MAKSUJÄRJESTELMÄN LOGISTIIKKAMODUULIN SUUNNITTELU JA TOTEUTUS

Jussi Hanski

Opinnäytetyö
Toukokuu 2010

Ohjelmistotekniikka
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN
KUVAILULEHTI

Tekijä(t) HANSKI, Jussi	Julkaisun laji Opinnäytetyö	Päivämäärä 17.05.2010
	Sivumäärä 30	Julkaisun kieli Suomi
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (X)

Työn nimi
MAKSUJÄRJESTELMÄN LOGISTIIKKAMODUULIN SUUNNITTELU JA TOTEUTUS

Koulutusohjelma
Ohjelmistotekniikka

Työn ohjaaja(t)
PELTOMÄKI, Juha

Toimeksiantaja(t)
Suomen Verkkomaksut Oy

Tiivistelmä
Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa järjestelmä lähetystenseurantaan Suomen Verkkomaksut Oy:lle. Järjestelmän tuli pystyä seuraamaan seurantakoodien avulla lähetettyjä paketteja eri logistiikkapalveluiden seurantajärjestelmistä. Järjestelmän piti tunnistaa milloin paketit ovat rekisteröityneet lähetystenseurantapalveluihin ja milloin paketit on toimitettu perille. Järjestelmän suunnittelu ja toteutus tehtiin osana isompaa maksujärjestelmäprojektia, joka tuli turvaamaan kuluttajienvälisessä etäkaupassa piileviä riskejä. Seurantajärjestelmä ohjelmoitiin käyttäen PHP-kieltä ja Zend Framework-ohjelmistokehystä.

Opinnäytetyössä käydään läpi toteutuksen vaatimia tekniikoita kuten SOAP ja WSDL. Opinnäytetyössä perehdytään myös Zend Framework –ohjelmistokehykseen ja siitä saataviin hyötyihin. Työssä käydään läpi ohjelmistokehyksen rakenne, sekä kuvataan ohjelmistokehyksen käyttöönottoprosessi. Työssä selvitetään aluksi mahdolliset liitynnät yleisimpiin logistiikkayritysten seurantajärjestelmiin.

Tämän jälkeen suunniteltiin ja toteutettiin tarvittava sovellus, joka vastaa annettuja vaatimuksia. Tuloksena työstä saatiin toimiva järjestelmä, joka pystyy seuraamaan lähetyksiä sekä Itellan-, että Matkahuollon-seurantapalveluista. Järjestelmästä luotiin helposti päivitettävä, jos uusia seurantapalveluita halutaan liittää mukaan.

Avainsanat (asiasanat)

logistiikkamoduuli, maksujärjestelmä, zend-Framework, php, Suomen Verkkomaksut Oy

Muut tiedot

Author(s)
HANSKI, Jussi

Type of publication
Bachelor´s Thesis

Date
17.05.2010

Pages
30

Language
Finnish

Confidential

( ) Until

Permission for web publication
(X)

Title
DESIGN AND IMPLEMENTATION OF LOGISTIC MODULE FOR PAYMENT SYSTEMS

Degree Programme
Software Engineering

Tutor(s)
PELTOMÄKI, Juha

Assigned by
Suomen Verkkomaksut Oy

Abstract
The goal of the thesis was to design and implement a packet tracking system for Suomen Verkkomaksut Oy. The system needed to be able to track packets from the tracking systems of logistic companies. The system needed to recognize when the packet was registered into the system and when it was delivered to the receiver. The systems design and implementation were carried out as a part of a larger payment system project, which was made to secure the trade between consumers. The tracking system was programmed using PHP-language and Zend Framework.

The thesis covers the techniques that were used in the implementations, such as SOAP and WSDL. It also covers Zend Framework's basic structure and advantages. There is also an example how to implement Zend Framework as a part of PHP-application. At first it was researched what logistic systems will be implemented.

After this the application, which fills all the requirements was designed and implemented. As a result system was gained that can track shipments through Itella’s and Matkahuolto’s tracking systems. The system is designed in a way that future updating and adding of a new logistic company’s tracking system is easy.

Keywords

logistic module, payment system, zend-Framework, php, Suomen Verkkomaksut Oy

Miscellaneous

SISÄLTÖ

Kuviot

LYHENTEET

API – Application Programming Interface

Käyttöliittymä, joka mahdollistaa eri ohjelmien tietojen ja pyyntöjen vaihdon keskenään.

BSD (lisenssi) – Berkley Software Distribution

Avoimen lähdekoodin lisenssi.

CSS – Cascading Style Sheets

Tyyliohjeet, joiden avulla voi määritellä esimerkiksi www-sivujen tyylit.

HTTP – Hypertext Transfer Protocol

Sovelluskerroksen protokolla, jota käytetään tiedon siirtoon verkossa.

INI – Initialization

Yleinen tiedostopääte sovelluksen tekstimuotoiselle asetustiedostolle.

JSON – JavaScript Object Notation

Yksinkertainen tiedonsiirtomuoto, jota on helppo käyttää JavaScript-ohjelmissa. Se on myös ohjelmointikieliriippumaton.

MVC – Model View Controller

Ohjelmistoarkkitehtuurimalli, jonka tarkoituksena on erottaa käyttöliittymä sovelluksen muusta koodista.

PDF – Portable Document Format

Adobe Systems Inc -yrityksen kehittämä järjestelmäriippumaton tiedostomuoto.

PHP – Hypertext Preprocessor

Ohjelmointikieli, jota käytetään verkkopalvelinympäristöjen ohjelmoinnissa.

PHTML

PHP-sivujen tiedostopääte, jota käytetään esimerkiksi silloin kun tiedosto sisältää PHP-koodin lisäksi HTML-koodia.

RSS – Really Simple Syndication

Verkkosyötemuoto, jota käytetään usein päivittyvän sisällön julkaisemiseen verkossa.

SOAP – Simple Object Access Protocol

Useiden protokollien yli toimiva XML-kieleen pohjautuva tietoliikenne määritelmä, joka mahdollistaa proseduurien etäkutsun.

TCP/IP – Transmission Control Protocol / Internet Protocol

Kuljetus- ja verkkokerroksen protokollayhdistelmä, jota käytetään kahden päätelaitteen välisessä tiedonsiirrossa.

URL – Uniform Resource Locator

Osoite, joka kuvaa sivun sijaintia verkossa.

WSDL – Web Service Description Language

XML-pohjainen kuvauskieli, jolla kuvataan verkosta löytyvää ohjelmointirajapintaa.

XML – eXtensible Markup Language

Rakenteellinen kuvauskieli, jonka avulla suurien tietomassojen jäsentäminen on mahdollista.

XML-RPC – eXtensible Markup Language – Remote Procedure Call

Proseduurikutsu, joka käyttää XML-kuvausta kutsun kuvauksena.

XSS – Cross site scripting

Verkkosivuhyökkäys, joka perustuu oman koodin syöttämiseen verkkosivuille.

1 TYÖN LÄHTÖKOHDAT

1.1 Tekijä

Tämän opinnäytetyön tekijänä toimi Jyväskylän ammattikorkeakoulun opiskelija, sekä Suomen Verkkomaksut Oy:n työntekijä Jussi Hanski. Työnkuvana Suomen Verkkomaksut Oy:ssä on tekninen asiakaspalvelu ja ohjelmistokehitys. Kielenä ohjelmistokehityksessä on pääosin PHP-ohjelmointikieli. Tekijällä on koulun kautta aiempaa kokemusta sovelluskehityksestä, sekä itse PHP-kielestä. Opinnäytetyön aihe, sekä toteutus muodostui normaalin työn ohessa, mutta opinnäytetyödokumentti toteutettiin työn ulkopuolella.

1.2 Toimeksiantajayritys

Toimeksiantajana opinnäytetyössä oli Suomen Verkkomaksut Oy. Suomen Verkkomaksut Oy on verkkokaupankäynnin maksuliikenteeseen erikoistunut yritys. Yritys on perustettu vuonna 2007 Lennu Keinäsen ja Niko Lehtosen toimesta. Yrityksen henkilöstön määrä vuonna 2010 oli yksitoista henkilöä ja toimipaikka sijaitsee Jyväskylässä Ylistönmäellä.

