Opinnäytetyö (AMK)

Tietojenkäsittely

Tietojärjestelmät

2010

Mikael Kulma

VERKKOKAUPAN TOTEUTUS ZEND FRAMEWORKILLA

– case lastenkirjakauppa.fi

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely | Tietojärjestelmät

Marraskuu 2010 | Sivumäärä: 101

Ohjaaja: Anne Jumppanen

Tekijä: Mikael Kulma

VERKKOKAUPAN TOTEUTUS ZEND FRAMEWORKILLA

Tämä tutkimus perehtyy verkkosovellusohjelmointiin PHP:lla LAMP ympäristössä. Tutkimuksen tavoitteena on selvittää mitä hyötyjä verkkosovellusohjelmoinnissa on saavutettavissa Zend Framework -MVC-kehysjärjestelmän käyttöönotolla ja mitä haasteita sen käyttöönotto asettaa ohjelmoijalle.

Tutkimuksen soveltavana osuutena toteutetaan verkkokauppasovellus, käyttäen Zend Framework -kehysjärjestelmää. Verkkokaupan toteutuksen yhteydessä toteutetaan uudelleenkäytettävä Zend Framework -sovelluksen mallikerroksen käsittelyyn tarkoitettu ORM-luokkakirjasto.

Verkkokaupan tilaaja on turkulainen Saarni-Kirjat-kirjakustantamo.

ASIASANAT:

Internet, verkkokauppa, WWW-sivustot, PHP, ohjelmointi, tiedontallennus, relaatiotietokannat

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Information systems

November 2010| Total number of pages:: 101

Instructor: Anne Jummpanen

Author: Mikael Kulma

DEVELOPING AN ONLINE WEBSTORE WITH ZEND FRAMEWORK

This thesis concerns web-software development with PHP in a LAMP server environment. The focus of this study is to research the benefits attainable in web application development by deploying the Zend Framework MVC-framework and to find out which challenges this deployment poses to the developer.

The empirical part of this study concerns the development of an online webstore application which uses the Zend Framework as its platform. A reusable ORM-framework for managing the model layer of a Zend Framework application is also developed during the development of the webstore application.

The online webstore application was commissioned by the Saarni-Kirjat publishing company, which is located in Turku.

KEYWORDS:

Internet, webstore, WWW-sites, PHP, programming, datasaving, relationdatabases

SISÄLTÖ

1 JOHDANTO

Tämä opinnäytetyö on toteutettu toimeksiantona kustannusosakeyhtiö Saarni-Kirjat-kirjakustantamolle. Kustannusosakeyhtiö Saarni-Kirjat on lastenkirjoihin sekä lasten kuvakirjoihin erikoistunut suomalainen kirjakustantamo, jonka kustantamia lastenkirjoja löytyy kirjakaupoista ja kirjakahviloista kautta maan.

Saarni-Kirjat tilasi uusitun verkkokauppasovelluksen keskittyessään aikaisempaa enemmän kirjojen jälleenmyyntiin. Kustantamon vanhat sivut eivät enää täyttäneet uuden toimintamallin sivustolle asettamia vaatimuksia. Uusi sivusto päätettiin toteuttaa täysin räätälöitynä ratkaisuna, mikä tarkoitti järjestelmän kehittämistä alusta lähtien uudelleen.

Tämän opinnäytetyön soveltavana osuutena toteutetaan kyseinen verkkokauppasivusto käyttäen Zend Framework -kehysjärjestelmää. Verkkokaupan toteutuksen yhteydessä toteutetaan myös uudelleenkäytettävä Zend Framework -sovelluksen mallikerroksen käsittelyyn tarkoitettu olio-relaatio-mallinnus-luokkakirjasto.

Tutkimuksen teoriaosa perehtyy verkkosovellusohjelmointiin PHP:lla. Sen tavoitteena on selvittää, mitä hyötyjä verkkosovellusohjelmoinnissa on saavutettavissa Zend Framework -MVC-kehysjärjestelmän käyttöönotolla ja mitä haasteita sen käyttöönotto asettaa ohjelmoijalle. Edelleen selvitetään, mitä etua on Zend Framework -kehysjärjestelmän käytöstä verkkokaupan ohjelmointialustana sekä minkälaisia välineitä Zend Framework tarjoaa ohjelmoijan käyttöön verkkokaupan sovelluskehitystä ajatellen.

Tutkimuksen lähteinä käytetään paitsi sekä alan kirjallisuutta että Internetistä saatavilla olevia lähteitä, myös tutkimuksen työvälineinä käytettävien ohjelmistojen referenssidokumentaatiota.

2 SOVELLUSOHJELMOINTI PHP:LLA

2.1 Käsiteltävä aihealue

Seuraavassa perehdytään lyhyesti PHP:n ominaisuuksiin, siltä osin kuin ne ovat tutkimuksen kannalta merkityksellisiä. Tämä tarkoittaa sitä, että jotkin PHP:n ominaisuudet käsitellään pintapuolisesti, ja toisia ei käsitellä lainkaan. PHP:n asennusta, sen välimuistiominaisuuksia tai laajennettavuutta ei käsitellä, sillä vaikka nämä ovat tärkeitä aiheita PHP:n tuntemuksen kannalta, ei niillä ole tutkimusongelman rajoissa riittävän suurta painoarvoa.

2.2 PHP: Hypertext Preprocessor

PHP, joka saa nimensä rekursiivisesta akronyymistä PHP: Hypertext Preprocessor on avoimeen lähdekoodiin perustuva skriptauskieli, joka soveltuu erityisen hyvin internetsivustojen ohjelmointiin. PHP:n syntaksi perustuu C-kieleen, Javaan ja Perliin ja se on varsin helppo oppia. (Thomson & Welling 2009, 3-5.)

Toisin kuin C tai Perl, PHP ei sisällä monimutkaisia komentoja HTML:n tulostusta varten. Sen sijaan PHP-sivut koostuvat HTML-koodista, jonka sisään on upotettu PHP-koodia. PHP-koodi sijoitetaan erityisten aloitus- ja lopetuskäskyjen, <?php ja ?> väliin. Nämä käskyt sallivat HTML-sivun siirtyä PHP-tilaan ja siitä pois tarpeen mukaan. (Achour ym. 2010a.) Alla on esimerkki HTML-dokumentista, johon on upotettu PHP-koodia.

```
<html>
    <head>
        <title>Yksinkertainen PHP-esimerkki</title>
    </head>
```

```
    <body>
        <p> <?php echo "Hei Maailma!"; ?> </p>
    </body>
</html>
```

PHP:n toiminnallisuutta kutsutaan palvelinpuolen skriptaukseksi. HTML:ää luova PHP-koodi suoritetaan palvelimella ja lopputulokseksi saatu HTML-dokumentti lähetetään takaisin asiakassovellukselle. Tällöin asiakas ei ole tietoinen palvelimella suoritettavasta prosessoinnista. (Ullman 2009, xv.)

PHP:n uusin vakaa versio tämän tutkimuksen kirjoitushetkellä on 5.3.3, ja se käyttää Zend Engine:n versiota 2.

2.3 PHP:n sovellusalueet

PHP:n merkittävin sovellusalue on palvelinpuolen skriptaus, mutta kieltä on mahdollista käyttää muihinkin tarkoituksiin.

PHP skriptejä voidaan käyttää

- palvelinpuolen skriptaukseen
- komentoriviskriptaukseen
- työpöytäsovellusten kirjoittamiseen. (Achour ym. 2010b.)

2.3.1 Palvelinpuolen skriptaus

Palvelinpuolen skriptaus on perinteisin PHP:n sovellusalue. Siihen tarvitaan PHP-parseri, joka useimmiten asennetaan WWW-palvelimen moduuliksi, WWW-palvelin ja WWW-selain. PHP-sovelluksen tulosteeseen päästään käsiksi WWW-selaimella pyytämällä palvelimelta sivu, johon on upotettu PHP-koodia. (Ullman 2009, xv.)

PHP:n toiminnallisuus ei rajoitu vain HTML:n tulostamiseen. PHP:lla on mahdollista tuottaa kuvia, PDF-tiedostoja ja jopa dynaamisesti luotuja Flash-videoita. Minkä tahansa tekstin, kuten XML:n tuottaminen on niin ikään helppoa. PHP kykenee luomaan sisällön automaattisesti ja tallentamaan lopputuloksen tiedostojärjestelmään tulostuksen sijaan, jolloin valmiita dokumentteja voidaan käyttää välimuistina dynaamiselle sisällölle. (Thomson & Welling 2009, 5.)

2.3.2 Komentoriviskriptaus

Palvelinpuolen skriptauksen lisäksi PHP-skripti on mahdollista suorittaa komentorivillä ilman WWW-palvelinta, tai selainta. Tällöin tarvitaan ainoastaan PHP-parseri. Tämänkaltainen käyttö soveltuu parhaiten ajoitettujen tehtävien suorittamiseen CRONin avulla. (Achour ym. 2010b.)

2.3.3 Työpöytäsovellusten kirjoittaminen

PHP ei todennäköisesti ole paras mahdollinen kieli työpöytäsovellusten ohjelmointiin. Tästä huolimatta mikäli ohjelmoija on kokenut PHP-ammattilainen, ja haluaa käyttää hyväkseen joitakin PHP:n edistyneempiä ominaisuuksia, voidaan PHP:ta käyttää myös kyseiseen tarkoitukseen PHP-GTK -kirjaston avulla. (Achour ym. 2010b.)

2.4 PHP ja tietokannat

PHP:n vahvin ja merkittävin ominaisuus on kattava tuki monille tietokannanhallintajärjestelmille. PHP tukee mm. IBM DB2, Informix, MySQL, Oracle, PostgreSQL, SQlite ja Sybase tietokantoja. (Ullman 2009, 354.)

Tämän lisäksi PHP tarjoaa tietokannan abstraktiokerroksen PDO:n. PDO sallii minkä tahansa sen tukeman tietokantajärjestelmän käytön yhtenäisen rajapinnan kautta. PHP tukee myös ODBC-yhteysstandardia. (Achour ym. 2010b.)

2.5 PHP ja tietoliikenne

PHP sisältää kattavan tuen yhteyksien muodostamiseen sovelluksen ulkopuolisiin palveluihin. Protokollat, kuten LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM kuuluvat tuen piiriin. PHP:lla on myös mahdollista käyttää pistokkeita ja kommunikoida käyttäen räätälöityjä protokollia. PHP tukee WDDX tiedonvaihtoa lähes kaikkien Web-ohjelmointikielien välillä. Kielellä on mahdollista instantioda Java-olioita ja käyttää niitä tavallisina PHP-olioina. Etäolioihin voidaan päästä käsiksi CORBA-laajennuksen avulla. (Achour ym. 2010b.)

2.6 Olio-ohjelmointi PHP:lla

PHP:n versiosta 5 alkaen kielen oliomallia on uudelleenkirjoitettu. Tavoitteena on ollut parempi suorituskyky ja uusien olio-ohjelmointiin liittyvien ominaisuuksien esittely. Tämä on merkittävä muutos verrattuna PHP 4:ään. (Gillmore 2008, 163.)

Versiossa 5 lisättyihin ominaisuuksiin kuuluvat näkyvyysmääreet, final-luokat ja metodit, rajapinnat, kloonaus ja parametrien tyyppivihjeet. Tämän lisäksi kieleen lisättiin uusia maagisia metodeja. (Gillmore 2008, 163.)

Toisin kuin PHP 4, PHP 5 käsittelee olioita viitemuuttujien avulla, mikä tarkoittaa sitä, että oliota symboloiva muuttuja ei sisällä arvonaan oliota, vaan ainoastaan viitteen siihen. Näitä viitemuuttujia kutsutaan PHP:ssa aliaksiksi. Kun olio annetaan parametriksi metodille, palautetaan tai sijoitetaan toiseen muuttujaan, käsiteltävät muuttujat eivät ole aliaksia, vaan sisältävät itse asiassa viitteen aliakseen, joka puolestaan viittaa itse olioon. (Achour ym. 2010g.)

Eräät PHP:n oliomallin ominaisuudet eroavat tavanomaisesta olio-ohjelmointiparadigmasta siinä määrin, että ne edellyttävät lähempää tarkastelua. Näihin kuuluvat

- luokkien automaattinen lataaminen
- maagiset metodit
- rakentajat ja purkajat
- metodien ja jäsenmuuttujien kuormittamien.

2.6.1 Luokkien automaattinen lataaminen

Luokkien automaattinen lataaminen helpottaa ohjelmoijan työtä tarjoamalla vaihtoehtoisen ratkaisun lähdekooditiedostojen lataamiseen.

Olio-ohjelmoinnin käytäntöjen mukaisesti ohjelmoijat usein sijoittavat yhden luokan lähdekoodin yhteen tiedostoon. Suuressa sovelluksessa tällaisia luokkatiedostoja kerääntyy valtavia määriä. Eräs PHP:n harmillinen edellytys on jokaisen tarvittavan luokan lähdekoodin sisältävän tiedoston liittäminen include-lauseella skriptin alussa. (Thomson & Welling 2009, 188.)

PHP 5 tarjoaa __autoload funktion, jota kutsutaan automaattisesti silloin, kun sovellus yrittää käyttää määrittelemätöntä luokkaa, tai rajapintaa. Kutsumalla tätä funktiota PHP saa viimeisen mahdollisuuden ladata kutsuttu luokka, ennen suorituksen lopettamista virheeseen. (Gillmore 2008, 192.) Määrittelemällä luokkien lataamiseen liittyvä toiminnallisuus __autoload funktiossa voidaan include-lauseet jättää kirjoittamatta.

2.6.2 Maagiset metodit

Funktiot __construct, __destruct, __call, __callStatic, __get, __set, __isset, __unset, __sleep, __wakeup, __toString, __invoke, __set_state ja __clone ovat PHP:n “maagisia” metodeita. Näitä nimiä ei tule käyttää ohjelmakoodissa, ellei tarkoitus ole käsitellä niihiin liittyvää automaattista toiminnallisuutta. PHP varaa kaikki ”__”-alkuiset funktionnimet kyseiseen käyttöön. (Achour ym. 2010c.)

Hyvä esimerkki maagisesta metodista on __set, jota kutsutaan silloin, kun olion määrittelemättömälle, tai näkyvyysmääreen piilottamalle jäsenmuuttujalle pyritään asettamaan arvoa. __set-metodin oletustoiminnallisuus asettaa oliolle tällöin uuden jäsenmuuttujan, mikäli metodille ei ole määritelty luokassa vaihtoehtoista toiminnallisuutta.

2.6.3 Rakentajat ja purkajat

PHP 5 sallii ohjelmoijan määritellä rakentajametodin luokalle maagisella metodilla __construct. Tämä metodi kutsutaan luokan jokaisen uuden instanssin luomisen yhteydessä, joten se soveltuu hyvin olion alustamiseen ennen käyttöä. (Gillmore 2008, 182)

Isäntäluokan rakentajaa ei kutsuta PHP:ssä automaattisesti, jos perivä luokka määrittelee __construct metodin. Mikäli näin halutaan tehdä, on isäntäluokan rakentajaa kutsuttava eksplisiittisesti perivän luokan rakentajassa. Tämä tapahtuu komennolla parent::__construct(). (Achour ym. 2010d.)

PHP 5 tarjoaa myös purkaja-metodin __destruct, joka kutsutaan kun kaikki viitteet tiettyyn olioon poistetaan tai kun olio tuhotaan eksplisiittisesti (Gillmore 2008, 186)

Samoin kun rakentajien kohdalla, isäntäluokan __destruct-metodia ei kutsuta automaattisesti, vaan se on kutsuttava komennolla parent::__destruct() perivän luokan purkaja-metodissa (Achour ym. 2010d).

2.6.4 Metodien ja jäsenmuuttujien kuormittaminen

PHP:n tulkinta kuormittamisesta eroaa termin määritelmästä useimmissa muissa olioperusteisissa ohjelmointikielissä. Perinteisesti kuormittaminen merkitsee sitä, että yhden luokan sisällä on mahdollista määritellä monta samannimistä metodia eri parametreilla. (Achour ym. 2010e.)

PHP:ssa kuormittaminen tarkoittaa olion metodien ja jäsenmuuttujien luomista dynaamisesti. Nämä dynaamiset entiteetit käsitellään maagisten metodien avulla. (Thomson & Welling 2009, 186-187.)

Kuormittavat metodit otetaan käyttöön kutsuttaessa olion sellaisia metodeja ja jäsenmuuttujia, joita ei ole määritelty luokassa tai jotka eivät ole näkyvissä kutsun näkyvyysalueella. (Achour ym. 2010e.)

Jäsenmuuttujien kuormittaminen käsitellään maagisilla metodeilla __set, __get, __isset ja __unset.

- __set suoritetaan kirjoitettaessa dataa määrittelemättömään jäsenmuuttujaan.
- __get suoritetaan luettaessa dataa määrittelemättömästä jäsenmuuttujasta.
- __isset suoritetaan kutsuttaessa funktioita isset, tai empty määrittelemättömällä jäsenmuuttujalla.
- __unset suoritetaan kutsuttaessa funktiota unset määrittelemättömällä jäsenmuuttujalla. (Achour ym. 2010e.)

Maagiset metodit saavat parametrikseen sen määrittelemättömän jäsenmuuttujan nimen, jota yritetään käsitellä. __set saa tämän lisäksi parametrikseen arvon, joka kyseiseen jäsenmuuttujaan yritetään sijoittaa. (Achour ym. 2010e).

Jäsenmuuttujien kuormittaminen toimii vain olion näkyvyysalueella. Maagisia metodeja ei suoriteta luokan näkyvyysalueella. Tästä syystä mikäli maagisia metodeja määritellään luokassa, ei niiden yhteydessä tule käyttää avainsanaa static (Achour ym. 2010e).

Metodien kuormittaminen käsitellään maagisilla metodeilla __call ja __callStatic.

- __call suoritetaan kutsuttaessa määrittelemätöntä metodia olion näkyvyysalueella.
- __callStatic suoritetaan kutsuttaessa määrittelemätöntä metodia luokan näkyvyysalueella. (Achour ym. 2010e.)

Metodit saavat parametrikseen sen määrittelemättömän metodin nimen, jota yritetään kutsua, sekä taulukon, joka sisältää määrittelemättömälle metodille annetut parametrit (Achour ym. 2010e).

2.7 PHP:n Reflektiivisyys

PHP 5 tarjoaa kattavan reflection-API:n. Sen avulla on mahdollista takaisinmallintaa luokkia, rajapintoja, funtioita, metodeja ja laajennuksia. Lisäksi reflection-API tarjoaa menetelmän luokkien, metodien ja funktioden kommenttilohkojen lukemiseen. (Thomson & Welling 2009, 190-191.)

2.8 PHP:n Nimiavaruudet

Eräs PHP 5.3:n merkittävimpiä uudistuksia oli nimiavaruuksien tuominen PHP:hen. Nimiavaruudet on suunniteltu ratkaisemaan kaksi merkittävää ongelmaa, joita ohjelmoijat kohtaavat kehittäessään uudelleenkäytettäviä koodielementtejä, kuten luokkia ja funktioita. Nämä ongelmat ovat nimeämiseen liittyvät sekaannukset ja koodin huono luettavuus (Achour ym. 2010f).

Nimeämisongelmat syntyvät sovelluskoodin sisältäessä luokan- tai funktionnimiä, jotka on jo määritelty PHP:n sisäisissä funktioissa, rajapinnoissa, tai luokissa tai jotka määritellään sovelluksen käyttämissä kolmannen osapuolen ohjelmistokirjastoissa. Tällöin PHP:n on mahdotonta tietää mitä samannimisistä luokista, tai funktioista sen tulisi kutsua. (Gillmore 2008, 208-208)

Koodin huono luettavuus on suoraan seurausta ohjelmoijien pyrkimyksistä kiertää nimeämiseen liittyviä ongelmia. Tavanomaisin menetelmä on keinotekoisen nimiavaruuden luominen liittämällä luokan koko nimiavaruus sen nimeen. Menetelmä on toimiva, mutta sen tuloksena syntyvät luokannimet ovat pitkiä ja sekoittavat koodia tarpeettomasti. (Achour ym. 2010f.)

