TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JANI TARMILA

B2B-VERKKOKAUPAN UUDISTAMINEN FUNKTIONAALISEN OHJELMOINNIN MENETELMIN

Diplomityö

Tarkastaja: professori Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan
tiedekuntaneuvoston
kokouksessa 5. toukokuuta 2014

TIIVISTELMÄ

JANI TARMILA: B2B-verkkokaupan uudistaminen funktionaalisen ohjelmoinnin menetelmin
Tampereen teknillinen yliopisto
Diplomityö, 45 sivua
Kesäkuu 2015
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen

Avainsanat: verkkokauppa, funktionaalinen ohjelmointi, web-sovelluksen uudistaminen

Verkossa tapahtuva kaupankäynti lisääntyy jatkuvasti ja myös yritysten välisen sähköisen tiedonsiirron merkitys on korostunut viime vuosina. Sähköinen kaupankäynti helpottaa toimintaa nopeuttamalla tiedonsiirron nopeutta ja vähentää inhimillisten virheiden mahdollisuutta. Yritysmaailmassa kaupankäynnin periaatteet poikkeavat kuitenkin kuluttajien välisestä kaupasta, mikä tulee huomioida myös toiminnan siirtyessä verkkoon.

Tässä diplomityössä käsitellään asiakasprojektina toteutettua B2B-verkkokaupan uudistusta. Uudistuksen lähtökohtana oli vanha verkkokauppa, jonka huono yhteensopivuus modernien selainten kanssa ja hankala käytettävyys muodostivat tärkeimmät kehityskohteet. Samalla haluttiin myös tarjota verkkokaupan käyttäjille uusia ominaisuuksia ja paremmat mahdollisuudet tehtyjen tilausten seurantaan.

Uudistus toteutettiin funktionaalista ohjelmointia tukevan ohjelmointikielen ja siihen liittyvien työkalujen avulla. Tässä työssä käydään läpi verkkokaupan perusajatukset, esitellään funktionaalista ohjelmointia yleisesti ja tarkastellaan verkkokauppauudistuksen etenemistä niin suunnittelun ja toteutusteknologioiden valinnan kuin toteutuksenkin kannalta. Lisäksi käsitellään valittujen teknologioiden toimivuutta ja uudistuksen tuloksena saavutettuja hyötyjä.

Uudistuksessa käytetyt toteutusteknologiat ja valitut työkalut vastasivat asetettuja tarpeita ja toimivat hyvin pieniä puutteita lukuun ottamatta. Projektin tuloksena syntynyt uusi verkkokauppa on aiempaa toimivampi moderneilla selaimilla ja uudet ominaisuudet parantavat tiedonkulkua käyttäjien suuntaan. Jatkossa myös verkkokaupan ylläpito on entistä helpompaa.

ABSTRACT

JANI TARMILA: Renewal of a B2B Online Shop Using Functional Programming
Tampere University of Technology
Master of Science Thesis, 45 pages
June 2015
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Professor Tommi Mikkonen

Keywords: online shopping, functional programming, web application renewal

Web based commerce is increasing constantly and the meaning of electronic data transfer between companies has also become more important in the last few years. Electronic commerce makes business easier by increasing data transfer speed and lessens the possibility of human errors. Business-to-business commerce differs from the retail commerce, which needs to be taken into account also when the business moves online.

This Master's thesis describes a customer project which implements the renewal of a B2B online shop. The base for the renewal was an old online shop, which had bad compatibility with modern browsers and also poor usability. These were the main focal points for the renewal. Along with the renewal the customer wanted to provide new features and better order tracking for the users of the online shop.

The renewal was implemented with a programming language that supports functional programming and related tools. This thesis handles the basic ideas of online shopping, gives a general view on functional programming and examines the progress of the renewal through planning, technology choices and the actual development work. In addition the thesis describes the operability of the chosen technologies and the advantages provided by the renewal.

The technologies and tools chosen for this renewal corresponded with the set needs and worked well except for a few exceptions. The new online shop that was created works better on modern browsers and the new features improve the flow of information toward the users. In the future the maintenance of the online shop is also easier than before.

ALKUSANAT

Tämän pitkällisen ajatusprosessin ja hieman lyhyemmän kirjoitusajan kuluessa syntynyt diplomityö syntyi Bitwise Oy:n toteuttaman verkkokaupan uudistusprojektin tuloksena. Haluan kiittää kaikkia projektin puitteissa töitä tehneitä henkilöitä ja erityisesti Bitwisen toimitusjohtajaa Tomi Mikkosta mahdollisuudesta käyttää työaikaa tämän työn kirjoittamiseen.

Lisäksi haluan kiittää asiakkaana toiminutta Adara Pakkaus Oy:tä luvasta tehdä diplomityöni tähän projektiin liittyen.

Erityiskiitokset ansaitsee työn tarkastaja professori Tommi Mikkonen, jonka rakentavien kommenttien ja palautteen perusteella työ saatiin maaliin kireästä aikataulusta huolimatta.

Lopuksi esitän kiitokset perheelleni ymmärryksestä ajankäytön suhteen ja tuesta diplomityön tekemisen kuluessa.

Tampereella, 18.5.2015

Jani Tarmila

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

ACID	Tietokantajärjestelmien periaate, joka turvaa tiedot atomisuuden, oikeellisuuden, eristyvyyden ja pysyvyyden kautta (Atomicity, Consistency, Isolation, Durability).
ASP.NET	Microsoftin kehittämä web-ohjelmistokehys, jonka mahdollistaa dynaamisten web-palveluiden rakentamisen.
B2B-kaupankäynti	Yritysten välinen kaupankäynti, jonka avulla yritykset hankkivat tarvitsemiaan tuotantohyödykkeitä (Business-to-business).
B2C-kaupankäynti	Asiakkaan ja yrityksen välinen kaupankäynti, jossa asiakas hankkii yritykseltä tarvitsemiaan tuotteita tai palveluita (Business-to-consumer).
BVP	Business VantagePoint, asiakkaan käyttämä toiminnanohjausjärjestelmä.
CSS	Tyylitiedostojen kuvauskieli, jolla määritellään verkkosivujen ulkoasu (Cascading Style Sheets).
CVP	Customer VantagePoint, asiakkaan aiempi verkkokauppasovellus.
ERP-järjestelmä	Tietojärjestelmä, jonka toiminnallisuus kattaa yrityksen toiminnan kaikki osa-alueet ostoista laskutukseen (Enterprise Resource Planning).
HTML	Kuvauskieli, jota käytetään verkkosivujen rakenteen kuvailuun (Hypertext Markup Language).
JDBC	Java-ohjelmointikielen rajapinta, jonka avulla voidaan käyttää relaatiotietokantaa (Java Database Connectivity).
jOOQ	Java-ohjelmointikielellä toteutettu kirjasto tietokantojen käsittelyyn (Java Object Oriented Querying).
JSON	Avoin tiedostomuoto datan siirtämiseen (JavaScript Object Notation).
MVC	Suunnittelumalli, jonka mukaisesti ohjelma jaetaan malliin, näkymään ja ohjaimeen (Model-View-Controller).
OMP	Asiakkaan käyttämä tuotannonohjausjärjestelmä.
QR-koodi	Kaksiulotteinen ruutukoodi, johon voidaan tallentaa tekstimuotoista tietoa (Quick Response).
REST	Arkkitehtuurimalli rajapintojen toteuttamiseen, perustuu HTTP-protokollaan (Representational State Transfer).
sbt	Build-työkalu Scala-sovellusten kääntämisen automatisointiin (Scala Build Tool).

1. JOHDANTO

Kuluttajien kaupankäynti verkossa yleistyy koko ajan [1] ja toisaalta myös yritykset ovat vähitellen siirtämässä keskinäistä kauppatoimintaansa verkkoon muun muassa erilaisten kustannushyötyjen vuoksi. Yritysten keskinäinen kaupankäynti noudattaa kuitenkin erilaisia periaatteita kuin kuluttajakauppa, mikä tulee ottaa erityisesti huomioon yritysten siirtyessä verkkokaupankäyntiin. Yritysmaailmassa toimintaympäristöt ovat suljettuja, mikä muuttaa toiminnan luonnetta poistamalla hintakilpailun merkityksen [2]. Hinnan sijasta yritysverkkokaupat kilpailevat ominaisuuksillaan, ajantasaisella tiedolla ja käytön helppoudella. Yritysten välisessä kaupassa verkkokaupat ovat enemmänkin työkaluja, joten niiden on oltava tehokkaita ja yksinkertaisia.

Tämän diplomityön tarkoituksena on kuvata verkkokaupan toteutusprojektin etenemistä ja tarkastella projektissa käytettyjä menetelmiä. Työ perustuu Bitwise Oy:n Adara Pakkaus Oy:lle toteuttamaan B2B-verkkokaupan uudistusprojektiin, jossa aiemmin toteutetun verkkokauppaympäristön tilalle toteutettiin uusi asiakkaan tarpeisiin räätälöity verkkokauppa. Diplomityö esittelee projektin etenemisen lähtökohdista määrittelyn ja suunnittelun kautta toteutukseen. Toteutuksen jälkeen pohditaan vielä käytettyjen menetelmien toimivuutta ja saavutettuja hyötyjä.

Projektin toteutuksen ja tämän diplomityön taustalla on ajatus funktionaalista ohjelmointia tukevien teknologioiden valinnasta. Funktionaalisuuden avulla voidaan parantaa verkkokaupan luotettavuutta ja ylläpidettävyyttä, mutta myös vähentää tarvittavan ohjelmakoodin määrää. Funktionaalisuuteen siirtyminen on kuitenkin oma haasteensa, sillä sen syvällinen ymmärtäminen vaatii funktionaalisten toimintatapojen omaksumista. Funktionaalisuuden kasvava suosio kannustaa kuitenkin käyttämään aikaa uusien taitojen opettelemiseen ja itsensä kehittämiseen.

Tämän diplomityön luvussa 2 esitellään verkkokauppojen yleisiä erityispiirteitä, kuvaillaan asiakkaan nykyinen järjestelmä ja käydään läpi uudistusprojektin lähtökohdat kehitystoiveiden ja käyttötapauskaavioiden avulla. Luku 3 tarjoaa teoriapohjan funktionaalisen ohjelmoinnin historiaan, periaatteisiin ja hyötyihin sekä haittoihin. Luvussa 4 käsitellään verkkokaupan suunnitteluun liittyviä näkökohtia lähtien erityistarpeista ja arkkitehtuurista sekä toteutusteknologioiden valinnasta. Luku 5 kuvailee projektiin valitut toteutusmenetelmät, käytetyt työkalut ja käyttöliittymän erikoisratkaisut. Luvussa 6 pohditaan valittujen teknologioiden ja menetelmien sopivuutta, työkalujen toimivuutta, projektin myötä saavutettuja hyötyjä ja muutamia jatkokehitysajatuksia.

2. LÄHTÖKOHDAT

Kaupankäynti verkossa on nopeaa ja se mahdollistaa ajantasaisen tiedon välittämisen. Kuluttajapuolella verkkokauppojen yleistyminen on lisännyt erityisesti hintakilpailua, mutta yritysmaailmassa tärkeämpää on nopea tiedonkulku. Lisääntyvässä määrin tärkeää on myös verkkokaupan käytön mahdollistaminen erilaisilla laitteilla.

Tässä luvussa esitellään yleisiä verkkokauppoihin liittyviä piirteitä sekä vertaillaan kuluttajaverkkokaupan ja yritysten välisen verkkokaupan eroja. Lisäksi esitellään asiakkaan alkuperäinen verkkokauppatoteutus ja käsitellään asiakkaan esittämiä uuteen verkkokauppaan liittyviä kehitystoiveita. Luvun lopussa esitellään verkkokaupan vaatimusten perusteella määritellyt käyttötapauskaaviot ja käydään lyhyesti läpi eri käyttötapaukset.

2.1 Verkkokauppojen erityispiirteet

Verkkokauppojen historia alkaa jo 1970-luvun lopulta [1], mutta niiden varsinainen yleistyminen alkoi vasta 1990-luvun puolivälissä World Wide Webin keksimisen myötä. Nykypäivänä suositut Amazon.com (http://www.amazon.com/) ja eBay (http://www.ebay.com/) toimivat alan pioneereina ja alkoivat myydä kuluttajille erilaisia tuotteita verkon välityksellä [3]. Varsinainen verkkokauppojen läpimurto on tosin tapahtunut vasta viime vuosina. Verkkokauppojen käytöstä on vähitellen tulossa osa arkipäivää, ja niiden tuotevalikoima on monipuolistunut vuosien varrella huomattavasti.

Lähtökohtaisesti verkkokaupat tarjoavat asiakkailleen mahdollisuuden hankkia tarjolla olevia tuotteita juuri asiakkaan haluamalla hetkellä ilman, että asiakkaan tarvitsee lähteä erikseen tarkastelemaan tuotevalikoimaa myymälään tai ottaa yhteyttä asiakaspalveluun tiettynä aikana tilauksen tekemiseksi [2]. Tämä mahdollistaa sen, että asiakas ei ole enää perinteisen kaupankäynnin tavoin sidottu aikaan ja paikkaan, vaan ostosten tekeminen onnistuu esimerkiksi keskellä yötä asiakkaan niin halutessa.

Verkkokaupoissa asiakkaalla on usein käytössään todellisen maailman metaforana toimiva ostoskori tai -kärry, johon tuotteet kerätään. Keräily tapahtuu erilaisten hakukoneominaisuuksien ja ehdotusten perusteella. Myyjä saattaa ostopäätöksiä vauhdittaakseen kerätä tietoja asiakkaan katselemista tuotteista ja aiemmista ostoista ja ehdottaa tätä kautta tuotteita, jotka saattaisivat olla asiakkaan kannalta mieluisia [2]. Kun asiakas on löytänyt haluamansa tuotteet, hän siirtyy maksamaan tuotteet kassalle. Maksuvälineinä toimivat verkkokaupasta riippuen esimerkiksi luotto- ja pankkikortit, lahjakortit, laskutus

tai vaikkapa Bitcoinin kaltaiset kryptovaluutat [4]. Suomessa yleinen maksutapa on verkkopankin kautta pankkitunnuksilla tehtävä maksu [5].

Kun tuotteet on maksettu, myyjä toimittaa ne asiakkaalle asiakkaan valitsemalla toimitustavalla. Toimitustapavaihtoehdot ovat kauppiaan päätettävissä, mutta useimmiten niitä on asiakkaan palvelemiseksi erilaisia. Tuotteet voidaan toimittaa esimerkiksi postitse suoraan asiakaan kotiin, pakettina postiin tai pakettiautomaattiin tai asiakas voi mahdollisesti noutaa tilaamansa tuotteet itse jostakin myyjän palvelupisteestä ja säästää näin postikuluissa [2]. Joidenkin tuotteiden, kuten pääsylippujen ja pelien tapauksessa myyjä voi lähettää tilauksen myös digitaalisessa muodossa, jolloin asiakas saa tilaamansa tuotteet lähes välittömästi tilaamisen jälkeen.

Kauppiaan kannalta verkkokauppa tuottaa kustannussäästöjä muun muassa toimitilavuokrissa ja markkinointikustannuksissa [2]. Lisäksi verkkokauppa laajentaa asiakaspotentiaalia, koska siihen ei liity toimitilojen sijainnista aiheutuvaa paikkarajoitetta. Asiakas puolestaan hyötyy verkkokauppaa käyttämällä valikoiman monipuolisuudesta, hintojen helposta vertailusta, muiden käyttäjien kirjoittamista arvioista ja siitä, että ostoksia voi tehdä suoraan kotoa haluamanaan aikana.

Verkkokauppaan liittyy etujen lisäksi myös ongelmia. Kaupankäynnissä tilaukset ja rahat liikkuvat digitaalisessa muodossa, joten asiakkaan on oltava tarkkana turvallisuuden suhteen. Koska myynnissä olevaa tuotetta ei pääse tarkastamaan fyysisesti ennen kaupantekoa, asiakkaan on luotettava myyjään tehdessään tilausta. Verkkokaupan perustamisen helppous mahdollistaa sen, että myyjien joukosta löytyy myös epärehellisiä toimijoita. Mikäli myyjä ei ole salannut verkkokaupan yhteyttä tai verkkokaupan suojaukset on toteutettu turvattomasti, voivat rikolliset päästä käsiksi asiakkaiden maksuvälinetietoihin ja käyttää niitä haluamallaan tavalla [1].

B2B-verkkokauppa eroaa B2C-verkkokaupasta monella eri tavalla, vaikka verkkokaupan perustoimintojen osalta ajatusmaailma on sama. B2B-kaupassa verkkokaupan asiakkaalla on todennäköisesti jo pitempiaikainen asiakassuhde tuotteiden myyjään, joten markkinoinnilla ja tuotteiden esittelyllä ei ole verkkokaupan tasolla yhtä suurta merkitystä kuin B2C-kaupassa [2]. Asiakas hankkii myytäviä tuotteita oman tuotantoprosessinsa tarpeisiin ja jalostaa ne eteenpäin, joten transaktioiden määrä on suurempi ja vakiintuneempi kuin kuluttajakaupassa. Asiakassuhteen merkitys korostuu siltäkin osin, että yritykset ovat voimakkaammin sitoutuneet tiettyihin toimittajiin ja tunnetuista toimittajista poikkeaminen voi aiheuttaa yllättäviä kustannuksia, sillä hankintapäätökset on tehty etukäteen asiakkaan ostoprosessin mukaisesti.

Yritysasiakkaiden ostamat tuotteet ovat yleensä vain osajoukko myyvän yrityksen tuotevalikoimasta, ja ostaja todennäköisesti tuntee tuotteet jo entuudestaan, joten tarkkojen

tuotetietojen tarjoamisen merkitys on huomattavasti pienempi kuluttajakauppaan verrattuna. Tuotteiden valinnan taustalla on pidempi prosessi ja mahdollinen kilpailuttaminen, joten myös tarkat hinnat ovat asiakkaan tiedossa jo ennen tilauksen tekemistä [2]. Myös maksu- ja toimitustavat ovat aiemmin tehtyjen sopimusten mukaisia, joten niihin ei oteta yhtä voimakkaasti kantaa B2B-verkkokaupassa. Tämän lisäksi B2B-verkkokaupat ovat yleensä suljettuja ja yrityksen tiedot ovat liiketoiminnallisista syistä nähtävissä vasta kirjautumisen jälkeen.

B2B-verkkokauppa toimii ostajien työkaluna, joten sen käytön on oltava nopeaa, helppoa ja tehokasta. Toisaalta verkkokauppa helpottaa parhaimmillaan tilausten käsittelyä myös myyjän kannalta, koska tilaukset tulevat sähköistä kanavaa pitkin puhelinsoittojen ja sähköpostien sijasta, mikä puolestaan mahdollistaa tehokkaamman ja nopeamman tilauskäsittelyn [2]. Tilauksissa saattaa esiintyä toistuvuutta, jonka automatisoiminen voi tarjota ostajalle lisähyötyjä ja kannustaa verkkokaupan käyttöä. Yritysten ERP-järjestelmien integrointi keskenään on monissa tapauksissa haastavaa järjestelmäerojen vuoksi, joten verkkokauppa toimii tämän kaltaisissa tapauksissa yksinkertaisena keinona mahdollistaa tilausten nopea tekeminen järjestelmistä riippumatta [6]. Toisaalta verkkokauppa voi myös myyjän ja asiakkaan välimaastossa toimivana mahdollistaa tiedonvälityksen erilaisten toiminnanohjausjärjestelmien välillä.

Tuotantoon liittyvästä luonteesta johtuen B2B-verkkokauppojen toimituksilla on usein selkeästi tiukemmat aikataulut kuin kuluttajakaupassa ja tilauksen tilan merkitys korostuu enemmän [2]. Asiakas saattaa myös haluta määritellä tilaukselleen tietyn toimituspäivän ennakolta varmistuakseen tilauksen saapumisesta ajallaan, ja myyjällä saattaa olla toimitusaikoihin liittyviä sopimusehtoja. Osa asiakkaan tilaamista tuotteista voi myös olla varastoinnissa, jolloin asiakkaan tulee nähdä kyseisten tuotteiden varastosaldot voidakseen varmistua.

