fatman Oy:lle. Uskon Fatman Oy:n myyjien pystyvän myymään ostotilaus-moduulia tulevaisuudessa muiden joukossa helpommin niin nykyisille kuin tulevillekin asiakkaille. Moduuli tulee silti todennäköisesti tarvitsemaan jatkokehitystä tulevaisuudessa. Asiakkailla on tapana pyytää hyvin yksityiskohtaisia ominaisuuksia omia tarpeitaan varten.

Parhaiten projektissa mielestäni onnistui itse ohjelmointi. Käyttäjätarinoita ja muita ongelmia projektin aikana oli useita, mutta onnistuin selvittämään ne. Lähes kaikki ratkaisut kehittelin itse, eikä muita yrityksen ohjelmistokehittäjiä tarvinnut projektin tiimoilta juurikaan häiritä. Vaikeinta opinnäytetyössä oli ehdottomasti itse tekstin tuottaminen. Koen myös, että lähdekoodia olisin voinut refaktoroida enemmän työn aikana. Aikarajoitteet vaikuttivat refaktoroinnin vähemmälle huomiolle jättämiseen.

Opinnäytetyön valmistumisen myötä opin lisää Fatman Oy:n sovelluksista ja ehkä hieman tavallista isompien tehtäväkokonaisuuksien lähestymisestä. Uskon, että ensin liiketoimintalogiikkaan tutustuminen ja sen pohjalta käyttäjätarinoihin paneutuminen oli järkevintä. Ohjelmoinnin näkökulmasta projektin myötä en juuri oppinut lisää uusia toimintatapoja. Koen kuitenkin kehittyneeni esimerkiksi .NET Frameworkin LINQ (Language Integrated Query) -komponentin käytössä. Se on ehdottomasti yksi suosikki ominaisuuksistani .NET-sovelluskehyksessä. LINQ:n avulla lähdekoodista saa helposti mielestäni selkeämpää ja helpommin luettavaa. LINQ:n käyttö poistaa myös useissa tilanteissa tarpeen ylenpalttisille ehtolauseille.

Lähteet

Agiledata 2018. Relational databases 101. Luettavissa: http://www.agiledata.org/essays/relationalDatabases.html. Luettu: 20.11.2018.

C# Corner 2018. What is Model and ViewModel in MVC pattern? Luettavissa: https://www.c-sharpcorner.com/UploadFile/abhikumarvatsa/what-is-model-and-viewmodel-in-mvc-pattern/. Luettu: 26.11.2018.

Git SCM 2009. Pro Git. Luettavissa: https://git-scm.com/book/fi/v1/Alkusanat-Versionhallinnasta. Luettu: 11.11.2018.

Martin, R. C. 2008. Clean Code: A Handbook of Agile Software Craftmanship. Prentice Hall.

Microsoft docs 2018. C# Guide. Luettavissa: : https://docs.microsoft.com/en-us/dotnet/csharp/. Luettu: 10.11.2018.

Scrum 2018. What is scrum? Luettavissa: https://www.scrum.org/resources/what-is-scrum. Luettu: 9.11.2018.

Service Architecture 2018. Service-Oriented Architecture (SOA) Definition. Luettavissa: https://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html. Luettu: 27.11.2018

W3Schools 2018. AJAX Introduction. Luettavissa: https://www.w3schools.com/xml/ajax_intro.asp. Luettu: 24.11.2018.

Wikipedia 2018. Javascript. Luettavissa: https://en.wikipedia.org/wiki/JavaScript. Luettu: 8.11.2018.