Asiakaskuntana Suomen Verkkomaksut Oy:llä ovat pääosin pienet ja keskisuuret yritykset, mutta löytyy asiakkaista myös suurempia yrityksiä. Suomen Verkkomaksut Oy:llä oli vuoden 2010 alussa yli tuhat asiakasta. Suomen Verkkomaksut Oy tarjoaa asiakkailleen yhden sopimuksen kautta kaikkia suomalaisia verkkopankkeja, Luottokunnan kautta luottokortilla maksamista sekä laskulla ja osamaksulla maksamista.

Verkkokauppiaan ei siis tarvitse tehdä integraatiota ja sopimusta jokaisen pankin välillä erikseen. Myös rahaliikenteen seuranta ja kirjanpitoraporttien tulostaminen Kauppiaspaneeli-työkalusta helpottaa verkkokauppiaan liiketoimintaa.

1.3 Työn tavoitteet

Työllä oli kaksi eri tavoitetta. Toinen oli itse sovelluksen toteuttaminen työnantajalle ja projektin valmiiksi saattaminen. Toinen taas oli itse opinnäytedokumentin tuottaminen. Dokumentissa perehdytään käytettyihin työkaluihin ja menetelmiin, käydään läpi työn tavoitteet, käsitellään työn toteutus sekä arvioidaan toteutettua sovellusta.

Opinnäytetyön aihe oli suunnitella ja toteuttaa logistiikan seurantajärjestelmä Suomen Verkkomaksut Oy:lle. Järjestelmän tehtävänä oli lähetysten tilannetietojen hakeminen logistiikkayritysten seurantapalveluista ja lähetysten tapahtumatilojen ilmoittaminen. Järjestelmä toteutettiin osaksi Jemma-palvelua, joka on tarkoitettu kuluttajien välisen kaupan turvaamiseksi.

Järjestelmän tehtävänä oli hoitaa seuraavia toimintoja:

- Seurantakoodien syöttö Jemmaan
- Paketin tilan seuranta eri seurantapalveluista
- Tapahtuman tilan vaihto toimituksen tilan mukaan
- Ilmoitusten lähettäminen tapahtuman osapuolille.

1.4 Kuluttajien välinen kauppa

Kuluttajien välinen kauppa tarkoittaa kahden yksityishenkilön välistä kauppatapahtumaa. Kuluttajansuojalaki ei koske kahden yksityishenkilön välistä kauppaa, joten ostajalla on yhtä suuri vastuu tapahtumassa kuin myyjälläkin. Myyjän tulee antaa paras mahdollinen kuvaus tuotteesta sekä kertoa sen viat tai puutteet. Ostajan tulee myös tarkastaa tuote huolella ennen ostoa, koska reklamointi ja palautus harvoin onnistuvat enää jälkikäteen.

Kuluttajien välinen kauppa verkossa lisääntyy koko ajan. Verkossa solmittu kauppa tuo omat haasteet tuotteen kunnon selvittämiseen. Kauppatapahtuma vaatii myös luottamusta, koska yleensä kauppa tapahtuu täysin ilman henkilökohtaista tapaamista. Maksu siirretään tilisiirtona ja tuote toimitetaan postitse. Tämä mahdollistaa sen, että kumpikin osapuolista voi tulla huijatuksi. Ostaja saattaa jäädä tuotteetta vaikka maksoi rahat myyjälle, tai myyjä ilman rahoja vaikka lähettikin tuotteen.

1.5 Jemma-palvelu

Suomen Verkkomaksut Oy toi markkinoille vuoden 2009 joulukuussa verkkopalvelun kuluttajien välisen kauppatapahtuman turvaamiseen. Tämä Jemma nimeä kantava palvelu toimii ostajan ja myyjän välissä turvaamassa kauppatapahtumaa.

Tapahtuma alkaa siitä, kun myyjä syöttää myytävän tuotteen tiedot Jemmaan. Kuvailun apuna voi käyttää esimerkiksi kuvia. Tämän jälkeen myyjä määrittelee tuotteelle hinnan ja kutsuu ostajan tapahtumaan. Seuraavaksi ostaja hyväksyy ja maksaa sovitun hinnan Jemmaan. Tämän jälkeen Jemma ilmoittaa myyjälle, että tuotteen voi postittaa. Myyjä voi syöttää paketin seurantakoodin Jemmaan, jolloin Jemma seuraa paketin kulkua myyjän käyttämästä logistiikkapalvelusta, esimerkiksi Itellan seurantapalvelusta. Kun paketti saapuu ostajalle, tulee hänen tarkastaa tuote ja hyväksyä se Jemmassa. Tämän hyväksymisen tai kahden päivän reklamaatioajan kuluttua Jemma tilittää sovitun summan myyjälle. Jos tuotteen toimituksen jälkeen ostaja haluaa reklamoida tuotteesta, silloin rahojen siirto jäädytetään, kunnes reklamaatio on suljettu. Tähän reklamaatioon voi kuulua uuden hinnan neuvottelu tai jopa kaupan purku.

2 TEKNIIKAT

2.1 Web Service

Web Service -palvelulla tarkoitetaan ohjelmointirajapintaa, joka mahdollistaa ohjelmistojärjestelmien käytön tietoverkon yli. Yleensä tämä etäkäyttö tapahtuu HTTP-protokollan tai muun TCP/IP-pohjaisen protokollan yli. Ohjelmointirajapinnat mahdollistavat siis palveluiden kytkeytymisen toisiinsa tietoliikenneverkon yli. Yleensä tällaisen ohjelmointirajapinnan kuvaukseen voidaan käyttää esimerkiksi WSDL-kuvauskieltä. (Booth, Haas, McCabe, Newcomer, Champion, Ferris & Orchard 2004.)

2.2 SOAP

SOAP on tietoliikennemääritelmä, jonka avulla asiakasohjelmisto ja varsinainen ohjelmointirajapinta ymmärtävät toisiaan. SOAP pohjautuu XML-kieleen ja sitä voidaan hyödyntää useiden tietoliikenneprotokollien yli. SOAP ei ota kantaa käytettyyn ohjelmointikieleen tai ympäristöön, eli Javalla toteutettu ohjelmointirajapinta pystyy vastaanottamaan PHP-ohjelmalta SOAP-kutsun ja palauttamaan vastaavasti SOAP-vastauksen.

SOAP viestin rakenne muodostuu kolmesta eri osasta:

- SOAP-ENV: Envelope
- SOAP-ENV: Header
- SOAP-ENV: Body

Envelope (eli kirjekuori) määrittelee käytetyn XML-kaavan. Header (eli otsikko) määrittelee viestin otsikkotietoja. Body (eli runko) sisältää lähetetyn viestin. (Gudgin, Hadley, Mendelsohn, Moreau, Nielsen, Karmarkar & Lafon 2007.)

2.3 WSDL

WSDL on XML-pohjainen kuvauskieli, jonka avulla kuvataan tietoverkosta löytyvää ohjelmointirajapintaa. Itse kuvaus on jaettu kahteen eri osa-alueeseen. Toinen osa on ns. abstrakti kuvaus ja toinen osa ns. konkreettinen kuvaus. Abstrakti kuvaus kuvaa palvelua ottamatta kantaa palvelun sijaintiin tai protokollaan, kun taas konkreettinen osa kuvaa myös käytetyn protokollan sekä palvelun osoitteen verkossa. (Chistensen, Curbera, Meredith, & Weerawarana 2001.)

2.4 Zend Framework

2.4.1 Yleistä

Zend Framework on Zend Technologies -nimisen yrityksen PHP5-ympäristölle toteuttama avoimeen lähdekoodiin perustuva ohjelmistokehys. Ohjelmistokehyksen avulla sovellusten kehitykseen ja ylläpitoon saadaan nopeutta, helppoutta ja tehokkuutta. Ohjelmistokehys on nopea ottaa käyttöön ja sen hyödyntämisen aloittaminen on helppoa. Verkko on täynnä ohjeita ja oppaita, joiden avulla ohjelmistokehyksen ominaisuuksien opettelu on nopeaa ja vaivatonta. (Allen, Lo & Brown 2009, 3−5.)