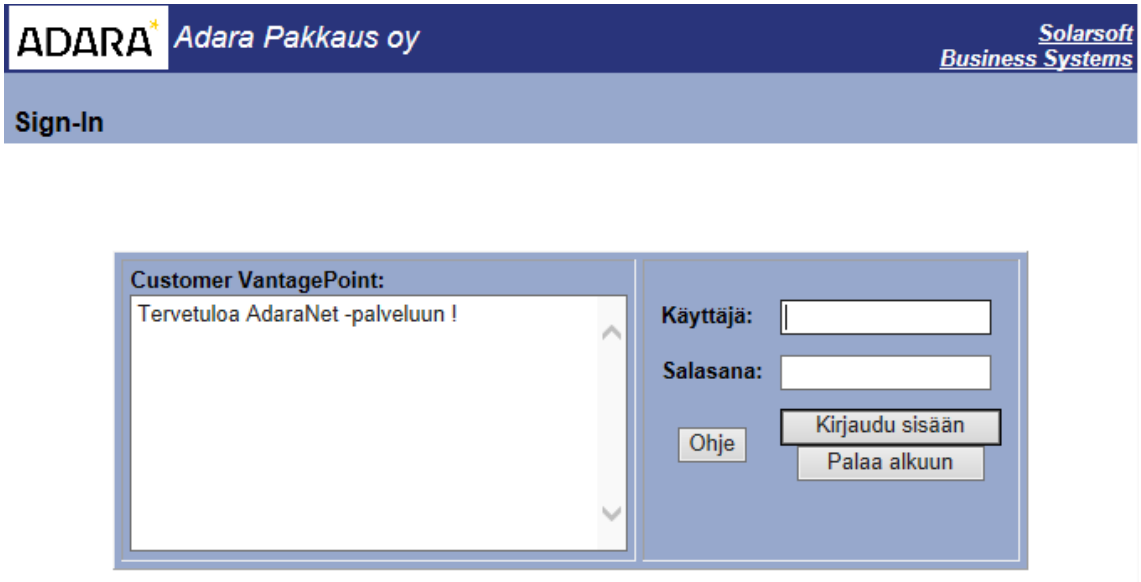
2.2 Asiakkaan alkuperäinen järjestelmä

Adara Pakkaus Oy:n aiemmin käyttämä B2B-verkkokauppasovellus on yhdysvaltalaisen Epicor Software Corporationin omistama Customer VantagePoint. CVP:n on alkujaan kehittänyt kanadalainen Solarsoft Business Systems, jonka Epicor osti lokakuussa 2012 [7]. Tässä yhteydessä myös CVP siirtyi Epicorin hallintaan. CVP on ASP.NET-sovellus, joka noutaa tarvittavan datan suoraan samaan tuotekokonaisuuteen kuuluvasta Business VantagePoint -tuotannonohjausjärjestelmästä. BVP vastaa asiakastietojen hallinnasta, tilauksista, varastonhallinnasta ja laskutuksesta. Tuotannonsuunnittelu tehdään OM Partnersin OMP-järjestelmässä. Tarkemmin järjestelmien väliseen kokonaisuuteen palataan kohdassa 4.2.

CVP on muiden Adaran järjestelmien tavoin suunniteltu aaltopahvivalmistajien käyttöön, joten se sisältää joitakin tavallisesta B2B-verkkokaupasta poikkeavia ominaisuuksia, kuten aaltopahviarkkien tilaustoiminnon. Järjestelmän perusominaisuuksiin kuuluu valmistetilausten tekeminen, tehtyjen tilausten tarkastelu, varastotilanteen seuranta, valmisteiden listaaminen, tarjouspyyntöjen tekeminen ja viestien lähettäminen asiakaspalveluun. CVP:ssä tilaukset lisätään tekemisen jälkeen ostoskoriin ja kun kaikki halutut tilaukset ovat ostoskorissa, ne voidaan lähettää asiakaspalvelun käsiteltäväksi.

Ostajat kirjautuvat CVP:hen omilla käyttäjätunnuksillaan. Ylläpitäjä voi liittää yksittäiseen käyttäjätunnukseen useampia asiakasnumeroita, jolloin käyttäjän on toimintoja käyttäessään valittava haluttu asiakasnumero joka kerta. Tämän lisäksi käyttäjätunnukselle on asetettu käyttäjäryhmä, jonka perusteella näytettäviä ominaisuuksia voidaan mukauttaa. Mukauttaminen tapahtuu näkymätasolla, joten vain kokonaisia sivuja on mahdollista näyttää tai piilottaa. Käyttäjähallinnan lisäksi ylläpitäjä voi muuttaa muutamia sivustolla näkyviä tiedoteviestejä ja määritellä kohteet, joihin käyttäjien lähettämät viestit ohjataan.

Kuvassa 2.1 nähtävä CVP on ulkoasultaan osittain vanhentunut ja hankalakäyttöinen. Käyttöliittymän suomennetut tekstit ovat joiltain osin epäloogisesti käännettyjä, mikä hankaloittaa käyttöä, koska käyttäjä joutuu pohtimaan mitä tekstit tarkoittavat. Myös lokalisointi on puutteellista, sillä esimerkiksi päivämäärät esitetään suomalaisia käyttäjiä ajatellen epäloogisessa formaatissa.



Kuva 2.1. *Customer VantagePointin sisäänkirjautumisnäkymä Internet Explorer 11:n yhteensopivuustilassa.*

2.3 Kehitystoiveet

Adaran verkkokaupan uudistus käynnistyi keväällä 2013 asiakkaan todettua, että aiempi verkkokauppatoteutus ei enää palvellut kaikkia heidän tarpeitaan. Eräs tärkeimmistä syistä uudistuksen aloittamiselle oli CVP:n huono selainyhteensopivuus modernien selainten kanssa, sillä CVP toimii vain Internet Explorer -selaimen versiolla 8 ja sitä vanhemmilla. Käytännössä uudemmissa Internet Explorerin versioissa saatavilla oleva yhteensopivuusnäkymä oli useissa tapauksissa Adaran asiakkaiden ainoa keino käyttää verkkokauppaa. Kaikissa tapauksissa Windows-ympäristöön rajattua Internet Exploreria ei kuitenkaan ollut ostajien käytettävissä, mikä vähensi verkkokaupan käyttäjäpotentiaalia.

Uudistusta lähestyttiin aluksi asiakkaan toiveiden kautta, sillä tarkoituksena ei ollut ottaa käyttöön mitään valmista verkkokauppa-alustaa, vaan toteuttaa täysin räätälöity kokonaisuus. Tällä tavoin voitiin varmistua siitä, että sovellusalakohtaisten erikoispiirteiden huomiointi ja olemassa olevien järjestelmien integrointi onnistuisivat mahdollisimman hyvin. Erityistä mietintää aiheutti verkkokaupan integroiminen käytössä olevaan ERP-järjestelmään, sillä verkkokaupan tilausten haluttiin siirtyvän automaattisesti puolin ja toisin. Uuden verkkokauppatoteutuksen työnimeksi valittiin Adara Shop.

Adaralle oli ensisijaisen tärkeää, että uudistetusta verkkokaupasta tulisi mahdollisimman helppokäyttöinen ja että tärkeimmät toimenpiteet olisi nopea tehdä. Verkkokaupan käyttö haluttiin mahdollistaa myös mobiililaitteilla, jotta käyttäjät voisivat tarvittaessa tehdä tilauksia kätevästi esimerkiksi tuotantotiloissa varastosaldojen tarkastelun yhteydessä.

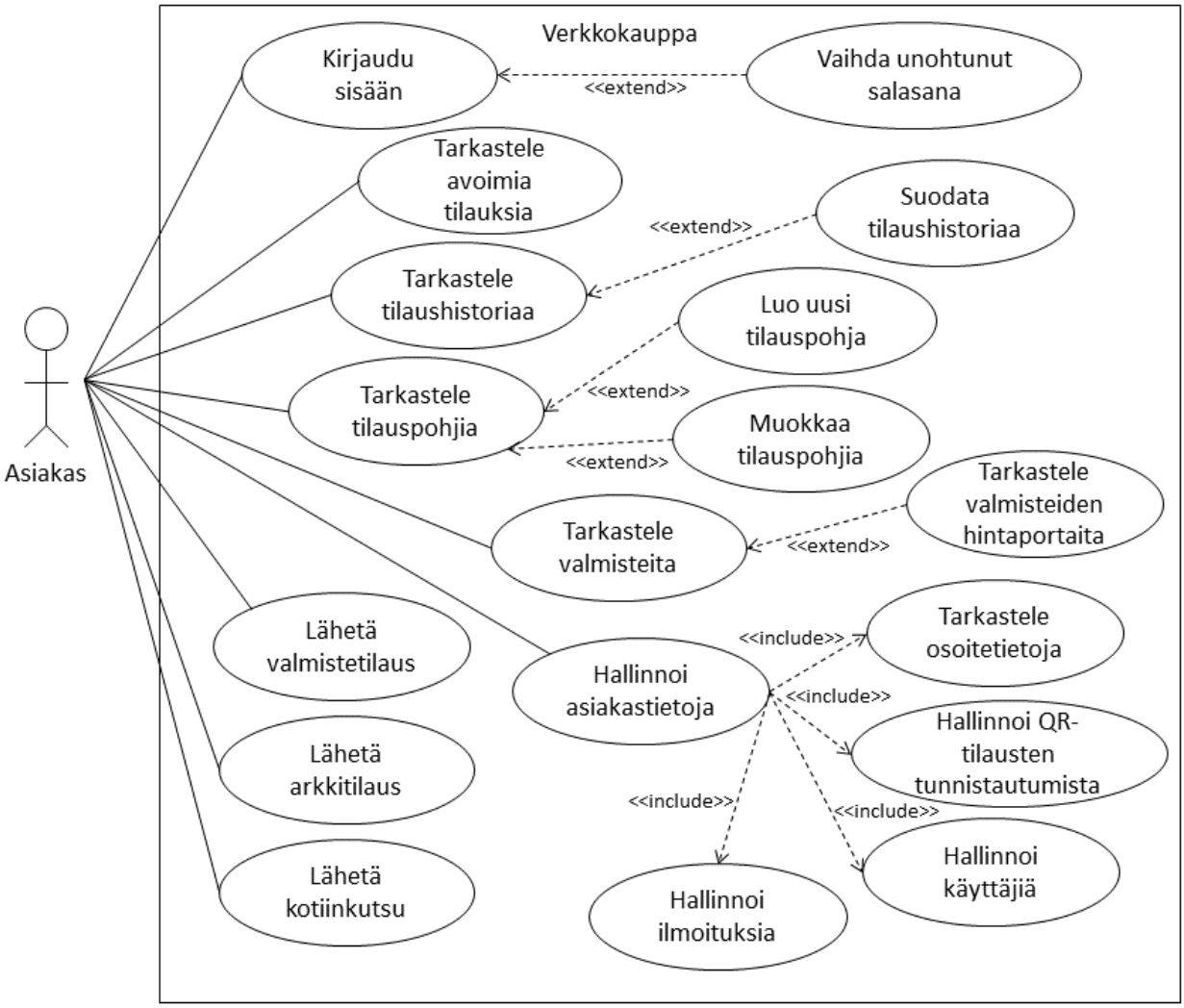
Verkkokaupan toiminnan kannalta oleellisiin perusominaisuuksiin kuuluvaksi määriteltiin aiemman verkkokauppatoteutuksen perusteella avointen tilausten listaaminen ja tarkastelu, tilaushistorian tarkastelu, valmistetilausten tekeminen, aaltopahviarkkitilausten tekeminen, valmisteiden listaaminen ja käyttäjätietojen hallinta. Näiden perusominaisuuksien lisäksi verkkokauppaan haluttiin toteuttaa palautelomake käyttäjäpalautteen keräämistä varten ja käyttöohje.

Nopean tilaamisen mahdollistamiseksi verkkokauppaan määriteltiin tilauspohjatoiminnallisuus, jonka kautta käyttäjät voisivat tallentaa usein tilattuja valmistekokonaisuuksia. Tähän ominaisuuteen päätettiin liittää mahdollisuus luoda tilauspohjia, joissa ennalta määrättyä tuotetta voitaisiin tilata nopeasti QR-koodiin sisällytetyn linkin avulla. QR-koodien käytön turvaamiseksi koodeille määriteltiin mahdollisuus asettaa koodien käytön yhteyteen tunnistautumismenetelmä. Lisäksi haluttiin mahdollistaa automaattisesti toistuvien tilausten tekeminen kätevästi, mikä päätettiin myös lisätä oman tyyppiseksi tilauspohjakseen.

Tilausten tilan etenemistä pidettiin verkkokaupan käyttäjien kannalta tärkeänä tietona, joten käyttäjille haluttiin antaa mahdollisuus tilaviestien tilaamiseen. Tähän liittyen määriteltiin myös yhdeksän erilaista tilaa, joissa verkkokaupan tilaukset ovat tilausprosessin aikana. Näiden tilojen esittäminen haluttiin lisätä myös tilauslistauksen yhteyteen.

2.4 Käyttötapaukset

Verkkokaupan ominaisuuksien määrittelyn yhteydessä huomattiin tarve kahdelle erilaiselle käyttäjäryhmälle, asiakkaille ja ylläpitäjille. Asiakkaan kannalta verkkokaupan tärkeimmät ominaisuudet on esitelty kuvan 2.2 käyttötapauskaaviossa. Käytännössä kaikki ominaisuudet lukuun ottamatta unohtuneen salasanan vaihtamista ovat sisäänkirjautumisen takana. Unohtuneen salasanan palauttamisen toteutuksessa on hyvä muistaa tietoturvaseikat, koska järjestelmä käsittelee kuitenkin yritysten väliseen liiketoimintaan liittyviä tietoja.



Kuva 2.2. *Verkkokaupan käyttötapauskaavio asiakkaan osalta.*

Verkkokaupassa asiakkaan tilaukset päätettiin jakaa kahteen eri näkymään niiden tilan mukaisesti ja tilaushistorianäkymään haluttiin suodatusmahdollisuus tilausajankohdan perusteella. Tilausten tarkastelunäkymässä näytetään listaus tilauksen perustiedoista, kuten tilausajankohdasta ja tilausnumerosta sekä tilausriveistä, joihin liittyy muun muassa toimitusajankohta ja tila.

Oman kokonaisuutensa muodostaa tilauspohjien tarkastelunäkymä, jossa voi tarkastella kolmen eri tyypin tilauspohjia ja luoda uusia sekä tietenkin muokata aiemmin luotuja. Tilauspohjiin liittyy tilattavien valmisteiden ja muiden tavallisten tilaustietojen lisäksi QR-tilauspohjien tapauksessa myös mahdollisesti tunnistustietoja ja automaattitilauspohjien tapauksessa tilaus- ja vahvistusaikaväli.

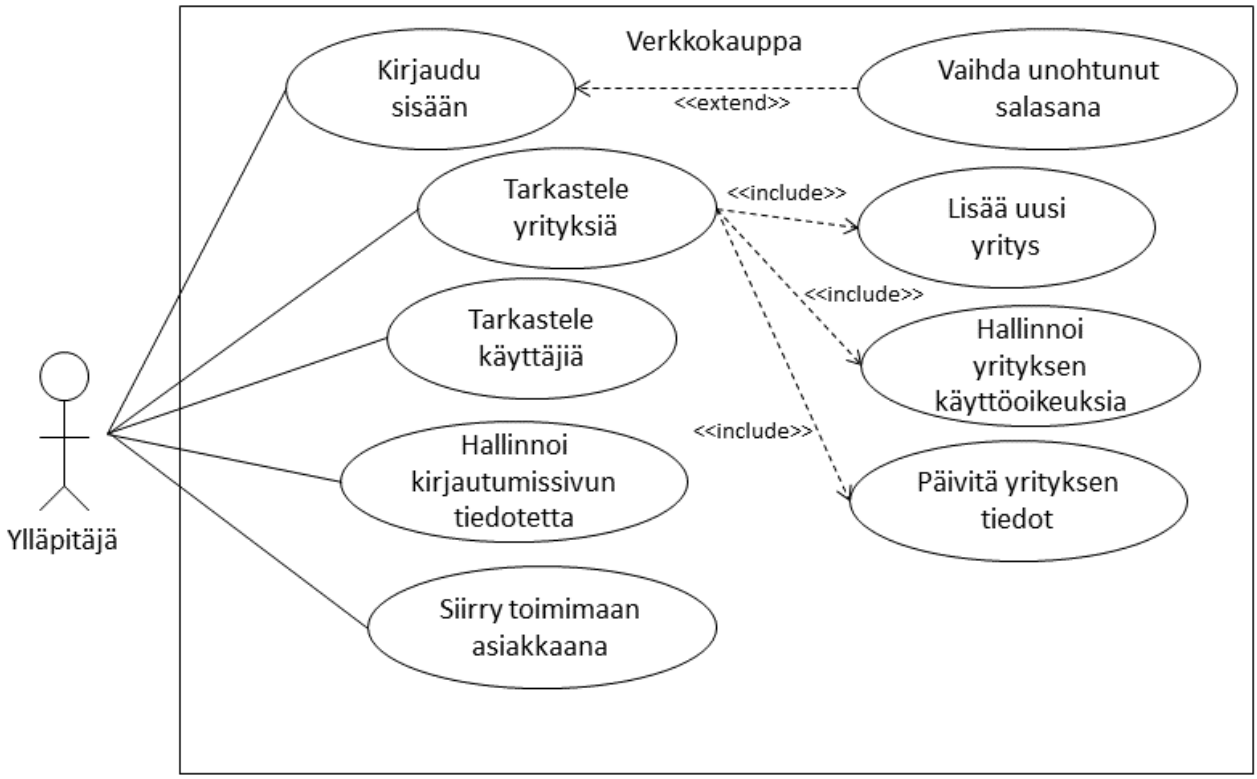
Valmisteiden tarkastelussa asiakas voi selata yritykselle linkitettyä valmistelistaa, jossa on listattuna muun muassa valmistenumero, asiakkaan oma vastaava numero ja valmisteen edellinen tilauspäivä. Tarkempien valmistetietojen osalta asiakas voi nähdä valmisteelle määritellyt kyseisellä hetkellä käytössä olevat hintaportaat.

Asiakastietojen hallintaan liittyen asiakas voi tarkastella toiminnanohjausjärjestelmästä luettuja osoitetietoja, jotka ovat käytettävissä verkkokaupan tilauksille, lisätä uusia käyttäjiä yrityksen tietoihin ja valita näille sopivat käyttöoikeudet sekä valita QR-tilausten yhteydessä käytettävään tunnistautumiseen liittyvät varmennusmenetelmät ja määritellä mahdolliset tunnistautumistiedot. Lisäksi asiakaskäyttäjä voi hallinnoida erilaisia ilmoituksia, kuten automaattisten tilauspohjien vahvistusten vastaanottajia, tilausten tilaviestejä ja myöhästymisilmoituksia.

Varsinaisia tilausten tekemistapoja on kolme erilaista. Valmistetilaus on tavallisin asiakkaan käyttämä tilauskeino, jossa tilattavat tuotteet valitaan asiakkaalle linkitettyjen valmisteiden joukosta. Kotiinkutsut koskevat puolestaan erikseen määriteltyä joukkoa valmisteista, joita varastoidaan asiakkaiden tarpeisiin. Näiden valmisteiden osalta kotiinkutsun tekemisnäkymässä näytetään varastosaldot, joiden perusteella tilaaja voi arvioida toimituksen kestoa.

Kolmas tilausvaihtoehto eli arkkitilaukset on selkeästi toimialakohtaisin. Käytännössä tämän tilausvaihtoehdon kautta voidaan tilata erilaisia aaltopahviarkkeja asiakkaan toivomien määritysten mukaisesti. Arkkitilauksiin liittyy esimerkiksi arkissa käytettävän pahvin laatu, valmiin arkin mitat ja arkkiin tehtävät nuuttaukset eli taivutukset.

Ylläpitäjän näkökulmasta kuvassa 2.3 esitellyt verkkokaupan käyttötapaukset painottuvat enemmän hallinnollisiin toimenpiteisiin. Sisäänkirjautuminen ja salasanahallinta onnistuvat asiakaskäyttäjän tavoin, mutta ylläpitäjä siirtyy kirjautumisen jälkeen ylläpitonäkymään.