Nimiavaruuden erotin PHP:ssa on kenoviiva. Alla on konkreettinen esimerkki luokasta jonka nimiavaruus on liitetty luokan nimeen, sekä esimerkki siitä, miten luokka tulee määritellä PHP 5.3:n tarjoamien nimiavaruuksien mukaisesti.

```
// vanha käytäntö:

class Zend_Db_Table_Row extends Zend_Db_Table_Row_Abstract {

    //…

}

// uusi käytäntö:

namespace Zend\Db\Table

class Row extends RowAbstract {

    //…

}
```

2.9 PHP:n ongelmia

PHP:n vahvuus on sen käytön edellyttämä vaatimaton perehtyneisyys. Kuka tahansa, joka hallitsee tietotekniikan perusteet, voi sen avulla ohjelmoida yksinkertaisia dynaamisia internetsivuja. Juuri kielen helppokäyttöisyys on johtanut siihen, että PHP kannattelee yli 20 miljoonaa internetsivustoa ympäri maailman. Tämä luku kasvaa tuhansilla joka päivä. (Lecky-Thompson ym. 2009.)

Toisaalta nimenomaan helppokäyttöisyys on johtanut yleisesti hyväksyttyjen ohjelmointimenetelmien puutteeseen. PHP:lle ei ole olemassa standardoituja parhaita käytäntöjä. Lyhyesti sanottuna osa tuotantokäytössä olevasta PHP koodista ei ole järin hyvää. (Lecky-Thompson ym. 2009.)

PHP on suunniteltu tukemaan sovelluslogiikan sisältävän koodin sijoittamista HTML-tiedostojen sisään. Tästä syystä jopa kieltä käyttävät ammattilaiset saattavat välttää ohjelmistotuotannon standardien soveltamista ohjelmoidessaan PHP:lla. (Zandstra 2008, 4)

Ongelmia syntyy, kun perinteisellä tavalla tuotettua sovellusta on tarpeen laajentaa. Tällöin yhden muutoksen tekeminen voi vaatia kymmenien PHP-koodia sisältävien sivujen muokkausta. (Zandstra 2008, 4)

Eräänlaisen ratkaisun näihin ongelmiin tarjoavat PHP:lle saatavilla olevat kehysjärjestelmät (Lecky-Thompson ym. 2009, 355).

3 KEHYSJÄRJESTELMÄT

3.1 Käsiteltävä aihealue

Seuraavassa tarkastellaan ensin kehysjärjestelmä-termin merkitystä yleisesti ja sen jälkeen perehdytään useimpien kehysjärjestelmien tarjoamiin ominaisuuksiin. Edelleen tarkastellaan sitä, mitä asioita on syytä huomioida kehysjärjestelmää valittaessa ja mitä käytännön hyötyjä kehysjärjestelmän käyttöönotosta on saatavissa.

On huomattava, että tässä tutkimuksessa keskitytään kehysjärjestelmiin PHP:n asiayhteydessä. Tarkasteltavat kehysjärjestelmiin liittyvät asiat, kuten suunnittelumallit, on ymmärrettävä PHP:n yleisimmän sovellusalueen eli verkkosovellusohjelmoinnin näkökulmasta.

3.2 Kehysjärjestelmän määritelmä

Kehysjärjestelmä on kokoelma valmiita luokkia, joiden tavoite on tehdä sovellusohjelmoinnista helpompaa ja nopeampaa. Kehysjärjestelmät toimivat usein hyvin läheisessä suhteessa alla olevaan palvelinympäristöön, ja käyttävät standardoidulla tavalla hyväkseen sen tarjoamia palveluita. Tällöin ohjelmoija on pakotettu noudattamaan järjestelmän ohjelmointikäytäntöjä. (McArthur 2008, 203.)

Monet kehysjärjestelmät määrittelevät sovellukselle oletusrakenteen, jota ohjelmoijan tulee seurata. Tämä voi olla yksinkertaisesti suositeltu hakemistorakenne, esimerkiksi sovelluksen toiminnallisuutta ja esityslogiikkaa käsittelevien tiedostojen jakaminen omiin hakemistoihinsa. Edistyneemmällä tasolla se voi tarkoittaa HTTP-pyynnön kapselointia kehysjärjestelmän luokkiin.

Tällöin järjestelmä ”reitittää” pyynnön sovellusluokille, jotka toteuttavat ennalta määritellyn rakenteen. (Lecky-Thompson ym. 2009, 356.)

Reitityksestä puhutaan usein niiden kehysjärjestelmien kohdalla, jotka toteuttavat MVC-suunnittelumallin. MVC:tä käsitellään lisää luvussa 3.5. Kyseisissä kehysjärjestelmissä HTTP-pyyntöä osoitteeseen /admin/edituser.php?id=1234 ei hyväksytä. Sen sijaan vastaava pyyntö osoitteeseen /user/edit/1234 saattaisi kutsua user-luokan edit-metodin. (Allen ym. 2008, 26.)

Tällainen lähestymistapa hävittää täysin käsitteen PHP:sta lineaarisena skriptauskielenä. Sovellus jaetaan sen sijaan yksittäisiin komponentteihin, jotka reagoivat käyttäjän syötteeseen. Lähestymistapa nopeuttaa sovelluskoodin kirjoittamista ja vähentää turhaa toistoa.

3.3 Kehysjärjestelmän tarjoamat palvelut

PHP:lle on olemassa huomattava määrä kehysjärjestelmiä. Lähes kaikki niistä ovat ilmaisia. Toisia ylläpidetään paremmin kuin toisia. Toisille on olemassa enemmän tukipalveluita kuin toisille. Vain yhdellä niistä voidaan sanoa olevan jonkinlainen hyväksyntä PHP:n kehittäjäyhteisöltä. Kaikki niistä pyrkivät silti tarjoamaan joukon ydinpalveluita. Näihin kuuluvat

- toiminnallisuuden ja esityslogiikan erottelu
- olio-relaatio-mallinnus
- URL-osoitteiden uudelleenkirjoitus
- validointi
- lomakkeiden tilan tallennus
- apuluokat. (Lecky-Thompson ym. 2009, 355-356.)

3.3.1 Toiminnallisuuden ja esityslogiikan erottelu

Kaikki PHP:n kehysjärjestelmät eivät toteuta MVC-suunnittelumallia. Tästä huolimatta lähes jokainen kehysjärjestelmä esittelee parhaat käytännöt toiminnallisuuden ja esityslogiikan erottamiseen toisistaan. (McArthur 2008, 203-204.)

Erottelu voidaan toteuttaa monella tapaa. Yksinkertaisimmillaan voidaan PHP-skriptit jakaa kahteen tiedostoon, toinen esityslogiikkaa ja toinen toiminnallisuutta varten. Toinen tapa on käyttää jotakin hyväksi havaittua esityslogiikan käsittelyyn tarkoitettua alustaa, kuten Smartya. Edistyneimmillään erottelu tarkoittaa MVC-arkkitehtuurin toteutusta. (Allen ym. 2008, 17.) Luvussa 3.5 tutustutaan lähemmin MVC-suunnittelumallin esittelemään tapaan toteuttaa erottelu.

3.3.2 Olio-relaatio-mallinnus

Olio-relaatio-mallinnus eli ORM tarkoittaa sovelluksen oliomallin tallennusta tietokantaan ilman SQL-lauseiden kirjoittamista. Kehysjärjestelmät tarjoavat useimmiten perustason tietokantatoiminnallisuuden valmiina. Tätä toiminnallisuutta kuvaa akronyymi CRUD, joka tulee sanoista create, read, update ja delete. Toisin sanoen kehysjärjestelmät sisältävät toiminnallisuuden tietokantaan tallennetun olion luomiseen, lukemiseen, päivittämiseen ja poistoon. Tämä helpottaa ohjelmoijan työtä vähentämällä kirjoitetun koodin määrää ja takaamalla oliomallin siirrettävyyden tietokanta-alustasta toiseen. (Lecky-Thompson ym. 2009, 358-359.)

3.3.3 URL-osoitteiden uudelleenkirjoitus

Websovellusten muututtua monimutkaisemmiksi myös niiden HTTP-GET-pyyntöihin käyttämät URL-osoitteet ovat pidentyneet. Useimmiten osoitteet ovat

epäselvä sekoitus istuntodataa ja pyynnön parametreja. Tällaiset osoitteet aiheuttavat ongelmia mm.

- kopioitaessa osoite sähköpostiviestiin
- pyrittäessä muistamaan osoite ulkoa
- hakukoneen indeksoidessa osoitteen.

Pitkät osoitteet voivat myös paljastaa osan sovelluksen toiminnallisuudesta ulkopuoliselle tarkkailijalle. Tällöin niiden voidaan sanoa aiheuttavan tietoturvariskin. (Lecky-Thompson ym. 2009, 357.)

Tämän vuoksi monet kehysjärjestelmät tarjoavat jonkinlaisen mekanismin URL-osoitteiden uudelleenkirjoittamista varten. Mekanismi voi olla staattinen, jolloin ohjelmoija määrittelee esim. osoitteen /home viittaavan home.php -tiedostoon, tai dynaaminen, kuten luvun 3.2 esimerkissä. (Allen ym. 2008, 26.)

URL-osoitteiden uudelleenkirjoitus toteutetaan yhteistyössä WWW-palvelimen kanssa. Palvelin ottaa HTTP-pyynnön vastaan ennen sen lähettämistä edelleen PHP:n käsiteltäväksi. Tällöin uudelleenkirjoituksen toteuttaminen on lähes mahdotonta ilman palvelintason muutoksia. Apachen tapauksessa uudelleenkirjoitus otetaan käyttöön asentamalla moduuli mod_rewrite. (Lecky-Thompson ym. 2009, 357.)

3.3.4 Validointi

Käyttäjän syötteen validointi on merkittävä osa internetsivustojen ohjelmointia. Osa validoinnista, kuten sähköpostiosoitteen oikeellisuuden tarkistaminen, on helposti toteutettavissa. Osa vaatii monimutkaisempaa toiminnallisuutta, kuten tietokantaan tallennettavan tiedon eheyden varmistaminen. Mikä tahansa validoinnin tehtävä onkin, yhtenäinen ja helposti ylläpidettävä lähestymistapa on toimivan sovelluksen perusedellytys. (Zend Technologies 2010b.)

Useimmat kehysjärjestelmät tarjoavat omat toteutuksensa validointitarkoituksiin. Nämä käsittävät luokat rajoitteille, virheiden raportoinnin ja tallentamisen sekä rajoitteiden monimuotoisuuden. (Lecky-Thompson ym. 2009, 357.)

Pienessä projektissa useimpia näistä tuskin tarvitaan. Ne ovat hyödyllisiä lähinnä yhtenäisen lähestymistapansa vuoksi, mikä tekee niistä helposti ylläpidettäviä.

3.3.5 Lomakkeiden tilan tallennus

Hyvin toimiva validointijärjestelmä ei ole täydellinen ilman lomakkeiden tilan tallennusta. Tallennuksen tavoite on varmistaa, että lomake-elementtien tila pysyy samana HTTP-pyyntöjen välillä. Tällöin esim. lomake, jonka kaikkia vaadittuja kenttiä ei ole täytetty, palautetaan käyttäjälle samassa tilassa, kuin se lähetettiin. (Zend Technologies 2010b.) Näin internetsivuston käyttö saadaan muistuttamaan enemmän työpöytäsovelluksen käyttöä.

Useimmat kehysjärjestelmät tarjoavat valmiin toiminnallisuuden lomakkeiden tilan tallennukseen (Lecky-Thompson ym. 2009, 357).

3.3.6 Apuluokat

Kehysjärjestelmät sisältävät usein joukon luokkia, jotka eivät näytä sopivan mihinkään yhteen kokonaisuuteen. Tällaiset luokat suorittavat toimintoja, kuten syötteen salaus, sähköpostien lähetys, käyttäjien autentikointi ja niin edelleen. Näitä luokkia kutsutaan apuluokiksi. Monet niistä sisältävät toiminnallisuutta jonka toivoisi löytyvän PHP:sta oletuksena. (Lecky-Thompson ym. 2009, 358.)

3.4 PHP:n kehysjärjestelmän valintakriteerit

Jokainen kehysjärjestelmä omaa tiettyjä etuja ja joitakin haittoja verrattuna kilpailijoihinsa. Järjestelmää valittaessa kannattaa vertailla viittä merkittävää osatekijää. Näihin kuuluvat

- arkkitehtuuri
- dokumentointi
- yhteisö
- tuki
- joustavuus. (McArthur 2008, 204.)

3.4.1 Arkkitehtuuri

Tärkein kehysjärjestelmän valintakriteeri on sen arkkitehtuuri. Sovellusohjelmoijan on hallittava käyttämänsä kehysjärjestelmän toiminnallisuus hyvin syvällisellä tasolla. Muussa tapauksessa kehysjärjestelmän käyttöönotto saattaa hidastaa, eikä nopeuttaa kehitystyötä. (McArthur 2008, 204.)

Kehysjärjestelmän tekniset vaatimukset, tiedostojen sijainti ja hakemistorakenne on selvitettävä tarkoin, jotta voidaan varmistua järjestelmän sopivuudesta käytössä olevaan palvelinympäristöön. (Allen 2009, 15-16.)

3.4.2 Dokumentaatio

Kehysjärjestelmän dokumentaatio saattaa joidenkin järjestelmien kohdalla merkitä pelkkää mukana tulevaa asennusohjetta, siinä missä toisaalla tarjotaan muodollisia valmennuskursseja. Dokumentaation laadukkuutta arvioidessa tulee

tarkastella kolmea osatekijää: ajankohtaisuutta, ymmärrettävyyttä ja kattavuutta. (McArthur 2008, 204.)

Dokumentaatio on hyödytöntä, jos käytössä oleva kehysjärjestelmän versio on uudempi kuin dokumentoitu versio tai jos dokumentaation teksti on epäselvää ja vaikeasti ymmärrettävissä. (McArthur 2008, 204.)

Toisaalta saatavilla on järjestelmiä, joiden esittelydokumentit ovat hyvinkin kattavia ja selkeitä, mutta joiden edistyneempi dokumentaatio on tuotettu automaattisesti lähdekoodin kommenteista. (Lecky-Thompson ym. 2009, 378.)

Hyvä dokumentaatio on perusedellytys kehysjärjestelmien tuottamien aika- ja kustannusäästöjen saavuttamiseksi (McArthur 2008, 204).

3.4.3 Yhteisö

Kehysjärjestelmän ympärillä olevan yhteisön tarkastelu on tärkeä osa järjestelmän arviointia. Tarkastelun kohteiksi kannattaa ottaa se, millä tavalla järjestelmän kehittämiseen on mahdollista osallistua ja kuinka pitkään järjestelmän kehittäjillä kestää vastata bugi-ilmoituksiin. (McArthur 2008, 205.)

Kehysjärjestelmää valitsevan organisaation kannalta yhteisön laajuus on merkittävä tekijä, sillä se vaikuttaa suoraan siihen, kuinka helppoa osaavan henkilöstön löytäminen on tarvittaessa (Lecky-Thompson ym. 2009, 379.)

3.4.4 Tuki

Tukipalveluiden tarjoaminen liittyy olennaisesti joidenkin kehysjärjestelmiä tuottavien organisaatioiden ansaintamalliin. Tällöin kehysjärjestelmä on ilmainen, mutta tukipalvelut ovat maksullisia. Toisille järjestelmille tukea ei tarjota lainkaan. Tällöin tuki on hankittava kolmannelta osapuolelta. (McArthur 2008, 205.)

Kehysjärjestelmäyhteisön postituslistat ja keskustelufoorumit tarjoavat ilmaista tukea. Tällainen tuki on usein tasoltaan vaihtelevaa, eikä vastaa luotettavuudessaan maksullista tukea. (McArthur 2008, 205.)

Ennen kehysjärjestelmän käyttöönottoa on tarkoin selvitettävä sille saatavilla olevan tuen määrä. Nopean ratkaisun puuttuminen ongelmatilanteissa voi osoittautua hyvinkin kalliiksi. (McArthur 2008, 205.)

3.4.5 Joustavuus

Kehysjärjestelmän joustavuudella tarkoitetaan sitä, miten helppoa se on integroida olemassa oleviin sovelluksiin, ja sitä minkälaisia teknisiä vaatimuksia se asettaa kehitysympäristölle. (Lecky-Thompson ym. 2009, 378.)

Eräät kehysjärjestelmät ovat huomattavan joustamattomia. Ne voivat asettaa tarkkoja rajoituksia koodin kirjoittamiselle ja organisoinnille, määritellä sallitut menetelmät esityslogiikan hallintaan tai sisältää räätälöityä syntaksia, joka on opiskeltava ennen järjestelmän käyttöä. (McArthur 2008, 205.)

Kehysjärjestelmiä arvioitaessa kannatta suosia järjestelmiä, jotka asettavat kehitystyölle mahdollisimman vähän ehdottomia rajoitteita.

3.5 MVC-suunnittelumalli

MVC, joka tulee sanoista model, view ja controller on suunnittelumalli, joka yksinkertaistaa sovelluksen kehitystä ja ylläpitoa. Tämä yksinkertaisuus saavutetaan erottelemalla sovellus kolmeen loogiseen kerrokseen: malliin, näkymään ja ohjaimeen. (Lyman 2009, 1).

Ohjelmistotuotannossa suunnittelumallien tavoitteena on lähestyä jonkin tietyn ongelman ratkaisemista yhtenäisellä ja testatulla tavalla. MVC-suunnittelumallin tapauksessa ongelma on se, miten erotella toisistaan sovelluksen käyttöliittymä, varsinaisen tiedon prosessointi ja sovelluksen sisäinen toiminnallisuus. (Lecky-

Thompson ym. 2009, 306.) Useimmat PHP:n kehysjärjestelmät toteuttavat MVC-suunnittelumallin.

Seuraavassa perehdytään siihen, miten MVC:n kerrokset tavanomaisesti toteutetaan verkkosovellusten ohjelmoinnissa.

3.5.1 Malli

Sovelluksen mallikerros on vastuussa sen tiedonkäsittelyn logiikasta. Se kapseloi tietokantayhteydet ja tarjoaa uudelleenkäytettävän luokkakirjaston. Mallista löytyy usein toiminnallisuus mm. tietokannan abstrahointiin, datan validointiin, sähköpostien lähettämiseen ja käyttäjien autentikointiin. (Lyman 2009, 2.)

3.5.2 Näkymä

Näkymäkerros on se osa sovellusta, jonka kehitys yleisesti mielletään websuunnitteluksi. Se hallinnoi sovelluksen käsittelemän tiedon ulkoasua ja tarjoaa käyttöliittymän tiedon keräämiseksi käyttäjiltä. Verkkosovelluksen tapauksessa yksinomaan näkymästä löytyviä teknologioita ovat mm. HTML, CSS ja Javascript. (Lecky-Thompson ym. 2009, 307.)

3.5.3 Ohjain

Ohjain yhdistää keskenään näkymän käyttöliittymän ja mallin prosessoinnin. Sen vastuualueeseen kuuluu käyttäjän syötteen kerääminen näkymältä ja asiaankuuluvien mallien hyödyntäminen. Ohjain kutsuu mallien toiminnallisuuksia ja tulkitsee palautetun tiedon, jotta se voidaan asettaa käyttäjän nähtäväksi näkymässä. (Lecky-Thompson ym. 2009, 308.)