2.4.2 Historia

PHP on ollut mukana dynaamisten verkkosivujen kehityksessä jo yli kymmenen vuotta. Alussa PHP-sivut olivat kirjoitettu niin, että PHP-koodi ja HTML-koodi oli sijoitettu samalle sivulle. Tämä malli sopi hyvin aluksi pienien verkkosivujen toteuttamiseen. Kun PHP ajan myötä kasvatti suosiotaan ja sovellukset kasvoivat vaativimmiksi, oli selvää, ettei PHP-koodin ja HTML-koodin sekoittaminen ollut pitemmän päälle hyvä ratkaisu. Tätä mallia lähdettiin muut-

tamaan erottamalla taustalogiikka pois varsinaisesta HTML-koodista. Sovellusta pilkottiin moniin eri tiedostoihin, jotka kaikki hoitivat oman osuutensa taustalogiikasta, tietokantayhteydestä tai muusta vastaavasta. Tällä tyylillä saatiin taas kaikille yhteisistä tiedostoista luotua kirjasto ja suurempien sovellusten toteuttaminen helpottui. PHP4:n ja varsinkin PHP5:n tuomat olio-ohjelmointimahdollisuudet mahdollistivat, että näistä kirjastoista muodostui ohjelmistokehyksiä, jotka helpottavat ohjelmistokehitystä sekä sovellusten ylläpitoa. Ohjelmistokehyksiä on ollut PHP-maailmassa jo vuosia, mutta vuonna 2005 Zend Technologies -yritys, joka on erikoistunut PHP-järjestelmien kehitykseen, aloitti vapaan lähdekoodin projektin, jonka tarkoituksena oli luoda PHP-ohjelmistokehys, joka toimisi tulevaisuuden PHP-sovellusten alustana. (Allen, Lo & Brown 2009, 4−5.)

2.4.3 Zend Frameworkin edut

Zend Framework -ohjelmistokehyksen eduista yhtenä mainittakoon käyttöönoton helppous. Kaikki yleisesti tarvittavat ominaisuudet löytyvät valmiiksi ohjelmistokehyksestä. Esimerkkeinä autentikointi-, haku-, sähköposti-, lokalisointi- ja tietokantayhteyskomponentit. Zend Framework -ohjelmistokehystä voi käyttää myös osissa. Eli on mahdollista ottaa käyttöön vain tarvittavat komponentit ja hyödyntää pelkästään niitä. Zend Framework -ohjelmistokehystä on myös mahdollista käyttää yhdessä muiden suosittujen PHP:n MVC-mallia toteuttavien ohjelmistokehyksien kanssa, kuten esim. CakePHP- ja CodeIgniter-ohjelmistokehyksen kanssa. (Allen, Lo & Brown 2009, 5−6.)

Zend Framework hyödyntää PHP5-ympäristön mahdollistamaa olio-ohjelmointilähestymistapaa ja nykyaikaisia suunnittelumalleja. Sen käyttöönotto on suunniteltu erittäin joustavaksi ilman, että se vaatisi käyttöönottajalta liikaa työtä. Komponentit on suunniteltu niin, että niissä olisi vähän riippuvuuksia toisiinsa, jotta niiden käyttöönotto olisi helppoa myös itsenäisinä komponentteina. (Mts. 6.)

Tällainen komponenttien toisistaan riippumattomuus helpottaa koodin ja koko ohjelmistokehyksen ymmärtämistä. Tiettyä komponenttia tarkasteltaessa ja käytettäessä ei siis tarvitse tietää juurikaan muista komponenteista tai niiden toiminnasta. Komponenttien syvällistä toimintaa ei välttämättä tarvitse ymmärtää, ennen kuin sitä voi käyttää. Kun kokemus komponentista karttuu, on uu-

sien ja monimutkaisempien ominaisuuksien käyttöönotto tehty käyttäjälle helpoksi ja suoraviivaiseksi. (Mts. 6−7.)

Ohjelmistokehyksen kaikista komponenteista löytyy kattava dokumentaatio, ja tämän voisi laskea yhdeksi ohjelmistokehyksen suureksi vahvuudeksi. Zend Framework asettaa dokumentaation yhtä suureen arvoon itse koodin kanssa. Ohjelmistokehys ei siis sisällä yhtään komponenttia, jota ei olisi dokumentoitu. Dokumentoinnin voi jakaa kahteen eri osaan: API- ja loppukäyttäjä-dokumenttiin. (Mts. 7.)

API-dokumentti tarkoittaa itse koodista löytyvää DocBlock-osaa, joka on eräänlainen kommentointi luokan, funktion tai luokan muuttujan yläpuolella. Tämä DocBlock-osa kuvaa lyhyesti toimintaa ja kertoo esimerkiksi funktion sisään ottamista parametreista sekä funktion palauttaman arvon tyypistä. DocBlock-standardia hyödyntävät myös sovelluskehitysympäristöt, kuten esimerkiksi Zend Studio, joka osaa tulkita DocBlock-osan parametrit ja hyödyntää niitä koodin automaattisessa ehdottamisessa. (Mts. 7.)

Loppukäyttäjä-dokumentti tarkoittaa verkosta löytyvää Zend Framework -käyttöohjetta, joka sisältää kaikki yksityiskohtaiset tiedot jokaisesta ohjelmistokehyksen komponentista ja niiden sisältämistä funktioista. Dokumentista löytyy myös esimerkkejä, kuinka ottaa komponentti käyttöön omassa ohjelmassa ja monimutkaisimmissa komponenteissa on myös selostettu komponentin toiminnan teoria. Nämä kaikki ominaisuudet tekevät Zend Framework -ohjelmistokehyksellä kehittämisestä yksinkertaista ja nopeaa. Koodi on myös helppo organisoida ja ohjelmistokehyksen rakenne tekee ylläpitämisestä ja vian etsinnästä helppoa. (Mts. 7−8.)

2.4.4 MVC-malli

Zend Framework toteuttaa niin kutsuttua MVC- eli Model-View-Controller –mallia (ks. kuvio 1). Model-View-Controller on yksi tunnetuimmista ohjelmiston suunnittelumalleista. Se erottelee toimintotason koodin näyttötason koodista. Se sai alkunsa Smalltalk-kielen yhteydessä vuonna 1979 ja on siitä levinnyt käyttöön jopa nykyaikaisiin verkkosovelluksiin.

Malli koostuu siis kolmesta eri osasta. Model, eli malli vastaa esimerkiksi sovelluksen tiedon tallentamisesta, ylläpidosta ja käsittelystä tietokantaan. View

eli näkymä vastaa tiedon näyttämisestä loppukäyttäjälle eli käyttöliittymästä. Controller eli ohjain vastaa käyttäjältä ja sovellukselta tulevan tiedon käsittelyssä sekä muokkaa näkymää ja mallia sen mukaan. Zend Framework siis toteuttaa MVC-mallin ja on yleensä ohjelmistokehyksellä tehtyjen sovelluksen ytimenä. (Allen, Lo & Brown 2009, 19−20.)

KUVIO 1. MVC-malli ohjelmistokehyksessä

2.4.5 Zend Frameworkin komponentteja

Ohjelmistokehyksen komponentit voisi jakaa kuuteen pääkategoriaan. MVC-komponentteihin, autentikointi- ja pääsyoikeuskomponentteihin, lokalisointikomponentteihin, ohjelmistojenvälisen kommunikoinnin komponentteihin, ohjelmistorajapintakomponentteihin ja ydinkomponentteihin, joita on vaikea muuten kategorioida. (Allen, Lo & Brown 2009, 9.)