Kuva 2.3. *Verkkokaupan käyttötapauskaavio ylläpitäjän osalta.*

Ylläpitäjä voi tarkastella verkkokauppaan jo lisättyjä yrityksiä, lisätä uusia yrityksiä, hallinnoida yrityksen mahdollisia käyttöoikeuksia ja käynnistää yrityksen tietojen päivityksen. Käyttöoikeustasot määritellään valmiiksi, jotta ylläpitäjän ei tarvitse käyttää erikseen aikaa oikeuksien tarkempaan säätämiseen. Yrityksen tietojen päivittämisen yhteydessä päivitetään esimerkiksi yrityksen valmiste- ja tilaustiedot ajan tasalle.

Käyttäjien tarkastelun kautta ylläpitäjä voi nähdä listan verkkokauppaan lisätyistä käyttäjistä tietoineen ja käyttöoikeuksineen. Verkkokaupan kirjautumisnäkymässä on CVP:n tavoin mahdollisuus lisätä tiedote, jonka avulla voidaan välittää informaatiota verkkokaupan käyttäjille esimerkiksi tuotantoseikkoihin ja uusiin ominaisuuksiin liittyen. Vain ylläpitäjälle kuuluvien ominaisuuksien lisäksi ylläpitäjällä on mahdollisuus siirtyä toimimaan asiakkaan tavoin valitsemalla yritys. Tällöin kaikki kuvan 2.2 mukaiset käyttötapaukset ovat ylläpitäjän suoritettavissa, mikä helpottaa yritysten tarkempien tietojen tarkastelua ja hallintaa.

3. FUNKTIONAALINEN OHJELMOINTI

Erilaiset ohjelmointiparadigmat tarjoavat vaihtoehtoisia tapoja ongelmanratkaisuun. Sopivan paradigman valinta on hyvä tehdä tapauskohtaisesti, jotta ohjelmointikielen ominaisuuksia voidaan hyödyntää parhaalla mahdollisella tavalla. Funktionaalinen ohjelmointi vaatii uudenlaisen ajattelutavan, mutta se tarjoaa hyötyä esimerkiksi lisäämällä ohjelmien luotettavuutta ja vähentää tarvittavan koodin määrää muun muassa tietorakenteiden sujuvamman käsittelyn kautta. Funktionaalisen ohjelmoinnin opetteleminen voi myös tarjota syvällisemmän ymmärryksen muihin ohjelmointiparadigmoihin ja helpottaa uuden oppimista tämän kautta.

Tässä luvussa esitellään tämän diplomityön taustalla olevan projektin toteutukseen valittua funktionaalista ohjelmointia. Ensin käydään lyhyesti läpi funktionaalisen ohjelmoinnin historiaa muutamien funktionaalisuutta sisältävien kielien kautta, sitten yleisiä periaatteita ja lopuksi hyötyjä muihin paradigmoihin verrattuna.

3.1 Funktionaalisuuden historiaa

Funktionaalisuuden alkuperä on matemaatikko Alonzo Churchin 1930-luvulla kehittämässä lambdakalkyylissä [8], joka on ennen tietokoneiden aikaa kehitetty formaalin laskennan malli. Sen lähtökohtana oli määritellä matematiikalle vaihtoehtoinen funktioihin joukkojen sijaan perustuva pohja. Lambdakalkyyli kehitettiin alkujaan matematiikan periaatteiden mallintamiseen, mutta se toimii myös lähes kaikkien funktionaalisten ohjelmointikielien pohjana. Lambdakalkyyli on Turing-täydellinen eli sillä voidaan laskea samat asiat kuin Turingin koneella [8]. Funktionaalisessa ohjelmoinnissa tyypittämätöntä lambdakalkyylia laajennetaan vakioilla ja tietotyypeillä [9].

Funktionaalisuuden kehityskulku käynnistyi yliopistoissa erilaisten opetus- ja tutkimusprojektien myötä. Aikaisin funktionaalisen ohjelmoinnin mahdollistavista ohjelmointikielistä on John McCarthyn 1950-luvun lopulla kehittämä Lisp [9]. Lispin lähtökohtana oli tarve symbolista dataa käsittelevän järjestelmän luomiseen. Lisp poikkesi rakenteeltaan aiemmista ohjelmointikielistä, ja se esitteli eräitä funktionaalisuuden perusperiaatteita esimerkiksi listojen käyttöön ja rekursioon liittyen. Myöhemmin monia eri murteita synnyttänyt Lisp ei kuitenkaan ole pelkästään funktionaalinen ohjelmointikieli, vaan siinä on myös proseduraalisen, reflektiivisen ja meta-ohjelmoinnin piirteitä, mikä tekee kielestä moniparadigmaisen. Lispin murteista erityisesti Scheme edustaa funktionaalisuutta [8].

Funktionaalisen ohjelmoinnin kehitys jatkui 1970-luvulla muiden muassa ISWIM-, PAL- ja SASL-ohjelmointikielten kautta [8]. ISWIM-kielestä ei syntynyt suoraa toteutusta vaan se jäi abstraktion tasolle. Ideat johtivat PAL-kieleen, joka taas puolestaan jalostui SASL-kieleksi. Puhtaasti funktionaalisen SASL-kielen kehittäjä David Turner hyödynsi sitä St Andrewsin yliopistossa opetuskäytössä Lispin tilalla, koska kieli muun muassa paransi ohjelmakoodin luettavuutta, poisti imperatiiviset ominaisuudet ja helpotti curry-muunnettujen funktioiden käsittelyä. Myöhemmin Turner muunsi SASL:n laiskaksi ja dynaamisesti tyypitetyksi ohjelmointikieleksi [8]. Nämä muutokset tekivät kielestä joustavamman ja yksinkertaistivat sen rakennetta joidenkin tietorakenteiden osalta.

Kehitys eteni samaan aikaan toisaalla yliopistomaailmassa APL-, FP- ja ML-ohjelmointikielien myötä [9]. APL hyödynsi taulukkopohjaisessa toimintatavassaan funktionaalisuuden ajatuksia käyttämättä siihen kuitenkaan lambdalausekkeita. FP oli ensimmäisiä laajemmalle levinneistä funktionaalisista kielistä. Myös FP:n perusajatukseen kuului lambdalaskentapohjan hylkääminen sen tuoman liiallisen vapauden vuoksi [9]. Lisäksi FP:ssä ei tarvita nimettyjä muuttujia sen funktiopohjaisuuden ansioista. ML syntyi metakielenä, mutta siitä muokattiin myöhemmin oma kokonaisuutensa [8]. Aikanaan ML oli käytännöllisimpiä funktionaalisia ohjelmointikieliä, koska se sisälsi muun muassa automaattisen tyyppipäättelyn ja mahdollisuuden käyttäjän määrittelemiin tietotyyppeihin sekä salli moniperiytymisen [9].

ML vaikutti esimerkiksi Mirandan ja Haskellin kehittymiseen. Miranda [8] sai alkunsa SASL-kielen perillisenä, kun David Turner jatkoi työtään funktionaalisuuden kehittäjänä. Vahvasti tyypitetty ja laiskaa laskentaa käyttävä Miranda on puhtaasti funktionaalinen kieli, joka sai ensimmäisenä lajissaan kaupallisen tuen. Miranda hyödyntää ohjelmakoodin rakenteessa sisennyksiä, mikä poistaa lohkosulkeiden ja rivin loppumerkkien tarpeen [8]. Mielenkiinto funktionaalisuutta kohtaan kasvoi edelleen ja vuonna 1987 puhtaasti funktionaalisten kielten joukkoon liittyi Haskell [9]. Haskell on standardoitu, vahvasti tyypitetty ohjelmointikieli, joka keräsi suosiota erityisesti tutkimuskäytössä ja opetuksessa. Mirandaan verrattuna Haskellin merkittävin lisäys on tyyppiluokkien mahdollistaminen [9]. Teollisuuskäytön kannalta on syytä myös mainita Ericssonin puhelinalan sovelluksiin kehittämä Erlang. Erlangia käytetiin Ericssonilla puhelinkeskusten ohjelmointiin sen vikasietoisuuteen ja rinnakkaisuuteen liittyvien ominaisuuksien vuoksi [10].

Nykypäivän kannalta huomattavia funktionaalisuuden edustajia ovat muiden muassa Clojure ja Scala. Clojure on Lispin murre, joten sen peruslähtökohdat ovat kantamuotonsa inspiroimia. Clojuren tavoin Scala toimii Java-virtuaalikoneen päällä ja yrittääkin sen myötä parantaa Javan kritisoituja ominaisuuksia tarjoamalla funktionaalisia piirteitä olio-ohjelmointiin yhdistettynä [11]. Funktionaalisia periaatteita ja toimintatapoja on viime aikoina vähitellen tuotu myös osaksi imperatiivisia ohjelmointikieliä. Esimerkiksi C# 3.0

ja Java 8 tarjoavat erilaisia rakenteita, joiden on tarkoitus suoraviivaistaa ohjelmointia ja mahdollistaa funktionaalisen tyylin käyttö.

3.2 Funktionaalisuuden periaatteita

Funktionaalisen ohjelmoinnin lähtökohtana voidaan pitää nimensä mukaisesti funktioita, joiden kautta ohjelman suoritus tapahtuu. Funktiot tuottavat matematiikan tapaan tuloksinaan arvoja, jotka riippuvat vain funktioille annetuista parametreista. Toisin sanottuna funktiot ovat sivuvaikutuksettomia eli eivät muuta ohjelman tilaa. Myöskään sijoituslausetta ei puhtaassa funktionaalisuudessa ole käytössä [12]. Tästä huolimatta useissa funktionaaliseksi kutsutuissa ohjelmointikielissä arvojen sijoittaminen on kuitenkin mahdollista, mikä vähentää kyseisten kielten puhtautta. Tämän kaltaisten kielten funktionaalisuutta tukee kuitenkin se, että niissä sijoitusta käytetään harvemmin kuin vastaavissa imperatiivista paradigmaa noudattavissa kielissä [13].

Funktioista käytetään funktionaalisessa ohjelmoinnissa nimitystä ensimmäisen luokan kansalaiset [9]. Tällä korostetaan sitä, että funktiot ovat erityisen tärkeässä asemassa eli käytännössä niitä voi käyttää toisten funktioiden parametreina ja paluuarvoina sekä tietorakenteiden sisältäminä alkioina. Muita funktioita parametreinaan saavat tai palauttavat funktiot ovat korkeamman asteen funktioita [12]. Ensimmäisen luokan kansalaisuuteen liittyvät mahdollisuudet muuttavat funktionaalisuudessa tarvittavaa ajattelutapaa muihin ohjelmointityyleihin verrattuna, mikä on hyvä muistaa huomioida myös ohjelmoitaessa.

Puhtaaseen funktionaalisuuteen liittyy myös aiemmin mainittu funktioiden sivuvaikutuksettomuus. Sivuvaikutuksettomuuden myötä funktioiden evaluointijärjestyksellä ei ole väliä, koska lopputulos on järjestyksestä riippumatta aina sama tietylle syötteelle [12]. Tämä mahdollistaa esimerkiksi laiskan tai rinnakkaisen evaluoinnin käytön, mikä voi osaltaan helpottaa ohjelman kirjoittamista tehokkaammaksi. Todellisuudessa myös erot evaluointijärjestyksessä vaikuttavat kuitenkin ohjelman tehokkuuteen ja saattavat aiheuttaa ikuisia silmukoita, joten aivan vapaaseen suoritusjärjestykseen ei kaikissa tapauksissa päästä [13]. Lisäksi käytännön tilanteista esimerkiksi syötteen lukeminen aiheuttaa väistämättä sivuvaikutuksia.

Tarkastellaan evaluointijärjestykseen liittyen esimerkkinä yksinkertaista kokonaisluvun palauttavaa Scala-funktiota, jossa on sivuvaikutuksena tekstirivin tulostaminen:

```
def palautaArvo() = {
      println(”Sivuvaikutus”)
      5
}
```

Scalassa voidaan funktion määrittelyn yhteydessä valita käytettävä evaluointijärjestys, joten toteutetaan kaksi esimerkkifunktiota, joissa käytetään eri järjestystä:

```
def applikatiivinen(x: Int) = {
      println(”Arvo 1: ” + x)
      println(”Arvo 2: ” + x)
}

def normaali(x: => Int) = {
      println(”Arvo 1: ” + x)
      println(”Arvo 2: ” + x)
}
```

Kun nämä funktiot suoritetaan funktioiden ensimmäisen luokan kansalaisuuteen perustuen siten, että niille annetaan parametrina aiemmin määritelty `palautaArvo`-funktio, saadaan `applikatiivinen`-funktion tulosteeksi:

```
Sivuvaikutus
Arvo 1: 5
Arvo 2: 5
```

Ja `normaali`-funktion tulosteeksi:

```
Sivuvaikutus
Arvo 1: 5
Sivuvaikutus
Arvo 2: 5
```

Tämä ero selittyy sillä, että nimiensä mukaisesti `applikatiivinen`-funktion arvon laskemisessa käytetään applikatiivista evaluointitapaa ja `normaali`-funktion laskennassa normaalia evaluointitapaa. Applikatiivisessa eli sisältä ulospäin suuntautuvassa laskentatavassa evaluoidaan ensin parametrin arvo, jota käytetään funktion arvon laskemiseen. Normaalissa eli ulkoa sisäänpäin suuntautuvassa järjestyksessä parametri evaluoidaan vasta tarvittaessa, mikä tarkoittaa tässä tapauksessa sen evaluoimista kahdesti [9].

Normaalin evaluoinnin tapauksessa voidaan välttyä käyttämättömien argumenttien evaluoimiselta, jolloin toiminta saattaa olla tehokkaampaa. Toisaalta esimerkkitapauksen kaltaisissa ohjelmissa, joissa käytetään samoja argumentteja useasti, tehdään ylimääräistä työtä [9]. Tehokkuutta tällaisissa tapauksissa parantaa laiska evaluointi, jossa yhdistetään normaalin evaluoinnin ajatus argumenttien evaluoinnista siihen, että kukin argumentti evaluoidaan vain kerran [12].

Sijoituslauseen puuttuminen puhtaasta funktionaalisuudesta aiheuttaa käytännön tasolla esimerkiksi sen ongelman, että for-silmukkaa ei silmukkamuuttujan käytön mahdottomuuden vuoksi voi hyödyntää. Oikeastaan edes tavalliset muuttujat eivät ole mahdollisia, koska alustamisen jälkeen arvojen muuttamiseen ei ole keinoja [13]. Tavallisen silmukan

sijaan onkin käytettävä rekursiota tai esimerkiksi funktion tuloksen muuntamista eri muotoon map-funktion avulla. Rekursion käyttö on järkevää myös sivuvaikutuksettomuuden säilyttämisen kannalta. Rekursiossa funktiota suoritetaan, kunnes rekursion päättymisehto täyttyy, joten erillistä silmukkamuuttujaa ei tarvita [11].

Sivuvaikutusten puuttumisen myötä funktionaalisessa ohjelmoinnissa painottuu kaavapäättelyn merkitys. Useissa moderneissa funktionaalisissa kielissä on perinteisempää case-rakennetta muistuttava mahdollisuus mallin tunnistamiseen (pattern matching), joka mahdollistaa monimutkaistenkin ehtorakenteiden kirjoittamisen kompaktisti [9]. Käsitellään esimerkkinä Scalan match-rakenne:

```
def mallinTunnistus(x: Any) = x match {
      case 2 => ”kaksi”
      case ”kolme” => 3
      case _ => ”jotain muuta”
}
```

Kun funktiota `mallinTunnistus` kutsutaan `mallinTunnistus(2)`, saadaan tuloksena teksti `”kaksi”`. Vastaavasti kutsu `mallinTunnistus(”kolme”)` palauttaa kokonaisluvun 3 ja mikä tahansa muu arvo tekstin `”jotain muuta”`. Scalan tapauksessa mallin tunnistamisen avulla voidaan jättää tyyppipäättelyn ja vertailun tekeminen ohjelmointikielen vastuulle, mikä tekee ohjelmakoodista huomattavasti kompaktimpaa ohjelmoijan kannalta [9].

3.3 Funktionaalisuuden hyötyjä ja haittoja

Kuten kaikissa eri ohjelmointiparadigmoissa, myös funktionaalisuudessa on omat etunsa ja huonot puolensa. Erityisesti puhdas funktionaalisuus parantaa ohjelmien luotettavuutta, koska sen periaatteiden mukaan ohjelmat käyttäytyvät aina yhtenäisellä tavalla. Koska suorituksen aikana ei voi tapahtua yllättäviä sivuvaikutuksia, ohjelmat ovat vakaampia ja niiden ylläpitäminen on helpompaa kuin vastaavien muilla paradigmoilla toteutettujen ohjelmien ylläpito [12]. Vastaavasti funktionaalisuus mahdollistaa monessa tapauksessa kompaktimman ja helpommin luettavan ohjelmakoodin.

Muuttujien puuttuminen puhtaasta funktionaalisuudesta on hyvä asia rinnakkaisuuden ja sitä kautta ohjelmien tehokkuuden parantamisen kannalta. Rinnakkaisuudessa on yleensä haasteena se, että usean säikeen toteutuksessa tiettyä dataa saa käsitellä vain yksi prosessi kerrallaan, jotta voidaan varmistua tiedon eheydestä. Muuttujien puuttuessa mahdolliset päällekkäisyydet eivät aiheuta ongelmia, koska prosessit eivät voi yksinkertaisesti muuttaa dataa. Tämä tekee rinnakkaisista operaatioista turvallisempia ja vähentää tarvetta erilaisten varautumiskeinojen käyttämiseen [11].

Funktionaalinen ohjelmointi on kasvattanut suosiotaan viime vuosina, mutta imperatiivinen ohjelmointi on edelleen sitä suositumpaa. Siirtymistä funktionaaliseen ohjelmointiin hidastavat erityisesti puutteelliset rajapinnat muihin ohjelmointikieliin [12]. Myös kehitystyökalut ovat jäljessä ominaisuuksiltaan ja monipuolisuudeltaan, mikä osaltaan vaikeuttaa ohjelmointia ja nostaa kynnystä siirtyä käyttämään funktionaalisia kieliä [14].

Uudenlaisten ohjelmointiparadigmojen opettelu on aina haastavaa, joten funktionaalisuuteen siirtyminen vaatii aikaa ja aivotyötä. Funktionaalinen ohjelmointi poikkeaa merkittävästi tyyliltään useimmille tutusta imperatiivisesta ohjelmoinnista, joten sen tehokkaaseen käyttöön vaaditaan syvällisemmän ymmärryksen kehittymistä. Taitava funktionaalinen ohjelmoija saattaa myös kirjoittaa paljon peräkkäisiä funktiokutsuja yksittäisellä rivillä suorittavaa ohjelmakoodia, jonka lukeminen on aloittelijalle todellinen haaste. Osa funktionaalisista kielistä mahdollistaa imperatiivisen tyylin käytön, mutta tällöin funktionaalisuuden hyödyt jäävät vähäisemmiksi [14]. Toisaalta eri paradigmojen välillä siirtyminen voi olla kätevää tilanteissa, joissa ratkottavat ongelmat ovat hyvin erityyppisiä.

4. VERKKOKAUPAN SUUNNITTELU

Kuten muidenkin ohjelmistojen tapauksessa, myös verkkokauppaa toteutettaessa on tärkeää kartoittaa siihen liittyvät vaatimukset ja ominaisuustoivomukset mahdollisimman aikaisessa vaiheessa. Näiden lisäksi toimintaympäristö asettaa omat rajoitteensa esimerkiksi valittavien teknologioiden ja arkkitehtuurin suhteen. Lisäksi on syytä huomioida mahdolliset integraatiot muihin järjestelmiin.

Tämä luku käsittelee suunnittelun lähtökohtana toimineet erityistarpeet ja haasteet, esittelee valitun arkkitehtuuriratkaisun ja perustelee tärkeimpien toteutusteknologioiden valintaa sekä kuvailee ne. Toteutusteknologioiden osalta rajaudutaan palvelinpäähän, käyttöliittymään liittyviä teknologiavalintoja käsitellään kohdassa 5.3.