3.6 MVC-kehysjärjestelmän tyypillinen työnjako

MVC-kehysjärjestelmän käyttöönotolle on olemassa useita syitä, koodin selkeydestä ylläpidon helppouteen. On kuitenkin vain yksi merkittävä syy siihen, miksi organisaatiot usein valitsevat MVC-kehysjärjestelmän. Tämä syy on järjestelmän mahdollistama orgaaninen työnjako. (McArthur 2008, 201.)

MVC sallii tehokkaan yhteistyön jakamalla työnteon kolmeen vastuualueeseen: sovellusohjelmointiin, web-suunnitteluun ja integrointiin. Tällöin luovat ja tekniset työntekijät voivat tulla keskenään helpommin toimeen. (McArthur 2008, 201).

Seuraavassa perehdytään eri vastuualueiden toimenkuvaan.

3.6.1 Sovellusohjelmoijat

Sovellusohjelmoijat työskentelevät mallien kanssa. Heidän osaamisensa liittyy useimmiten mm. PHP:hen, tietokantoihin, algoritmeihin ja sovellusarkkitehtuuriin. Sovellusohjelmoijat ovat usein vastuussa sovelluksen toiminnallisuuden suunnittelusta ja tuottavat rajapintoja, joiden avulla on mahdollista päästä käsiksi mallien tiedon prosessointiin. (McArthur 2008, 202.)

3.6.2 Websuunnittelijat

Websuunnittelijat käsittelevät näkymiä ja ovat vastuussa sovelluksen ulkoasusta ja käyttöliittymästä. Heidän osaamisalueensa käsittää HTML:n, CSS:n, Javascriptin ja graafisen suunnittelun. Useimmiten websuunnittelijat ovat yhteydessä sekä organisaation sisäisiin, että ulkoisiin vastuuhenkilöihin suunnitellessaan ja esitellessään sovelluksen käyttöliittymää. (McArthur 2008, 202.)

3.6.3 Integrointi

Integrointi tapahtuu sovelluksen ohjainkerroksessa. Se yhdistää sovellusohjelmoijien ja websuunnittelijoiden aikaansaannokset keskenään. Ohjainten kehittäjät ovat tavanomaisesti kokemattomampia, kuin sovellusohjelmoijat. Heidän työnkuvaansa kuuluu valmiiden aihioiden työstäminen ja niiden osien muuttaminen dynaamiseksi. He ovat myös vastuussa tiedon siirtämisestä käyttöliittymän lomakkeista mallien prosessoitavaksi ja prosessoinnin tuloksen palauttamisesta käyttöliittymälle. (McArthur 2008, 202.)

4 ZEND FRAMEWORK

4.1 Käsiteltävä aihealue

Seuraavassa perehdytään tarkemmin yhteen PHP:n kehysjärjestelmään: Zend Frameworkiin. Tarkastelun kohteena ovat ensin järjestelmän pääpiirteet, sen suunnittelufilosofia ja se, miten järjestelmä täyttää aikaisemmin mainitut kehysjärjestelmän valintakriteerit. Sen jälkeen tutustutaan joihinkin yksittäisiin komponentteihin tarkemmin.

4.2 Zend Framework-kehysjärjestelmän pääpiirteet

Vuonna 2005 PHP:n erikoistuva yritys nimeltään Zend Technologies aloitti yhteistyöprojektinsa PHP:n käytön edistämiseksi. Projekti käsittää kolme osa-aluetta: Eclipse liitännäisen PDT:n, Zend Frameworkin ja Zend Developer Zone websivuston. (Allen ym. 2008, 9.)

Zend Framework on avoimen lähdekoodin projekti, joka tarjoaa kehysjärjestelmän PHP:lle. Sen tavoite on luoda standardeja ohjelmointikäytäntöjä käsittävä järjestelmä, johon tulevaisuuden PHP-sovellukset voivat perustua. (Zend Technologies 2010c.)

Kehysjärjestelmä koostuu joukosta komponentteja, jotka on ryhmitelty muutaman ylemmän tason moduulin alle. Se tarjoaa kaiken tarvittavan yrityskäyttöön tarkoitettujen verkkosovellusten kehittämiseksi. Järjestelmä on hyvin joustava ja se sallii sovelluskehittäjän käyttöön ottaa vain kulloinkin tarvitsemansa komponentit. (Allen ym. 2008, 10.)

4.3 Zend Frameworkin tavoitteet

Zend Technologies on julkistanut ne tavoitteet, jotka yhdessä muodostavat Zend Frameworkin suunnittelufilosofian. (Allen ym. 2008, 14.)

4.3.1 Korkealaatuiset komponentit

Kaikki Zend Frameworkiin integroitu koodi on erittäin korkeatasoista. Tällä tarkoitetaan sitä, että koodi on kirjoitettu käyttäen PHP 5:n uusimpia ominaisuuksia ja että se ei aiheuta ainuttakaan ilmoitusta PHP parserilta. Toisin sanoen se on E_STRICT yhteensopivaa. (Zend Technologies 2010c.)

Tästä on hyötyä sovellusohjelmoinnissa, sillä tällöin ohjelmoija voi olla varma siitä, että kaikki parserin viestit johtuvat sovelluksen omasta koodista, eivätkä kehysjärjestelmästä.

Zend Technologies sisällyttää järjestelmän dokumentaation korkeatasoisuuden vaatimukseen. Komponentin ohjeistus on yhtä tärkeää kuin sen muodostava koodi. (Allen ym. 2008, 15.) Tämä on merkittävä asia järjestelmää valittaessa ja on eräs Zend Frameworkin vahvuuksista.

Edelleen korkeatasoisuuden tavoitteeseen kuuluu se, että koko sovellus on mahdollista toteuttaa käyttäen vain Zend Frameworkin komponentteja. Kolmannen osapuolen ohjelmakirjastoja ei tarvita, vaikka niiden käyttö on mahdollista niin mikäli niin halutaan. (Zend Technologies 2010c.)

Tämä takaa kehysjärjestelmällä tuotetun sovelluksen eheyden. Komponenttien käyttö ja nimeäminen sekä niiden toiminnallisuus ja hakemistorakenne ovat samanlaisia sovelluksesta riippumatta.

Tästä huolimatta Zend Framework on modulaarinen ja sen komponentit sisältävät hyvin vähän keskinäisiä riippuvuussuhteita (Allen ym. 2008, 15).

4.3.2 Yksinkertaisuus

Toinen kehysjärjestelmän suunnittelun tavoite kiteytyy mantraan ”Älä muuta PHP:ta” (Allen ym. 2008, 15). PHP on perimmiltään yksinkertainen ja käytännönläheinen. Zend Frameworkin tavoite on säilyttää tämä yksinkertaisuus tarjoamalla sovelluskehittäjille helppoja ratkaisuja haastaviin ongelmiin.

4.3.3 Suojattu tekijänoikeus

Kaikki Zend Frameworkin kehittäjät ovat allekirjoittaneet lisensointisopimuksen, joka määrittelee työn omistusoikeuden. Sopimuksessa allekirjoittaja takaa sen, että hänellä on parhaan tietonsa mukaan oikeus lahjoittaa työnsä tulos Zend Technologiesin käyttöön ja toisaalta sen, ettei kenenkään kolmannen osapuolen oikeuksia loukata. (Zend Technologies 2010c.) Tämän sopimuksen tavoite on suojata kehysjärjestelmän käyttäjiä mahdollisilta tekijänoikeusristiriidoilta.

4.3.4 Tuki

Eräs itsestään selvä, mutta siitä huolimatta huomionarvoinen asia on se, että Zend Framework on Zend Technologiesin tukema projekti. Tämä tarkoittaa sitä, että on hyvin epätodennäköistä, että kehysjärjestelmän kehitys loppuisi ydinkehittäjien joutilaisuuden vuoksi, tai että kehysjärjestelmä ei tukisi uusimpia PHP:n versioita. (Allen ym. 2008, 15.)

Zend Technologiesilla on riittävästi resursseja takaamaan se, että sekä järjestelmän koodi että sen dokumentaatio ovat aina ajan tasalla. (Allen ym. 2008, 15.)

Zend Technologies tarjoaa kehysjärjestelmän käyttäjille valmennuskursseja, konsultointia, sekä tukipalveluita. Tämän lisäksi saatavilla on yksityiskohtainen dokumentointi jokaiselle järjestelmän komponentille ja laaja kehittäjäyhteisö.

4.4 MVC-suunnittelumalli Zendin mukaan

Zend Frameworkin ohjainjärjestelmä on MVC-suunnittelumallin implementaatio (Lyman 2009, 1).

Kuten luvussa 3.5 todettiin, MVC jakaa sovelluksen kolmeen loogiseen kerrokseen: malliin, näkymään ja ohjaimeen. Seuraavassa perehdytään siihen, miten Zend Framework toteuttaa tämän suunnittelumallin.

4.4.1 Ohjain

Zend_Controller moduulin luokat toteuttavat ohjain-osan Zend Frameworkin MCV-implementaatiosta. Ne esittelevät etuohjain-suunnittelumallin, jossa sovelluksen vastaanottamat HTTP-pyynnöt jaetaan sivuohjaimille yhdestä etuohjain-luokasta käsin. (Lyman 2009, 2.)

Etuohjain tarjoaa mahdollisuuden käyttää liitännäisiä tämän prosessin jokaisessa vaiheessa, sekä antaa ohjelmoijan käyttöön joukon tapahtumia, jotka ajetaan aina prosessin saavuttaessa tietyn vaiheen. (Allen ym. 2008, 11.) Tämä sallii kehysjärjestelmän toiminnallisuuden helpon mukautettavuuden, mahdollisimman vähäisellä työmäärällä.

4.4.2 Näkymä

Zend Frameworkin näkymä-toteutus on nimeltään Zend_View. Se on PHP-aihio-järjestelmä, mikä tarkoittaa sitä, että näkymät ovat käytännössä perinteisiä PHP-tiedostoja, joissa koodi on upotettu HTML:n sekaan (Lyman 2009, 3). Tämän lisäksi Zend_View tarjoaa liitännäisjärjestelmän uudelleenkäytettävien elementtien luomista varten. (Allen ym. 2008, 11.)

Zend_View on suunniteltu niin, että sen tietyt ominaisuudet on mahdollista korvata kolmannen osapuolen koodilla. Se on myös mahdollista korvata kokonaan toisella aihiojärjestelmällä, kuten Smartylla. (Allen ym. 2008, 11.)

4.4.3 Malli

Zend_Db_Table on perusta, jolle Zend Frameworkin mallit voidaan rakentaa. Se toteuttaa table-data-gateway-suunnittelumallin, joka tarjoaa oliopohjaisen käyttöliittymän tietokantataulun tietoihin. (Lyman 2009, 2.)

Zend_Db_Tablea tukee Zend_Db tietokanta-abstraktiokerros, joka takaa tietokannasta riippumattoman pääsyn useimpiin saatavilla oleviin tietokannanhallintajärjestelmiin. Näihin kuuluvat mm. MySQL, Postgres, SQL Server, Oracle ja SQLite. (Allen ym. 2008, 11.)

Varsinaista Zend_Model luokkaa ei ole olemassa. Tämä johtuu siitä, että toisin kuin näkymä- ja ohjainkomponentit, mallikomponenttien toteutus vaihtelee huomattavasti sovelluksesta toiseen. Zend Frameworkin kehittäjäyhteisö ei määrittele mallitoteutusta, sillä sen aiheuttamat rajoitukset sovellusohjelmoinnille ovat painoarvoltaan suurempia, kuin sellaisesta mahdollisesti saatava lisäarvo. (Zend Technologies, 2010a.)

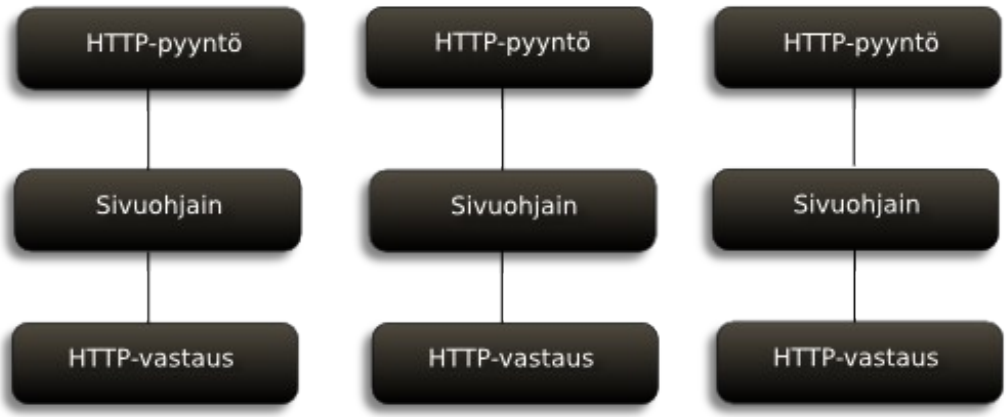
Yksinkertaisimmillaan Zend Frameworkin malli voi olla Zend_Db_Tablesta periytetty luokka. Tässä luokassa määritellään sen tietokantataulun nimi, jota halutaan käyttää, sekä taulun pääavain. Tällainen mallitoteutus pakottaa siirtämään osan sovelluksen datan käsittelystä ohjainkerrokseen, mikä voi aiheuttaa tarpeetonta toistoa. Kyseessä on tällöin niin sanottu fat-controller-suunnittelumalli. (Lyman. 2009, 1). Eräänlainen mallitoteutus apuluokkineen löytyy luvusta 6.

4.5 Etuohjain-suunnittelumalli

Verkkosovellusten HTTP-pyynnön käsittely voidaan jakaa karkeasti kahteen eri toimintamalliin: etuohjaimeen ja sivuohjaimeen (Allen ym. 2008, 24).

Sivuohjain käyttää erillisiä tiedostoja jokaiselle sovelluksen sivulle. Tämä on perinteinen tapa kehittää PHP-sovelluksia. Selain lähettää pyynnön

palvelimelle, joka vastaa lähettämällä takaisin pyydetyn sivun, prosessoituaan ensin sen sisältämän PHP-koodin. Tällöin sovelluksen toiminnallisuuden hallinta on jaettu useisiin tiedostoihin ja saman koodin toistaminen useissa tiedostoissa on lähes välttämätöntä. (Allen ym. 2008, 24). Kuva 1 havainnollistaa sivuohjaimen toimintaa.



Kuva 1: Sivuohjaimen toiminta

Zend Frameworkin käyttämä etuohjain-suunnittelumalli keskittää kaikki HTTP-pyynnöt yhteen tiedostoon. Tavanomaisesti tämä tiedosto on index.php. Se sijaitsee verkkosovelluksen julkisessa juurihakemistossa. (Zend Technologies, 2010d.)

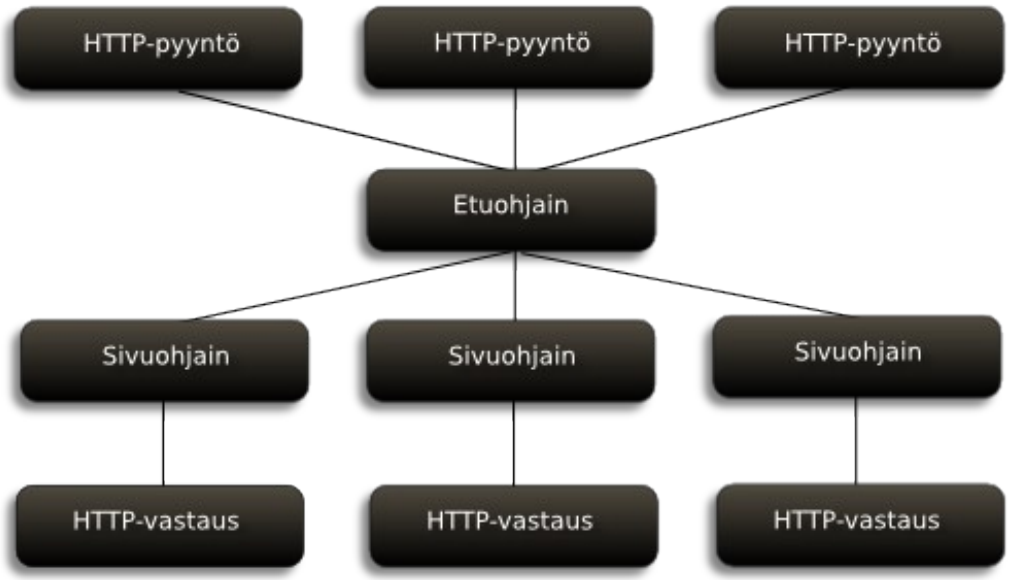
Etuohjain valitsee HTTP-pyynnön parametrien perusteella sovellus-, eli sivuohjaimen, joka sisältää pyydetyn sivun käsittelyä ohjaavan logiikan. Sivuohjaimesta kutsutaan action-parametrin määrittelemä metodi.

Etuohjaimen käytöllä on useita etuja sivuohjaimeen nähden. Ensinnäkin koodin toisto vähenee ja toisaalta sovelluksen käsittämät URL-polut on helpompi erottaa varsinaisen sisällön luovasta koodista.

Näytettävä sivu valitaan kolmen HTTP-pyynnön sisältämän GET-parametrin perusteella. Nämä parametrit ovat module, controller ja action. Module-

parametri on mahdollista jättää pois, jolloin järjestelmä käyttää oletusmoduulia. Tällöin pyyntö /index.php?controller=product&action=details ajaa oletusmoduulin product-ohjaimen details-metodin. (Allen ym. 2008, 24).

Tosiasiassa tämänkaltaisia URL-osoitteita näkee harvoin. Sen sijaan Zend Framework uudelleenkirjoittaa osoitteet muotoon. /module/controller/action. Tällöin edellämainittu osoite on muotoa /product/details. (Zend Technologies, 2010d.) Kuva 2 havainnollistaa etuohjaimen toimintaa.



Kuva 2: Etuohjaimen toiminta

4.6 Zend_Controller_Front

Zend Frameworkin etuohjaimen koodi on jaettu moneen yhteistyössä toimivaan luokkaan. Nämä luokat toteuttavat joustavan ratkaisun HTTP-pyynnön reitittämiseksi oikeaan paikkaan. (Allen ym. 2008, 25). Seuraavassa perehdytään näihin luokkiin tarkemmin.

4.6.1 HTTP-pyyntö

HTTP-pyyntö kapseloidaan Zend_Controller_Request_Http-luokan instanssiin. Se sisältää kaiken sovelluksen ulkopuolisen datan. Toisin sanoen se tallentaa PHP:n superglobaalien muuttujien, $_GET, $_POST, $_COOKIE, $_SERVER ja $_ENV sisällön sekä sovelluksen juuripolun. Reititin lisää tähän tietoon moduulin, sivuohjaimen ja action:n nimen ne selvitettyään. (Zend Technologies, 2010d.)

Zend_Controller_Request_Http sisältää funktion getParams, jonka avulla sovelluksen on mahdollista päästä käsiksi pyynnön parametreihin. (Allen ym. 2008, 26).

4.6.2 Reititys

Reititys selvittää mitä ohjainta ja action-metodia on kutsuttava pyynnön parametrien perusteella. Tämän toteuttavat luokat, jotka implementoivat Zend_Controller_Router_Interface–rajapinnan.
Zend_Controller_Router_Rewrite tarjoaa konkreettisen rajanpinnan implementaation ja on Zend Frameworkin oletusreititin. (Zend Technologies, 2010d.)

Zend_Controller_Router_Rewrite pilkkoo pyynnön URL-osoitteen erillisiksi parametreiksi. Mikäli osoite on muotoa /index.php?controller=product&action=details, ottaa reititin parametrit suoraan $_GET taulukosta. Kuten luvussa 4.5 todettiin, tällaiset osoitteet ovat harvinaisia. Useimmiten osoite tulee muodossa /product/details, jolloin reititin käyttää $_SERVER[”REQUEST_URI”] superglobaalia muuttujaa parametrien selvittämiseen. (Allen ym. 2008, 26).