MVC-komponentit

MVC-komponentit tarjoavat työkalut siihen, että sovelluksessa voidaan erottaa toimintatason koodi näyttötason koodista. Zend_Controller-komponentti vastaa siis ohjainosaa MVC-mallissa, ja taas Zend_View vastaa näkymäosaa. Zend_Db, joka hoitaa tietokantayhteyttä, voi esimerkiksi vastata sitten taas malliosaa MVC-mallissa. Zend_Controller hyödyntää niin sanottua Front Controller- suunnittelumallia, jossa tämä Front Controller jakaa pyynnön eteenpäin oikealle kontrollerille ja sen sisältämälle toiminnalle. Zend_View, joka vastaa tiedon näyttämisestä loppukäyttäjälle, tarjoaa PHP-pohjaisen sivupohjajärjestelmän, jossa näkymä-tiedostot voidaan kirjoittaa PHP:lla. Zend_View tarjoaa myös mahdollisuuden ottaa käyttöön jonkun ulkopuolisen sivupohjajärjestelmän. Tarjolla on myös Zend_Layout, joka helpottaa monille sivuille samanlaisen ilmeen saamisesta. Zend_View tarjoaa myös niin sanottua Helper-liitännäisjärjestelmää, jonka avulla voi luoda helposti uudelleenkäytettävää näyttökoodia. (Mts. 10−12.)

Zend_Db_Table toteuttaa niin sanotun Table Data Gateway -suunnittelumallin, jota voidaan käyttää mallin toteutuksen pohjana MVC-mallissa. Zend_Db_Table hyödyntää Zend_Db:tä, joka tarjoaa olio-ohjelmointimaisen lähestymistavan tietokantayhteyteen, jossa käytettävyys pysyy samana vaikka tietokantakantajärjestelmä vaihtelee. Yhteys voidaan muodostaa esimerkiksi seuraaviin tietokantoihin:

- MySQL
- PostgreSQL
- SQL Server
- Oracle
- SQLite.

MVC-kategorian luokat hyödyntävät joitain ydinkategorian luokkia, esimerkiksi Zend_Config-luokkaa. Sen avulla voidaan lukea esimerkiksi tietokantayhteyteen tarvittavat parametrit joko PHP-taulukosta, XML-tiedostosta tai INI-tiedostosta. Se voi sisältää myös eri parametrit ohjelmistokehityksen eri vaiheille, esimerkiksi kehitys-, testaus- ja tuotantovaiheelle. (Mts. 10−12.)

Turvallisuuden näkökulmasta Zend_Validate ja Zend_Filter auttavat ohjelmistokehittäjää varmentamaan syötetyn tiedon oikeellisuutta ja helpottavat turvallisten sovellusten rakentamista. Niiden avulla saadaan moni verkkosivuilla helposti esiintyvä tietoturva-aukko paikattua. (Mts. 10−12.)

Autentikointi- ja pääsyoikeuskomponentit

Verkkosovelluksille on yleistä, että käyttäjä tunnistetaan ja tietyille käyttäjille, sekä käyttäjäryhmille annetaan eri oikeuksia sovelluksessa. Tätä varten Zend Framework sisältää Zend_Auth- ja Zend_Acl-komponentit. Ohjelmistokehityksessä joustavuus on suuressa osassa ja tästä johtuen jos ohjelmistokehyksen autentikointimetodit eivät riitä, voi ohjelmistokehittäjä halutessaan toteuttaa ja muokata omiin tarpeisiinsa soveltuvan autentikontimetodin. Yleensä autentikoinnista saatu lipuke tallennetaan Zend_Session avulla ja Zend_Acl voit tämän jälkeen hyödyntää lipuketta ja määritellä, onko käyttäjällä oikeuksia ohjelmiston resursseihin. (Mts. 12.)

Lokalisointikomponentit

Lokalisointikomponentit auttavat sovelluksen muokkausta kansainvälisiin ympäristöihin. Tarjolla on Zend_Locale-luokka, jonka Zend_Currency- ja Zend_Measure-komponentit vastaavat oikeiden yksiköiden ja kielen käytöstä. Zend_Translate-komponentti taas vastaa tekstin näytöstä sivuilla käyttäjälle sopivalla kielellä. (Mts. 12−13.)

Ohjelmistojen välisen kommunikoinnin komponentit

Ohjelmistojen väliseen kommunikointiin löytyy myös komponentteja. Esimerkiksi Zend_Http_Client-komponentin avulla on mahdollista hakea tietoa toisilta verkkosivuilta ja palveluista. Zend_XmlRpc_Client-komponentilla on mahdollista käsitellä XML-RPC-yhteyksiä ja Zend_Json-komponentilla voi luoda ja lukea JSON-tietoa. (Mts. 13.)

Ohjelmistorajapintakomponentit

Ohjelmistorajapintakomponentit auttavat verkosta löytyvien palveluiden hyödyntämisen omassa sovelluksessa. Esimerkkinä voisi käyttää erilaisia RSS-lähteitä, Googlen, Yahoo!:n ja Amazonin verkkopalveluita. (Mts. 13.)

Ydinkomponentit

Ydinkomponenteiksi nimettiin muuten vaikeasti kategorioitavia komponentteja. Tämä komponenttien paletti tarjoaa paljon ohjelmistokehittäjän työtä helpottavia komponentteja. Kategoriasta löytyy esimerkiksi komponentteja lokitiedon käsittelyyn, sähköpostin lähettämiseen, PDF-tiedoston käsittelyyn, hakuominaisuuksien toteuttamiseen, välimuistin käyttöönottoon ja lomakkeiden luomiseen. Ohjelmistokehykseen lisätään koko ajan uusia komponentteja, jotka kaikki noudattavat samoja ohjelmistosuunnittelun filosofioita, jotka varmistavat ohjelmistokehyksen laadun ja jatkuvuuden tulevaisuudessa. (Mts. 14−15.)

2.4.6 Zend Frameworkin filosofia

Zend Framework sisältää ainoastaan korkealaatuista koodia. Tämä tarkoittaa sitä, että yksikään koodinpätkä kirjastossa ei synnytä viestejä PHP-jäsentäjältä. PHP-ympäristössä voi siis käyttää E_STRICT-asetusta ja jos jä-

sentäjä palauttaa virheen, voi ohjelmistokehittäjä olla varma siitä, että kaikki virheet aiheutuvat omasta koodista, ei ohjelmistokehyksen. Tämä auttaa vian etsinnässä huomattavasti. Tähän laadukkuuteen vaikuttaa myös dokumentaation taso, koska dokumentaatio asetetaan yhtä suureen arvoon kun itse koodi.

Zend Framework on myös yritysystävällisen BSD-lisenssin alainen ja sitä kehitetään niin, että se on täysin kopiosuojavapaata koodia. Näin sitä voi käyttää kuka tahansa, mihin tarkoitukseen tahansa ilman minkäänlaisia lakiesteitä. (Allen, Lo & Brown 2009, 15−16.)

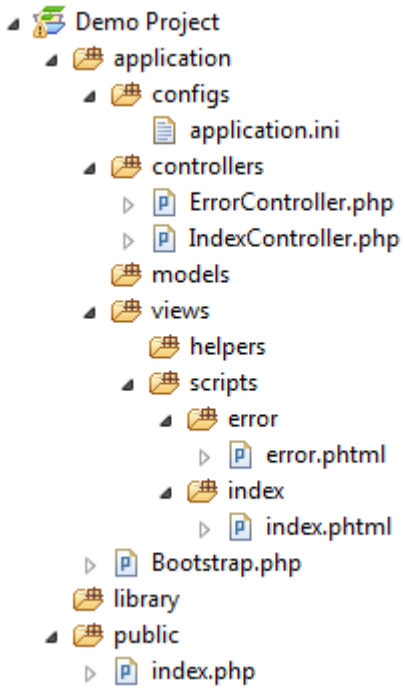
2.4.7 Zend Frameworkin käyttöönotto käytännössä

Zend Framework on mahdollista ottaa käyttöön helposti esimerkiksi Zend Studio -ohjelman avulla luomalla uuden Zend Framework–projektin projektivelhon avulla tai sitten luomalla hakemistorakenteen ydintiedostoineen käsin. Tämä onnistuu esimerkiksi käytettävällä sovelluskehittimellä tai vaikka suoraan komentokehotteen avulla.