4.1 Erityistarpeet ja yleiset haasteet

Valmiin verkkokauppasovelluksen käyttö rajattiin pois mahdollisuuksista projektin aikaisessa vaiheessa, koska todettiin että valmiin ratkaisun mukauttaminen ja integroiminen olisivat vaatineet huomattavasti työtä sovellusalan seikoista ja asiakkaan käytännöistä johtuen. Tätä näkemystä tuki myös se seikka, että oman räätälöidyn ratkaisun myötä verkkokaupan ominaisuuksista saadaan juuri toivotun kaltaiset, omaan toimintatapaan sopivat ja yhtenäisen ilmeen mukaiset erityisesti tilanteessa, jossa mikään valmis ratkaisu ei sellaisenaan sovellu käyttöönotettavaksi.

Verkkokaupan eri tilanteissa tarvitsemat tiedot, kuten tilaukset, valmisteiden hinnat ja asiakkaiden osoitteet ovat tallennettuna Adaran BVP-toiminnanohjausjärjestelmässä. Alustavien selvitysten myötä ilmeni, että BVP:ssä ei ole erillisiä selkeitä rajapintoja tilauksiin ja asiakkaisiin liittyvien tietojen käsittelyyn tai noutamiseen, lukuun ottamatta erikseen räätälöityä mahdollisuutta yksinkertaisten tilausten tuomiseen järjestelmään. Joitakin olemassa olevia rajapintoja järjestelmästä löytyi tämän lisäksi, mutta ne liittyivät lähinnä varaston ja raaka-aineiden hallintaan. Tämä tarkoitti käytännössä sitä, että toiminnanohjausjärjestelmän datan lukeminen oli toteutettava suoraan BVP:n Oracle-tietokannan kautta. Suunniteltaessa uuden verkkokaupan arkkitehtuuria tämä haaste oli hyvä huomioida siltä osin, että toiminnanohjausjärjestelmän mahdollisiin muutoksiin ja päivityksiin varauduttiin jo ennalta.

BVP:n tietokannan tarkempi tutkiminen paljasti jo alkuvaiheessa, että tietokannan rakenne ja sisältö tulisivat aiheuttamaan päänvaivaa toteutuksen edessä. Tietokantataulujen nimet ja sarakkeiden nimet olivat muutaman kirjaimen lyhenteitä, mikä oli selkeä haaste niiden sisällön ja tietokannan rakenteen selvittämisessä. Jonkin verran kuvauksia ja selityksiä kannan rakenteen osalta oli tarjolla, mutta suurimmaksi osaksi dokumentaatio

osoittautui puutteelliseksi. Rakenteen selvittämistä hankaloitti myös se, että minkäänlaisia vierasavaimia tietokannan taulujen väliltä ei löytynyt, vaikka tietyt arvot selvästi viittasivatkin toisiinsa. Ehkäpä suurin ongelma tietokannan sisällön osalta oli se, että eri sarakkeiden tietotyypit oli toteutusvaiheessa valittu huonosti. Tämä tarkoitti sitä, että esimerkiksi kellonaikoja on esitetty viiden merkin tekstijonoina ja erilaisia tilatietoja yhden merkin kirjainkoodilla tai Y/N-yhdistelmällä boolean-arvon sijaan. Hankaluuksia datan käsittelyssä ja arvojen vertailussa aiheutti erityisesti myös erilaisten merkkijonoina tallennettujen kenttien vakiomittaisuus, koska kenttien sisältö on tästä johtuen usein täydennetty välilyönneillä määrämittaansa. Oman lisänsä kokonaisuuteen toi vielä tietokantayhteyden hitaus ja epävarmuus.

Verkkokaupan kannalta eräs tärkeä lähtökohta ovat sen kautta myytävät tuotteet. Adaran tapauksessa myytävät tuotteet ovat erilaisia aaltopahviarkkeja ja -valmisteita. Suurin osa tuotteista valmistetaan asiakaskohtaisten vaatimusten mukaisesti, mutta mukana on myös yleisiä tuotteita, jotka perustuvat esimerkiksi standardien mukaisiin laatikkomalleihin. Valmisteiden lisäksi asiakkaat tilaavat arkkeja, jotka tuotetaan tilausten mittojen ja materiaalipyyntöjen mukaisesti. Verkkokaupan tuotteet ovat siis osittain kiinteitä ja osittain mukautettavia, mikä oli muistettava tilaustoimintoja toteutettaessa.

Julkisista B2C-verkkokaupoista poiketen B2B-verkkokaupat ovat yleensä vain yritysten olemassa oleville asiakkaille rajattuja. Myös Adara Shopin tapauksessa suljetun ympäristön käyttäminen oli tarpeellista muun muassa siksi, että erilaisten asiakaskohtaisesti rajattujen tuotteiden ja hintojen näyttäminen eri asiakkaille mahdollistuisi. Hintoihin liittyi myös tarve hintaportaiden käytöstä, mikä tarkoittaa käytännössä tuotteiden kappalehinnan muuttumista tilatun määrän mukaisesti. Hintaportaat ovat voimassa tietyillä aikaväleillä, joten hintojen käytön yhteydessä tulee valita kyseisellä hetkellä voimassa olevat portaat, jotta voidaan varmistua hintojen oikeellisuudesta.

Suljettuun ympäristöön liittyvät myös erilaiset käyttöoikeudet, joihin liittyen kävi ilmi, että yhteen käyttäjään saattaa liittyä useampia eri asiakasnumerolla olevia yrityksiä toiminnanohjausjärjestelmässä. Tämä oli haaste siksi, että yrityksillä saattaa olla päällekkäisiä tuotteita erilaisilla hinnoitteluilla, mikä voi puolestaan aiheuttaa virheellisten hintojen näyttämistä, ellei yrityksen valintaa ole toteutettu huomioimaan tätä seikkaa. Toisaalta joillain yrityksillä saattoi olla myös useampia asiakasnumeroita yritysrakenteista johtuen, joten yksittäiselle käyttäjälle oli mahdollistettava käytettävän asiakasnumeron valinta ennalta määritetystä joukosta.

Aiemmin BVP-järjestelmään toteutetun tilausten tuontimenetelmän mukaisesti tilausten siirtäminen toiminnanohjausjärjestelmään tapahtuu kirjoittamalla tilauksen tiedot suoraan tietokantaan. Tilauksia ei saa kuitenkaan lisätä suoraan tilaukset sisältävään tietokantatauluun, vaan tätä varten on käytössä erillinen välitaulu, josta BVP siirtää ulkoisista

järjestelmistä tulleet tilaukset ajastetusti omiksi sisäisiksi tilauksikseen. Tilauksen lisäämisen ja ajastetun prosessin suorittamisen välisenä aikana tilaukset eivät näy järjestelmässä. Mikäli tilauksen tiedot ovat jollain tavoin virheellisiä, tilauksen siirto epäonnistuu ja tiedot jäävät vain välitauluun. BVP:n tilaustietoja ei voitu siis käyttää sellaisenaan, kun asiakkaalle haluttiin näyttää myös siirtymistä odottavat tilaukset heti niiden tekemisen jälkeen.

Koska asiakkaan tilaamat tuotteet valmistetaan useimmiten vasta tilauksen vastaanottamisen jälkeen, liittyy yksittäiseen BVP:n tilausriviin tieto sen etenemisestä tuotantoketjussa. BVP:ssä etenemistä ei voi seurata suoraviivaisesti johtuen siitä, että tarkemmat tiedot eri vaiheiden tilasta ovat jaettuna eri tuotantovaiheisiin liittyvien tietokantataulujen taakse. Asiakkaalle näytettävä tila haluttiin kuitenkin yksinkertaistaa, joten tilaseurantaa jouduttiin purkamaan selkeämmäksi kokonaisuudeksi. Oman haasteensa aiheutti myös se, että tilausten tilojen muutoksista ei ole mahdollista saada ilmoituksia, vaan tilaa joutuu tarkastelemaan manuaalisesti joka kerta erikseen, kun tilatietoa tarvitaan.

BVP:stä löytyy useita tuhansia erilaisia tuotteita, joista vain tietty valikoima liittyy tiettyyn asiakkaaseen. Tuotteiden yhdistäminen asiakkuuteen on ratkaistu niin sanotulla ristikytkentätaululla, jossa tuotteet linkitetään tuotekohtaisesti jokainen omalla rivillään yksittäiseen asiakasnumeroon. Tässä yhteydessä tuotteelle voidaan myös määrittää asiakkaan käyttämä tuotenumero, jotta tuotteiden tunnistaminen on helpompaa myös asiakkaalle. Ristikytkentätaulun tarkempi tarkastelu paljasti kuitenkin, että jokainen yksittäinen tuote on kytketty asiakkaaseen useampaan kertaan eri kolmikirjaimisen parametrin avulla. Parametri indikoi kytkentärivin käyttötarkoituksen, joita ovat käytännössä näkyvyys vanhassa verkkokaupassa, linkitys asiakkaaseen ja linkitys asiakkaan tuotenumeroon. Tämän kaltainen saman tiedon toistaminen useaan kertaan koettiin redundantiksi, joten toimintaa haluttiin yksinkertaistaa.

Tuotteiden kaltaisen haasteen muodostivat toimitusosoitteet, joiden tapauksessa BVP:hen on määritelty useita erilaisia mahdollisia osoitekokonaisuuksia. Osa osoitteista oli vanhentuneita tai osittain päällekkäisiä tietojensa osalta, mikä tarkoitti sitä, että kaikkia osoitteita ei voitu antaa verkkokaupan puolella asiakkaan valittavaksi. Vanhan verkkokauppatoteutuksen tapauksessa tämä oli toteutettu myös eräänlaisen ristikytkentätaulun avulla, mikä osoittautui kohtalaisen suoraviivaiseksi toimintatavaksi myös jatkon kannalta.

4.2 Arkkitehtuuri

Tilausten, tuotteiden ja asiakkaiden tietojen hallinnointi ja käsittely tapahtuu BVP-toiminnanohjausjärjestelmässä, joka on verkkokaupan kannalta tärkein ulkoinen järjestelmä. Selkeiden rajapintojen puutteen vuoksi käytännössä ainoa mahdollinen keino noutaa tietoja BVP:stä on sen tietokannan lukeminen suoraan. Tämä rajoite synnytti tarpeen sille, että BVP:n tietokannan käsittelyyn liittyvät toimenpiteet olisivat erillään muusta

verkkokauppaan liittyvästä toiminnallisuudesta, jotta mahdolliset muutokset ja päivitykset aiheuttaisivat mahdollisimman vähän tarpeita muuttaa verkkokaupan toteutusta.

Aiemman verkkokauppatoteutuksen CVP:n toimintaa tutkimalla saatiin selville, että kyseinen järjestelmä käytti BVP:n tietokantaa suoraan, toimien käytännössä vain näkymänä tietokannan sisältöön. Koska verkkokauppauudistuksen myötä kaupan käyttäjien määrää oli tarkoitus lisätä voimakkaasti, tietokannan suora käyttö nähtiin ongelmallisena sen suorituskyvyn kannalta. Erityinen riski muodostui siitä seikasta, että verkkokaupan lisääntyvä käyttö saattaisi jumiuttaa toiminnanohjausjärjestelmän, mikä tietäisi isoja ongelmia koko yrityksen toiminnan kannalta. Toisaalta myös tietoturvasyistä aiempi toimintatapa oli epäilyttävä, koska se mahdollisti potentiaalisesti luvattoman pääsyn toiminnanohjausjärjestelmän tietokantaan.

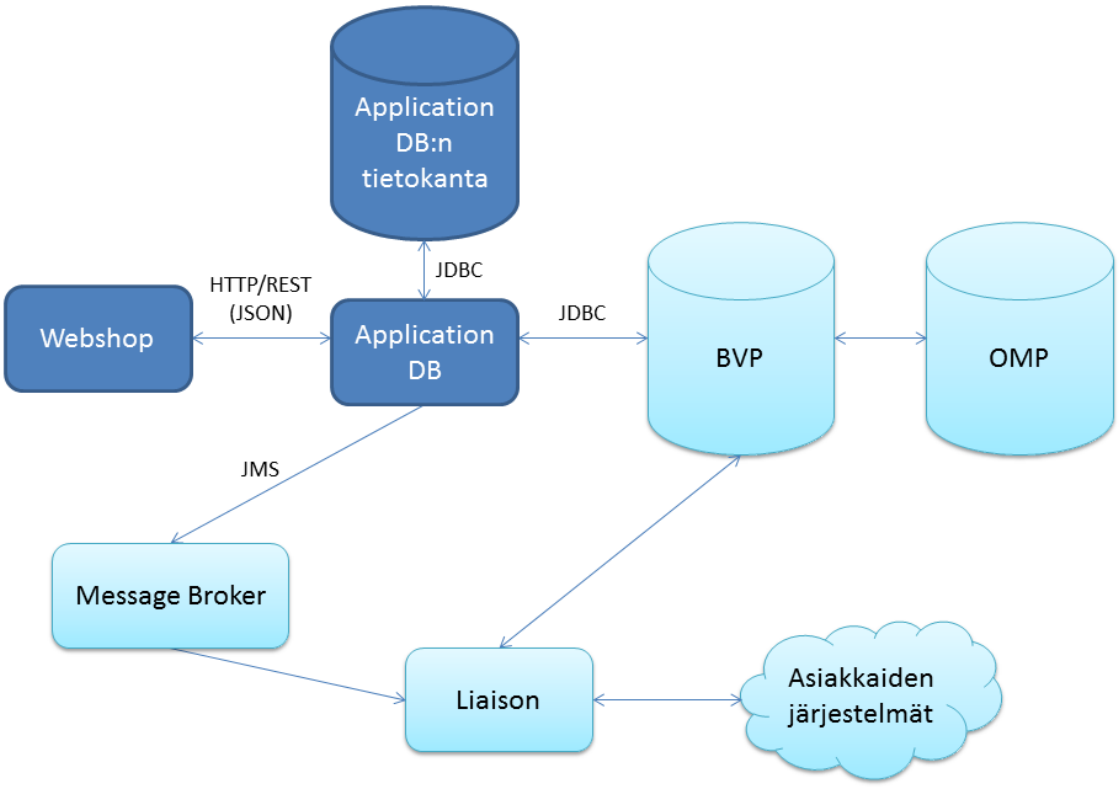
Näiden lähtötietojen perusteella verkkokaupan toteutuksen osalta päädyttiin kaksiosaiseen rakenteeseen, jossa varsinaisen verkkoon näkyvän osan toteuttaa Webshop ja ulkoisen järjestelmien ja tietokannan käsittelystä vastaa Application DB. Kommunikointi näiden kahden osan välillä tapahtuu HTTP-protokollan yli käyttäen REST-arkkitehtuurimallin mukaista rajapintaa. Tiedot siirretään web-sovelluksissa yleisesti käytetyssä JSON-muodossa. Verkkokauppaan liittyvien tietojen tallennusta varten Application DB:hen liittyy oma tietokanta, jonka kanssa kommunikoidaan JDBC-rajapinnan kautta.

Verkkokauppakokonaisuus ja muut siihen liittyvät järjestelmät on esitetty kuvassa 4.1. Kuvassa vaaleansinisellä pohjalla ovat ulkoiset järjestelmät ja toimijat, joiden toimintaan tämän projektin puitteissa ei tehty muutoksia. Tummansinisellä on korostettu verkkokauppauudistuksen myötä toteutetut osat eli Webshop, Application DB ja Application DB:n tietokanta. Ulkoisiin järjestelmiin liittyy BVP-toiminnanohjausjärjestelmän lisäksi OMP-tuotannonohjausjärjestelmä, Message Broker -viestinvälityskokonaisuus sekä Liaison Technologiesin toteuttamat integraatiopalvelut, joiden kautta vastaanotetaan tietoja asiakkailta sekä tavarantoimittajilta.

Ulkoisten järjestelmien osalta vastuunjako on toteutettu siten, että BVP ja OMP vastaavat toiminnan- ja tuotannonohjauksesta. BVP ja OMP siirtävät tietoa keskenään tietokantojensa välillä, mutta käytännössä kaikki verkkokaupan tarvitsema data löytyy BVP:n tietokannasta, joten tarvetta OMP:iin integroitumiseen ei ole. Yhteys OMP:iin on kuitenkin tärkeä tuotannon tilan seuraamisen kannalta.

Message Broker on Bitwisen toteuttama viestinvälityskomponentti, joka mahdollistaa datan lähettämisen ja vastaanottamisen eri ulkoisista lähteistä ja samalla tarvittaessa myös muuntamisen formaatista toiseen. Myös Liaison Technologiesin integraatiopalvelut toimivat tiedonsiirto- ja muuntokanavana erityisesti tapauksissa, joissa rajapinnat on määritelty ennen Message Brokerin toteuttamista. Message Brokeria käytetään esimerkiksi ti-

lausvahvistusten lähettämisessä asiakkaan suuntaan tietyissä erikoistapauksissa. Liaisonin kautta tuodaan ja viedään muun muassa tilauksiin liittyviä tietoja ja varastonhallinnan dataa.



Kuva 4.1. *Verkkokaupan liitynnät muihin järjestelmiin.*

Application DB:n toiminnallisuudet on eriytetty Webshopista ensisijaisesti sen takia, että tällä tavoin Application DB on voitu eristää ulkoverkosta, mikä parantaa verkkokaupan tietoturvaa huomattavasti aiempaan verrattuna. Tämän lisäksi Application DB mahdollistaa BVP:n käyttämisen abstrahoinnin Webshopin näkökulmasta, mikä on käytännöllistä esimerkiksi siinä tapauksessa, jos toiminnanohjausjärjestelmää vaihdetaan jossain vaiheessa.

Application DB:n oma tietokanta parantaa BVP:n luotettavuutta, koska sinne voidaan väliaikaisvarastoida verkkokaupan tarvitsemaa dataa ja koska verkkokaupan käyttäjät eivät tällä tavoin suoraan kuormita BVP:n tietokantaa. Koska BVP:n tietokannan datan rakenne on haastava esittämisen kannalta, mahdollistaa oma tietokantaratkaisu myös datan

rakenteen parantamisen ja muuntamisen paremmin verkkokaupassa esitettäväksi. Application DB pystyy tarvittaessa myös tarjoamaan muunlaisia palveluita, joilla dataa voidaan jalostaa aiempaa pidemmälle.

4.3 Toteutusteknologioiden valinta

Verkkokaupan uudistuksen lähtökohtana oli päivittää se vastaamaan nykypäivän tarpeita sekä toiminnallisuuksien että ulkoasun osalta. Käytettävät toteutusteknologiat valittiin huomioiden niiden ajantasaisuus ja toisaalta myös se, että valitut teknologiat olivat jo todellisessa käytössä eri toimijoilla. Tämän ajattelun myötä voitiin varmistua siitä, että mitään isoja yllätyksiä ei teknologioiden suhteen projektin edetessä kohdattaisi. Koska olemassa olevat verkkokaupparatkaisut oli rajattu toteutusvaihtoehtojen ulkopuolelle, tuli teknologiavalinnat tehdä myös ajatellen koko verkkokaupan toimintaa ja integroitumista muihin järjestelmiin.

4.3.1 Scala ja Play Framework

Erilaisia web-sovelluskehyksiä ja erityisesti mielipiteitä niihin liittyen, on tarjolla valtavasti, joten suuresta joukosta yhden valitseminen on vähintäänkin haaste. Projektin toteuttajilla oli aiempia kokemuksia ensisijaisesti PHP- ja Java-pohjaisista sovelluskehyksistä, mutta tätä ei haluttu pitää minkäänlaisena rajoituksena valinnan osalta. Koska myös asiakkaan suunnalta toteutusteknologiavalinta oli vapaa, päätettiin apuna hyödyntää Bitwisen kokeneiden työntekijöiden näkemyksiä. Yhdessä käydyissä keskusteluissa esille nousi funktionaalinen ohjelmointi ja erityisesti Scala.