4.6.3 Jakaminen

Jakaminen (dispatching) on se prosessi, joka varsinaisesti toteuttaa oikean ohjaimen oikean action-metodin kutsumisen. Zend_Controller_Dispatcher_Standard tarjoaa toiminnallisuuden lähes kaikkiin käyttötapauksiin ja on Zend Frameworkin oletusluokka kyseiseen tarkoitukseen. (Lyman 2009, 2.)

Jakajan suorittamaan työhön kuuluvat ohjainluokan ja action-metodin nimien formatointi ja metodin kutsuminen. Se panee käytäntöön Zend Frameworkin luokkien nimeämistä koskevat rajoitukset, kuten sen että ohjainluokan nimi on koostuttava vain kirjaimista ja että nimi on kirjoitettava isolla alkukirjaimella. (Allen ym. 2008, 26).

Jakajan dispatch-metodi lataa tarvittavan ohjainluokan, instantioi sen ja sen jälkeen kutsuu pyynnön parametreissa määritellyn action-metodin. (Allen ym. 2008, 26).

4.6.4 Sivuohjain

Zend_Controller_Action on abstrakti luokka, josta kaikki sivuohjaimet periytetään. Jakaja varmistaa sen, että kaikki ohjaimet periytyvät tästä luokasta, voidakseen olla varma siitä, että tietyt metodit ovat saatavilla. Sivuohjaimet nimetään lisäämällä ”Controller”-sana halutun sivunimen perään. (Zend Technologies, 2010d.)

Sivuohjain sisältää pyynnön, eli Zend_Controller_Request_Http:n instanssin parametrien lukemista varten ja vastauksen, eli Zend_Controller_Response_Http:n instanssin vastaukseen kirjoittamista varten. Luokan muu toiminnallisuus keskittyy siihen, että sivuohjainten kirjoittaminen ja sen muutosten hallinta on sovellusohjelmoijalle mahdollisimman helppoa. Se

tarjoaa metodit parametrien lukemiseen ja asettamiseen sekä pyynnön uudelleenohjaukseen. (Allen ym. 2008, 26).

Sivuohjaimen sisältämät action-metodit nimetään lisäämällä haluttuun sivun nimeen sana ”Action”. Tällöin esim. productController-sivuohjain voisi sisältää detailsAction- ja listingAction-nimisiä funktioita. Näitä funktioita kutsutaan pyynnön sisältäessä niihin ohjaavat parametrit. (Allen ym. 2008, 26-27).

Zend_Controller_Action sisältää kaksi eri tasoa joilla on mahdollista suorittaa toiminnallisuutta valitusta action-metodista huolimatta: init- ja pre- sekä post-dispatch metodit (Zend Technologies, 2010e.)

Init-metodi korvaa sivuohjaimen rakentajametodin. Se eroaa rakentajasta siinä, että se ei vaadi parametreja, eikä isäntäluokan rakentajaa ole pakollista kutsua. (Zend Technologies, 2010e.)

PreDispatch kutsutaan aina ennen action-metodia ja postDispatch sen jälkeen. PreDispatch-metodia voidaan käyttää esim. autentikoinnin tarkistamiseen, tai kutsutun action-metodin ohittamiseen tietyissä olosuhteissa. PostDispatch on hyödyllinen esim. sivuaihion muutoksiin. (Zend Technologies, 2010e.)

4.6.5 Vastaus

Etuohjainketjun viimeinen linkki on HTTP-vastaus. Verkkosovelluksessa tämä vastuualue kuuluu Zend_Controller_Reponse_Http-luokalle. Tämän luokan instanssi on käytännössä paikka, johon kaikki ohjainprosessin tulokset tallennetaan. Olion tilaa on mahdollista muuttaa koko prosessin aikana. Tästä on hyötyä erityisesti etuohjaimen liitännäisten toiminnallisuudessa. (Allen ym. 2008, 27.)

Zend_Controller_Response_Http sisältää kolmenlaista tietoa: selaimelle takaisin lähetettävien HTTP-otsikoiden tiedot ja HTTP-vastauskoodin, joka oletuksena on 200, vastauksen sisältämän datan ja sovelluksen aiheuttamat

poikkeukset. Etuohjain hallinnoi poikkeuksia ja varmistaa, että tuotantokäytössä niiden sisältämää dataa ei lähetetä selaimelle. (Zend Technologies, 2010d.)

4.6.6 Etuohjaimen liitännäiset

Etuohjain sisältää liitännäisjärjestelmän, jonka avulla on mahdollista suorittaa omaa koodia tietyissä etuohjainprosessin vaiheissa. Liitännäiset periytetään Zend_Controller_Plugin_Abstract-luokasta, joka tarjoaa kuusi tyhjää metodia, jotka ohjelmoija voi ylikirjoittaa ja joita etuohjain kutsuu suorituksensa aikana. Näihin kuuluvat

1. routeStartup(), joka kutsutaan ennen reitittimen suoritusta
2. dispatchLoopStartup(), joka kutsutaan ennen jakajan suoritusta
3. preDispatch(), joka kutsutaan ennen action-funktion suoritusta
4. postDispatch(), joka kustutaan action-funktion suorituksen jälkeen
5. dispatchLoopShutdown(), joka kutsutaan kun jakaja on valmis
6. routeShutdown(), joka kutsutaan kun reititin on valmis. (Allen ym. 2008, 28.)

Zend Framework käyttää hyväkseen tätä toiminnallisuutta ErrorHandler-liitännäisessä, joka ohjaa selaimen virhesivulle sillon, kun reitityksessä tapahtuu virhe (Zend Technologies, 2010f).

4.7 Zend_View

Zend_View-moduuli erottaa sovelluksen näkymäkomponentin muusta sovelluksesta. Kuten luvussa 4.4.2 todettiin, kyseessä on PHP-aihio-järjestelmä. Koodi Zend_View'n aihioissa on natiivia PHP:ta eikä pseudokieltä, kuten on laita esim. Smartyn tapauksessa. (Lyman 2009, 3.)

4.7.1 Datan siirto näkymään

Jotta näkymän olisi mahdollista esittää mallien prosessoinnin tuloksena syntyvää dataa, on ohjaimen siirrettävä data näkymään. Tämä tapahtuu komennolla:

```
$view->assign('tervehdys', 'Hei Maailma!');
```

Komento asettaa näkymään uuden tervehdys-nimisen muuttujan ja antaa sen arvoksi merkkijonon ”Hei Maailma!”. Komento voidaan antaa myös muodossa:

```
$view->tervehdys = ”Hei Maailma!”;
```

Tällöin PHP kutsuu maagista metodia __set, joka puolestaan kutsuu assign-metodin (Zend Technologies, 2010g).

4.7.2 Näkymän aihio

Näkymän aihio on tavanomainen PHP-tiedosto, jonka näkyvyysalue on sisällytetty Zend_View-luokan instanssiin. Tämä tarkoittaa sitä, että sillä on pääsy kaikkiin Zend_View:n metodeihin ja muuttujiin aivan kuten luokan sisäisellä metodilla. (Allen ym. 2008, 28.)

Näkymään lisätty data tallennetaan Zend_View-instanssin julkisiksi jäsenmuuttujiksi, jolloin aihio pääsee niihin käsiksi suoraan. Näkymä tarjoaa myös liitännäisjärjestelmän, joka helpottaa aihioiden kirjoittamista (Zend Technologies, 2010g). Seuraavassa on esimerkki Zend_View-aihiosta.

```
<h1>Esimerkkiaihio</h1>

<p><?php echo $this->tervehdys;?></p>
```

4.7.3 Näkymän liitännäiset

Zend Framework sisältää useita valmiita näkymän liitännäisiä ja tarjoaa käytännön omien liitännäisten kirjoittamiseksi. Yleisimmin käytetty näkymän

liitännäinen on escape-funktio, joka on sisäänrakennettu Zend_View'iin. Muut löytyvät omasta hakemistostaan, josta Zend_View lataa ne käyttöön automaattisesti. (Allen ym. 2008, 29).

Sekä liitännäisluokka että sen toiminnallisuuden toteuttava metodi on nimettävä sillä nimellä, jolla liitännäistä halutaan kutsua. Zend_View toteuttaa maagisen metodin __call, joka etsii ja lataa luokan tämän nimen ja sille määritellyn nimiavaruuden perusteella. Jotta liitännäinen olisi mahdollista löytää on se rekisteröitävä näkymään komennolla:

```
$view->setHelperPath('/polku/liitännäiseen','liitännäisen nimiavaruus');
```

Näkymän liitännäiset tarjoavat helpon tavan erotella usein käytetty koodi näkymän aihioista. Tämä selventää aihion koodia ja helpottaa sen ylläpidettävyyttä. (Zend Technologies, 2010g.)

4.8 Zend_Db

Zend_Db on Zend Frameworkin tietokanta-asbtraktiokirjasto. Se tarjoaa toiminnallisuuden sovelluksen erottamiseen sen käyttämästä tietokannanhallintajärjestelmästä. Tämä on hyödyllistä silloin kun sovellus on tarpeen siirtää tietokannasta toiseen (Allen ym. 2008, 31).

Zend_Db toteuttaa tehdas-suunnittelumallin. Tarvittava tietokantaluokka instantioidaan tehtaalle annettujen parametrien perusteella. Esim. MySQL-tietokanta otetaan käyttöön seuraavasti:

```
$params = array ('host' => 'localhost',
     'username' => 'käyttäjänimi',
     'password' => 'salasana',
     'dbname'=> 'tietokannannimi');
$db = Zend_Db::factory('PDO_MYSQL', $params);
```

Zend_Db käyttää hyväkseen PHP:n PDO-laajennusta, joka tukee useita tietokannanhallintajärjestelmiä. Oracle- ja DB2-tuki toteutetaan käyttäen järjestelmien omia ajureita. (Zend Technologies, 2010h.)

Zend_Db lisää PDO:n abstraktiokerroksen päälle profilointitoiminnallisuuden ja standardit tietokannan käsittelykäskyt rivien hakemiseksi, lisäämiseksi, päivittämiseksi ja poistamiseksi. (Allen ym. 2008, 31).

4.9 Zend_Db_Table

Tietokanta-abstraktion lisäksi Zend Framework tarjoaa Zend_Db_Table moduulin, joka toteuttaa table-data-gateway-suunnittelumallin. Se sallii oliopohjaisen pääsyn tietokantataulun tietoihin. (Allen ym. 2008, 32).

Tauluja edustavat luokat periytetään Zend_Db_Table_Abstract –luokasta. Esimerkiksi tuotetietoja sisältävän product-taulun luokka luodaan seuraavasti:

```
Class Product extends Zend_Db_Table_Abstract  {

     protected $_name = 'product';

}
```

$_name-jäsenmuuttuja määrittelee käytettävän tietokantataulun nimen. Se voidaan jättää määrittelemättä, jolloin Zend_Db_Table käyttää luokan nimeä. (Zend Technologies, 2010i.).

Oletusarvoisesti Zend_Db_Table käyttää taulun pääavaimena ”id”-saraketta. Mikäli tätä halutaan muuttaa voidaan pääavain määritellä $_primary-jäsenmuuttujan avulla seuraavasti:

```
class Product extends Zend_Db_Table_Abstract {

     protected $_name = 'product';

     protected $_primary = 'product_id';

}
```

Tällöin Product-luokka käyttää taulunaan product-taulua ja pääavaimenaan product_id:tä. (Allen ym. 2008, 32).

Zend_Db_Table tarjoaa joukon funktioita tiedon keräämiseen taulusta. Näihin kuuluvat find, fetchRow ja fetchAll. (Zend Technologies, 2010i.)

Find hakee rivin tietokannasta pääavaimen perusteella. Se palauttaa Zend_Db_Table_RowSet-olion, joka on kokoelma Zend_Db_Table_Row-olioita. Zend_Db_Table_Row on yhtä tietokannan riviä edustava olio. (Allen ym. 2008, 32.)

Fetchrow hakee rivin sille parametrina annetun WHERE-lauseen perusteella. Se paluttaa Zend_Db_Table_Row-olion. FetchAll hakee joukon rivejä ja palauttaa Zend_Db_Table_RowSet-olion. (Allen ym. 2008, 32).

Zend_Db_Table sisältää myös funktiot tietokantarivien lisäämiseen, päivittämiseen ja poistoon (Zend Technologies, 2010i).

4.10 Zend_Validate ja Zend_Filter

Zend_Validate-komponentti tarjoaa joukon usein tarvittavia validaattoreita. Näihin kuuluvat mm. sähköpostiosoitteiden ja luottokorttinumeroiden validointi, sekä tietokannan rivien vertaaminen syötteen sisältöön. Tämän lisäksi se sisältää validaattoreiden ketjutusmekanismin, jonka avulla useita validaattoreita voidaan sitoa yhteen lähteeseen, käyttäjän määrittelemässä järjestyksessä. (Zend Technologies, 2010b.)

Validaattori tutkii syötteen, asettaa sille reunaehtoja ja palauttaa boolean arvon, joka kertoo, täyttääkö syöte sille määritellyt ehdot. Mikäli syöte ei täytä ehtoja validaattori voi lisäksi palauttaa tiedon siitä, mitkä ehdot jäivät täyttymättä. (Zend Technologies, 2010b.)

Zend_Filter-komponentti tarjoaa joukon datan suodattamiseen tarkoitettuja luokkia. Ne muokkaavat syötettä poistamalla sen osia, tai lisäämällä siihen tilanvaihtosekvenssejä. (Zend Technologies, 2010b.) Verkkosovelluksessa suodatusta voidaan käyttää mm. HTML-entiteettien poistamiseen käyttäjän syötteestä.

4.11 Zend_Form

Zend_Form yksinkertaistaa verkkosovelluksen lomakkeiden luomista ja käsittelyä. Se sisältää seuraavat toiminnallisuudet:

- Syötteen suodatus ja validointi
- Lomake-elementtien järjestys
- Lomakkeiden tulostus sivulle
- Elementtien ryhmittely
- Elementti- ja lomaketason konfigurointi. (Zend Technologies, 2010d.)

Zend_Form käyttää useita Zend Frameworkin komponentteja näiden toimintojen toteuttamiseen. Näihin kuuluvat mm. Zend_Validate, Zend_Filter ja Zend_View. (Zend Technologies, 2010b.)

5 LASTENKIRJAKAUPPA.FI

5.1 Saarni-Kirjat

Tämän tutkimuksen soveltavana osuutena toteutettiin verkkokauppasovellus Saarni-Kirjat-kustantannusosakeyhtiölle. Kustannusosakeyhtiö Saarni-Kirjat on lastenkirjoihin sekä lasten kuvakirjoihin erikoistunut suomalainen kirjakustantamo, jonka kustantamia lastenkirjoja löytyy kirjakaupoista ja kirjakahviloista kautta maan.

Saarni-Kirjoja on saatavilla myös Junibackenin lastenkirjakaupasta Tukholmasta.

Kustannusosakeyhtiö Saarni-Kirjat toimii Turussa, Suomen kirjallisen kulttuurin, kirjapainotaidon ja kustannustoiminnan vanhalla ja perinteikkäällä syntymäseudulla.

5.2 Järjestelmän tarve

Saarni-kirjat on aikaisemmin keskittynyt lähinnä kustannustoimintaan ja kirjojen myymiseen yksityisille organisaatioille sekä julkisen sektorin toimijoille. Jonkinlaiset verkkosivut ovat jo olemassa ja niitä on menestyksekkäästi käytetty kustantamon yhteystietojen esilletuomiseen.

Olemassa olevat sivut eivät kuitenkaan täytä kustantamon muuttuneiden tarpeiden vaatimuksia riittävän kattavasti.

Saarni-kirjat on laajentamassa toimintaansa, keskittyen kustannustoiminnan lisäksi myös jälleenmyyntiin. Tällöin toimiva ja ulkoasultaan miellyttävä verkkokauppa on oleellinen asia.

Tämän tutkimuksen soveltavan osuuden tavoite on toteuttaa muuttuneen toimintamallin vaatimukset täyttävä verkkokauppasivusto. Verkkokauppa tulee saamaan nimekseen Suomen Lastenkirjakauppa, ja se tullaan julkaisemaan osoitteeseen www.lastenkirjakauppa.fi

5.3 Järjestelmän vaatimukset

Lastenkirjakaupan toteutukselle asetettiin alusta lähtien kaksi huomionarvoista haastetta. Ensinnäkin kaiken sivuston sisällön on oltava saatavilla sekä suomeksi että ruotsiksi ja toisaalta sivujen on tarjottava menetelmä tuoteryhmien luomiseen ja poistoon.

Tuoteryhmien luomisella tarkoitetaan tässä sitä, että yksittäinen tuote on oltava liitettävissä tuoteryhmään, jolloin sen ominaisuudet määräytyvät tuoteryhmän mukaan. Tuotteella ei tule olla muita ominaisuuksia kuin nimenomaan sille kuuluvat ominaisuudet. Mikäli kaupassa on myynnissä lippuja tai leluja, ei niiden ominaisuuksiin tule kuulua esim. ISBN-tunnusta, joka on yksinomaan kirjojen ominaisuus. Tämä harmillinen puute löytyy monista tuotantokäytössä olevista alun perin kirjakaupoiksi tarkoitetuista verkkokaupoista ja sitä haluttiin tietoisesti välttää.

5.4 Toteutusmenetelmät

Lastenkirjakauppa päätettiin toteuttaa ketterän sovelluskehityksen periaatteita noudattaen. Vaikka kehittäjäryhmän puutteesta johtuen mitään varsinaista metodia ei voitu ottaa käyttöön, eikä sellaisen käyttöönotosta olisi yhden työntekijän ollessa kyseessä ollut mitään hyötyä, seurattiin kehitystyössä silti eräitä ketteristä metodeista löytyviä suosituksia.

5.4.1 Projektin rajaus

Projektin rajat määriteltiin kattavan vaatimusmäärittelyn ja toiminnallisuutta kuvaavien dokumenttien sijaan konseptin avulla. Konsepti kuvaa karkeasti sovelluksesta löytyvän toiminnallisuuden ja sitä on mahdollista muuttaa kehitystyön aikana, mikäli tarvetta muutoksiin ilmenee. Tällaiset muutokset tehdään konseptin asettamissa raameissa, jolloin suuremmat muutokset vaativat projektin rajojen uudelleenmäärittelyn ja siten myös uuden konseptin.

Tämä takaa sen, ettei projekti missään vaiheessa riistäydy käsistä ja kasva odotettua suuremmaksi ja toisaalta sen, että asiakaan on mahdollista ohjata projektin etenemistä.

5.4.2 Yhteys asiakkaaseen

Asiakkaaseen pidettiin jatkuvasti yhteyttä projektin aikana. Keskeneräistä järjestelmää käytiin läpi asiakkaan kanssa moneen kertaan, jotta voitiin varmistaa lopullisen tuotteen vastaavan asiakkaan tarpeita. Projektin etenemistä ohjattiin asiakkaalta saadun palautteen perusteella.

5.4.3 Käyttäjäkertomukset

Lastenkirjakaupassa hyödynnettiin käyttäjäkertomuksia testauksen välineinä. Ne eivät olleet kehityksen aikana merkittävässä osassa muutoin kuin asiakkaan toteuttamissa käytettävyyskokeilussa.

Näitä varten kehitettiin joukko skenaarioita, joita Lastenkirjakaupan mahdolliset asiakkaat tulisivat seuraamaan, ja kokeiltiin miten helppoa niiden läpivieminen on. Näistä kokeiluista saatuja huomioita käytettiin hyödyksi kaupan kehittämisessä.

5.5 Konsepti

Lastenkirjakaupan konsepti rakennettiin asiakkaan lähettämän kertomuksen perusteella. Kertomus sisälsi tekstimuotoisen kuvauksen siitä, mitä kaupan tulee sisältää ja miten sen tulee toimia.