Normaalissa PHP-skriptissä perinteinen ”Hello World” -tekstin tulostus ruudulle onnistuu yhden PHP-tiedoston avulla. Tiedoston tarvitsee sisältää vain seuraava koodi:

<?php echo ’Hello World’;

Zend Framework -ohjelmistokehyksen tapauksessa kyseisen tuloksen saaminen vaatii ensin ohjelmistokehysympäristön rakentamisen. Kuviossa 2 esitetään ohjelmistokehyksen perusrunko.



KUVIO 2. Esimerkki ohjelmistokehyksen perusrungosta

Seuraavaksi tarkastellaan projektin kansioiden ja tiedostojen merkitystä.

Application-kansio

Application-kansio sisältää kaiken sovelluksen ajamiseen tarvittavan sovelluskohtaisen koodin. Se sisältää kansiot models, views ja controllers, jotka korostavat ohjelmistokehyksen suhdetta MVC-malliin. Application-kansioon ei tule olla suoraa pääsyä ulkopuolisilta. Sinne siis ei tule voida viitata suoraan URL-osoitteella. Kansioon voi myös lisätä omia kansioita esimerkiksi sovelluksen asetustiedostoille.

Library-kansio

Nimensä mukaisesti Library-kansio sisältää koodikirjastoja. Yleensä tästä kansiosta löytyy itse Zend Framework -koodikirjasto. Kansio sisältää yleensä myös sovelluskehittäjän omia kirjastoja, joita sovelluksessa hyödynnetään. Kirjastot voi tosin tallentaa myös palvelimen omaan niin sanottuun include-kansioon, johon kaikilla palvelimen PHP-sovelluksilla on pääsy. Tämän kansion polku on määritetty PHP-järjestelmän php.ini-tiedostossa.

Public-kansio

Public-kansio on ainoa Zend Framework -projektin kansio johon järjestelmän ulkopuolelta on pääsyoikeus. Tämä parantaa ohjelmiston, sekä koko palvelimen tietoturvaa. Kansiosta löytyy vain yksi PHP-tiedosto: index.php. Kansio sisältää myös kaiken ulkopuolelta ladattavissa olevan materiaalin, esimerkiksi CSS-tiedostot, JavaScript-tiedostot, kuvat ja muut multimediatiedostot.

Index.php

Index.php on siis ainoa PHP-tiedosto palvelimella, joka on ulkopuolisille näkyvissä. Tämä on sovelluksen esilataustiedosto, joka alustaa ympäristön, asettaa sovelluksen käyttämät polut sekä noutaa ja suorittaa sovelluksen.

Kuviosta 3 voi nähdä, kuinka index.php aluksi määrittää sovelluksen polun. Seuraavaksi määritellään sovelluksen ympäristö, joka voi olla jo ennalta määritetty järjestelmän asetustiedostoissa tai sitten se määritellään tässä. Ympäristöllä tarkoitetaan esimerkiksi tuotantoympäristöä (production) tai testausympäristöä (testing). Tämän jälkeen määritellään, kuinka ympäristön virheet käsitellään ja näytetäänkö virheitä loppukäyttäjälle. Näiden jälkeen määritellään sovelluksen sijainti, joka on tarpeellinen tieto PHP 5.1:sen mukanaan tuomalle päivämäärä- ja aika-funktioille.

Seuraavaksi määritellään koodikirjaston polku ja kun se on määritetty, voidaankin jo hakea ja luoda itse ohjelmakehyksen Application-olio Zend_Application-luokasta. Application-olion avulla voidaan sitten suorittaa itse sovellus.

Sovelluksen suorittaakseen Application-olio kutsuu ohjelmistokehyksen Zend_Application_Bootstrap-luokkaa, joka vastaan ohjelmistokehyksen esilatauksesta. Application-olio myös luo sovellukselle Zend_Loader_Autoloader-olion, joka vastaa siitä, että kun sovelluksessa kutsutaan luokkia, voidaan niitä ladata dynaamisesti ilman tarvetta liittää luokkatiedostoon ennen luokan käyttämistä. Zend_Autoloader liittää itsensä __Autoload() PHP-funktioon tarjoten todellisen nimiavaruutta hyödyntävän automaattisen luokkalatauksen. Oletuksena Autoloader-luokka löytää Zend- ja ZendX-nimiavaruudet, mutta sille voidaan määritellä myös muita nimiavaruuksia, jos ominaisuutta halutaan hyödyntää myös muissa omissa kirjastoissa.

Bootstrap-olio alustaa Zend_Controller_Front-olion, joka vastaa request-olion ja response-olion luomisesta tarvittaessa, välittää pyyntöjä, suorittaa ohjaimet sekä palauttaa ohjelman lopputuloksen. Zend_Controller_Front-olio myös ottaa kiinni sovelluksen ajon aikana palautetut poikkeukset ja välittää ne virheenkäsittelijäohjaimelle.

```
<?php

// Define path to application directory
defined('APPLICATION_PATH')
    || define('APPLICATION_PATH', realpath(dirname(__FILE__) .
'/../application'));

// Define application environment
defined('APPLICATION_ENV')
    || define('APPLICATION_ENV', (getenv('APPLICATION_ENV') ? ge-
tenv('APPLICATION_ENV') : 'production'));
error_reporting(E_ALL|E_STRICT);
ini_set('display_errors', false);
date_default_timezone_set('Europe/Helsinki');

// Ensure library/ is on include_path
set_include_path(implode(PATH_SEPARATOR, array(
    realpath(APPLICATION_PATH . '/../library'),
    get_include_path(),
)));

/** Zend_Application */
require_once 'Zend/Application.php';

// Create application, bootstrap, and run
$application = new Zend_Application(
    APPLICATION_ENV,
    APPLICATION_PATH . '/configs/application.ini'
);
$application->bootstrap()->run();
```

KUVIO 3. Esimerkki index.php-tiedostosta

Apache .htaccess-tiedosto

Jotta kaikki muut verkon kutsut kuin kuvien, JavaScriptien, CSS-tiedostojen ja muiden vastaavien tiedostojen pyynnöt ohjautuisivat index.php-tiedostoon, täytyy palvelimen Apache-ohjelman ohjaussääntöjä määritellä. Tämä onnistuu joko Apachen httpd.conf-tiedostosta tai projektin paikallisesta .htaccess-tiedostosta public-kansiossa.

Kuvion 4 esimerkissä .htaccess-tiedostoon määritellään ehto, jos tiedostoa ei löydy pyydetystä polusta ohjataan pyynnöt suoraan index.php-tiedostoon.

```
# Rewrite rules for Zend Framework
RewriteEngine on
RewriteCond % {REQUEST_FILENAME} ! -f
RewriteRule .* index.php
```

KUVIO 4. Esimerkki .htaccess-tiedostosta

IndexController.php-tiedosto

IndexController on ohjelman oletusohjain. Kaikki ilman ohjainpyyntöä tulleet pyynnöt ohjataan sille. Itse ohjainluokan nimeämiskäytännöt ovat erittäin tiukat. Luokkien nimien tulee olla muotoa (ohjaimen nimi)Controller(), ja ohjaimen sisältämien toimintojen nimet (toiminnon nimi)Action(). Ohjaimen oletustoiminto on nimeltään indexAction().