Scala on ohjelmointikieli, jossa sekoittuvat toisaalta sekä olio-ohjelmoinnin että funktionaalisuuden piirteet. Kielen nimi on lyhenne sanoista scalable language, mikä sisältää perusajatuksen sen skaalautuvuudesta ja muuntuvuudesta eri käyttötarkoituksiin sekä laajentuvuudesta käyttäjänsä mukana. Scala on puhdas oliokieli, jossa kaikki arvot ovat olioita ja operaatiot funktiokutsuja, mutta kieleen kuuluu myös laaja tuki erilaisille funktionaalisen ohjelmoinnin menetelmille. Käytännössä tämä näkyy ohjelmoijalle siten, että Scala mahdollistaa siirtymän funktionaalisuuteen pakottamatta tekemään kaikkea funktionaalisesti heti alusta alkaen [15].

Funktionaalisuus tarjoaa paljon erilaisia tapoja esimerkiksi tietorakenteiden suoraviivaisempaan käsittelyyn ja muuntamiseen muodosta toiseen [15]. Scala mahdollistaa myös koodin määrän vähentämisen esimerkiksi Javaan verrattuna tämän kaltaisissa tilanteissa. Tämä perustuu muun muassa syntaktisiin seikkoihin, kuten puolipisteiden pois jättämisen mahdollistamiseen, mutta myös kielen kannalta konkreettisempiin seikkoihin, kuten funktioiden ketjuttamiseen ja lambda-laskennan juurista kumpuaviin lähtökohtiin.

Lyhyenä esimerkkinä Scalan suurimmista eroista ei-funktionaalisiin kieliin nähden voidaan tarkastella seuraavaa Java-koodia:

```
List<String> sanat = Arrays.asList("Aakkonen", "Auto", "Kynä", "Avain",
                                   "Kirja", "Pöytä");

Map<Character, List<String>> tulos = new HashMap<Character, List<String>>();

for(String sana : sanat) {
       char ekaKirjain = sana.charAt(0);

       if(!tulos.containsKey(ekaKirjain)) {
              tulos.put(ekaKirjain, new ArrayList<String>());
       }

       tulos.get(ekaKirjain).add(sana);
}

for(List<String> lista : tulos.values()) {
       Collections.sort(lista);
}

System.out.println(tulos);
```

Ohjelma 1. *Sanalistan ryhmitteleminen alkukirjaimen perusteella (Java 7).*

Ohjelma 1 muodostaa saamastaan sanalistasta alkukirjaimen perusteella järjestetyn avain-arvoparikokonaisuuden, jossa avaimena toimii alkukirjain ja arvona lista kyseisellä alkukirjaimella alkavista sanalistan sanoista. Vastaava ohjelma voidaan toteuttaa Scalalla huomattavasti kompaktimmin:

```
val sanat = List("Aakkonen", "Auto", "Kynä", "Avain", "Kirja", "Pöytä")
val tulos = sanat.sorted.groupBy(_.head)
println(tulos)
```

Scalan tapauksessa voidaan välttää silmukoiden käyttöä säiliöiden sisäänrakennetuilla funktioilla. Käyttämällä `groupBy`-funktiota saadaan myös kätevästi muodostettua halutunlainen tietorakenne määrittämällä vain haluttu avain, joka on tässä tapauksessa merkkijonon pää eli ensimmäinen merkki. Esimerkin suhteen kannattaa huomata myös se, että Scala käyttää tyyppipäättelyä, mikä mahdollistaa arvojen tyypin pois jättämisen, mikäli se on pääteltävissä kontekstista [15]. Myös tämän kaltaiset ominaisuudet tukevat Scalan valintaa projektiin, koska ne kasvattavat tuottavuutta.

Siirtymää Scalan käyttöön helpottaa se seikka, että Scalaa ajetaan Java-virtuaalikoneen päällä. Käytännössä tämä tarkoittaa sitä, että Java- ja Scala-luokkia voi käyttää samassa sovelluksessa ja niiden välillä voi tehdä viittauksia Scala-kääntäjän sisältämän Java-kääntäjän myötävaikutuksella [15]. Tämä yhteensopivuus mahdollistaa siis kaikkien Java-maailman kirjastojen ja työkalujen käytön Scala-maailmassa kätevästi ilman ylimääräistä vaivaa, mikä vähentää tarvetta vanhojen ratkaisujen uudelleenkeksimiseen.

Ohjelmointikielen valinnan jälkeen mahdollisten web-sovelluskehysten määrä väheni huomattavasti, mutta Scalallekin vaihtoehtoja oli useampia. Muun muassa Scalatran ja Liftin sijasta valinnan kohde oli Play Framework. Tätä valintaa perusteltiin sillä, että Play Framework on osa Scalan kehittäneen Martin Oderskyn perustaman Typesafe Inc:n alustakokonaisuutta [16]. Tätä seikkaa pidettiin hyvänä sen vuoksi, että alustakokonaisuuteen kuuluu muitakin projektin kannalta tarpeellisia teknologioita, kuten työkalut tietokannan olio-relaatio-mallinnukseen ja ajastettuun viestienvälitykseen.

Play Framework on Scalalla toteutettu MVC-arkkitehtuuria noudattava web-sovelluskehys, jota voi käyttää joko Javalla tai Scalalla toimivien rajapintojen kautta. Play Framework sisältää integroidun testauskehyksen, mutta myös erillisten testauskeinojen käyttö on mahdollista. Play Frameworkiin on integroitu Netty-webpalvelin, mikä tekee palveluiden tuotantoon siirtämisestä kohtalaisen suoraviivaista poistamalla tarpeen Apachen tyyppisen erillisen palvelinratkaisun konfigurointiin [17].

REST-rajapintojen toteuttaminen on yksinkertaista Playn reititystoimintojen avulla, mikä parantaa sovellusten rakennetta [17]. Play sisältää myös oman Scala-pohjaisen toimintatapansa sivupohjien luomiseen, mikä yksinkertaistaa sovelluksen rakennetta, koska sivupohjien käyttöön ei tarvita erillistä teknologiaa. Kehittäjän kannalta Playn parhaita ominaisuuksia on sen sivunlatauksen yhteydessä tarvittaessa tapahtuva automaattinen kääntäminen ja virheiden näyttäminen suoraan selainikkunassa.

4.3.2 PostgreSQL

Tietokantojen tapauksessa toistuu ohjelmointikielen ja sovelluskehyksen valinnasta tuttu valinnan vaikeus monipuolisen tarjonnan takia. Uudistusprojektin tietokantavalinta Application DB:n oman tietokannan osalta perustui pääosin aiempiin hyviin kokemuksiin PostgreSQL:n käytöstä ja myös muihin tietokantajärjestelmiin verrattuna avoimiin lisenssiehtoihin [18].

PostgreSQL on laajasti käytetty avoimen lähdekoodin tietokannanhallintajärjestelmä, joka noudattaa ACID-periaatetta [19]. ACID-periaate varmistaa järjestelmän tietojen eheyden atomisuuden, oikeellisuuden, eristyvyyden ja pysyvyyden kautta. Atomisuuden käsite tarkoittaa tietokannan transaktioiden eli toimenpiteiden suorittamista joko kokonaan tai epäonnistuttaessa ei ollenkaan. Oikeellisuus takaa, että transaktiot muuttavat tietokannan tilan kelvollisesta tilasta toiseen kelvolliseen tilaan. Eristyvyyden merkitys on se, että toimenpiteet voidaan suorittaa rinnakkain tai sarjassa, eikä tulos riipu valitusta tavasta. Pysyvyys takaa, että transaktion suorittamisen jälkeen muutokset pysyvät tallessa.

PostgreSQL sisältää monipuolisesti erilaisia tietotyyppejä, mikä mahdollistaa datan käsittelyn ja tallentamisen tarkoituksenmukaisessa muodossa [18]. Tietokannan käyttämisen näkökulmasta tärkeä seikka ovat rajapinnat, joista tämän projektin puitteissa päädyttiin aiempien teknologiavalintojen perusteella valitsemaan JDBC. PostgreSQL on myös laajennettavissa esimerkiksi paikkatietoa vaativien tarpeiden kannalta, mutta tämän projektin osalta tärkeämpää on tietojen tallennuksen varmuus ja vakaus.

4.3.3 Slick

Tietokannan kanssa kommunikointi voidaan toteuttaa puhtaana SQL:nä, mutta tällainen ratkaisu ei ole helposti ylläpidettävissä ja vaatii huomattavan määrän käsityötä. Käytännössä järkevämpää on käyttää olio-relaatio-mallinnusta, jonka avulla voidaan hyödyntää käytössä olevan ohjelmointikielen rakenteita datan kuvaamiseen ja kontrollointiin. Scalan ja Play Frameworkin tapauksessa Typesafen suosittelema Slick mahdollistaa tietokannan käsittelyn funktionaalisin menetelmin [20]. Slick valittiin mukaan projektin teknologiavalikoimaan Typesafen suositusten myötä ja myös siksi, että se sopii hyvin mukaan muiden kokonaisuuden osien kanssa.

Slickin avulla tietojen mallinnus ja tietokantakyselyt voidaan kirjoittaa SQL:n sijasta Scalana. Tätä kautta voidaan käyttää Scalan staattista tyypitystä, tietorakenteita ja case class -rakenteiden tarjoamia etuja datan käsittelyssä [21]. SQL:n käytön välttäminen tekee myös mahdollisten muutosten tekemisestä helpompaa, koska Slickin avulla toteutetut mallit ovat käytettävissä kyselyissä, mikä vähentää toisteisen koodin määrää. Slick mahdollistaa kuitenkin tarvittaessa myös SQL-kyselyjen tekemisen, mikäli jokin tarvittava toimenpide ei onnistu kätevästi korkeammalla abstraktiotasolla.

4.3.4 jOOQ

Verkkokaupan toiminnan kannalta tärkeä datan lähde on BVP:n Oracle-tietokanta. Kun tietokantaa tutkittiin tarkemmin suunnitteluvaiheessa, sen rakenne osoittautui monelta osin hankalaksi hallita. Tietokannan rakenteen hallintaa ja datan noutoa varten etsittiin hyviä toimintatapoja, joista käteväksi osoittautui Javalla toteutettu jOOQ-kirjasto [22].

Käyttämällä jOOQ:a BVP:n tietokannan käsiteltävistä tauluista voitiin generoida oliomalli, jonka avulla datan käsittely ja kyselyjen teko onnistuu puhtaan SQL:n käyttöä suoraviivaisemmin. Ilman tämän kaltaista työkalua tietokannan käyttöön olisi vaadittu paljon enemmän selvitystyötä ja käsin tapahtuvaa kyselyjen muodostamista. jOOQ ei ole yhtä kompleksinen kuin monet muut vastaavat mallinnustyökalut, mikä sopi hyvin BVP:n hankalarakenteisen datan noutamiseen [22]. Samalla päästiin eroon myös Oracle-tietokannan mahdollisista poikkeavuuksista ja isoimmista vierasavainten puuttumiseen liittyvistä ongelmista.

5. TOTEUTUS

Verkkokaupan tärkeimpien ominaisuuksien määrittelemisen ja kokonaisuuden rakenteen suunnittelemisen sekä toteutusteknologioiden valinnan jälkeen prosessin seuraava luonteva osa on toteutuksen aloittaminen. Toteutusvaiheessa aiemmat päätökset muuttuvat konkreettisiksi ja samalla voidaan arvioida niiden toimivuutta sovelluksen kannalta.

Tässä luvussa esitellään projektin aikana käytetyt toteutusmenetelmät ja työkalut sekä käydään läpi erilaiset rajapintakokonaisuudet muihin järjestelmiin. Lisäksi luvussa tarkastellaan verkkokaupan käyttöliittymän toteutusta ja siihen liittyviä ratkaisuja.

5.1 Toteutusmenetelmät

Verkkokaupan toteutus päätettiin tehdä ketterän ohjelmistokehityksen mukaisesti, koska määrittelyn ja suunnittelun jälkeen todettiin, että kaikkien verkkokaupan ominaisuuksien riittävän tarkka ja huolellinen kuvailu olisi turhan aikaa vievää. Ketterää lähestymistapaa tuki myös se seikka, että verkkokaupan kannalta tärkein rajapinta eli yhteys toiminnanohjausjärjestelmä BVP:hen oli edelleen osittain hämärän peitossa puutteellisen dokumentaation vuoksi.

Mitään erityistä menetelmävaihtoehtoa, kuten Scrumia tai XP:tä, ei haluttu käyttää, mutta ajatuksena oli kuitenkin hyödyntää iteratiivisuutta, jotta asiakas voisi antaa palautetta tehdyistä muutoksista ja toteutetuista ominaisuuksista mahdollisimman nopeasti. Myös projektin eteneminen oli selkeämmin havainnoitavissa tämän myötä. Iteraation kestoksi valittiin aluksi kaksi viikkoa ja samalla sovittiin, että kestoa voitiin tarvittaessa muuttaa, jos sillä hetkellä toteutuksessa olevat ominaisuudet vaatisivatkin enemmän aikaa tai vastaavasti valmistuisivat arvioitua nopeammin.

Asiakas järjesti projektin alkuvaiheessa koulutuksen, jossa käytiin läpi toimialakohtaista tietoa sekä toiminnanohjausjärjestelmän perustoiminnallisuudet ja myös jossain määrin järjestelmän ylläpitoa sekä tietokannan rakennetta. Tämä koulutus muodosti perusymmärryksen BVP:n kannalta, mikä nopeutti järjestelmän toimintaperiaatteen selvittämistä toteutuksen myöhemmissä vaiheissa.

Toteutus alkoi rinnakkain kahden muun samalle asiakkaalle toimitettavan projektin kanssa. Ensisijaisena vastuuhenkilönä näiden kaikkien kolmen projektin etenemisestä asiakkaan suuntaan toimi Bitwisen projektipäällikkö. Varsinainen ohjelmointi jakautui kahdelle ohjelmistosuunnittelijalle, jotka työskentelivät projektin ajan samassa työhuoneessa. Tämän lisäksi projektin alkuvaiheessa hyödynnettiin erillisen graafikon osaamista ulkoasun ja toimintojen sijoittelun osalta.

Asiakkaan toivomat ominaisuudet jaettiin alussa tehtäviin, joille annettiin prioriteetit toteutustärkeyden mukaan. Alustava jako tehtiin järjestelmän käyttötapausten mukaisesti. Tehtäville arvioitiin myös alustavat työmäärät, joiden perusteella iteraatiojaksoon mahtuvien tehtävien määrä oli helpompi valita. Uusia tehtäviä lisättiin tarpeen mukaan ja vanhoja päivitettiin sisällöltään, prioriteetiltaan ja työmäärältään asiakkaan toiveiden mukaisesti. Myös asiakkaalta tulleet muutospyynnöt käsiteltiin tätä kautta. Tehtyjen muutosten seuraamista ja hallinnointia varten käyttöön otettiin versionhallintajärjestelmä.

Iteraatiojakson mukaisin väliajoin pidettiin asiakkaan kanssa yhteinen palaveri, jossa käytiin läpi edellisen jakson aikana toteutettuja asioita ja mahdollisesti heränneitä kysymyksiä. Uudet ominaisuudet ja näkymät käytiin läpi demonstroimalla niiden toiminta kehitysversiolla. Samalla asiakas sai antaa palautetta näkemästään ja toivoa muutoksia tarpeen mukaan. Palaverissa valittiin myös seuraavan jakson toteutettavat asiat ja tehtävät korjaukset. Näille määriteltiin palaverin jälkeen työmääräarviot toteuttajien kesken. Kommunikointi asiakkaan suuntaan iteraatioiden aikana tapahtui ensisijaisesti sähköpostin välityksellä.

Kun verkkokaupan toteutus oli saatu siihen vaiheeseen, että ensimmäiset ominaisuudet olivat toimintakunnossa, perustettiin järjestelmälle asiakkaan sisäverkkoon rajattu testiympäristö, jota päivitettiin iteraatiojaksojen mukaisesti. Tämä mahdollisti sen, että asiakas pääsi itsekin helposti testaamaan toteutuksessa olevaa sovellusta ja ominaisuuksia. Verkkokaupan testiympäristö yhdistettiin toiminnanohjausjärjestelmän testiympäristöön, mikä mahdollisti samalla myös järjestelmien välisen integraation testaamisen oikeassa toimintaympäristössä.

Verkkokaupan testaamisessa käytettiin yksikkötestejä muun muassa Webshop-osuuden kontrollereiden sekä Application DB:n toiminnanohjausjärjestelmään liittyvien tuontiominaisuuksien ja näiden välisten rajapintojen testaamiseen. Yksikkötestauksessa hyödynnettiin valitun sovelluskehyksen tarjoamia yksikkötestausmenetelmiä. Järjestelmätestausta tehtiin käyttöliittymän kautta automatisoiduilla selaintesteillä.

Dokumentaation osalta verkkokaupalle kirjoitettiin wiki-pohjainen käyttöohje, jota käyttäjät pääsevät lukemaan suoraan verkkokaupan eri näkymistä. Tämän lisäksi toteutuksen aikana kerätyt huomiot kerättiin ylläpito-ohjeeseen, jota täydennettiin vielä toteutuksen loppupuolella sovelluksen käyttöönottoon ja päivittämiseen liittyvillä seikoilla.

Kun suurin osa ominaisuuksista oli saatu toteutettua ja asiakas oli testannut ne toimiviksi, perustettiin verkkokaupan varsinainen tuotantoympäristö. Alkuun tuotantoympäristöön tuotiin vanhan verkkokaupan aktiivisimmat viisi käyttäjäyritystä, jotta palautetta saataisiin myös todellisilta loppukäyttäjiltä. Muutaman viikon testiajanjakson jälkeen uudistuneen verkkokaupan käyttöönotto aloitettiin myös muiden yritysten osalta.

5.2 Käytetyt työkalut

Erilaisten yksinkertaisten sovellusten toteuttaminen onnistuu periaatteessa vaikkapa pelkän tekstieditorin, komentorivin ja kääntäjän avulla, mutta toimivat työkalut tehostavat ja nopeuttavat työtä. Sovelluksen rakenteen ja toimintojen monimutkaistuessa hyvien työkalujen merkitys korostuu entisestään. Verkkokaupan toteutuksessa käytettyjen työkalujen valinnat perustuivat aiemmin tehtyihin teknologiavalintoihin ja toteuttajien aiempaan tietämykseen erilaisten työkalujen käytöstä.

5.2.1 Scala IDE

Ohjelmoinnin kannalta tärkein työkalu on jonkinlainen editori, jolla ohjelmakoodia muokataan, mutta ohjelmointiympäristöllä työskentelyä voidaan tehostaa erilaisten lisäominaisuuksien avulla. Scalan myötä ohjelmointiympäristön valinta kohdistui Scala IDE:een [23] ja käytännössä koko verkkokaupan kehitystyö tehtiin sitä käyttäen. Scala IDE on rakennettu Java-ohjelmoijille tutun Eclipse-ohjelmointiympäristön päälle siten, että mukaan on lisätty tuki Scala-kääntäjälle ja Scala-projekteille. Scala IDE:n voi ladata joko omana erillisenä pakettinaan tai Eclipsen päivitystenhallinnan kautta lisäosana. Ympäristön kehittäjätiimin jäseniin kuuluu myös Scalan ja Play Frameworkin kehittäjinä tunnetun Typesafen henkilökuntaa.

Kaiken kaikkiaan Scala IDE:n ominaisuudet ovat Eclipseä aiemmin käyttäneiden kannalta tuttuja ja intuitiivisia. Scala IDE:een on lisätty muun muassa Scalaa tukeva funktioiden nimien täydennys ja syntaksin korostus sekä velhotoiminnot luokkien ja olioiden generointia varten [23]. Koodin kirjoittamisen kannalta käteviä ovat automaattisen muotoilun ja sisennyksen aputoiminnot, jotka helpottavat rakenteen ylläpitämistä. Hankalampien ongelmatilanteiden selvittämisen kannalta on myös hyvä seikka, että ohjelmakoodin debuggaaminen onnistuu mukaan integroidulla Scala-debuggerilla. Scala IDE sisältää myös tuen Play Frameworkin reititystoiminnallisuudelle ja sivupohjakielelle, mikä helpottaa käyttöliittymän toteuttamista.