Kertomuksen pohjalta lastenkirjakaupan määriteltiin kattavan kolme laajempaa osiota: julkisen puolen, joka on kaupan asiakkaille näkyvä osa sivustoa, hallintapuolen, josta käsin on mahdollista ylläpitää kauppaa, ja blogin, joka toimii mainoskanavana Saarni-Kirjojen järjestämille tapahtumille.

Julkinen puoli käsittää kaupan etusivun, ostoskorin, tilauksen tekemiseen liittyvät sivut, asiakkaiden omat sivut, tuotelistauksen, sekä Lastenkirjakaupan toimintaan liittyvät sivut, kuten toimitusehdot ja yhteystiedot. Seuraavassa esitellään asiakkaan hyväksymän konseptin sisältö.

Etusivu sisältää seuraavat toiminnallisuudet:

- Tuotteiden ja tuoteryhmien listaus.
- Blogin syöte.
- Ostokorin sisällön tiivistelmä.
- Asiakkaiden sisäänkirjautuminen.
- Lastenkirjakaupan esittely.

Tuotelistaussivu sisältää seuraavat ominaisuudet:

- Tuotteiden ja tuoteryhmien listaus
- Tuotteiden haku
- Ostoskorin sisällön tiivistelmä

Ostoskorisivu sisältää seuraavat ominaisuudet:

- Ostoskorin sisällön tarkka listaus
- Mahdollisuus poistaa tuotteita
- Mahdollisuus tyhjentää kori
- Maksutavan valinta

Tilaussivu sisältää seuraavat ominaisuudet:

- Asiakkaan sisäänkirjautuminen
- Tallennettujen yhteystietojen haku
- Tilauksen lähettäminen

Asiakkaan oma sivu sisältää seuraavat ominaisuudet:

- Asiakkaan sisäänkirjautuminen
- Tallennettujen yhteystietojen muuttaminen
- Tilausten etenemisen seuranta

Hallintapuoli käsittää tilausten, tuotteiden ja asiakkaiden hallinnan sekä rajoitetun sisällönhallinnan.

Tilausten hallintaan kuuluvat

- Tilausten tarkastelu
- Tilausten haku ja suodatus
- Tilausten edistymistä kuvaavan tilan vaihtaminen.

Tuotehallintaan kuuluvat

- Tuotteiden lisäys ja poisto
- Tuotteiden tuoteryhmän valinta
- Tuoteryhmien hallinta
- Tuotteiden ominaisuuksien muuttaminen
- Tuotteiden haku ja suodatus
- Tuotteiden julkaiseminen ja piilottaminen

Asiakkaiden hallintaan kuuluvat

- Asiakkaiden tietojen muuttaminen
- Asiakkaiden listaaminen
- Asiakkaiden haku ja suodatus

Sisällönhallintaan kuuluvat

- Toimitusehtojen ja yhteystietojen muuttaminen
- Etusivun osien sisällön muuttaminen

Blogi käsittää merkintöjen hallinnan ja ylläpidon, mahdollisuuden hakea ja suodattaa blogimerkintöjä sekä blogimerkintöjen kommentoinnin. Sen julkisen puolen toiminnallisuuteen kuuluvat

- Merkintöjen selaus ja haku
- Merkintöjen kommentointi.

Sen ylläpitopuoleen kuuluvat

- Merkintöjen haku ja suodatus

- Merkintöjen kirjoittaminen
- Merkintöjen poisto
- Merkintöjen julkaiseminen ja piilottaminen

5.6 Toteutusalustan valinta

Lastenkirjakaupan toteutusalustaksi valittiin Zend Framework -kehysjärjestelmä. Kuten luvussa 4 todettiin Zend Framework sisältää MVC-toteutuksen lisäksi kattavan dokumentaation ja tuen, sekä laajan apuluokkakirjaston.

Zend Frameworkin huomattavimpiin etuihin Lastenkirjakaupan tapauksessa kuuluvat tuki monikieliselle toteutukselle Zend_Translaten avulla ja lomakkeiden käsittely Zend_Forminn ja sen hyväkseen käyttämien Zend_Validaten ja Zend_Filterin avulla.

Verkkokauppa julkaistaan Webbinen.fi -webhotelliin, jossa on käytössä Apache-WWW-palvelin ja PHP:n versio 5.2. Hotelli tarjoaa MySQL-tietokannan, joka otetaan käyttöön kaupan tuotantoonoton yhteydessä. Kehityksen aikana käytetään SQLite-tietokantaa, joka on helposti siirrettävissä.

6 LASTENKIRJAKAUPAN TOTEUTUS

6.1 Käsiteltävä aihealue

Seuraavassa perehdytään Lastenkirjakauppa-sivuston varsinaiseen toteutukseen Zend Framework -kehysjärjestelmällä, siltä osin kuin se on tutkimusongelman rajoissa mielekästä. Tämä tarkoittaa sitä, että tietokannan ja käyttöliittymän suunnittelua käsitellään vain pintapuolisesti.

6.2 Ulkoasun suunnittelu

Lastenkirjakaupan toteutus aloitettiin ulkoasun suunnittelulla. Ulkoasun avulla asiakkaan on mahdollista hahmottaa, miten konseptissa määritelty toiminnallisuus tullaan toteuttamaan. Kun ulkoasulle on saatu asiakkaan hyväksyntä, voidaan olla varmoja siitä, ettei kehitystyön aikana tulla tekemään turhaa työtä.

Ulkoasua suunniteltaessa ei vielä kirjoitettu varsinaista koodia. Sen sijaan sivuaihiot piirrettiin vektoripiirto-ohjelmalla ja valmiit kuvat lähetettiin asiakkaan kommentoitavaksi.

Kun asiakas oli kommentoinut lähetettyjä kuvia, tehtiin niihin tarvittavat muutokset ja käytiin ne uudelleen läpi. Kun asiakas oli ulkoasuun tyytyväinen, sivuston asettelu lyötiin lukkoon ja siirryttiin tietokannan suunnitteluun.

6.3 Tietokannan suunnittelu

Tietokannan suunnittelu on läheisessä yhteydessä sovelluksen mallikerroksen suunnitteluun. Tietokanta on se paikka, johon mallit tulevat tallentamaan tietonsa, joten tietokantatauluja suunniteltaessa on huomioitava malli-luokkien edellyttämät vaatimukset.

Tietokantaa suunniteltaessa määriteltiin sovelluksen tärkeimmät malli-oliot ja niiden suhteet toisiinsa. Todettiin, että Lastenkirjakauppa tulee sisältämään seuraavat mallit:

- Tuote
- Tuoteryhmä
- Ikäryhmä
- Tilaus
- Maksutapa
- Osoite
- Asiakas
- Käyttäjä
- Blogimerkintä
- Blogikommentti
- Tunniste

Näistä jokaiselle luotiin oma tietokantataulunsa. Taulujen väliset suhteet toteutettiin käyttäen vierasavaimia ja liitostauluja. Useita kieliversioita käsittäviä malleja varten luotiin asiaankuuluville tauluille lapsitaulut, jotka sisältävät mallin eri kieliversioiden tiedot.

Jokaisen mallin yksityiskohtainen läpikäynti ei tässä ole mielekästä. Sen sijaan seuraavassa perehdytään tarkemmin kahden yksittäisen mallin, tuoteryhmän ja tilauksen, tietokantatoteutukseen.

6.3.1 Tuoteryhmät

Lastenkirjakaupan eräänä haasteena oli tuoteryhmien muokattavuus. Tämä toteutettiin tietokantatasolla sijoittamalla tuotteiden ominaisuuksien tiedot tuotetauluun ja ominaisuuksien nimet tuoteryhmätauluun. Tietyn tuotteen ominaisuudet saadaan siten selville yhdistämällä tuote- ja tuoteryhmätaulut toisiinsa.

Tuoteryhmätaulu product_category luotiin seuraavalla SQLiten syntaksin mukaisella SQL-kyselyllä:

```
CREATE TABLE product_category (

     id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

     is_age_grouped INTEGER NOT NULL,

     is_public INTEGER NOT NULL,

     is_deleted INTEGER NOT NULL

);
```

Taulu sisältää tiedot siitä, jaetaanko kyseinen tuoteryhmä edelleen ikäryhmiin, onko se julkaistu, toisin sanoen näkyykö se kirjakaupan julkisella puolella ja siitä onko se poistettu. Lastenkirjakaupan tuoteryhmien hallinnassa tuoteryhmän poistaminen merkitsee kyseisen tuoteryhmän poistetuksi, muttei poista sitä tietokannasta. Tämän jälkeen tuoteryhmä on löydettävissä roskakori-linkin takaa, josta käsin se on mahdollista poistaa kokonaan. Sen kokonaan poistaminen poistaa myös kaikki siihen kuuluvat tuotteet, joten se ei ole ensimmäinen tarjottu vaihtoehto.

Tuoteryhmästä löytyvät ominaisuuksien nimet tallennetaan product_category_data-tauluun. Tämä johtuu siitä, että nimien on oltava saatavilla useilla kielillä. Taulu luotiin seuraavalla SQL-kyselyllä.

```
CREATE TABLE product_category_data (
```

```
    id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

    lang_code TEXT NOT NULL,

    category_id INTEGER NOT NULL,

    name TEXT NOT NULL,

    property1_key TEXT NULL,

    property2_key TEXT NULL,

    property3_key TEXT NULL,

    property4_key TEXT NULL,

    property5_key TEXT NULL,

    property6_key TEXT NULL,

    property7_key TEXT NULL,

    property8_key TEXT NULL,

    property9_key TEXT NULL,

    property10_key TEXT NULL,

    CONSTRAINT "product_category_data_has_parent" FOREIGN KEY ("category_id")
REFERENCES "product_category" ("id")

);
```

Vaikka tuoteryhmät ovat vapaasti muokattavissa päätettiin mahdollisten ominaisuuksien määrä rajoittaa kymmeneen. Periaatteessa ominaisuuksien määrä olisi mahdollista olla rajoittamatta luomalla ominaisuuksille oma taulunsa ja liittämällä se liitostaulun avulla tuoteryhmä-, tai tuotetauluun. Tällainen toiminnallisuus ei kuitenkaan ole järin välttämätöntä tai edes haluttua. WWW-sivua tulostettaessa on itse asiassa hyödyllistä tietää se, ettei ominaisuuksia voi olla kymmentä enempää. Tämä myös vähentää turhia kyselyitä, ominaisuuksien löytyessä suoraan tuoteryhmätaulusta.

Tuoteryhmän pari eli tuotetaulu luotiin seuraavalla SQL-kyselyllä:

```
CREATE TABLE product (
     id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
     availability_id INTEGER NOT NULL,
     category_id INTEGER NOT NULL,
     price REAL NOT NULL,
     times_added_to_cart INTEGER NOT NULL,
     times_ordered INTEGER NOT NULL,
     rank INTEGER NOT NULL,
     date_updated TEXT NOT NULL,
     date_created TEXT NOT NULL,
     is_public INTEGER NOT NULL,
     is_deleted INTEGER NOT NULL,
     small_image BLOB NULL,
     large_image BLOB NULL,
     CONSTRAINT "product_has_existing_category" FOREIGN KEY ("category_id")
REFERENCES "product_category" ("id")
);
```

Taulu sisältää mm. category_id-sarakkeen, joka on tuoteryhmätauluun viittaava vierasavain. Taulun tuoteryhmään liittyvät ominaisuudet ovat saatavissa product_data-taulusta useana kieliversiona. Product_data-taulu luotiin seuraavalla SQL-kyselyllä:

```
CREATE TABLE product_data (
     id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
     lang_code TEXT NOT NULL,
     product_id INTEGER NOT NULL,
```

```
    title TEXT NOT NULL,

    property1 TEXT NULL,

    property2 TEXT NULL,

    property3 TEXT NULL,

    property4 TEXT NULL,

    property5 TEXT NULL,

    property6 TEXT NULL,

    property7 TEXT NULL,

    property8 TEXT NULL,

    property9 TEXT NULL,

    property10 TEXT NULL,

    short_desc TEXT NULL,

    long_desc TEXT NULL,

    CONSTRAINT "product_data_has_parent" FOREIGN KEY ("product_id")
REFERENCES "product" ("id")

);
```

Taulu sisältää tuotenimike-sarakkeen sekä tuoteryhmätaulun ominaisuuksien nimiä vastaavat ominaisuuksien arvot. Tiedot ovat haettavissa taulusta tuotteen eli product-taulun id-sarakkeen ja halutun kieliversion koodin perusteella.

6.3.2 Tilaukset

Toinen esimerkki tietokantamallien toteutuksesta on tilauksen tallentaminen.

Tilauksen todettiin koostuvan sen tehneen asiakkaan tiedoista, siihen kuuluvien tuotteiden tiedoista, sen tilasta, maksutavasta ja lähetyspäivämäärästä. Tilaustaulu ”orders” luotiin seuraavalla SQL-kyselyllä:

```
CREATE TABLE orders (

     id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

     customer_id INTEGER NOT NULL,

     date_created TEXT NOT NULL,

     payment_method_id INTEGER NOT NULL,

     order_status_id INTEGER NOT NULL,

     CONSTRAINT "order_has_existing_customer" FOREIGN KEY ("customer_id")
REFERENCES "customer" ("id"),

     CONSTRAINT "order_has_existing_payment_method" FOREIGN KEY
("payment_method_id") REFERENCES "payment_method" ("id"),

     CONSTRAINT "order_has_existing_status" FOREIGN KEY ("order_status_id")
REFERENCES "order_status" ("id")

);
```

Tilaustaulun kohdalla poikettiin yleisestä taulujen nimeämiskäytännöstä käyttämällä taulun nimessä monikkoa. Tämä johtuu siitä, että order on varattu sana SQL:ssä. Varattujen sanojen käyttö taulun tai sarakkeen nimissä aiheuttaa tarpeettomia hankaluuksia sovellusohjelmoijalle, sillä tällöin on huolehdittava siitä, että sana suljetaan jokaisessa kyselyssä tietokannanhallintaohjelmiston määrittelemien tilanvaihtosekvenssien sisään. Ongelmia saattaa syntyä eritoten kolmannen osapuolen kirjastojen kanssa, jotka eivät välttämättä toteuta tätä vaatimusta.

Tilauksen suhteet toteutettiin määrittelemällä liitettävään tauluun viittaava vierasavain orders-taulussa. Asiakkaan tapauksessa tämä avain on ”customer_id”. Vierasavaimet lisättiin myös maksutapa- ja tilaoliota varten. Tilaukseen kuuluvat tuotteet määriteltiin tilaukseen viittaavalla vierasavaimella ”order_product”-liitostaulussa. Taulu luotiin seuraavalla kyselyllä:

```
CREATE TABLE order_product (
```

```
    id INTEGER NOT NULL PRIMARY KEY,

    order_id INTEGER NOT NULL,

    product_id INTERGER NOT NULL,

    product_price REAL NOT NULL,

    quantity INTEGER NOT NULL,

    CONSTRAINT "order_product_has_existing_order" FOREIGN KEY ("order_id")
REFERENCES "orders" ("id")

);
```

Taulu sisältää vierasavaimen product-tauluun, joka sisältää tuote-mallin tiedot. Tämän lisäksi se sisältää tuotteiden määrän ja tuotteen hinnan tilauksen lähetyshetkellä. On huomattava, että product_id-sarakkeelle ei ole määritelty rajoitetta. Tämä johtuu siitä, että kyseinen tuote voidaan poistaa kokonaan järjestelmästä. Rajoite tekisi poiston mahdottomaksi. Tällaista tilannetta varten tilatun tuotteen tuotenimike tallennetaan order_product_data-tauluun, josta se on noudettavissa useana kieliversiona, vaikka varsinainen tuote olisikin poistettu. Taulu luotiin seuraavalla kyselyllä:

```
CREATE TABLE order_product_data (

    id INTEGER NOT NULL PRIMARY KEY,

    order_product_id INTEGER NOT NULL,

    lang_code TEXT NOT NULL,

    title TEXT NOT NULL,

    CONSTRAINT "order_product_data_has_parent" FOREIGN KEY
("order_product_id") REFERENCES "order_product" ("id")

);
```

Tuotenimike on haettavissa taulusta tilaukseen kuuluvan tuotteen, eli order_product-taulun id-sarakkeen ja halutun kieliversion koodin perusteella.

6.4 ORM-kehys

Ennen kuin tietokantasuunnittelusta siirryttiin mallien suunnitteluun toteutettiin Lastenkirjakaupalle mallien käsittelyyn tarkoitetut luokat.

Kuten luvussa 4.4.3 todettiin, Zend Framework ei tarjoa valmista mallitoteutusta. Tämä tarkoittaa paitsi sitä, että Zend_Model-luokkaa ei ole olemassa, myös sitä, että kehysjärjestelmästä ei löydy mallien käsittelyyn liittyvää valmista toiminnallisuutta.

Zend_Db-tietokanta-abstraktiokerros ja Zend_Db_Table tarjoavat oliopohjaisen pääsyn tietokantataulun tietoihin, mutta varsinaisen ORM, eli olio-relaatio-mallinnus-kehyksen toteutus jää sovellusohjelmoijan harteille.

Seuraavassa perehdytään Lastenkirjakaupan ORM-kehyksen toteutukseen. Kehys asetettiin Lkk_Model_-nimiavaruuden alle. Nimiavaruus on lisätty Zend Frameworkin __autoload-toteutuksen polkuihin, jolloin sen alla olevat luokat löytyvät automaattisesti.

6.4.1 Entiteetit

Lkk_Model_Entity_Abstract on abstrakti luokka, joka luo pohjan sovelluksen malliluokkien toteutukselle. Kaikki lastenkirjakaupan mallit eli entiteetit periytetään tästä isäntäluokasta.

Luokka sisältää toteutukset maagisille metodeille __set ja __get, joiden avulla toisaalta taataan se, ettei entiteetille ole mahdollista lisätä siihen kuulumattomia jäsenmuuttujia ja toisaalta tarjotaan yksinkertainen tapa käsitellä sen suojattuja jäsenmuuttujia. Se sisältää myös mahdollisuuden asettaa kaikki entiteetin jäsenmuuttujat taulukon avulla.

Luokka ei sisällä paljon koodia, joten seuraavassa on sen toteutus kokonaisuudessaan kommenttilohkoista riisuttuna.

```php
<?php

abstract class Lkk_Model_Entity_Abstract {

	public function __construct(array $options = null) {

		If (is_array($options)) {

			$this->setOptions($options);

		}

	 }

	public function __set($property, $value) {

		$method = 'set' . $property;

		if (!method_exists($this, $method)) {

		throw new Exception('Error: '. $property . ' is not a property of
' . get_class($this));

		}

		$this->$method($value);

	}

	public function __get($property) {

		$method = 'get' . $property;

		if (!method_exists($this, $method)) {

		throw new Exception('Error: '. $property . ' is not a property of
' . get_class($this));

		}

		return $this->$method();

	}

	public function setOptions(array $options) {

		$methods = get_class_methods($this);
```

```
            foreach ($options as $key => $value) {
                $method = 'set' . ucfirst($key);
                if (in_array($method, $methods)) {
                    $this->$method($value);
                }
            }
            return $this;
        }
}
```

Luokan setOptions-metodi palauttaa luokan sen instanssin, johon se kuuluu, jolloin metodeita on mahdollista ketjuttaa.