Kuten kuviosta 5 voi havaita, IndexController jatkaa Zend_Controller_Action-luokkaa, joka sisältää pyyntö- ja palutus-oliot. Näiden avulla päästään käsiksi esimerkiksi pyynnöltä saatuihin tietoihin. Init-funktiossa voitaisiin määritellä suoritettavaksi asioita, jotka suoritetaan ennen kaikkien ohjaimen toimintojen suoritusta. Ennen kuin ohjain suorittaa toiminnon, luo se Zend_View-olion ja asettaa sen luokan käyttöön nimellä $view. Tämän avulla voimme siirtää tietoa ohjaimelta näkymälle. Heti kun toiminto on kokonaan suoritettu renderöidään automaattisesti toimintoa vastaava näkymämalli response-, eli paluuolioon. Vastaava näkymämalli löytyy kansiosta view/scripts/(ohjaimen nimi)/(toiminnon nimi). Näkymämallit ovat .phtml-päätteisiä tiedostoja, eli kyseisen toiminnon näkymämalli olisi nimeltään index.phtml.

```php
<?php

class IndexController extends Zend_Controller_Action
{

    public function init()
    {
        /* Initialize action controller here */
    }

    public function indexAction()
    {
        $this->view->assign('title', 'Hello World');
    }


}
```

KUVIO 5. Esimerkki IndexController.php-tiedostosta

Index.phtml tiedosto:

Koska index.phtml-tiedoston suorittaa luokka, joka on Zend_View-luokan jäsenluokka, on kaikki muuttujat, jotka määritimme ohjaimessa $view-oliolle, myös käytössä näkymässä $this-olion kautta.

Kuviossa 6 on esitetty esimerkkinä index.phtml-tiedosto. Tässä esimerkissä asetetaan sivun otsikon $this-olion kautta. Samalla tulostetaan kyseisen otsikon myös sivun runkoon.

```php
<!DOCTYPE html PUBLIC "-//W3C/DTD_XHTML 1.0 Strict//EN"
           "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xm1ns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
<meta http-equiv="Content-type" content="text/html; charset=utf-8"
/>
<title>
           <?php echo $this->escape($this->title);?>
</title>
</head>
<body>
           <h1><?php echo $this->escape($this->title);?></h1>
</body>
</html>
```

KUVIO 6. Esimerkki Index.phtml-tiedostosta

2.4.8 Yhteenveto

Kokemattomalle ohjelmoijalle Zend Framework voi vaikuttaa monimutkaiselta pystyttää, mutta muutamassa tunnissa ohjeiden avulla myös kokemattomampi ohjelmoija on pystyttänyt toimivan ohjelmistokehyksen. Ohjelmistokehyksen todelliset edut alkavat vasta, kun sovellus alkaa kasvaa ja monimutkaistua. Ohjelmistokehyksen ominaisuuksiin kuitenkin kuuluu hyvänä puolena se, että kaikkea ei tarvitse tietää ennen kuin komponentteja voi alkaa hyödyntää. Tämä pätee myös ohjelmistokehyksen perusrakenteeseen. Ohjelmistokehittäjän ei tarvitse ymmärtää kaikkia pyyntöjen ja palautusten ohjaukseen liittyvää, jotta voisi hyödyntää ohjelmistokehystä omassa sovelluksessaan. (Allen, Lo & Brown 2009, 17−39.)

3 SUUNNITELMA

3.1 Yleistä

Projektissa käytettiin niin sanottuja ketteriä menetelmiä (Scrum), jotka jättivät perinteisen suunnitteluosuuden pienempään rooliin. Jokaista pientä ominaisuutta ei tarvinnut kirjoittaa paperille ja miettiä valmiiksi. Alustavan suunnitelman jälkeen sovellusta lähdettiin purkamaan pienempiin osiin ja niistä osista se purettiin pienempiin tehtäviin, joita sitten ohjelmoitiin. Aika ajoin tarkasteltiin kokonaisuuden lopputulosta ja valittiin seuraavat toteutettavat ominaisuudet. Opinnäytetyön aiheena on tämän isomman Jemma-projektin logistiikkaseuranta-osakokonaisuus.

3.2 Palvelin

Järjestelmän palvelinalustana toimii Crescom Oy:n ylläpitämä Linux-palvelin. Palvelimella toimiva Apache-palvelinohjelmisto hoitaa tiedostojen, tässä tapauksessa verkkosivujen jakamisen HTTP-yhteyden yli. Palvelimella toimii myös PHP-ohjelmointiympäristö, sekä MySQL-tietokantapalvelin. Palvelimelle oli myös asennettu tuki SOAP 1.2 -yhteyskäytännölle.

3.3 Sovellusvaatimukset

Järjestelmän tuli ottaa käyttäjältä vastaan seurantakoodeja ja seurata seurantakoodeja vastaavien lähetyksen kulkua eri logistiikkapalveluiden seurantajärjestelmistä. Kun lähetyksessä tapahtui tilamuutos, tuli järjestelmän päivittää

tieto Jemman järjestelmään. Jemmassa tämä tieto sitten tuli käsitellä ja tarvittaessa muuttaa kauppatapahtuman tilaa tai lähettää sähköposti-ilmoituksia tapahtumaan liittyen.

3.4 Sovelluksen rakenne

Sovellus rakentui kahdesta eri osasta. Itsenäisestä Web Service- eli ohjelmistorajapinta-osasta, joka toimii rajapintana eri seurantajärjestelmien ja Jemman järjestelmän välillä. Toinen osa sovellusta on itse Jemman logistiikkajärjestelmä. Tämä järjestelmä hoitaa paketin tilakyselyt ohjelmistorajapinnan paketinseurannalta, automaattiset tapahtuman tilamuutokset, sähköpostin lähetyksen tapahtumatilan muutoksista, sekä seurantakoodien hallinnan.

3.5 Seurantajärjestelmät

Aluksi tehtiin selvitys mahdollisista seurantajärjestelmistä, joita tultaisiin hyödyntämään. Selvityksessä käytiin sähköpostikeskusteluja eri logistiikkayritysten kanssa. Selvitystyö joidenkin logistiikkayritysten kanssa venyi liian pitkäksi, joten aluksi jouduttiin ottamaan vain kaksi yleisintä logistiikkayritystä mukaan. Jos kaikki selvityksessä olleet järjestelmät olisi yritetty liittää kerralla mukaan, niin se olisi aiheuttanut huomattavan viipeen koko projektin toteutukseen.

3.5.1 Itella

Itellalta löytyy kaikille julkinen lähetyksen seurantajärjestelmä, joka toimii HTML-verkkosivuna. Tähän sivuun voi syöttää seurantakoodin ja sivu listaa lähetyksen etenemisen HTML-tauluna. Taulusta selviää lähetyksen tila, sekä se missä lähetys milloinkin on ollut.

3.5.2 Matkahuolto

Matkahuollolta löytyy julkinen lähetyksen seurantajärjestelmä HTML-verkkosivuna. Tämä järjestelmä kertoo HTML-taulussa lähetyksen vastaanottopaikan, määränpään, sekä milloin lähetys on lähtenyt liikkeelle, saapunut määränpäähän ja milloin se on luovutettu vastaanottajalle.

3.6 Ohjelmistorajapinta-palvelu

Ohjelmistorajapinta-palvelun tuli vastaanottaa seurantapyyntö. Tämän jälkeen hakea pyydetyllä seurantakoodilla oikeasta seurantajärjestelmästä lähetyksen tilatieto ja palauttaa se. Palvelun tuli myös palauttaa tarvittaessa virhe jos pakettia ei löydy tai jos palvelua yritetään käyttää väärillä tunnuksilla.

Molempien seurantapalveluiden, Itellan ja Matkahuollon tapauksessa järjestelmän tulee pyytää HTML-sivua julkisesta paketinseurantajärjestelmästä. Tämän jälkeen järjestelmä lukee sivulta paketin tilatiedot, käsittelee ne ja muuttaa oikeaan muotoon, jotta ne voidaan palauttaa kyselyn lähettäjälle.

3.7 SOAP-palvelu

Itse paketinseuranta-ohjelmistorajapintaa ei liitetty kiinteäksi osaksi Jemma-palvelua. Kutsut ohjelmistorajapintaan tehdään käyttäen SOAP-protokollaa. Tämä mahdollistaa sen, että ohjelmistorajapintaa voidaan kutsua mistä vain. SOAP-protokolla on tuettu myös useissa eri ohjelmointikielissä.