5.2.2 sbt

Sovelluksen kääntämistä helpottavat build-työkalut, joista Scalan kanssa suosituin on sbt [24]. Sbt tarjoaa suoraviivaiset keinot Scala-koodin kääntämiseen ja se integroituu myös Play Frameworkin kanssa. Kääntämisprosessin määrittely tehdään Scalalla kirjoitettujen sbt-tiedostojen ja Build.scala-tiedostojen kautta. Näihin Scala-ohjelmia muistuttaviin tiedostoihin kirjataan tarvittavat kirjastot versioineen ja määritellään sovelluksen osat riippuvuuksineen projekteina. Konfiguraationsa avulla sbt vastaa myös paketinhallinnasta.

Sbt sisältää oman komentokehotteensa, jonka kautta sitä käytetään [24]. Yhdellä yksinkertaisella komennolla onnistuvat esimerkiksi sovelluksen käynnistäminen ajoon kehitysmoodissa, yksikkötestien ajaminen, projektikansioiden siivoaminen ja lisäosan avulla

Eclipsen ymmärtämien projektitiedostojen generointi. Myös sovelluksen paketointi julkaisua varten onnistuu kätevästi yhden komennon avulla. Käytännössä sbt osaa generoida esimerkiksi käyttöympäristöön siirtämisen kannalta kätevän zip-paketin, jonka sisältä löytyy Java-virtuaalikoneella ajettava pakettikokonaisuus ja myös kokonaisuuden käynnistämisen mahdollistava skriptitiedosto.

Verkkokaupan toteutuksessa sovellus jaettiin toimintojen mukaan kymmeneen erilaiseen sbt-projektiin, joista kaksi paketoivat muut ajettaviksi kokonaisuuksiksi. Yhden buildtiedoston käyttäminen mahdollisti kaikkien riippuvuuksien määrittelyn samassa paikassa. Ulkoisia kirjastoja otettiin käyttöön sbt:n avulla reilu kaksikymmentä.

5.2.3 JIRA

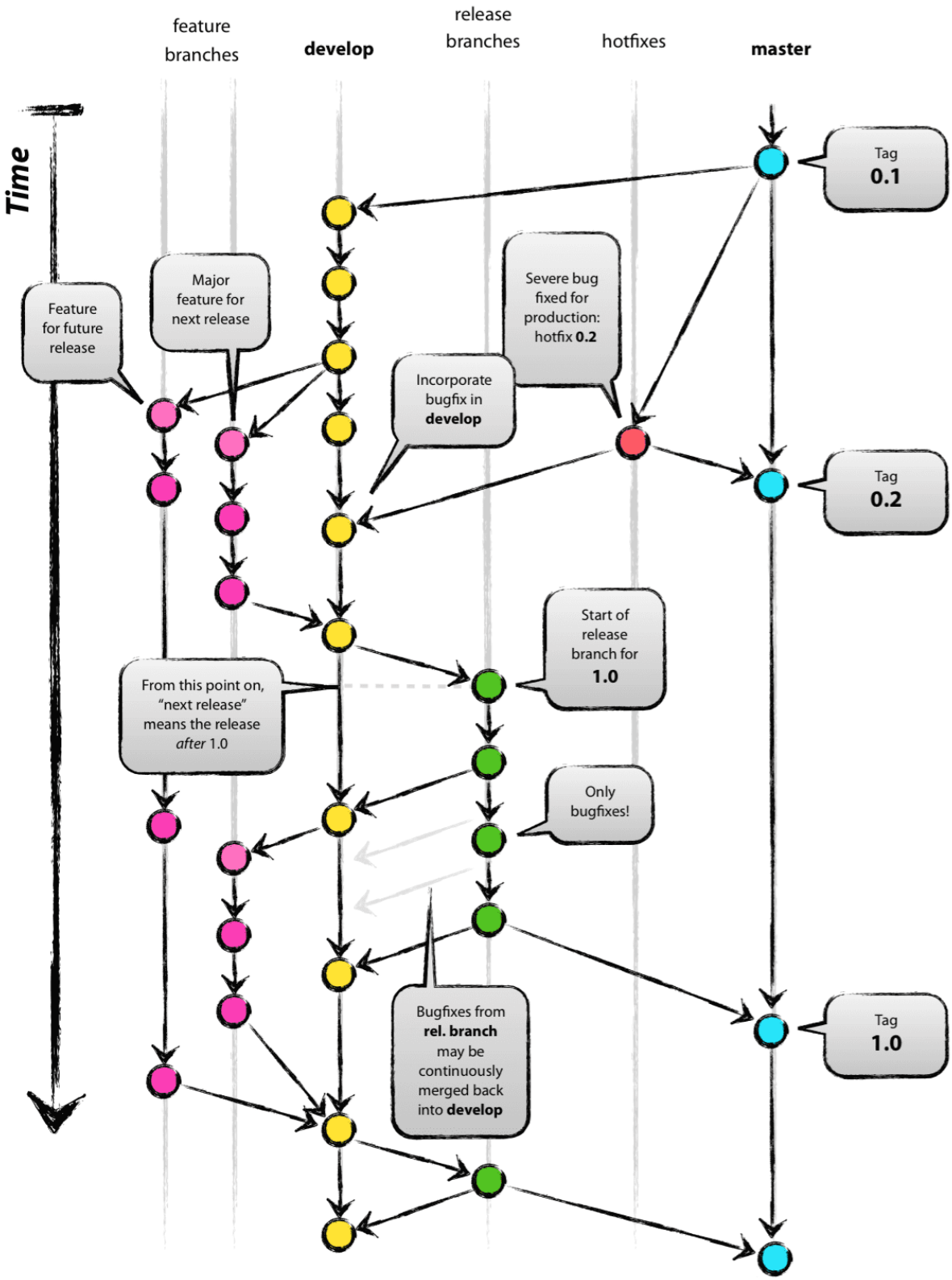
Toteutettavien ominaisuuksien ja muutospyyntöjen hallinnan helpottamiseksi projektissa päätettiin käyttää JIRA-tehtävienhallintaohjelmistoa. JIRA [25] on Atlassianin kehittämä monipuolisia projektinhallintaominaisuuksia sisältävä sovellus, jonka avulla projektin toteutuksen etenemiseen liittyvät seikat on yksinkertaista kerätä yhteen paikkaan. JIRA:ssa tehtävät voivat sisältää erilaisia tietoja kuvauksesta vastuuhenkilöön ja kommenteista työmääräarvioon. Tehtäviin voi liittää myös tietoja sovelluksen versioihin liittyen ja JIRA:n integroiminen versionhallintaan tarjoaa mahdollisuuden bugien sekä muiden ongelmien tilan seuraamiseen. Projektin aikana JIRA:aan kirjattiin noin kaksisataa tehtävää ja sitä käytettiin myös apuna iteraatiopalavereissa, kun tarkasteltiin jo toteutettuja ja tulevia ominaisuuksia sekä projektin yleistä etenemistä.

Koska toteutus oli päätetty tehdä ketterien menetelmien mukaisesti, otettiin käyttöön JIRA:n Agile-lisäosa [26]. JIRA Agile parantaa ketterien projektien hallintaa uusien tehtävienhallintanäkymien avulla. Projektin kaikki tehtävät näytetään suunnittelunäkymässä ryhmiteltynä iteraatiojaksoihin ja tehtäviin, jotka eivät vielä kuulu mihinkään jaksoon. Samassa näkymässä näytetään myös tehtävien työmääräarviot, mikä on kätevää oikean tehtävämäärän valinnan kannalta. Tehtävien järjestys listalla vastaa niiden toteutusprioriteettia, ylempänä olevat tehtävät ovat korkeammalla prioriteetilla. Kun haluttu iteraatiojakso on määritelty, se voidaan käynnistää. Tämän jälkeen käytettävissä on työskentelynäkymä, jossa tehtävät on jaettu kolmeen palstaan niiden tilan mukaan. Tässä näkymässä tehtävien siirtäminen toteutukseen tai valmiiksi on suoraviivaista ja samalla näkymä havainnollistaa kyseisen jakson tilanteen nopeasti.

5.2.4 Git

Verkkokauppaprojektin tuotosten taltioimiseen käytettiin Git-versionhallintaa [27], joka valikoitui käytettäväksi lähinnä aiempien projektien positiivisten kokemusten perusteella ja siksi, että valmis infrastruktuuri Gitin käyttöön oli jo olemassa. Git on alkujaan Linus Torvaldsin kehittämä hajautettu versionhallintajärjestelmä, joka mahdollistaa isojenkin projektien tehokkaan käsittelemisen. Gitissä muutokset siirretään työhakemistosta ensin

paikalliselle staging area -alueelle, josta ne puolestaan siirretään commit-toimenpiteellä palvelimelle. Tämän menettelyn myötä voidaan tarvittaessa siirtää vain palasia tiedostoista, mikä on hyödyllistä monissa tilanteissa.



Kuva 5.1. *Gitflow-toimintamalli [28].*

Git soveltuu hyvin sovelluskehitykseen, jossa toteutetaan useita ominaisuuksia yhtä aikaa, sillä se tarjoaa nopeat työkalut versioiden haarautumiseen ja haarojen yhdistämiseen. Toteutetun projektin tapauksessa Gitin käytössä hyödynnettiin kuvassa 5.1 kuvattua Gitflow-menetelmää [28]. Gitflow yhtenäistää Gitin käyttöä ja tarjoaa pohjan erilaisille toimintamalleille. Gitflow on toimintamalli, jossa versionhallintaan perustetaan ensin kaksi oletushaaraa, master ja develop. Master-haaran tilan tulee olla aina tuotantovalmis, kun taas develop-haara sisältää viimeisimmät kehitystulokset. Develop-haara yhdistetään julkaisun yhteydessä master-haaraan.

Näiden päähaarojen lisäksi käytetään ominaisuushaaroja, julkaisuhaaroja ja paikkaushaaroja [28]. Ominaisuushaarat ovat develop-haarasta erillään kehitettyjä ominaisuuksien mukaan erotettuja kokonaisuuksia, jotka yhdistetään takaisin develop-haaraan niiden valmistuttua. Julkaisuhaarat ovat julkaisun valmistelua ja ne sisältävät vain pieniä muutoksia, joiden avulla kehitysversio julkaistaan viimeisteltäväksi. Valmistuttuaan julkaisuhaarat yhdistetään master-haaraan uutena versiona ja mahdolliset muutokset tuodaan develop-haaraan jatkokehitystä varten. Paikkaushaarat lähtevät master-haarasta ja sisältävät korjauksia jo tuotantokäytössä olevaan sisältöön. Paikkaukset lisätään master-haaraan uuden version luomien myötä ja myös develop-haaraan, jotta tuotantoversiot sisältävät jatkossakin tehdyt korjaukset.

5.3 Käyttöliittymä

Asiakkaan toiveet käyttöliittymän suhteen liittyivät helppokäyttöisyyteen ja toimivuuteen eri moderneilla selaimilla. Käyttöliittymän toteuttamiseen käytettiin sivujen rakenteen määrittelyyn HTML5:tä, ulkoasun määrittelyyn CSS3:a ja dynaamisten toiminnallisuuksien mahdollistamiseen JavaScriptiä. Nämä yhdistettiin Play Frameworkin sivupohjamekanismiin, joka mahdollistaa Scala-koodin käyttämisen sivujen rakenteen muodostamisen apuna. Play Frameworkin sivupohjat [29] ovat HTML-sivuja, joita kuorrutetaan oman syntaksin avulla Scalan rakenteilla.

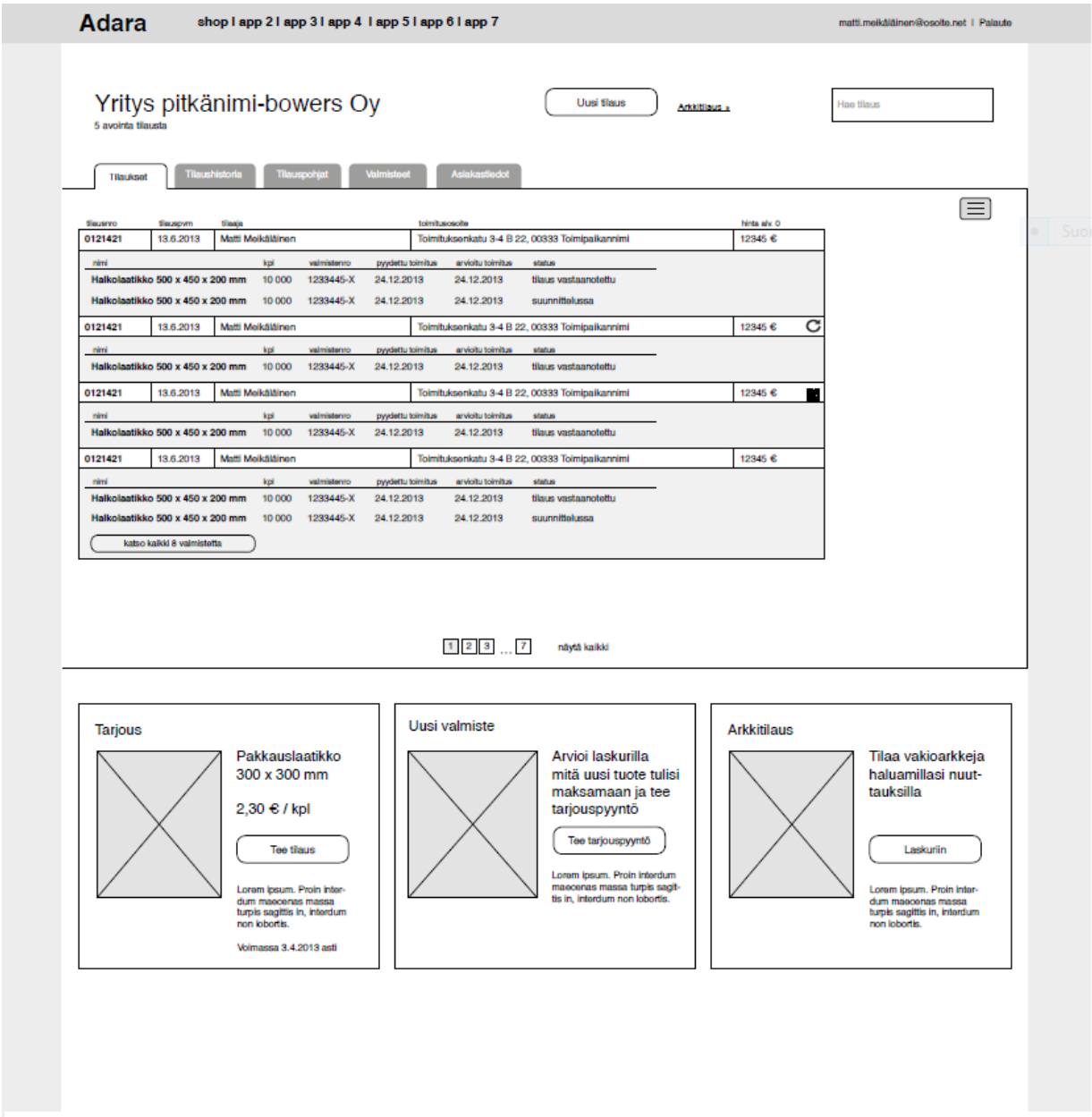
Tarkastellaan seuraavaa esimerkkiä erittäin yksinkertaisesta sivupohjasta:

```
@(asiakas: Asiakas, tilaukset: List[Tilaus])

<h1>Tervetuloa, @asiakas.nimi!</h1>

<ul>
@for(tilaus <- tilaukset) {
  <li>@tilaus.tilausnumero</li>
}
</ul>
```

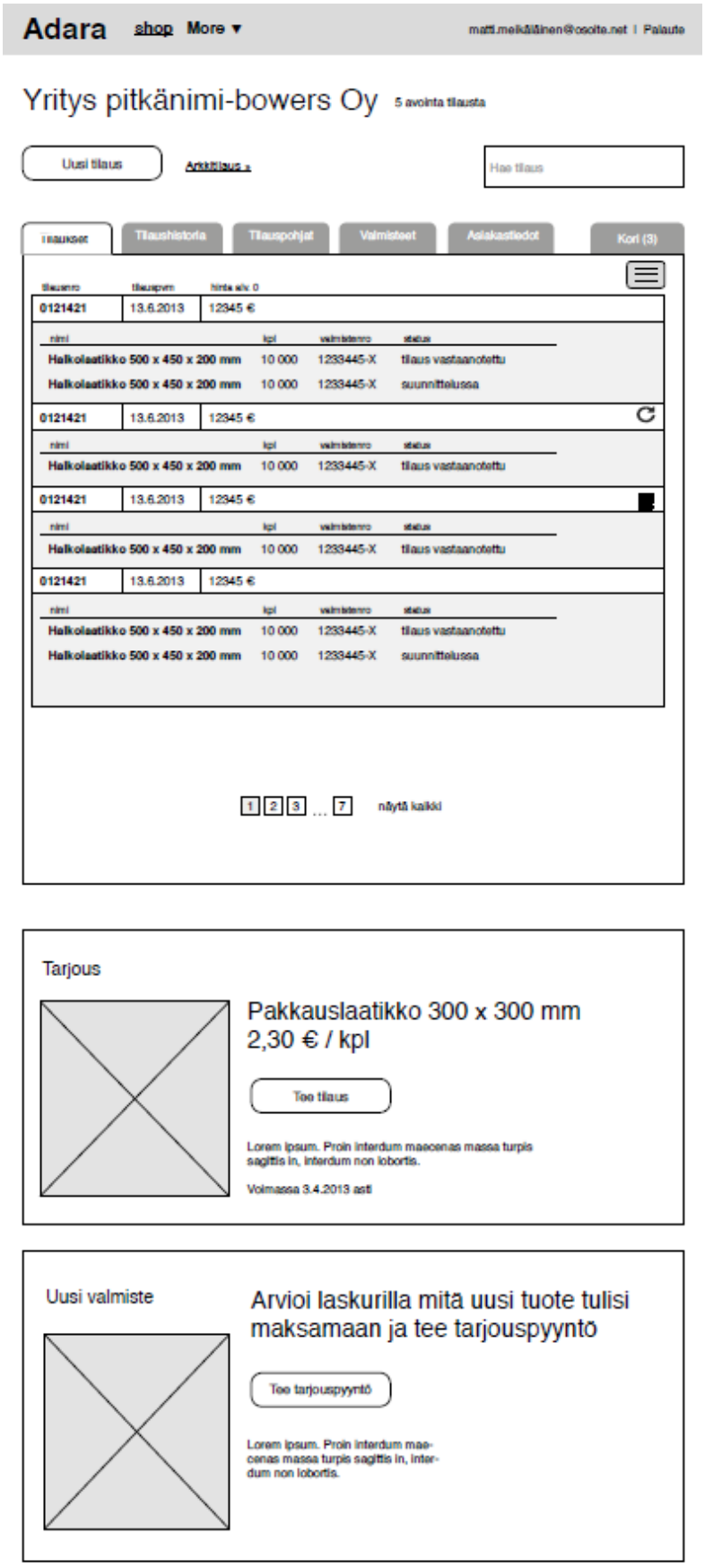
Sivupohjille voidaan antaa kontrollereista kutsuttaessa parametreja, jotka sisältävät muodostettavalla sivulla esitettävän datan. Tässä esimerkissä sivupohja saa parametreinaan asiakkaan ja listan asiakkaan tilauksista. @-merkin avulla sivupohjakielessä käytetään

näitä parametreja ja tulostetaan tarvittaessa niiden sisältämiä arvoja, kuten asiakkaan nimi tai tilauksen tilausnumero. @-merkillä voidaan myös käyttää funktioita, esimerkissä forsilmukkaa käytetään koko tilauslistan numeroiden tulostamiseen. Scala-pohjainen syntaksi kirjoitetaan HTML-merkkauksen keskelle. Käytettävissä on lisäksi muun muassa ehtorakenteita ja funktioita muistuttavia uudelleenkäytettäviä koodilohkoja [29]. Include-mekanismin avulla voidaan välttää toistoa sivurakenteissa. Play Frameworkin sivupohjamoottori tukee oletuksena yleisimpiä sivupohjatyyppejä, kuten HTML:ää ja XML:ää, mutta myös omien tyyppien määrittely on tarvittaessa mahdollista.