Lastenkirjakauppaan on lisätty Entity_-nimiavaruus entiteettejä varten. Entiteetit on nimettävä Entity_Nimi-kaavan mukaisesti. Seuraavassa on yksinkertainen mallitoteutus osoitemallille Entity_Address.

```
<?php
class Entity_Address extends Lkk_Model_Entity_Abstract {
    protected $id;
    protected $city;
    protected $streetAddress;
    protected $zipCode;
    public function __toString() {
        return $this->streetAddress . " " . $this->zipCode . " " . $this-
>city;
    }
    public function setId($id) {
```

```
        $this->id = (int) $id;
        return $this;
    }
    public function getId() {
        return $this->id;
    }
    public function setCity($text) {
        $this->city = $text;
        return $this;
    }
    public function getCity() {
        return $this->city;
    }
    public function setStreetAddress($text) {
        $this->streetAddress = $text;
        return $this;
    }
    public function getStreetAddress() {
         return $this->streetAddress;
    }
    public function setZipCode($num) {
        $this->zipCode = $num;
        return $this;
    }
```

```
    public function getZipCode() {
        return $this->zipCode;
    }
}
```

Samoin, kuin Lkk_Model_Entity_Abstract-luokan setOptions-metodi, myös Entity_Address luokan asettajametodit palauttavat luokan instanssin. Tällöin instanssin ominaisuudet on mahdollista asettaa seuraavasti.

```
$address = new Entity_Address();
$address->setId(1)
        ->setStreetAddress(”Katuosoite”)
        ->setZipCode(12345)
        ->setCity(”Kaupunki”);
```

Tosiasiassa entiteetin id-jäsenmuuttujaa ei aseteta eksplisiittisesti. Lkk_Model_Mapper_DbTable_Interface-rajapinnan toteuttavat luokat asettavat sen automaattisesti, joko haettaessa entiteetti tietokannasta tai tallennettaessa se sinne.

6.4.2 Entiteettien hallinta

Lkk_Model_EntityManager tarjoaa facade-suunnittelumallin mukaisen rajapinnan ORM-kehyksen toiminnallisuuteen. Sen avulla on mahdollista paikallistaa ja tallentaa entiteettejä. Ohjainluokat käyttävät yksinomaan sitä entiteettien hallintaan.

Luokan sydän on getMapperFor-metodi, joka toimii luomalla entiteetin instanssin sille parametrina annetun $identifier-merkkijonon perusteella. Identifier, eli tunniste on se osa entiteetin luokan nimeä, joka tulee merkkijonon ”Entity_” jälkeen. Osoitemallin tapauksessa tunniste on ”Address”. Tunnisteen

perusteella luotu instanssi annetaan edelleen parametriksi Lkk_Model_Mapper_Context-oliolle, joka puolestaan välitetään Lkk_Model_Mapper_Factory-tehdasluokalle. Tehdasluokka palauttaa entiteetille kuuluvan mallintaja-olion, jonka avulla entiteettejä on mahdollista paikallistaa ja tallentaa. Seuraavassa on getMapperFor-metodi kokonaisuudessaan.

```
public function getMapperFor($identifier) {

     $entityClass = "Entity_" . ucfirst($identifier);

     $context= new Lkk_Model_Mapper_Context(new $entityClass());

     return Lkk_Model_Mapper_Factory::getInstance($context);

     }
```

Luokkaa käytetään seuraavasti:

```
$em = new Lkk_Model_EntityManager();

$product = $em->getMapperFor(”product”)->fetchSingle();
```

Lkk_Model_EntityManager toteuttaa maagiset metodit __call ja __callStatic, joiden avulla sen käyttö on yksinkertaisempaa. __call mahdollistaa getMapperFor-metodin kutsumisen tyylikkäämmin. Seuraavassa on sen toteutus.

```
public function __call($func, $args) {

     if (strpos($func, "Mapper") !== false) {

          $param = substr($func, 0, -6);

          Return $this->getMapperFor($param);

     }

}
```

Tällöin entiteetin mallintaja-luokkaa on mahdollista kutsua seuraavasti:

```
$em = new Lkk_Model_EntityManager();
```

```
$product = $em->productMapper()->fetchSingle();
```

__call toimii olion näkyvyysalueella. __callStatic totetuttaa saman toiminnallisuuden luokan näkyvyysalueella. Sen avulla mallintaja-luokkaa on mahdollista kutsua staattisesti.

```
$product = Lkk_Model_EntityManager::productMapper()->fetchSingle();
```

6.4.3 Olio-relaatio-mallinnuksen rajapinta

Entiteetin mallintamisen sen tallennuspaikkaan toteuttavat luokat, jotka implementoivat Lkk_Model_Mapper_Interface-rajapinnan. Seuraavassa on rajapinnan toteutus.

```
<?php

interface Lkk_Model_Mapper_Interface {

      public function __construct(Lkk_Model_Mapper_Context $context);

      public function find($param = null);

      public function persist(Lkk_Model_Entity_Abstract $entity);

      public function delete(Lkk_Model_Entity_Abstract $entity);

}
```

Rajapinta määrittelee sen, että toteuttavan luokan rakentaja-metodin on otettava parametrikseen Lkk_Model_Mapper_Context-luokan instanssi. Edelleen luokasta on löydyttävä julkiset metodit find, persist ja delete.

6.4.4 Olio-relaatio-mallinnuksen konteksti

Lkk_Model_Mapper_Context-luokka sisältää tiedot siitä, mikä entiteetti on kyseessä, mikä sille kuuluva mallintaja-luokka on ja siitä mikä Lkk_Model_Gateway_Abstract-luokan perillinen edustaa entiteetille kuuluvaa

tietokantataulua, jos sellainen on olemassa. Kaikkia entiteettejä ei tallenneta tietokantaan, joten kaikille ei myöskään kuulu Lkk_Model_Gateway-luokkaa.

Lkk_Model_Mapper_Context selvittää entiteetin mallintaja-luokan pyydettäessä getMapperClass-metodin avulla. Se toimii tutkimalla entiteettiluokan kommenttilohkoa ja etsimällä siitä ”@Mapper”-tunnistetta. Kommenttilohkon läpikäymisessä käytetään hyväksi PHP:n Reflection-API:a sekä sitä laajentavaa Zend_Reflection-luokkakirjastoa. Mikäli tunniste löytyy palautetaan metodin kutsujalle asiaankuuluvan mallintaja-luokan nimi merkkijonona. Seuraavassa on getMapperClass-metodin toteutus.

```
public function getmapperClass() {
     if (isset($this->mapperClass)) {
          return $this->mapperClass;
     }
     $ref = new ReflectionObject($this->entity);
     try {
          $zendRefDoc = new Zend_Reflection_Docblock($ref);
          if ($zendRefDoc->hasTag("Mapper")) {
               $tag = $zendRefDoc->getTag("Mapper");
               $mapperClass = $tag->getDescription();
               if (class_exists($mapperClass)) {
                    $this->mapperClass = $mapperClass;
                    return $this->getMapperClass();
               }
          }
     } catch(Exception $e) {}
```

```
            return null;
        }
```

6.4.5 Mallintajatehdas

Mallintaja-luokan varsinainen alustaminen toteutetaan Lkk_Model_Mapper_Factory-tehdasluokassa. EntityManager kutsuu tehdasluokan getInstance-metodia, joka käyttää hyväkseen yllä kuvattua Lkk_Model_Mapper_Context-luokan getMapperClass-metodia. Seuraavassa on tehdasluokan getInstance-metodin toteutus.

```
public static function getInstance(Lkk_Model_Mapper_Context $context) {
        if (!is_null($mapperClass = $context->getMapperClass())) {
            if (isset(self::$mappers[$mapperClass])) {
                return self::$mappers[$mapperClass];
            } else {
                self::$mappers[$mapperClass] = new $mapperClass($context);
                return self::$mappers[$mapperClass];

            }
        }
        return self::getDefaultMapper($context);
}
```

Tehdasluokka tarkastaa onko sille parametrinä annetusta Lkk_Model_Mapper_Context-oliosta saatavilla mallintaja-luokan nimeä. Mikäli Lkk_Model_Mapper_Context::getMapperClass palauttaa null-arvon, kutsutaan tehdasluokasta getDefaultMapper-metodia, joka palauttaa oletusmallintajan eli Lkk_Model_Mapper_DbTable-luokan instanssin. Tällöin tietokantaan tallennettaville entiteeteille ei ole tarpeen määritellä @Mapper-tunnistetta.

Suurin osa entiteeteistä tallennetaan tietokantaan, joten tämä vähentää turhaa toistoa.

Tehdasluokka tallentaa alustamansa mallintaja-luokat staattiseen taulukkoon, jolloin yhdestä luokasta alustetaan vain yksi instanssi. Yksi instanssi riittää, sillä kulloinkin käytössä olevasta Lkk_Model_Mapper_Context-luokasta saadaan selville sekä entiteetti että tietokantataulu, jolloin tietyn entiteetin mallintaja-luokan metodit on mahdollista toteuttaa kyseisen entiteetin asiayhteydessä.

6.4.6 Entiteettien tallennus istuntoon

Entiteetin tallentamisen istuntoon toteuttaa Lkk_Model_Mapper_Session-luokka. Se käyttää hyväkseen Zend_Session-istuntojenhallintakomponenttia. Se sisältää tyhjän toteutuksen Lkk_Model_Mapper_Interface-rajapinnan määrittelemälle persist-metodille. Tämä johtuu siitä, että istuntoon tallennettu tieto tallentuu automaattisesti sitä muutettaessa.

Luokan rakentajametodissa alustetaan Zend_Session_Namespace-olio, jonka avulla istuntoon tallennettua entiteettiä hallinnoidaan. Seuraavassa on rakentajametodin toteutus.

```
public function __construct(Lkk_Model_Mapper_Context $context) {

     $this->context = $context;

     $this->session = new Zend_Session_Namespace();

}
```

Istuntoon tallennettu olio paikallistetaan Lkk_Model_Mapper_Interface-rajapinnan määrittelemällä find-metodilla. Seuraavassa on find-metodin toteutus.

```
public function find($param = null) {

     $identifier = $this->getContext()->getIdentifier();
```

```
        if (isset($this->session->$identifier)) {
            return $this->session->$identifier;
        } else {
            $entity = $this->getContext()->getEntityClass(true);
            $this->session->$identifier = $entity;
            return $this->session->$identifier;
        }
}
```

Metodi tarkastaa, onko istuntoon tallennettu Lkk_Model_Mapper_Context-olion määrittelemää entiteettiä. Mikäli entiteetti löytyy, se palautetaan, ja jos sitä ei löydy tallennetaan istuntoon entiteetin uusi instanssi. Luokkaa käytetään seuraavasti:

```
$em = new Lkk_Model_EntityManager();
$cart = $em->cartMapper()->find();
```

Kuten yllä olevasta koodista ilmenee, voivat ohjainluokat paikallistaa istuntoon tallennettuja oliota täsmälleen samalla tavalla kuin tietokantaan tallennettuja olioita. Toisin sanoen ohjainluokan käyttäessä yhtenäistä facade-rajapintaa ORM-kehyksen toiminnallisuuteen, sen ei tarvitse huolehtia siitä, mihin olio on tallennettu.

6.4.7 Entiteetin tietokantarajapinta

Entiteetin tallentamisen tietokantaan toteuttaa Lkk_Model_Mapper_DbTable-luokka. Tälle luokalle on määritelty oma rajapintansa Lkk_Model_Mapper_Dbtable_Interface, joka periytetään Lkk_Model_Mapper_Interface -rajapinnasta.

```
<?php
```

```
interface Lkk_Model_Mapper_DbTable_Interface extends
Lkk_Model_Mapper_Interface {

     public function fetchSingle(Lkk_Model_Mapper_DbTable_Criteria $criteria);

     public function fetchCollection(Lkk_Model_Mapper_DbTable_Criteria
$criteria = null);

     public function fetchPage($page = 1, $count = 20,
Lkk_Model_Mapper_DbTable_Criteria $criteria = null);

}
```

Rajapinta määrittelee tietokannan käsittelyssä tarvittavat metodit fetchSingle, fetchCollection ja fetchPage. Nämä metodit käyttävät hyväkseen Lkk_Model_Mapper_Criteria-luokan tarjoamaa toiminnallisuutta SQL-kyselyjen muodostamiseen.

6.4.8 Entiteettien paikallistamisen rajaaminen

Lkk_Model_Mapper_Criteria käärii Zend_Db_Table_Select-luokan. Zend_Db_Table_Select tarjoaa oliopohjaisen käyttöliittymän SQL-kyselyn muodostamiseen.

Lkk_Model_Mapper_Criterian avulla on mahdollista asettaa tiettyjä SQL-kyselyä rajaavia ehtoja, joiden perusteella luokka luo ja palauttaa Zend_Db_Table_Select-olion. Seuraavassa on esimerkki EntityManagerin suorittamasta entiteettien paikallistamisesta, jossa käytetään Lkk_Model_Mapper_DbTable_Criteria-luokkaa rajoittamaan tulosjoukkoa.

```
$em = new Lkk_Model_EntityManager();

$criteria = new Lkk_Model_Mapper_DbTable_Criteria();

$criteria->setLimit(4)

          ->setOrder(”dateCreated”)

          ->addEquals(”categoryId”, 1);
```

```
$products = $em->productMapper()->fetchCollection($criteria);
```

Lkk_Model_Mapper_DbTable_Criteria käyttää entiteettien jäsenmuuttujien nimiä ehtojen asettamiseen. Tämä eroaa Zend_Db_Table_Selectistä, joka käyttää tietokantataulun sarakkeiden nimiä. Tästä on hyötyä ohjainluokkia kehitettäessä. Lkk_Model_Mapper_DbTable_Criteria sallii ohjaimen hienosäätää kyselyä tarpeen mukaan, silti säilyttäen ohjaimen eriytettynä varsinaisesta tietokannasta.

6.4.9 Entiteetin tietokantatauluolio

Yksinkertaisimmillaan Zend Frameworkin malliluokka voidaan luoda periyttämällä tietokantataulua edustava luokka Zend_Db_Table_Abstract-luokasta. Zend_Db_Table_Abstract sisältää valmiin toiminnallisuuden tietokantataulun tietojen luomiseen, lukemiseen, päivittämiseen ja poistoon. Kuten luvussa 4 todettiin, luokka toteuttaa table-data-gateway-suunnittelumallin, joka tarkoittaa sitä, että se toimii ikään kuin yhdyskäytävänä tietokantataulun tietoihin.

Lastenkirjakaupan ORM-kehys käyttää hyväkseen tätä luokkaa tietokantataulujen käsittelyssä. Jokaiselle tietokantaan tallennettavalle entiteetille luodaan Lkk_Model_Gateway_Abstract-luokan perivä tietokantayhdyskäytävä. Näille luokille on lastenkirjakaupassa varattu nimiavaruus Gateway_. Luokat tulee nimetä Gateway_Nimi -kaavan mukaisesti. Tällöin osoitemallia Entity_Address vastaava tietokantayhdyskäytävä on nimeltään Gateway_Address.

Lkk_Model_Gateway_Abstract perii Zend_Db_Table_Abstract-luokan, jolloin se sisältää saman tietokannankäsittelyyn liittyvän toiminnallisuuden kuin Zend Frameworkin oletusmallit. Tämän lisäksi luokka sisältää refrerenceMap-taulukon, joka listaa tietokantataulun sarakkeiden nimet ja niitä vastaavat

entiteettien jäsenmuuttujien nimet. Alla on esimerkkitaulukko Gateway_Address -luokasta.

```
protected $referenceMap = Array (
     "id"            => "address.id",
     "city"          => "address.city",
     "streetAddress" => "address.street_address",
     "zipCode"       => "address.zip_code"
);
```

Lkk_Model_Gateway_Abstract sisältää metodit getColumnName ja getPropertyName, joiden avulla on mahdollista selvittää jäsenmuuttujan nimeä vastaava tietokantataulun sarakenimi ja päinvastoin. Alla on getColumnName-metodin toteutus.

```
public function getColumnName($property) {
     if (array_key_exists((String)$property, $ref = $this->getReferenceMap()))
{
          return $ref[$property];
     }
     return null;
}
```

Metodi käy läpi referenceMap-taulukon ja etsii sen indeksistä parametrina annettua $property-merkkijonoa. Mikäli merkkijono löytyy, metodi palauttaa indeksiä vastaavan alkion. Monet lastenkirjakaupan ORM-kehyksen luokat käyttävät hyväkseen tätä toiminnallisuutta eriyttääkseen oman toimintansa alla olevasta tietokannasta. Tällöin ORM-kehyksen luokat voivat käsitellä entiteettejä ja niiden tietokanta-vastineita tietämättä mitään varsinaisen tietokannan toteutuksesta.

6.4.10 Entiteetin konvertointi

Jotta Zend_Db_Table-komponenttia voitaisiin hyödyntää entiteettien tallentamisessa tietokantaan, on entiteetti konvertoitava tietokantariviä edustavaksi Zend_Db_Table_Row-olioksi. Samoin, jotta sen avulla olisi mahdollista hakea tietoa tietokannasta, on kyselyn tuloksena palautettava Zend_Db_Table_Row-, tai Zend_Db_Table_RowSet-olio konvertoitava takaisin entiteetiksi tai entiteettien kokoelmaksi.

Tämä toiminnallisuus on Lkk_Model_Gateway_Converter-luokan vastuulla. Luokka sisältää metodit rowToObj, rowSetToCollection ja objToRow, jotka toteuttavat konvertoinnin.

RowToObj-metodin toteutus on yksinkertainen. Sen tarvitsee vain käydä läpi tietokantayhdyskäytävästä löytyvä referenceMap-taulukko ja asettaa parametrinä annetun Zend_Db_Table_Row olion ominaisuudet referenceMap-taulukon tietojen avulla. Seuraavassa on metodin toteutus.

```
public function rowToObj(Zend_Db_Table_Row_Abstract $row) {

     $t = $row->getTable();

     $obj = $this->getEntityForRow($row);

     foreach ($ref = $t->getReferenceMap() as $prop => $col) {

          $col = array_pop(explode(".", $col));

          if ((isset($row->$col))) {

               $method = "set" . ucfirst($prop);

               if (method_exists($obj, $method)) {

                    $obj->$method($row->$col);

               }

          }
```

```
        }

        unset($t);

        return $obj;

}
```

RowSetToCollection-metodi on niin ikään yksinkertainen. Se käy läpi sille parametrina annetun Zend_Db_Table_RowSet-olion ja kutsuu rowToObj -metodia, jokaisen sen sisältämän Zend_Db_Table_Row-olion kohdalla. RowToObj-metodin palauttama Zend_Db_Table_Row tallennetaan taulukkoon, joka lopuksi palautetaan. Seuraavassa on sen toteutus:

```
public function rowSetToCollection(Zend_Db_Table_Rowset_Abstract $rowSet) {

        $collection = array();

        foreach ($rowSet as $row) {

                $collection[] = $this->rowToObj($row);

        }

        return $collection;

}
```

ObjToRow-metodi sen sijaan vaatii monimutkaisemman toteutuksen. Tämä johtuu siitä, että Zend_Db_Table_Row sisältää tiedon siitä, onko olio kiinnitetty tietokantaan vai ei. Toisin sanoen, edustaako se uutta riviä, jota ei vielä ole tallennettu vai olemassa olevaa riviä. Tästä johtuen objToRow-metodissa on ensin tarkastettava se, onko konvertoitava entiteetti jo tallennettu vai onko se luotu new-avainsanaa käyttäen. Seuraavassa on metodin toteutus:

```
public function objToRow(Lkk_Model_Entity_Abstract $obj) {

                $t = $this->getTableForEntity($obj);

                // do some setup to handle composite primary keys

                $tableName = $t->info("name");
```

```
            $primaryKeys = $t->info("primary");
            $primaryValues = Array();
            $isExistingRow = true;
            foreach ($primaryKeys as $key) {
                  $property = $t->getPropertyName($tableName . "." . $key);
                  if (is_null($obj->$property)) {
                        $isExistingRow = false;
                        break;
                  } else {
                        $primaryValues[] = $obj->$property;
                  }
            }
            if ($isExistingRow) {
                  return $this->objToExistingRow($obj, $t, $primaryValues);
            } else {
                  return $this->objToNewRow($obj, $t);
            }
      }
```

Mikäli kyseessä on olemassa oleva rivi, kutsuu metodi objToExistingRow-metodia, joka palauttaa tietokantaan kiinnitetyn riviolion. Muussa tapauksessa kutsutaan objToNewRow-metodia, joka palauttaa tallentamattoman rivin.