Ohjelmointirajapinnan kuvaamiseen käytetään WSDL-dokumenttia. Tämä on XML-pohjainen kuvauskieli, jolla kuvataan palvelun operatiivista tietoa. Tämän avulla rajapintaa hyödyntävät sovellukset osaavat tuoda rajapinnalle tarvittavat parametrit ja saavat tietoa minkä muotoista tietoa palvelu palauttaa.

3.8 Ohjelmointirajapinnan hyödyntäminen Jemmassa

3.8.1 Seurantakoodien hallinta

Myyjän tulee voida syöttää seurantakoodeja Jemman tapahtumanäkymään. Järjestelmän tulee tarkistaa seurantakoodin oikeellisuus. Seurantakoodia, joka liittyy jo aiemmin toimitettuun lähetykseen, ei pidä pystyä syöttämään. Jos seurantakoodia ei heti löydy järjestelmästä, voi olla kyse kirjoitusvirheestä, joten tällöin seurantakoodi piti olla mahdollista poistaa.

3.8.2 Seurannan tausta-ajo

Jemman tulee seurata paketteja tausta-ajona. Seurattavien lähetysten tilatietoja tulee kysellä seurantapalvelulta tunnin välein. Jos paketin tila on muuttunut, päivitetään tarvittaessa tapahtuman tila uuteen, sekä lähetetään ostajalle tai myyjälle sähköposti-ilmoitus tilamuutoksesta.

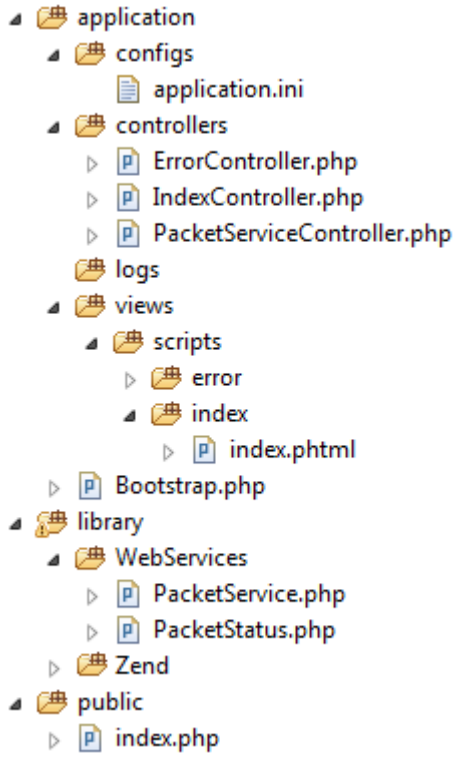
3.8.3 Viestit kuluttajille

Jemma lähettää sähköpostiviestin ostajalle, kun myyjä syöttää seurantakoodin järjestelmään. Ostajalle tulee myös ilmoittaa kun paketti on noudettavissa. Myyjälle tulee ilmoittaa kun ostaja on noutanut paketin.

4 TOTEUTUS

4.1 Ohjelmistorajapinta

Ohjelmistorajapintaa lähdettiin toteuttamaan omana PHP-sovelluksena. Sovellus toteutettiin hyödyntäen Zend Framework -ohjelmistokehystä. Tämä ohjelmistokehyksen valinta määritteli sitten sovelluksen kuvion 7 mukaiseksi.



KUVIO 7. Ohjelmistorajapintasovelluksen kansiorakenne

Public-kansion index.php, joka ainoa julkiseen verkkoon näkyvä PHP-tiedosto. Se hoitaa sovelluksen ajamisen määrittelemällä ja suorittamalla Zend_Application-olion, jonka ympärille koko sovellus rakentuu.

Application-kansio sisältää Bootstrap.php-tiedoston, joka esilataa sovellukselle yhteisiä olioita ja rekistereitä. Yhtenä esimerkkinä mainittakoon log-olio, jota

käytetään projektin ajonaikaiseen lokin kirjoittamiseen. Loki kirjoitetaan sovelluksessa log-kansioon. Application-kansio sisältää myös controller-kansion, joka sisältää sovelluksen ohjaimet, views-kansion sovelluksen käyttämille näkymille ja configs-kansion sovelluksen asetustiedostolle. Asetustiedostoina sovelluksessa on configs-kansiosta löytyvä application.ini, joka sisältää sovelluksen vakioita ja asetuksia. Se sisältää esimerkiksi asetuksen virheiden näytöstä loppukäyttäjälle, esimääritetyn polun sovellukselle, sekä mallien ja koodikirjaston polut. Koodikirjasto löytyy library-kansiosta. Se sisältää sovelluksen omien luokkien lisäksi myös koko Zend Framework -koodikirjaston.

4.2 SOAP-palvelu

SOAP-palvelu rakennettiin käyttäen Zendin omaa Zend_Soap_Server-luokkaa. Kun tämän PacketServiceController -nimeä kantavan luokan indexAction-funktiota kutsutaan, tarkastetaan onko kutsuja lähettänyt WSDL-parametrin. Jos WSDL-parametri löytyy vastaanotetuista parametreista, niin kutsujalle tullaan näyttämään SOAP-palvelun WSDL-dokumentti. Ohjelmistokehyksen oma Zend_Soap_AutoDiscover-luokka hoitaa dokumentin rakentamisen suoraan WebServices_PacketService-luokasta. Tilanteessa, missä WSDL-parametria ei tuoda, luodaan Zend_Soap_Server-olio. Tämä olio sisällyttää itseensä WebServices_PacketService-luokan, jota voidaan tämän jälkeen käyttää SOAP-rajapinnan yli.

4.3 PacketService-luokka

Tämän luokan tehtävänä on hakea pakettien seurantatiedot seurantapalveluista. Paketin seuraamisesta vastaa findPacket()-funktio, joka ottaa parametreina seurantakoodin lisäksi käytetyn seurantapalvelun tiedon, sekä käyttäjänimen ja salasanan. Näillä parametreilla voidaan hakea paketin tiedot palautettavaksi kutsun lähettäjälle, sekä varmistaa, että kutsujalla on myös oikeus käyttää palvelua.

4.3.1 Autentikonti

Kun findPacket()-funktio saa kutsun parametreilla, tarkastetaan ensin onko käyttäjänimi sekä salasana oikein. Tapauksessa, missä tunnukset eivät täsmää, palauttaa findPacket()-funktio SOAP-virheen viestillä ”Authorization failu-

re”. Jos käyttäjänimi ja salasana täsmäävät, voidaan kutsua paketinseuranta-järjestelmää vastaavaa funktiota.

4.3.2 Seurantapalveluiden HTML-parserointi

Molemmille seurantajärjestelmille, Itellalle ja Matkahuollolle tehtiin siis oma funktio, joka hoitaa paketin seurantatiedon muodostamisen. Molemmissa seurantajärjestelmissä pakettitiedon hakeminen perustui siis HTML-sivun parsimiseen. Tähän on tarjolla hyvä domDocument-luokka, joka on osa PHP-ydintä. Tämän luokan avulla koko HTML-sivun sisältö voidaan käydä läpi ja jakaa sivu HTML-solmuihin. Näiden solmujen avulla sivulta oli helppo etsiä tarvittava tieto paketin tilan selvittämiseksi. (PHP: Dom 2010.)

4.3.3 Virheiden käsittely

Virheitä sovelluksessa saattoi syntyä esimerkiksi väärän käyttäjänimen ja salasanan käytön seurauksena. Myös väärän seurantakoodin tai seurantapalvelun tuonti järjestelmään palauttaa virheen, koska tällöin sivun parserointi ei onnistu. Virheet palauttavat SoapFault-luokan olion poikkeuksena, joka sisältää tietoina esimerkiksi virheen kuvauksen, sekä virhekoodin. Tämän tiedon avulla ohjelmistorajapintaa käyttävä taho voi sitten havaita poikkeuksen ja suorittaa tarvittavia toimintoja.

Kaikki pyynnöt, tapahtumat ja virheet luokan sisällä kirjataan järjestelmän lokiin. Lokin avulla voidaan jälkeenpäin tutkia missä virhe on tapahtunut ja mikä sen on aiheuttanut.