Koska verkkokaupasta haluttiin myös mobiililaitteilla toimiva, päätettiin määrittelyä ja toivottuja ominaisuuksia lähestyä responsiivisen eli eri laitteille mukautuvan ulkoasun suunnalta. Ensimmäisessä vaiheessa graafikko toteutti kuvissa 5.2 ja 5.3 esimerkkeinä



Kuva 5.2. *Verkkokaupan käyttöliittymän hahmotelma isoille näytöille.*

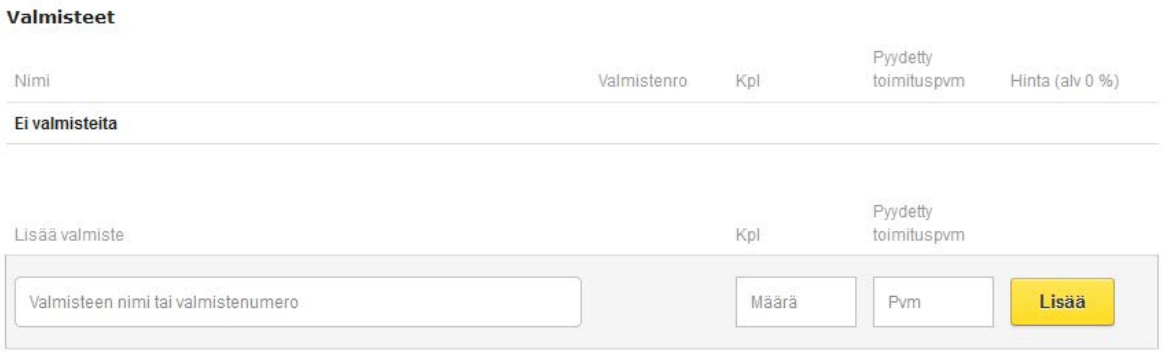


Kuva 5.3. *Verkkokaupan käyttöliittymän hahmotelma mobiililaitteille.*

nähtävät hahmotelmat verkkokaupan perusnäkymistä. Hahmotelmat edustivat kolmea eri laitekokoa: tietokoneen näyttöä, tablettilaitetta ja kännykkää tai muuta vastaavaa pieninäyttöistä laitetta.

Hahmotelmien kautta voitiin helposti havainnollistaa toimintojen sijoittelua jo aikaisessa vaiheessa ja ne toimivat myös hyvänä pohjana tehtävien määrittelylle ja myöhemmin todelliselle toteutukselle. Tarkastelemalla hahmotelmia voidaan löytää responsiivisen suunnittelun kannalta tärkeitä perusperiaatteita, jotka liittyvät erityisesti näyttötilan leveyteen. Yläpalkin valikko pienenee mobiililaitteilla pudotusvalikoksi, mikä vähentää tilantarvetta, mutta pitää ominaisuudet tästä huolimatta saatavilla. Tilauspainikkeiden sijoittelu vaihtaa paikkaa käytettävissä olevan tilan mukaan, pienellä näytöllä ne siirtyvät alemmas. Myös tilausrivien tiedot mukautuvat pienellä näytöllä siten, että vähemmän kriittiset tiedot piilotetaan. Alareunan nostopaikat noudattavat mobiililaitteella blokkimaista asettelua, jossa ne asetellaan allekkain.

Tilausten syöttämisestä haluttiin tehdä mahdollisimman intuitiivista käyttäjien kannalta ja samalla myös tilattavien valmisteiden hinnat haluttiin näyttää ajantasaiseen tietoon perustuen. Tätä varten tilausnäkymään toteutettiin kuvassa 5.4 nähtävä lisäystoiminto, jonka avulla käyttäjä voi lisätä tilaukselle tuotteen valmisteen syöttää sen nimeä, toimittajan valmistenumeroa tai asiakkaan omaa valmistenumeroa. Tämän jälkeen kentän alapuolella näytetään asiakkaan valmistelistauksen perusteella ehdotukset, joista klikkaamalla käyttäjä voi valita haluamansa valmisteen. Lopuksi tilausriville kerrotaan vielä haluttu kappalemäärä ja pyydetty toimituspäivä. Toimituspäivän valintaa helpottamaan kyseiseen kenttään lisättiin kalenterivalitsin.

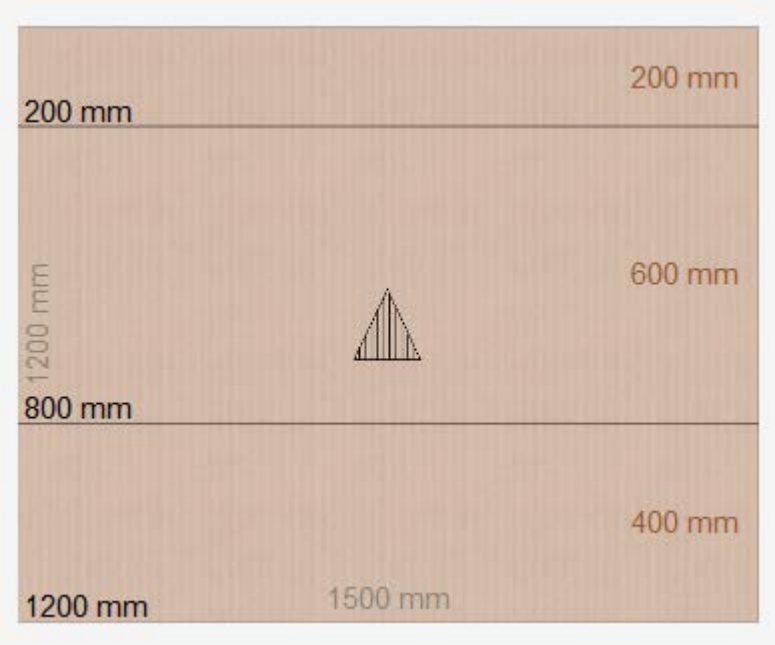


Kuva 5.4. *Verkkokaupan tilausnäkymän valmistevalinta.*

Kun käyttäjä napsauttaa tilausrivin lisäyspainiketta, siirretään valmisterivi ylempään lisättyjen valmisteiden listaan. Tilausrivien kautta käyttäjä pääsee myös halutessaan tarkastelemaan valmisteiden tarkempia tietoja niiden nimiä klikkaamalla. Tässä yhteydessä valmisteelle päätellään hinta tarkastelemalla voimassa olevia hintaportaita ja tilattua kappalemäärää. Kappalemäärän ja toimituspäivän muuttaminen on mahdollista vielä rivikohtaisesti lisäämisen jälkeenkin. Tällaisten muutosten yhteydessä päivitetään myös tilausrivin hinta. Kuluttajaverkkokaupoista poikkeava toimintatapa tekee tilausten tekemisestä

nopeampaa ja tehokkaampaa, koska tilausrivien lisäys onnistuu suoraviivaisesti ilman tarvetta tuotteiden selaamiselle.

Eräs toimialaan liittyvä erityisesti tätä verkkokauppaa varten luotu käyttöliittymäratkaisu löytyy arkkitilausnäkymästä. Arkkitilausnäkymässä käyttäjät määrittelevät tilattavalle arkille erilaisia vaatimuksia käytettävistä materiaaleista ja toimitusajasta arkin kokoon. Asiakkaalle haluttiin havainnollistaa syötettyjen tietojen perusteella saavutettava lopputulos, mistä johtuen näkymään toteutettiin kuvassa 5.5 nähtävä esikatselukuva arkista.



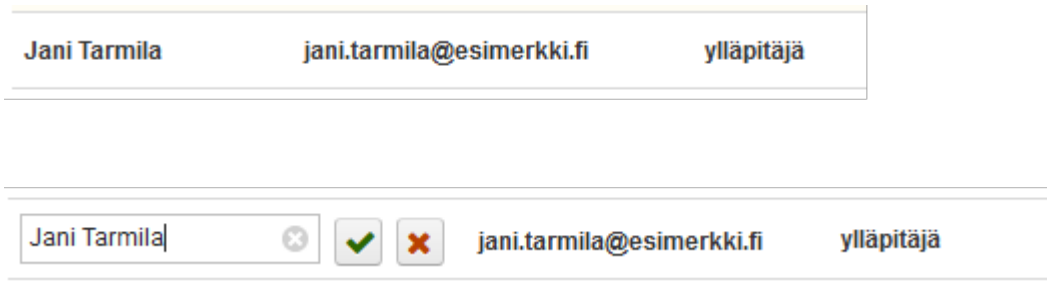
Kuva 5.5. *Arkkitilausnäkymän esikatselu.*

JavaScript-piirtokirjaston avulla canvas-pohjaisesti toteutetusta arkin esikatselukuvasta asiakas näkee selkeästi tilattavan arkin mittasuhteet ja nuuttausten eli arkkiin valmiiksi tehtävien taitosten etäisyydet toisistaan ja arkin yläreunasta. Keskellä oleva symboli kuvaa arkin aaltojen suunnan, mikä on tärkeä tieto erilaisia pakkauksia suunniteltaessa. Esikatselun taustalla käytetty aaltopahvikuva kiinnittää esikatselukuvan paremmin aihepiiriinsä. Esikatselua päivitetään dynaamisesti käyttäjän syöttämien tietojen mukaan, mikä mahdollistaa esimerkiksi nuuttausten asettelun testaamisen ja virheellisten valintojen indikoinnin myös kuvallisesti.

Aiemmassa verkkokaupassa useamman asiakasnumeron käyttö oli toteutettu siten, että käyttäjälle voitiin lisätä useampia asiakasnumeroita, joiden kaikkien tiedot näytettiin yhtä aikaa esimerkiksi listausnäkymissä. Käytännössä tämä teki listausnäkymistä raskaita ja osittain vaikeasti tulkittavia. Lisäksi tilauksen teon yhteydessä ja muissa näkymissä, joissa tiedot pitää kohdistaa tietylle asiakasnumerolle, pakotettiin käyttäjä joka kerta valitsemaan haluttu asiakasnumero uudestaan.

Uudistuksen yhteydessä toiminta useamman asiakasnumeron kanssa muutettiin selkeämmäksi. Kukin asiakasnumero vastaa yhtä yritystä, johon puolestaan voidaan erikseen liittää käyttäjiä. Yksittäinen käyttäjä voidaan lisätä haluttuihin yrityksiin, jonka jälkeen käyttäjälle näytetään mahdollisuus yrityksen valintaan ja vaihtamiseen. Tällä tavoin jokaisessa näkymässä näytetään aina vain yhden yrityksen tietoja, mikä vähentää sekavuutta ja poistaa tarpeen yrityksen jatkuvalle uudelleenvalinnalle.

Kuten muissakin tietojärjestelmissä, myös verkkokaupassa on tietoja, joita käyttäjät eivät joudu muokkaamaan kovinkaan usein, mutta muokkauksen täytyy kuitenkin olla mahdollista. Erillisten muokkausnäkymien toteuttaminen ja ylläpito lisäävät työkuormaa, mutta näkymät saattavat olla myös käyttäjän kannalta haasteellisia, koska yhteys muokattavaan tietoon voi kadota siirryttäessä näkymästä toiseen. Verkkokaupan toteutuksen yhteydessä tähän seikkaan kiinnitettiin huomiota ottamalla käyttöön in-place-muokkausmahdollisuus muun muassa käyttäjätilien muokkaamisen yhteydessä.



Kuva 5.6. *Käyttäjän nimen muokkaaminen in-place-muokkaimella.*

Kuvassa 5.6 ylempänä on esimerkki käyttäjälistauksen rivistä, jolla esitetään käyttäjään liittyviä perustietoja. Kun käyttäjä vie hiiren osoittimen jonkin tiedon päälle, näytetään ohjeteksti, joka kehottaa klikkaamaan tiedon muokkaamiseksi. Käyttäjän klikattua kyseistä tietoa vaihdetaan tiedon paikalle dynaamisesti JavaScriptilla muokkausnäkymä, jossa tietoa voi muokata ja tehdyt muokkaukset tallentaa tai hylätä. Tällä tavoin toimimalla käyttäjä näkee selkeästi, mitä tietoa on muokkaamassa ja muokkauksen jälkeen lista on välittömästi ajan tasalla myös muokattujen tietojen osalta.

6. ARVIOINTI

Web-sovellusten uudistusprojektit ovat aina haastavia, toisaalta johtuen aiempien käytössä olleiden järjestelmien asettamista lähtökohdista ja mahdollisesta käyttäjien muutosvastarinnasta, mutta myös tarpeellisten ominaisuuksien löytämisen näkökulmasta. Projektin toteutuksessa on hyvä muistaa huomioida asiakkaan toiveet ja tarpeet sekä tehdä järjestelmän käytöstä mahdollisimman helppoa loppukäyttäjien kannalta.

Tässä luvussa pohditaan verkkokauppauudistukseen valittujen toteutusteknologioiden sopivuutta ja käytettyjen työkalujen toimivuutta. Lisäksi luku esittelee uudistuksella saavutetut hyödyt ja käy läpi projektin aikana ja myöhemmin heränneet jatkokehitysajatukset.

6.1 Toteutusteknologioiden ja menetelmien sopivuus

Asiakkaan asettamien lähtökohtien mukaan toteutusteknologioiden valinta oli melko vapaa, joten mitään erityistä pakkoa funktionaalisuutta tukevien teknologioiden valintaan ei olisi ollut. Koska vapaus oli kuitenkin olemassa, ei valintaa haluttu rajata jo aiemmin kokeiltuihin ja tunnettuihin teknologioihin. Valinnan tekemisessä hyödynnettiin kokeneempien työntekijöiden näkemyksiä ja erilaisia listauksia, joissa kuvailtiin teknologioiden suosion voimakkuutta.

Funktionaalisuuden suuntaan valintaa ohjasi halu oppia uutta ja erilaista perinteisiin proseduraalisiin ja olio-ohjelmointiin perustuviin ohjelmointikieliin. Lopullisiin valintoihin vaikutti se myös, että Typesafella oli valmiiksi tarjolla monipuolinen alustakokonaisuus [16], johon suuri osa valituista teknologioista kuului. Kokonaisuus vahvisti uskoa siihen, että teknologiat toimisivat hyvin yhdessä ja niiden mahdollisiin ongelmiin olisi tarjolla apua eri kanavien välityksellä. Toisaalta myös muilta ohjelmistokehittäjiltä oli kerääntynyt hyviä kokemuksia valittujen teknologioiden osalta ja ne olivat heidän mukaansa käytössä myös heidän projekteissaan.

Scalan oppimiskynnys vaikutti alkuun korkealta, mutta sen tiivis liitto Javan kanssa [15] pienensi pahimpia riskejä ja uhkakuvia. Eräs ehdoton etu oli myös Javalle toteutettujen kirjastojen käyttömahdollisuus projektissa, sillä tämä laajensi valmiiden mahdollisuuksien hyödyntämistä merkittävästi. Erityisesti projektin alussa aikaa kului Scalan hienouksien haltuun ottamiseen, mutta työtuntien lisääntymisen myötä itsevarmuus ominaisuuksien käyttöön kasvoi ja ymmärryksen lisääntyessä erityisesti Scalan tietorakenteiden ja niiden välisten muunnosten hyödyt tulivat selkeämmin esille. Käytännön tasolla Scalan käyttö tehosti työskentelyä, koska sen kompaktimpi syntaksi vähensi tarvittavan koodin määrää.

Play Framework tarjosi hyvän lähtökohdan web-sovelluksen kehittämiselle, koska se sisälsi kaikki yleisimmät tarvitut ominaisuudet web-sivujen muodostamisesta lähtien. Erityisen kätevää oli Playn tarjoama tapa paketoida sovellus zip-tiedostoon käynnistysskripteineen, mikä helpotti uusien versioiden julkaisemista niin testi- kuin tuotantoympäristöönkin. Tämä mahdollisti kohtuullisen vaivattoman julkaisuprosessin, mutta toisaalta pienienkin muutosten teko vaati aina uuden paketoinnin, mikä oli toisinaan bugikorjausten kannalta rasite. Prosessi oli kuitenkin niin suoraviivainen, että kovin suurta ajanhukkaa tästä ongelmasta ei syntynyt.

Myös kehitysaikana Play Framework tarjosi etua automaattisen käännöksen myötä. Kehittäjän kannalta oli kätevää, että kehitysympäristön ja selaimen lisäksi ei tarvinnut erikseen käydä vielä joka välissä käskyttämässä kääntäjää, vaan sovelluskehys osasi hoitaa tämän tarvittaessa. Käyttöliittymän toteuttamisen yhteydessä haasteeksi muodostui Playn validoinnin vanhanaikaisuus. Validointi onnistui kyllä palvelinpäässä lomakkeen rakenteen mukaan kohtalaisen monipuolisesti, mutta minkäänlaista mahdollisuutta dynaamiseen validointiin Play ei sisältänyt. Tämä oli lievä pettymys, koska sivulatauksen odottaminen esimerkiksi lomakkeen lähettämisen yhteydessä rasittaa palvelintakin turhaan.

Kaikki verkkokaupan kannalta tärkeät ominaisuudet, joita Play ei itsessään sisältänyt, löytyivät valmiina lisäosina tai Java-kirjastoina, joiden käyttöönotto oli yksinkertaista. Ainoat isommat haasteet esiintyivät autentikoinnin toteuttavassa kirjastossa. Osa kirjaston dokumentaation mukaisista funktioista ja ominaisuuksista ei tuntunut toimivan halutulla tavalla. Myös dokumentaation sisältävä sivusto oli välillä pois käytöstä, mikä hankaloitti autentikaation rakentamista. Mikäli tarpeena olisi ollut monimutkaisempi autentikaatiojärjestelmä, olisi ollut järkevää harkita jo oman lisäosan toteuttamista, mutta haasteista huolimatta valittu ratkaisu riitti tähän sovellukseen.

Application DB:n oman tietokannan osalta projektin aikana kaikki sujui hyvin. PostgreSQL osoittautui toimivaksi valinnaksi ja se sisälsi kaikki tarvitut ominaisuudet ja toimi myös tuotantokäytössä luotettavasti. MySQL:ään verrattuna erityisesti komentorivin kautta tietokannan tarkasteluun käytettävät komennot vaativat dokumentaation tarkastelua, mutta käytön myötä nekin tulivat tutuiksi.

Tietokannan hallintaan käytetty Slick vaati enemmän käsityötä kuin esimerkiksi Hibernaten kaltaiset automatisoidummin toimivat olio-relaatio-mallinnustyökalut, mutta vastavuoroisesti Slickin etuna olivat paremmat mahdollisuudet erilaisten piirteiden tarkempaan määrittelyyn käsin. Slickin kautta tietokannan käyttö oli melko yksinkertaista, mutta esimerkiksi lisäysoperaatioiden syntaksi oli alkuun haasteellinen Scala-luonteensa takia. Tässäkin tapauksessa pahimman oppimiskynnyksen ylittämisen jälkeen käyttö oli nopeaa.

Verkkokaupan toteuttamisen kannalta kenties kaikkein suurimman haasteen aiheutti asiakkaan toiminnanohjausjärjestelmään integroituminen. Tässä apuna käytettiin jOOQ-kirjastoa, joka osasi muodostaa tietokannasta oliomallin yhden komentorivikomennon avulla. Ilman jOOQ:n käyttöä tietokannan rakenteen selvittelyyn ja mallintamiseen sekä käyttöön olisi tuhlaantunut huomattavasti paljon isompi aika, koska rakenne oli niin epäselvä. Mitään ihmeitä jOOQ:n avulla ei kuitenkaan saatu aikaan, mutta sen tarjoama apu oli kuitenkin korvaamatonta tätä rajapintaa rakennettaessa. Tietty myös projektin edetessä ymmärrys toiminnanohjausjärjestelmän tietokannan suhteen karttui ja tätä myötä toiminta sen kanssa nopeutui.