6.4.11 Entiteetin tallennus tietokantaan

Entiteetin tallentamisen tietokantaan toteuttaa Lkk_Model_Mapper_DbTable-luokka. Se hyödyntää Lkk_Model_Mapper_DbTable_Criteria ja

Lkk_Model_Gateway_Converter luokkia, sekä Lkk_Model_Gateway_Abstract-luokasta periytettyjä tietokantayhdyskäytävää tallentamisen toteuttamiseen.

Alla on luokan sisältämän fetchSingle-metodin toteutus. FetchSingle paikallistaa yksittäisen entiteetin sille parametrina annetun Lkk_Model_DbTable_Criteria -olion ehtojen perusteella.

```
public function fetchSingle(Lkk_Model_Mapper_DbTable_Criteria $criteria) {
     $t = $this->getTable();
     $criteria->setTable($t);
     if ($row = $t->fetchRow($criteria->getSelect())) {
          return $this->getConverter()->rowToObj($row);
     }
     return null;
}
```

Tietokantaan tallentaminen tapahtuu Lkk_Model_Mapper_Interface-rajapinnan määrittelemän persist-metodin toteutuksella.

```
public function persist(Lkk_Model_Entity_Abstract $entity) {
     $row = $this->getConverter()->objToRow($entity);
     $row->save();
     if (method_exists($entity, "setId") && method_exists($entity, "getId")) {
          if ($entity->getId() === null) {
               $entity->setId($row->id);
          }
     }
}
```

Metodi asettaa tallennettavan entiteetin id-jäsenmuuttujan arvon tallennuksen jälkeen, mikäli se on asettamatta.

6.5 Mallien toteutus

Kun mallien käsittelyyn tarvittavat luokat olivat valmiina, siirryttiin varsinaisten malliluokkien toteuttamiseen. Tästä huolimatta ORM-kehyksen kehitystä jatkettiin koko Lastenkirjakaupan toteutusprojektin ajan ja siihen tehtiin tarvittaessa muutoksia ja laajennuksia.

Seuraavassa esitellään ensin yksinkertainen mallitoteutus. Sen jälkeen perehdytään monikielisten mallien toteutukseen ja lopuksi selvitetään miten malliluokkaa on mahdollista käyttää hyväksi käyttäjien autentikoinnissa.

6.5.1 Yksinkertainen mallitoteutus

Yksinkertaisimmillaan Lastenkirjakaupan malliluokka sisältää metodit luokan jäsenmuuttujien arvon asettamiseen ja lukemiseen. Hyvä esimerkki tällaisesta luokasta on luvussa 6.4.1 esitelty osoite-entiteetti.

Entiteetin koodia ei toisteta tässä. Sen sijaan alla on kokonaisuudessaan sille kuuluva tietokantayhdyskäytäväluokka.

```
<?php

class Gateway_Address extends Lkk_Model_Gateway_Abstract {

      protected $_primary = 'id';

      protected $_name = 'address';

      protected $referenceMap = Array (

            "id"            => "address.id",

            "city"                    => "address.city",

            "streetAddress" => "address.street_address",
```

```
            "zipCode"       => "address.zip_code"
      );
}
```

Koska ORM-kehys hoitaa entiteettien tallentamisen ja paikallistamisen, ei tämän lisäksi vaadita muuta kuin varsinainen tietokantataulu, johon mallin tiedot tallennetaan.

Tämä vähentää kirjoitettavan koodin määrää ja siten vähentää turhaa toistoa. Sen lisäksi virheitä syntyy sitä vähemmän, mitä vähemmän koodia on kirjoitettava.

6.5.2 Mallin tietojen automaattinen haku tietokannasta

Monet lastenkirjakaupan entiteetit sisältävät jäsenmuuttujinaan toisen entiteetin. Näiden entiteettien alustaminen toteutetaan laiskasti, mikä tarkoittaa sitä, että niiden sisältämät tiedot haetaan tietokannasta vasta pyydettäessä.

Seuraavassa on Entity_Order-tilausentiteetin getCustomer-metodi, joka palauttaa tilausentiteetille kuuluvan asiakasentiteetin.

```
public function getCustomer() {
      if (isset($this->customer)) {
            return $this->customer;
      } else {
            $em = new Lkk_Model_EntityManager();
            if ($customer =  $em->customerMapper()->find($this-
>getCustomerId())) {
                  $this->setCustomer($customer);
                  return $this->getCustomer();
            }
```

```
    }
    return null;
}
```

Metodi tarkastaa ensin onko tilaukseen jo tallennettu asiakasentiteettiä. Mikäli sellainen löytyy, se palautetaan. Muussa tapauksessa metodi paikallistaa asiakkaan Lkk_Model_EntityManager-luokan avulla ja tallentaa sen jäsenmuuttujakseen, minkä jälkeen se kutsuu uudelleen getCustomer-metodia.

6.5.3 Useita kieliversioita sisältävä malli

Useita kieliversioita sisältävät entiteetit paikallistavat tietyn kieliversioentiteetin tiedot yllä kuvatulla tavalla. Kieliversioiden paikallistaminen tapahtuu kuitenkin samalla tavalla jokaisen useita kieliversioita sisältävän entiteetin kohdalla, joten tätä varten päätettiin luoda isäntäluokka Entity_MultiLangEntity.

Entity_MultiLangEntity sisältää metodin getLangData kieliversioentiteetin paikallistamiseen. Seuraavassa on sen toteutus.

```
public function getLangData($lang) {
    if (array_key_exists($lang, $this->getData())) {
        $data = $this->getData();
        return $data[$lang];
    } else {
        $ns = explode("_", get_class($this));
        $identifier =  array_pop($ns);
        $identifier[0] = strtolower($identifier[0]);
        $criteria = new Lkk_Model_Mapper_DbTable_Criteria();
        $criteria->addEquals("langCode", $lang);
```

```
            $idProperty = $identifier . "Id";
            $criteria->addEquals($idProperty, $this->getId());
            $mapperName = $identifier . "DataMapper";
            $em = new Lkk_Model_EntityManager();
            if ($data = $em->$mapperName()->fetchSingle($criteria)) {
                  $langData = $this->setLangData($lang, $data);
                  $criteria->clearAll();
                  return $langData;
            }
      }
      return null;
}
```

Metodi tallentaa paikallistamansa kieliversioentiteetit $data-taulukkoon, jolloin ne on tarpeen hakea vain kerran.

6.5.4 Autentikoitava malli

Lastenkirjakaupassa on käytössä kahdenlaisia käyttäjiä. Ensimmäinen on pääkäyttäjä, jolla on pääsy kaupan hallintapuolelle ja toinen kaupan asiakas, jonka on kirjauduttava sisään voidakseen selata tilauksiaan ja kommentoidakseen blogimerkintöjä.

Pääkäyttäjä mallinnetaan Entity_User-luokan avulla ja asiakas Entity_WebUser -luokan avulla. Entity_WebUser sisältää asiakasentiteetin, joka puolestaan sisältää asiakkaan tilauksen käsittelyssä tarvittavat tiedot.

Käyttäjien autentikointiin käytetään Zend Frameworkin Auth-komponenttia. Tätä varten Lastenkirjakaupalle luotiin uusi mallintajaluokka

Lkk_Model_Mapper_DbTable_Auth, joka käärii Zend_Auth-komponentin. Lkk_Model_Mapper_DbTable_Auth perii Lkk_Model_Mapper_DbTable-luokan ja lisää siihen autentikoinnissa tarvittavan toiminnallisuuden.

Tämä tarkoittaa sitä, että entiteettien, jotka käyttävät hyväkseen autentikointitoiminnallisuutta, on otettava Lkk_Model_Mapper_DbTable_Auth -mallintajaluokka käyttöön lisäämällä kommenttilohkoonsa tunniste @Mapper Lkk_Model_Mapper_DbTable_Auth. Tämän lisäksi Lkk_Model_Mapper_DbTable_Auth vaatii kolme muuta tunnistetta. Näihin kuuluvat @AuthCredential, jonka avulla määritellään autentikoinnin käyttäjänimenä käytettävä jäsenmuuttuja ja @AuthSecret, jonka avulla määritellään autentikoinnissa käytettävä salasana, sekä @AuthGateway, joka määrittelee autentikoinnissa käytettävän tietokantayhdyskäytävän. Seuraavassa on Entity_WebUser luokan kommenttilohkototeutus:

```
<?php

/**

 * WebUser entity: Models the publicly available user

 * @Mapper Lkk_Model_Mapper_DbTable_Auth

 * @AuthGateway webUser

 * @AuthCredential userName

 * @AuthSecret password

 */

class Entity_WebUser extends Lkk_Model_Entity_Auth_Abstract {

     // ….

}
```

Kuten yllä olevasta koodista käy ilmi, autentikoitavat entiteetit periytetään Lkk_Model_Entity_Auth_Abstract-luokasta, joka perii

Lkk_Model_Entity_Abstract-luokan. Lkk_Model_Entity_Auth_Abstract lisää Lkk_Model_Entity_Abstract-luokkaan isAuthenticated-jäsenmuuttujan sekä metodit sen asettamiseksi ja lukemiseksi. Lkk_Model_Mapper_DbTable_Auth käyttää tätä jäsenmuuttujaa merkitsemään entiteetin autentikoiduksi.

Lkk_Model_Mapper_DbTable_Auth-luokalle luotiin myös rajapinta Lkk_Model_Mapper_Auth, jonka toteuttamalla on mahdollista toteuttaa vaihtoehtoisia autentikointimenetelmiä. Seuraavassa on sen toteutus.

```
<?php

interface Lkk_Model_Mapper_Auth_Interface extends Lkk_Model_Mapper_Interface {

     public function getCurrent();

     public function authenticate(Lkk_Model_Entity_Auth_Abstract $entity);

     public function logOut();

}
```

6.6 Ohjaimien toteutus

Malliluokkien toteutuksen jälkeen toteutettiin lastenkirjakaupan ohjainluokat. Kuten luvussa 4 todettiin, Zend Frameworkissa ohjainluokat periytetään Zend_Controller_Action-luokasta.

Ohjainluokat yhdistävät mallit ja näkymät toisiinsa. Tästä johtuen niiden toiminnallisuus on usein samankaltainen luokasta riippumatta. Useimmiten niiden toiminta rajoittuu tiedon hakemiseen tietokannasta ja sen välittämiseen näkymälle, sekä käyttäjän syötteen käsittelyyn ja tallentamiseen edelleen tietokantaan.

Ohjainluokkien samankaltaisuudesta johtuen kaikkien ohjainten yksityiskohtainen käsittely ei tässä ole mielekästä. Sen sijaan perehdytään

tarkemmin vain yhteen ohjaimeen. Tiettyjen näkymien erityisvaatimuksia ohjainluokille käsitellään myöhemmin näkymien käsittelyn yhteydessä.

6.6.1 Indeksiohjain

Indeksiohjain on se sivuohjain, jolle Zend Frameworkin jakaja oletuksena ohjaa HTTP-pyynnön. Siihen päästään käsiksi joko lähettämällä Lastenkirjakauppaan pyyntö, jossa ei määritellä sivuohjainta eikä action-metodia, toisin sanoen kirjoittamalla WWW-selaimen osoitekenttään www.lastenkirjakauppa.fi, tai määrittelemällä sivuohjaimeksi index, toisin sanoen kirjoittamalla WWW-selaimen osoitekenttään www.lastenkirjakauppa.fi/index. Ohjain luodaan seuraavasti.

```
<?php

class IndexController extends Zend_Controller_Action {

     //...

}
```

Lastenkirjakaupan indeksiohjain sisältää vain yhden action-metodin. Tämä metodi on nimeltään indexAction. IndexAction kutsutaan sivuohjaimesta oletuksena, mikäli action-metodia ei määritellä. Se on myös mahdollista kutsua eksplisiittisesti, jolloin pyyntö on lähettävä osoitteeseen www.lastenkirjakauppa.fi/index/index.

Metodi sisältää kolme vaihetta, joista ensimmäinen on pyynnön parametrien selvittäminen. Se toteutetaan seuraavasti:

```
public function indexAction() {

     if (!$agegroup = $this->getRequest()->getParam("agegroup")) {

          $agegroup = "all";

     }
```

```
$this->view->agegroup = $agegroup;
if (!$category = $this->getRequest()->getParam("category")) {
     $category = 1;
}
if (!is_numeric($category)) {
     $category = 1;
}
$this->view->category = $category;
if (!$lang = $this->getRequest()->getParam("lang")) {
     $lang = "fi";
}
$this->view->lang = $lang;
     // jatkuu...
```

Kun pyynnön parametrit on käsitelty asetetaan näkymälle Zend_Form-luokasta periytetty kirjautumislomake-olio.

```
$webUserForm = new Form_WebUserLogin();
$webUserForm->setLang($this->view->lang);
$this->view->webUserForm = $webUserForm;
```

Tämän jälkeen siirrytään tiedon hakemiseen tietokannasta. Se toteutetaan seuraavasti:

```
$em = new Lkk_Model_EntityManager();
$this->view->webUser = $em->webUserMapper()->getCurrent();
if ($this->view->webUser->isAuthenticated) {
     $this->view->customer = $this->view->webUser->getCustomer();
```

```
        }

        $this->view->ageGroups = $em->ageGroupMapper()->fetchCollection();

        $criteria = new Lkk_Model_Mapper_DbTable_Criteria();

        $criteria->addEquals("isPublic", 1);

        $this->view->categories = $em->categoryMapper()-
>fetchCollection($criteria);

        $criteria->addEquals("isDeleted", 0)

                  ->addOrder("rank")

                  ->setLimit(2);

        $this->view->topProducts = $em->productMapper()-
>fetchCollection($criteria);

        $criteria->clearAll()

                  ->addEquals("contentTag", "leftblock")

                  ->addEquals("langCode", $lang);

        $this->view->leftBlock = $em->editableContentMapper()-
>fetchSingle($criteria);

        $criteria->clearAll()

                  ->addEquals("isPublic", 1)

                  ->addEquals("isDeleted", 0)

                  ->addEquals("categoryId", $category);

        if (!is_numeric($agegroup)) {

             $this->view->products = $em->productMapper()-
>fetchCollection($criteria);

        } else {

             $criteria->addManyToManyJoinCondition("product", "ageGroup")

                       ->addEquals("id", $agegroup, "ageGroup");
```

```
          $this->view->products = $em->productMapper()-
>fetchCollection($criteria);

     }

     // jatkuu...
```

Lopuksi käsitellään näkymälle asetetun lomakkeen lähettäminen takaisin.

```
if ($this->getRequest()->isPost()) {



     $postData = $this->getRequest()->getPost();

     if (isset($postData["webuser_submitlogin"])) {

          if ($this->view->webUserForm->isValid($this->getRequest()-
>getPost())) {

               $userToLogin = new Entity_WebUser();

               $userName = (!empty($postData['webuser_login_username']))?
$postData['webuser_login_username']: ' ';

               $userToLogin->setUserName($userName)

                    >setPassword($postData['webuser_login_password']);

               $em->webUserMapper()->authenticate($userToLogin);

               $url = $this->view->url(array('controller' => 'index',
'action' => 'index', "lang" => $this->view->lang), null, true);

               header('Location: '.$url);

               exit;

          }

     }

}
```

6.7 Näkymien toteutus

Lastenkirjakaupan näkymät toteutettiin osittain samanaikaisesti ohjainluokkien toteutuksen kanssa. Toteutuksessa edettiin niin, että ensin projektin alkuvaiheessa määritelty ulkoasu taitettiin XTHML- ja CSS-mallinnuskielillä ja sen jälkeen siihen lisättiin dynaamiset osiot, joiden toiminnallisuus toteutettiin ohjainluokassa.

Kuten luvussa 4 todettiin, ovat Zend Frameworkin näkymät perinteisiä PHP-skriptejä. Ne koostuvat siis HTML-koodista, jonka sisään on upotettu PHP-koodia. Tämä antaa näkymille pääsyn koko sovelluksen toiminnallisuuteen, mikä merkitsee sitä, että näkymässä on mahdollista tehdä tietokantakyselyjä ja käsitellä entiteettejä aivan kuten ohjainluokassa. Tästä on hyötyä siinä mielessä, että se antaa sovellusohjelmoijalle liikkumavaraa, mutta toisaalta sen voidaan nähdä antavan näkymille likaa vapauksia. Tämä voi johtaa MVC -suunnittelumallin käytäntöjen rikkomiseen. On ohjelmoijasta itsestään kiinni pitäydytäänkö MVC:n käytännöissä ja missä määrin näin tehdään.

Seuraavassa käsitellään indeksiohjaimen indeksinäkymä Lastenkirjakaupan julkiselta puolelta ja toimitusehdot -sivun sisällönhallintanäkymä hallintapuolelta.

6.7.1 Indeksinäkymä

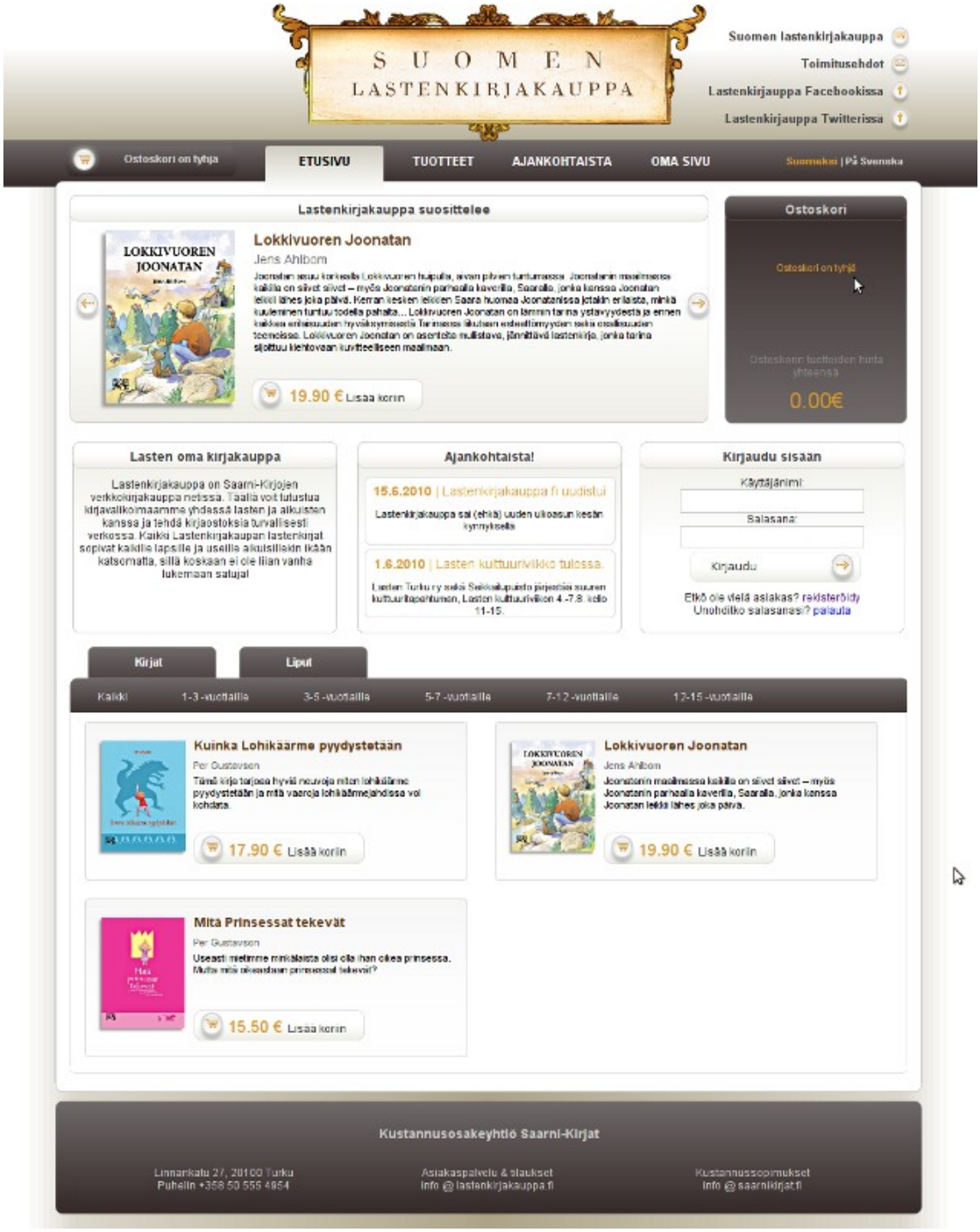
Lastenkirjakaupan Indeksinäkymä sisältää tuotteiden listauksen, asiakkaan sisäänkirjautumislomakkeen, blogin syötteen ja ostoskorin.