4.4 Ohjelmistorajapinnan käyttö Jemmasta

Jotta toteutettua ohjelmistorajapintaa voitaisiin hyödyntää itse Jemmassa, luotiin Jemmaan PacketTracking-luokka. Tämä luokka tulisi vastamaan seurantapyynnöistä ohjelmistorajapinnan findPacket()-funktiolle.

PacketTracking-luokka on Jemman koodikirjastosta löytyvä luokka, jonka tehtävä on ottaa yhteys ohjelmistorajapintaan ja pyytää paketin tilatieto. Luokan getPacketInfo()-funktio ottaa parametreina paketin seurantakoodin, sekä seurantapalvelun tiedon. Tämän jälkeen Zend_Soap_Client-olion avulla kysyy ohjelmistorajapinnalta paketin tilatiedon. Jos ohjelmistorajapinta palauttaa sovellukselle poikkeuksen, niin tämän jälkeen tarkastetaan mikä poikkeuksen

aiheuttanut virhe oli kyseessä ja suoritetaan virheen vaatimat toiminnot. Tämän jälkeen palautetaan tapauskohtainen paluutieto takaisin paluuarvona. Jos pakettia ei esimerkiksi löydetä, palautetaan tilatieto, joka kuvaa pakettia, jota ei löydetty järjestelmästä. Kaikki luokan sisällä tapahtuneet pyynnöt ja palautteet kirjoitetaan järjestelmän lokitiedostoon, jotta virhetilanteissa voitaisiin nähdä mitä pyynnöissä on tapahtunut.

4.5 Seurantakoodien hallinta

Seurantakoodien syöttö ja hallinta toteutettiin Jemman kauppatapahtumanäkymään. Tästä näkymästä myyjä pystyi siirtymään linkillä seurantakoodien syöttönäkymään. Ostajalle ja myyjälle tapahtumanäkymä tarjosi linkin sivulle, josta seurantakoodeja ja pakettien tiloja pystyi tarkkailemaan.

4.6 Ajastetut seurantatehtävät

Ajastettuja tehtäviä varten luotiin oma Zend-projekti. Tähän projektiin lisättiin ohjain updatePacketStatuses, joka päivitti kaikkien seurannassa olevien pakettien tilatiedot. Projektiin lisättiin myös tehtävä updateTransactionStates, joka vastasi siitä, että jos kauppatapahtuman kaikki paketit on toimitettu ostajalle ja asetettu reklamaatioaika on kulunut, voidaan tapahtuma siirtää eteenpäin seuraavaan tilaan.

Tehtävien ajastus ja ajo hoidettiin Linux-palvelimen omalla crontab-ajastustoiminnolla, joka suorittaa komentosarjoja ennaltamäärättyinä kellonaikoina.

4.7 Palvelun testaus

Koko seurantaprosessin suoritusta testattiin ensin vanhojen lähetysten seurantakoodeilla. Kun koko Jemma-palvelua testattiin, lähetettiin myös oikeita lähetyksiä logistiikkayritysten seurantaan ja tarkasteltiin järjestelmän toimintaa lähetysten edetessä itse järjestelmästä, sekä järjestelmän lokeista. Testaajina toimi ohjelmoijien lisäksi yrityksen muuta henkilökuntaa, joilla osalla oli kokemusta testauksesta, ja osalla ei. Testausta suoritettiin luomalla testaajien välisiä kauppatapahtumia, joihin liitettiin aitoja lähetettyjä paketteja. Kun testaaja havaitsi virheen, oli se sitten kirjoitusvirhe tai virhe sovelluksen toiminnassa, pystyi hän sovelluksesta löytyvällä raportointityökalulla kirjoittamaan virheestä

kuvauksen ja lähettämään sen suoraan kuvankaappauksilla varustettuna projektinhallintajärjestelmään. Projektinhallintajärjestelmässä kyseiset virheet näkyvät sitten tehtävinä, joista ohjelmoija voi ottaa ne itselleen hoidettavaksi.

5 POHDINTA

Selvitys siitä, mitä logistiikkapalveluita otetaan mukaan ja minkälaisia rajapintoja lähetystenseurantaan oli tarjolla, osoittautui erittäin aikaa vieväksi tehtäväksi. Kaikista suunnitelluista logistiikkapalveluista ei ehditty saamaan edes kunnon vastausta, kun ominaisuutta olisi pitänyt lähteä toteuttamaan. Aikaa siis tälle selvitykselle olisi pitänyt varata enemmän.

Zend Framework -ohjelmistokehyksen käyttäminen sovelluksen toteutuksessa osoittautui erittäin hyväksi ratkaisuksi. Ohjelmistokehyksen kautta sovelluksen jakaminen MVC-mallin mukaiseksi helpotti osaltaan kokonaisuuden hahmottamista, sekä selkeytti sovelluksen rakennetta. Vikojen etsiminen helpottui, kun näytettävän koodin sijainti oli erillään tietokannan tauluja kuvaavan luokan koodista. Ohjelmistokehyksen sisältämät valmiit luokat helpottivat ja nopeuttivat ominaisuuksien toteuttamista, koska kaikkea ei tarvinnut lähteä tekemään itse. Esimerkiksi SOAP-palvelun pystytys oli erittäin helppoa, koska ohjelmistokehys sisälsi valmiin luokan tätä tarkoitusta varten. Ohjelmistokehyksen käyttäminen sovelluksessa vähentää myös syntyvän dokumentaation määrää, koska kaikki ohjelmistorajapinnan luokat on jo dokumentoitu, joten sovelluksen kehittäjän ei tarvitse dokumentoida kuin oma tuotoksensa kokonaisuudesta.

Palvelun testaus ja virheiden korjaus lähestyvän julkaisupäivän painostamana oli sekava prosessi. Aikaa ei ollut tarpeeksi, eikä testaussuunnitelmaa ollut laadittu. Tämä johti siihen, että testauksesta ei jäänyt minkäänlaista dokumentaatiota, eikä testausta voitu suorittaa suunnitellusti. Testaajat lähinnä testasivat omasta mielestään kaikki ominaisuudet läpi ja tämän jälkeen totesivat ominaisuuksien toimivan. Parempi lähestymistapa olisi voinut olla testaussuunnitelma, joka käy ominaisuudet yksitellen läpi ja jossa on jo valmiiksi mietitty erilaiset skenaariot mistä ominaisuuden täytyy selvitä. Tämä dokumentaatio pitäisi luoda ennen ominaisuuden tekemistä, koska se olisi myös hyvä apu toteutusvaiheessa. Ohjelmoija näkisi testit määrityksinä, joita ominaisuuden pitää täyttää, ja voisi itse testata nämä skenaariot toteutusvaiheessa. Testa-

uksessa olisi voinut myös hyödyntää yksikkötestausta sekä automatisoituun testaukseen sopivia PHP-kirjastoja, kuten PHPUnit.

Tuloksena saatiin toimiva järjestelmä, joka täytti sille annetut vaatimukset. Jatkokehityksen kannalta seurantaa hoitavasta järjestelmästä tehtiin mukautuva ja helposti päivitettävä, jotta tulevaisuuden seurantarajapinnat olisi helppo lisätä järjestelmään. Seurantajärjestelmä on tällä hetkellä käytössä Jemma-palvelussa.

LÄHTEET

Allen, R., Lo, N. & Brown, S. 2009. Zend Framework in Action.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. & Orchard, D. 2004. Web Service Architecture. Viitattu 20.3.2010. http://www.w3.org/TR/ws-arch

Brainalliance. 2009. Zend Framework-kurssimateriaali

Chistensen, E., Curbera, F., Meredith, G. & Weerawarana, S. 2001. Web Service Description Language. Viitattu 20.3.2010. http://www.w3.org/TR/wsdl

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H., Karmarkar, A. & Lafon, Y. 2007. SOAP version 1.2 Part 1: Messaging Framework (Second Edition). Viitattu 20.3.2010. http://www.w3.org/TR/soap12-part1

PHP: Dom 2010. Viitattu 20.3.2010. http://www.php.net/manual/en/book.dom.php