Eräs valitun arkkitehtuurin aiheuttama ongelma oli kokonaisuuksien käyttämien oliomallien yhteensovittaminen. Käytännössä oma mallinsa oli toiminnanohjausjärjestelmästä tulevalle datalle, Application DB:n sisäiselle datalle ja Webshopin tietojen esittämiseen käyttämälle mallikerrokselle. Toiminnanohjausjärjestelmän tapauksessa malli oli muunnettava käsin tarvittavilta osin Application DB:n sisäiseen muotoon, eikä mallia ollut järkevä yhtenäistää toiminnanohjausjärjestelmän tietokannan vähintäänkin haastavan rakenteen vuoksi. Webshopin ja Application DB:n välinen tiedonsiirto aiheutti käytännössä sen, että tälläkään välillä mallit eivät voineet olla täysin samanlaisia, mutta automaattinen JSON-muunnos yksinkertaisti niiden siirtämistä huomattavasti toiseen tapaukseen verrattuna.

Kaiken kaikkiaan minkään valitun toteutusmenetelmän kanssa ei projektin aikana ilmennyt kovinkaan suuria ongelmia. Oppimiseen käytetty aika oli hieman alussa arvioitua suurempi, mutta sitä selittävät teknologioiden vieraus ja funktionaalisen paradigman erilaisuus. Nämä eivät lopulta muodostuneet minkäänlaiseksi ongelmaksi, sillä kaikkiin haasteisiin onnistuttiin löytämään toteutuksen kannalta sopiva ratkaisu. Loppujen lopuksi funktionaalisuuden merkitys toteutusmenetelmien kannalta oli kohtuullisen pieni, mutta sen käytöstä saatu toimintavarmuus oli lopputuloksen kannalta tärkeää.

Projektin käynnistyessä valittu ketterä kehitystapa osoittautui toimivaksi, koska se mahdollisti projektin etenemisen helpon ja ajantasaisen seurannan sekä tarjosi asiakkaalle mahdollisuuden tehdä muutoksia haluamallaan tavalla myös matkan varrella ilman isompaa byrokratiaa tai sopimusmuutoksia. Palavereiden toistuminen tietyllä aikataululla mahdollisti sen, että sopivan ajankohdan valitsemiseen ei yleensä kulunut ylimääräistä aikaa.

Ketteryyden kannalta haasteena voidaan pitää sitä, että sekä toteuttajalta että asiakkaalta vaaditaan jatkuvaa aktiivisuutta järjestelmän testaamisen suhteen, jotta korjaustarpeet löydetään mahdollisimman aikaisessa vaiheessa. Projektin keskivaiheilla testausaktiivisuudessa koettiin pieni notkahdus, joka näkyi lisääntyneinä korjausmäärinä lähestyttäessä loppua.

Ketterän kehityksen käyttöä tuki myös se, että tärkeimmän rajapinnan muodostanut toiminnanohjausjärjestelmä oli erityisesti projektin alussa toteuttajille entuudestaan tuntematon. Mikäli toteutus olisi tehty porrastetummalla mallilla, olisi alkuvaiheessa pitänyt käyttää huomattavan paljon aikaa toiminnanohjausjärjestelmään liittyvien seikkojen selvittämiseen, mikä olisi puolestaan hidastanut projektin käynnistymistä merkittävästi. Nyt tietämys karttui vähitellen ominaisuuksien toteuttamisen myötä, joten käyttökelpoisia ominaisuuksia saatiin testaukseen nopeammin.

Alkuperäisiin alustaviin arvioihin verrattuna projektin kesto venyi jonkin verran, mikä johtui pääosin siitä, että alkuvaiheessa kaikki ominaisuudet eivät olleet riittävän tarkasti määriteltyjä ja ulkoiset rajapinnat olivat entuudestaan tuntemattomia. Projektin edetessä toimintoihin lisättiin erilaisia käyttäjän toimintaa helpottavia ominaisuuksia, jotka myös kasvattivat työmäärää. Uusien teknologioiden opettelun vaikutus ei ollut merkittävä lopullista työmäärää ajatellen, sillä oppiminen tapahtui yleensä toteuttamisen yhteydessä.

6.2 Käytettyjen työkalujen toimivuus

Toteuttamiseen käytetyt työkalut valittiin sekä teknologioiden yhteydestä kerättyjen suositusten että aiempien käyttökokemusten perusteella. Scalan yhteydessä tarvittavat työkalut olivat toteuttajille entuudestaan enimmäkseen tuntemattomia, mikä aiheutti tässäkin yhteydessä tarpeen uuden oppimiselle. Projektinhallintaan liittyvien työkalujen tuntemus kuitenkin kompensoi tätä oppimistarvetta, sillä niiden käyttöön oli muodostunut rutiineja jo aiemmissa projekteissa.

Suurimmat ongelmat työkalujen suhteen liittyivät käytettyyn ohjelmointiympäristöön eli Scala IDE:een. Tämä oli ikävää siinä mielessä, että kyseinen työkalu oli projektin kannalta ehdottomasti käytetyin, sillä koko projektin toteutus kirjoitettiin sen avulla. Ongelmat esiintyivät ensisijaisesti ympäristön yleisenä hitautena ja kaatuiluna, mikä hidasti päivittäisen työn tekemistä. Joissain tapauksissa yksittäisen tiedoston avaaminen saattoi kestää kymmeniä sekunteja ja toisinaan ympäristö hyytyi täysin tallentaessaan muutettua tiedostoa. Pahimmillaan tämä aiheutti tehtyjen muutosten katoamisen ympäristön kaatuessa ilman mitään sen kummempaa syytä.

Muita ongelmia Scala IDE:een liittyen olivat kummalliset virheilmoitukset integroituun Scala-kääntäjään liittyen ja kääntäjän monessa tilanteessa esittämät turhat varoitukset ja väärät virheilmoitukset. Ympäristöllä tuntui olevan ongelmia etenkin joidenkin riippuvuuksien ymmärtämisessä tiedostojen muuttuessa. Työskentelyä hidasti myös automaattisen täydennyksen satunnainen jumiutuminen kesken toimenpiteen tai ehdotusten puuttuminen kokonaan.

Ohjelman hallitsemattomiin keskeytymisiin ja automaattiseen täydennykseen liittyvät ongelmat johtuivat ilmeisesti pääosin taustalla olevan Eclipsen muistinkäytöstä ja Scala

IDE:n käytön vaatimien lisäosien epäkypsyydestä. Myöhemmät versiot vaikuttavat korjanneen pahimmat ongelmat tähän liittyen, mutta korjauksista ei harmittavasti ollut hyötyä vielä verkkokaupan kehitysprosessin aikana.

Käännöstyökaluna käytetty sbt toimi enimmäkseen moitteettomasti ja nopeutti projektin ajamista ja julkaisemista. Projektin keskivaiheilla sbt:n kanssa kohdattiin joitain riippuvuusongelmia, jotka johtuivat lähinnä riippuvuuksista ja paketinhallinnan puutteista tiettyjen kirjastoversioiden suhteen. Nämä ongelmat ratkesivat kuitenkin taustalla, kun paketinhallinnan sisältämät kirjastoversiot päivittyivät vastaamaan tarvittuja. Käytännössä eniten opettelua sbt:n osalta kului sen määrittelytiedostojen rakenteen ymmärtämiseen, mutta perusrakenteen määrittelyn jälkeen lisäysten tekeminen oli yksinkertaista.

JIRA:n toimi tehtävienhallinnan apuna sujuvasti ja mahdollisti projektin etenemisen helpon seurannan ja kätevän työnjaon. Opetteluun ei tarvittu ylimääräistä aikaa, koska JIRA oli koko projektiryhmälle jo entuudestaan tuttu. Koska käytetty JIRA-instanssi oli useamman projektin kesken jaettu, jouduttiin kesken projektin vaihtamaan käytössä olevaa lisenssiä tarvittavan henkilömäärän kasvaessa. Tästä ei kuitenkaan aiheutunut pitkää katkoa tehtävienhallinnan toiminnan kannalta.

Myös versionhallinnassa käytetty Git oli projektiryhmän jäsenille aiemmista projekteista tuttu, joten sen käyttö sujui jo rutiinilla. Valittu toimintamalli Gitflow tehosti toimintaa ja yhtenäisti versionhallintapuun rakennetta. Malli helpotti myös julkaistavien versioiden hallintaa ja muutosten vientiä oikeisiin paikkoihin. Ominaisuushaarat mahdollistivat useamman ominaisuuden yhtäaikaisen kehittämisen toisistaan riippumatta.

6.3 Saavutetut hyödyt

Uudistuksen tärkeimmät lähtökohdat olivat toimivuus moderneilla selaimilla ja erilaisilla laitteilla sekä käyttäjien toiminnan helpottaminen ja nopeuttaminen. Modernien selainten kannalta tavoite saavutettiin käyttämällä käyttöliittymän toteuttamiseen teknologioita, joiden asema on vakiintunut web-sovellusten osalta. Erilaisten laitteiden toimivuus varmistettiin toteuttamalla käyttöliittymä responsiivisena huomoiden kolmen eri laitekoon käyttötilanteet. Samalla mahdollistettiin myös esimerkiksi verkkokaupan QR-tilauspohjien kannalta tärkeä mobiililaitekäyttö. Käyttäjien kannalta uudistus toi mukanaan myös paljon erilaisia käteviä ominaisuuksia ja mahdollisuuden entistä parempaan sekä selkeämpään tilausten tilan seurantaan.

Monipuolistamalla käyttäjien mahdollisuuksia eri selainten ja laitteiden käytön kautta voitiin verkkokauppaan lisätä myös uusia käyttäjiä, jotka eivät voineet käyttää vanhaa verkkokauppatoteutusta. Käytännössä tämä tarkoitti sähköisten kanavien kautta tulevien tilausten lisääntymistä, mikä mahdollistaa toiminnan tehokkaamman suunnittelun ja vä-

hentää turhaa tilausten toisteista näppäilyä käsin esimerkiksi sähköpostin tietojen perusteella. Uudenlaiset tilausominaisuudet, kuten QR-tilaukset tekivät tilausten tekemisestä entistä kätevämpää myös käyttäjien näkökulmasta.

Toteutetun verkkokauppauudistuksen tuloksena syntyi entistä helpommin ylläpidettävä ja juuri asiakkaan tarpeisiin ja toiveisiin räätälöity ratkaisu. Aiempaan verrattuna muutosten tekeminen on jatkossa huomattavasti nopeampaa ja yksinkertaisempaa, mikä kannustaa innovoimaan uudenlaisia toimintoja myös tulevaisuudessa. Jatkoprojektien kannalta hyödyksi on myös verkkokauppauudistuksen aikana kehittynyt toimialatuntemus ja asiakkaan toimintatapojen tuntemus, mikä vähentää tarvittavan tutkimustyön määrää uusia projekteja aloitettaessa.

6.4 Jatkokehitysajatukset

Verkkokauppauudistuksen aikana syntyi muutamia ideoita jatkokehityksen kannalta. Nämä ideat on hyvä pitää mielessä, kun verkkokaupan toimintaa mietitään tulevaisuutta ajatellen ja ne yleistyvät myös useimpiin muihin vastaaviin ympäristöihin ja sovelluksiin. Käytännössä ajatukset pitää kuitenkin päivittää ajan tasalle toteutushetkellä, sillä toimintaympäristöt muuttuvat jatkuvasti.

Toteutuksen myötä verkkokauppaan rakennettiin rajapinta, jonka kautta toiminnanohjausjärjestelmän tietokannan sisältämän datan käyttäminen onnistuu aiempaa suoraviivaisemmin. Tätä toteutettua rajapintaa on järkevää hyödyntää, kun järjestelmän dataa tarvitaan muissa yhteyksissä, kuten erilaisissa raportointijärjestelmissä, sillä sen kautta saatava tieto on formatoitu jo valmiiksi kohtuullisen käyttökelpoiseen muotoon.

Mikäli verkkokaupan taustalla toimiva toiminnanohjausjärjestelmä vaihtuu jossain vaiheessa, tulee tässä yhteydessä rakentaa myös sitä käyttävät rajapinnat uudelleen. Samassa yhteydessä on hyvä myös laajentaa verkkokaupan ominaisuuksia tukemaan uuden toiminnanohjausjärjestelmän tarjoamia mahdollisuuksia. Tätä kautta voidaan potentiaalisesti mahdollistaa entistä reaaliaikaisemmat tiedot ja monipuolisemmat tilausmahdollisuudet.

Verkkokaupan ja sähköisen kaupankäynnin lisääntymisen myötä digitaalisessa muodossa liikkuvan datan määrä kasvaa. Tämä vähentää mahdollisten inhimillisten virheiden määrää, koska tietoa ei kopioida käsin paikasta toiseen. Jatkossa verkkokaupan yhteyteen voidaan tarvittaessa toteuttaa kohtuullisen pienellä työmäärällä myös muunlaisia rajapintoja tiedon vastaanottamiseen tai lähettämiseen.

7. YHTEENVETO

Tämän diplomityön kuvaileman projektin tuloksena syntyi funktionaalisen ohjelmoinnin mahdollistavien menetelmien ja työkalujen avulla toteutettu verkkokauppa. Verkkokauppauudistuksen myötä asiakas sai tarpeisiinsa räätälöidyn verkkokaupan, jonka ylläpito ja jatkokehitys ovat aiempaa helpompia. Näiden seikkojen lisäksi uudistettu verkkokauppa toimii laajemmin erilaisilla laitteilla kännyköistä tietokoneisiin ja kaikilla moderneilla selaimilla, koska toteutuksessa hyödynnettiin responsiivista suunnittelua. Toteutettu verkkokauppa on osoittanut toimivuutensa myös käytännössä, sillä sen kautta on tähän mennessä tehty tuhansia tilauksia.

Asiakkaan olemassa olevaan toiminnanohjausjärjestelmään integroituminen osoittautui toteutuksen kannalta projektin haastavimmaksi osa-alueeksi. Rajapintojen puute pakotti lähestymistavaksi tietojen haun suoraan järjestelmän tietokannasta, mikä monimutkaisti sovelluksen arkkitehtuuria. Onnistuneet teknologiavalinnat vähensivät kuitenkin tästä aiheutunutta ylimääräistä selvitystyötä ja paransivat tietojen käsittelyn abstrahointia.

Työn lähtökohtana toimineet funktionaalisuutta tukevat työkalut osoittautuivat pääosin toimiviksi, mutta erityisesti kehitysympäristön toteutusaikaiset versiot olivat vaihtelevan epävakaita. Kehitysympäristön hidastelu ja kaatuilu oli kuitenkin lähinnä ärsyttävää, eikä vaikuttanut suuresti toteutuksen etenemiseen. Muut valitut työkalut olivat tutumpia aiemmista projekteista, mikä osaltaan kompensoi funktionaalisuuden opetteluun liittyvää ylimääräistä työkuormaa.

Itse funktionaalisuuden merkitys jäi toteutuksen kannalta kohtalaisen vähäiseen asemaan, sillä ohjelmointikieleksi valittu Scala tuki myös imperatiivista ohjelmointia. Tästä huolimatta Scalan käyttö helpotti päivittäistä työtä muun muassa vähentämällä koodirivien määrää kompaktimman syntaksin kautta. Myös tiiviistä yhteydestä Javaan oli hyötyä toteutuksessa.

Toteuttajien näkökulmasta uusien teknologioiden käytöstä huolimatta toteutusprojekti onnistui hyvin, mikä kannustaa käyttämään valittuja teknologioita jatkossakin. Uusien teknologioiden opettelu on aina haaste, mutta onnistumiset rakentavat syvempää ymmärrystä eri toimintatapoihin. Työkalujen kehitystyö jatkuu yhä aktiivisena, joten pahimmat niiden aiheuttamat ongelmat on epäilemättä saatu jo osittain korjattua.

LÄHTEET

[1] E. Tkacz, A. Kapczynski, Internet – Technical Development and Applications, Springer Berlin Heidelberg, 2009, 270 p.

[2] Q. Zheng, Introduction to E-commerce, Springer Berlin Heidelberg, 2009, 517 p.

[3] K. Webley, A Brief History of Online Shopping, TIME, verkkosivu. Saatavissa (viitattu 24.3.2014):
http://content.time.com/time/business/article/0,8599,2004089,00.html

[4] M.B. Taylor, Bitcoin and the age of Bespoke Silicon, 2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), Montreal, Canada, September 29 - October 4, 2013, 10 p.

[5] Suomalainen maksaa verkossa mieluiten perinteisin tavoin, Taloussanomat, verkkosivu, 11.6.2010. Saatavissa (viitattu 26.3.2014):
http://www.taloussanomat.fi/raha/2010/06/11/suomalainen-maksaa-verkossa-mieluiten-perinteisin-tavoin/20108347/139

[6] W. Qunli, Analysis on the Integration of ERP with E-business, 2010 Third International Symposium on Intelligent Information Technology and Security Informatics (IITSI), Jinggangshan, China, 2-4 April 2010, pp. 81-83.

[7] Epicor Completes Acquisition of Solarsoft Business Systems; Reports Preliminary Fiscal 2012 Financial Results, Marketwire, October 15 2012. Saatavissa:
http://finance.yahoo.com/news/epicor-completes-acquisition-solarsoft-business-130000481.html

[8] D.A. Turner, Some History of Functional Programming Languages, St Andrews University, United Kingdom, June 12 2012, 20 p.

[9] P. Hudak, Conception, Evolution, and Application of Functional Programming Languages, ACM Computing Surveys, Vol. 21, No. 3, September 1989, pp. 359-411.

[10] J. Armstrong, Erlang, Communications of the ACM, Vol. 53, No. 9, September 2010, pp. 68-75.

[11] K. Hinsen, The Promises of Functional Programming, Computing in Science & Engineering, Vol. 11, No. 4, July/August 2009, pp. 86-90.

[12] M.L. Scott, Programming Language Pragmatics, Morgan Kaufmann, 2000, 856 p.

[13] M. Harsu, Ohjelmointikielet, Talentum, 2005, 322 s.

[14] P. Wadler, Why no one uses functional languages, SIGPLAN Notices, Vol. 33, No. 8, August 1998, pp. 23-27.

[15] M. Odersky, What is Scala, The Scala Programming Language, verkkosivu. Saatavissa (viitattu 10.5.2015): http://scala-lang.org/what-is-scala.html

[16] Platform for Reactive Applications, Typesafe, verkkosivu. Saatavissa (viitattu 11.5.2015): http://www.typesafe.com/products/typesafe-reactive-platform

[17] Build Modern & Scalable Web Apps with Java and Scala, Play Framework, verkkosivu. Saatavissa (viitattu 11.5.2015): https://www.playframework.com

[18] PostgreSQL: About, verkkosivu. Saatavissa (viitattu 11.5.2015): http://www.postgresql.org/about/

[19] T. Haerder, A. Reuter, Principles of transaction-oriented database recovery, ACM Computing Surveys (CSUR), Vol. 15, No. 4, December 1983, pp. 287-317.

[20] Connect Reactive Applications to Data Sources with Slick, Typesafe, verkkosivu. Saatavissa (viitattu 11.5.2015): http://www.typesafe.com/community/core-tools/slick

[21] A. Mackler, Learning Slick, 2015. Saatavissa (viitattu 11.5.2015): https://mackler.org/LearningSlick2/

[22] jOOQ: The easiest way to write SQL in Java, verkkosivu. Saatavissa (viitattu 11.5.2015): http://www.jooq.org/

[23] Scala IDE for Eclipse, verkkosivu. Saatavissa (viitattu 14.5.2015): http://scala-ide.org/

[24] Getting Started with sbt, verkkosivu. Saatavissa (viitattu 15.5.2015): http://www.scala-sbt.org/0.13/tutorial/index.html

[25] JIRA - Issue & Project Tracking Software, Atlassian, verkkosivu. Saatavissa (viitattu 15.5.2015): https://www.atlassian.com/software/jira/

[26] JIRA Agile, Atlassian, verkkosivu. Saatavissa (viitattu 15.5.2015): https://www.atlassian.com/software/jira/agile/

[27] About - Git, verkkosivu. Saatavissa (viitattu 16.5.2015): https://git-scm.com/about

[28] V. Driessen, A successful Git branching model, January 05 2010, verkkosivu. Saatavissa (viitattu 16.5.2015): http://nvie.com/posts/a-successful-git-branching-model/

[29] Scala Templates, Play Framework, verkkosivu. Saatavissa (viitattu 16.5.2015): https://www.playframework.com/documentation/2.3.x/ScalaTemplates