Blogin syöte ja ostoskori toteutettiin käyttäen Zend Frameworkin tarjoamaa Zend_View_Helper_Action-liitännäistä. Tämän liitännäisen avulla on mahdollista kutsua näkymästä toisen ohjaimen action-metodia ja tulostaa sen näkymä kutsuvan näkymän sisään. Tällöin ostoskorin sisällön ja blogin syötteen listaava elementti voidaan kutsua monessa näkymässä, eikä niitä tarvitse toteuttaa uudelleen jokaiselle näkymälle. Liitännäistä kutsutaan seuraavasti:

```
<?php echo $this->action('sidebar', 'cart'); ?>
```

Samoin kuin ostoskori ja blogin syöte, myös asiakkaan sisäänkirjautumislomake on uudelleenkäytettävä elementti. Sama Zend_Formin perivä lomakeolio voidaan tulostaa monessa näkymässä

Kuva 1 näyttää Lastenkirjakaupan indeksinäkymän sellaisena, kuin se tulostetaan silloin, kun asiakas ei ole kirjautunut sisään. Asiakkaan kirjauduttua korvataan kirjautumislomake tervehdystekstillä ja sivun yläreunan oikealle puolelle tulostetaan käyttäjän tiedot ja uloskirjautumislinkki.



Kuva 3: Lastenkirjakaupan indeksinäkymä

6.7.2 Toimitusehtojen sisällönhallintanäkymä

Lastenkirjakaupan hallintapuoli toteuttaa rajoitetun sisällönhallinnan. Hallinnasta käsin on mahdollista muokata toimitusehdot- ja suomen lastenkirjakauppa -sivujen sisältöä sekä rajoitetusti eräitä yksittäisiä sivuston elementtejä.

Sisällönhallintaa varten Lastenkirjakaupalle toteutettiin tekstinkäsittelykomponentti Javascriptillä. Komponentti käyttää hyväkseen nykyaikaisten selainten tarjoamaa execCommand-metodia, sekä mahdollisuutta muuttaa iframe-elementti muokattavaksi komennolla designMode = ”on”.

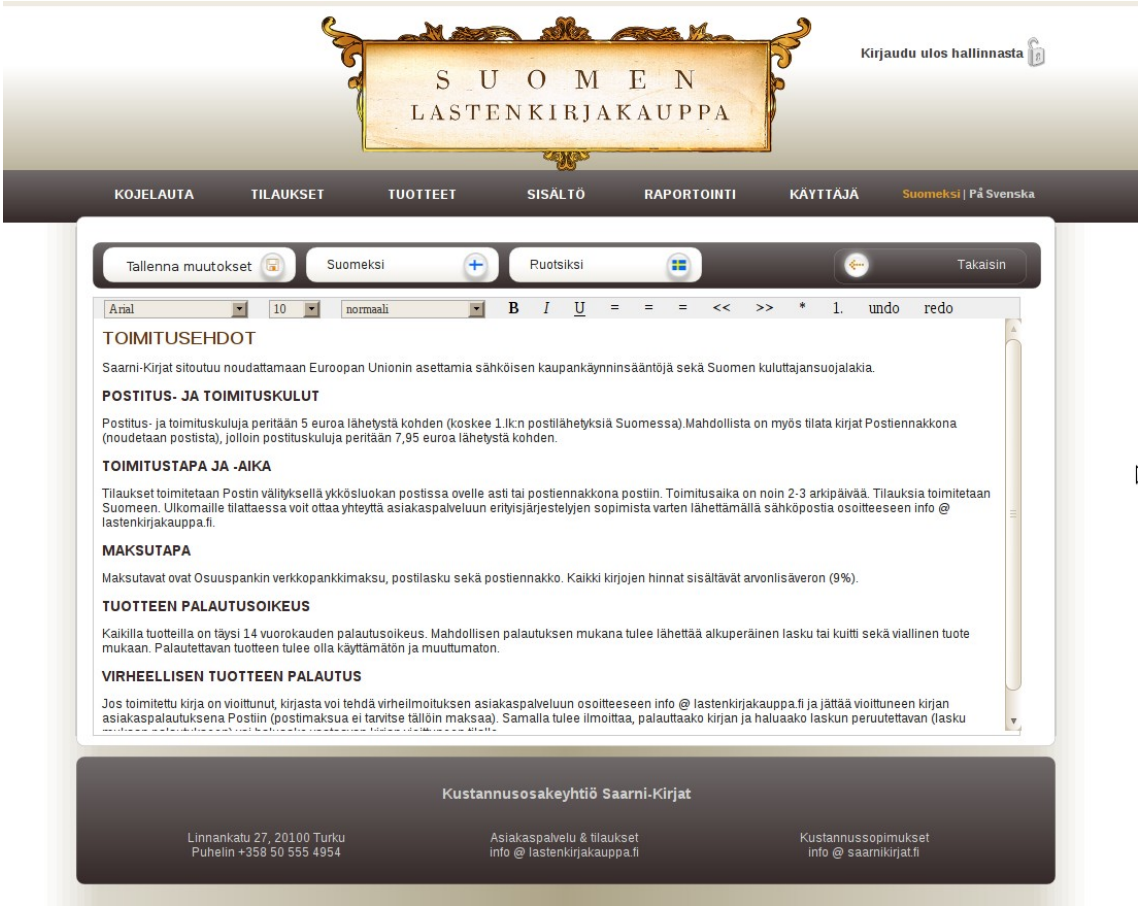
Komponentti toimii kopioimalla textarea-elementin sisällön ja syöttämällä sen muokattavaan iframe-elementtiin. Tämän jälkeen textarea-elementti piilotetaan ja iframen päälle luodaan työkalupalkki, joka sisältää tekstinkäsittelyssä tarvittavat työkalut. Näihin kuuluvat kirjasintyypin ja sen koon valinta, tyylin valinta, tekstin lihavointi, kursivointi ja alleviivaus, tekstin sisennys ja asettelu, listojen käyttö, sekä toiminnon peruutus ja palautus.

Valmis teksti tallennetaan siirtämällä iframe-elementin sisältämä HTML-koodi takaisin textarea-elementtiin ja tallentamalla sen sisältö tavalliseen tapaan tietokantaan lähettämällä lomake takaisin palvelimelle.

Tekstinkäsittelykomponentti otetaan käyttöön seuraavasti:

```
var editor = new rte({

          textareaid: 'contentdata',

          submitid: 'submit',

          stylesheeturl: '/css/editor.css',

          toolbar: 'defaults'

     });
```

Kuva 2 näyttää toimitusehdot-sivun sisällönhallintanäkymän.



Kuva 4: Sisällönhallinnan toimitusehdot-sivu

6.8 Sovelluksen käyttöönotto

Lastenkirjakauppa otettiin käyttöön marraskuussa 2010 siirtämällä se webbinen.fi-webhotellin julkiseen juurihakemistoon. Käyttöönoton yhteydessä tietokanta vaihdettiin SQLitesta MySQL-tietokantaan.

6.8.1 Käyttöönoton ongelmia

Verkkokauppasovelluksen käyttöönotossa ilmeni eräitä ongelmia, jotka lopulta ratkaistiin muuttamalla osia kaupan ORM-kehyksen koodista, sekä muuttamalla palvelimen asetuksia .htaccess-tiedoston avulla.

Eniten työtä aiheutti Webbinen.fi-webhotellin vanhentunut PHP-versio. Webhotellissa käytössä oleva versio ei tukenut PHP 5.3:ssa esiteltyjä uusia ominaisuuksia, joita Lastenkirjakaupassa käytettiin. Maagista metodia __callStatic ei tuettu, kuten ei myöskään PHP:n myöhäistä staattista sidontaa. Samaten niinkin yksinkertainen asia, kuin PHP:n sisäinen lcfirst-funktio puuttui Webbisen PHP-versiosta.

Nämä ongelmat ratkaistiin etsimällä kaupan koodista ne kohdat, jotka käyttivät kyseisiä ominaisuuksia ja muuttamalla niiden toiminnallisuus yhteensopivaksi vanhemman PHP-version kanssa.

Htaccess-tiedostoa editoitiin, jotta palvelimella oletuksena käytössä oleva tietoturva-asetus magic_quotes_gpc saatiin poistettua käytöstä. Asetus lisää sekä HTTP-POST- että HTTP-GET-pyyntöjen dataan sekä evästeiden sisältöön tilanvaihtosekvenssit automaattisesti. Tämä tuotti ongelmia tallennettaessa HTML-koodia, joka luonnollisesti sisältää paljon lainausmerkkejä. Lainausmerkit kuuluvat niihin merkkeihin, joihin magic_quotes_gpc-asetus lisää tilanvaihtosekvenssit.

Ylipäätään todettiin, että sovelluksen tietoturvasta on syytä huolehtia muulla tavalla kuin luottamalla palvelimen automaattisesti sotkemaan syötteeseen.

7 TUTKIMUKSEN TULOKSET

Tämän tutkimuksen tavoite oli selvittää mitä hyötyjä verkkosovellusohjelmoinnissa on saavutettavissa Zend Framework -kehysjärjestelmän käyttöönotolla ja mitä haasteita sen käyttöönotto asettaa ohjelmoijalle.

7.1 Zend Frameworkin edut

Lastenkirjakaupan kehityksen aikana todettiin Zend Frameworkin sisältävän lukuisia hyödyllisiä komponentteja, jotka säästävät sovellusohjelmoijan aikaa huomattavasti.

Kattavan valmiin komponenttikirjaston ohella Zend Framework tarjoaa kaksi huomattavaa etua. Näihin kuuluvat koodin autonomia ja sen helppo jatkokehitettävyys.

7.1.1 Koodin autonomia

Zend Frameworkilla tuotetun koodin voidaan sanoa olevan autonomista kahdellakin tavalla.

Esinnäkin koodi ei ole sidottu sen kehittäjään, sillä kuka tahansa kehysjärjestelmän toiminnallisuuden tunteva sovellusohjelmoija voi lyhyen tutustumisen jälkeen ymmärtää suurimman osan sovelluksen toiminnasta.

Toisaalta Zend Frameworkin koodi on pitkälti eriytettyä, mikä tarkoittaa sitä, että mikään komponentti ei ole välttämättä riippuvainen toisesta. Tällöin on mahdollista työskennellä samanaikaisesti esimerkiksi näkymien ja mallien kanssa, ilman että kumpikaan sovelluskerros kärsii.

7.1.2 Jatkokehitettävyys

Sovelluksen helppo jatkokehitettävyys on suoraan seurausta koodin autonomiasta. Kun sovelluksen toiminnallisuus on toteutettu tiettyjen rajojen sisällä, on sen jatkokehitys huomattavasti helpompaa.

7.2 Zend Frameworkin asettamat haasteet

Zend Framework on ideaalinen kehysjärjestelmä opiskelun alustaksi, sillä ohjelmoijan on mahdollista käyttää siitä vain tarvitsemiaan komponentteja.

Kehysjärjestelmä sallii ohjelmoijan perehtyä vaativampaan toiminnallisuuteen omaan tahtiinsa ja sisällyttää uusia luokkia sovellukseensa niihin tutustuttuaan. Tällainen joustava rakenne on omiaan vähentämään kehysjärjestelmän käyttöönoton haasteita.

On silti sanottava, että Lastenkirjakaupan toteutuksen aikana eräät Zend Frameworkin komponentit aiheuttivat käyttöön otettaessa enemmän päänvaivaa kuin toiset. Tällainen komponentti on mm. Zend_Form, jonka täydellinen hallitseminen vie aikaa. Niin ikään Zend Frameworkin HTTP-pyynnön käsittelyn täydellinen ymmärtäminen vaatii opiskelua.

Näiden haasteiden selättäminen ei tosin ole perusedellytys kehysjärjestelmän käytölle. Zend Frameworkin perustoiminnallisuutta on mahdollista käyttää hyvin vähäisellä tuntemuksella. Se ei niinkään pakota sovellusohjelmoijaa opiskelemaan kattavaa luokkakirjastoa, kuin tarjoaa valmiita ratkaisuja silloin, kun sellaisia tarvitaan.

7.3 Zend Framework verkkokaupan alustana

Lastenkirjakaupan kehityksen aikana tultiin siihen tulokseen, että Zend Framework sopii erinomaisesti verkkokaupan toteutusalustaksi.

Kehysjärjestelmän tarjoamat syötteen validointiin ja suodattamiseen tarkoitetut komponentit sopivat verkkokaupan tarpeisiin hyvin.

8 JOHTOPÄÄTÖKSET

8.1 Kriittinen arviointi

Pohdittaessa Lastenkirjakaupan toteutusta jälkeenpäin, eräs asia edellyttää kriittistä tarkastelua. Tämä asia on Lastenkirjakaupan ORM-kehyksen toteutus.

Eräs Zend Frameworkin huomattavimmista eduista on koodin samankaltaisuus sovelluksesta toiseen. Oman ORM-kehyksen toteuttaminen ikään kuin kumoaa kehysjärjestelmän rakenteellisuudesta saatavan hyödyn muuttamalla osan sovelluksesta täysin vieraaksi ulkopuoliselle tarkastelijalle. Se edellyttää siten huomattavasti enemmän perehtymistä kuin Zend Frameworkin oletusluokkien käyttö.

Tämä ongelma on mahdollista ratkaista kahdella tapaa. Ensimmäinen vaihtoehto on pysyttäytyä yksinomaan Zend Frameworkin tarjoaman mallitoiminnallisuuden piirissä ja hyväksyä sen mukanaan tuoma tarpeeton saman koodin toisto. Toinen vaihtoehto on ottaa käyttöön jokin kolmannen osapuolen ORM-kehys, kuten Doctrine. Tällöin on mahdollista pitää kiinni kolmannen osapuolen kirjaston luomasta rakenteellisuudesta ja silti käyttää kattavaa ORM-kehystä.

8.2 Tutkimuksen hyödynnettävyys

Tätä tutkimusta on sen itsestään selvän hyödyn, eli Lastenkirjakauppa-verkkokaupan ohella mahdollista hyödyntää Zend Frameworkin opiskelun lähteenä. Se saattaa sisältää joitain näkemyksiä, jotka ovat hyödyllisiä kehysjärjestelmään vasta tutustuvalle.

Toisaalta tutkimus sisältää kattavan ORM-kehyksen toteutuksen, jolloin siitä voidaan nähdä olevan hyötyä myös ohjeena tai vertailukohtana Zend Frameworkin ORM-kehyksen toteutuksessa.

LÄHTEET

Allen, R.; Brown, S &. Lo, N. 2008. Zend Framework in action. Greenwich:Manning.

Achour, M.; Betz, F.; Dovgal, A.; Lopes, N.; Magnusson, H.; Richter, G. & Seguy, D. & Vrana J. 2010a. PHP Manual. What is PHP?. the PHP Documentation Group. Viitattu 25.9.2010 http://www.php.net/manual/en/intro-whatis.php.

Achour, M.; Betz, F.; Dovgal, A.; Lopes, N.; Magnusson, H.; Richter, G. & Seguy, D. & Vrana J. 2010b. PHP Manual. What can PHP do?. the PHP Documentation Group. Viitattu 25.9.2010 http://www.php.net/manual/en/intro-whatcando.php.

Achour, M.; Betz, F.; Dovgal, A.; Lopes, N.; Magnusson, H.; Richter, G. & Seguy, D. & Vrana J. 2010c. PHP Manual. Classes and Objects. Magic Methods. the PHP Documentation Group. Viitattu 25.9.2010 http:/php.net/manual/en/language.oop5.magic.php.

Achour, M.; Betz, F.; Dovgal, A.; Lopes, N.; Magnusson, H.; Richter, G. & Seguy, D. & Vrana J. 2010d. PHP Manual. Classes and Objects. Constructors and Destructors. the PHP Documentation Group. Viitattu 25.9.2010 http://php.net/manual/en/language.oop5.decon.php.

Achour, M.; Betz, F.; Dovgal, A.; Lopes, N.; Magnusson, H.; Richter, G. & Seguy, D. & Vrana J. 2010e. PHP Manual. Classes and Objects. Overloading. the PHP Documentation Group. Viitattu 25.9.2010 http://php.net/manual/en/language.oop5.overloading.php.

Achour, M.; Betz, F.; Dovgal, A.; Lopes, N.; Magnusson, H.; Richter, G. & Seguy, D. & Vrana J. 2010f. PHP Manual. Namespaces. Namespace overview. the PHP Documentation Group. Viitattu 25.9.2010 http://fi2.php.net/manual/en/language.namespaces.rationale.php.

Achour, M.; Betz, F.; Dovgal, A.; Lopes, N.; Magnusson, H.; Richter, G. & Seguy, D. & Vrana J. 2010g. PHP Manual. Classes and Objects. Objects and references. the PHP Documentation Group. Viitattu 25.9.2010 http://php.net/manual/en/language.oop5.references.php.

Gilmore, J. W. 2008. Beginning PHP and MySQL: From Novice to Professional. Third Edition. New York:Appress.

Lecky-Thompson, E.; Myer, T. & Nowicki, S. D. 2009. Professional PHP6. Indianapolis: Wiley Publishing, Inc.

Lyman, F. 2009. Pro Zend Framework Techniques. New York:Appress.

McArthur, K. 2008. Pro PHP: Patterns, Frameworks, Testing and More. New York:Appress.

Thomson, L. & Welling, L. 2009. PHP and MySQL Web Development. Fourth Edition. Massachusetts:Addison-Wesley.

Ullman, L. 2009. PHP For the Web. Third Edition. Berkeley: Peachpit Press

Zandstra, M. 2008. PHP Objects, Patterns, and Practice. Second Edition. New York:Appress.

Zend Technologies, 2010a. Frequently Asked Questions. Viitattu 15.10.2010 http://framework.zend.com/about/faq.

Zend Technologies, 2010b. Zend Framework manual. Zend Form. Viitattu 15.10.2010 http://framework.zend.com/manual/1.11/en/zend.validate.introduction.html.

Zend Technologies, 2010c. About Zend Framework. Viitattu 21.11.2010 http://framework.zend.com/about/overview.

Zend Technologies, 2010d. Zend Framework manual. Zend Controller Basics. Viitattu 21.11.2010 http://framework.zend.com/manual/en/zend.controller.basics.html.

Zend Technologies, 2010e. Zend Framework manual. Action Controllers. Viitattu 21.11.2010 http://framework.zend.com/manual/en/zend.controller.action.html

Zend Technologies, 2010f. Zend Framework manual. Plugins. Viitattu 21.11.2010 http://framework.zend.com/manual/en/zend.controller.plugins.html

Zend Technologies, 2010g. Zend Framework manual. View Scripts. Viitattu 21.11.2010 http://framework.zend.com/manual/en/zend.view.scripts.html

Zend Technologies, 2010h. Zend Framework manual. Zend_Db_Adapter. Viitattu 21.11.2010 http://framework.zend.com/manual/en/zend.db.adapter.html