Tampereen Ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Miika Koskela

Opinnäytetyö

WWW-ohjelmistokehys - esittelyssä Zend Framework

Työn ohjaaja lehtori Petri Heliniemi
Tampere 06/2010

Tekijä	Miika Koskela
Työ	WWW-ohjelmistokehys – esittelyssä Zend Framework
Sivumäärä	40
Valmistumisaika	kesäkuu 2010
Työn ohjaaja	lehtori Petri Heliniemi

TIIVISTELMÄ

Tässä opinnäytetyössä tarkastellaan Zend Framework -ohjelmistokehystä korkeammalta tasolta. Työssä ei käydä läpi eri komponentteja yksityiskohtaisesti, vaan perehdytään siihen, miten tämä ohjelmistokehys oikeastaan toimii.

Vaikka Zend Frameworkin komponentit ovat helppokäyttöisiä ja dokumentointi melko hyvä, laajan sovelluskokonaisuuden rakentamiseen tarvitaan kuitenkin ohjelmistokehyksen arkkitehtuurin ja toimintaperiaatteiden hyvää tuntemusta.

Aluksi tarkastellaan ohjelmistokehystä lyhyeksi, jonka jälkeen esitellään varsinainen työn aihe. Ohjelmistokehyksen esittelyn jälkeen rakennetaan esimerkkisovellus, jossa hyödynnetään opittuja tietoja ja taitoja.

Avainsanat PHP, Zend Framework, ohjelmistokehys, arkkitehtuuri

Author	Miika Koskela
Thesis	Web application framework – introducing Zend Framework
Pages	40
Graduation time	June 2010
Thesis Supervisor	lecturer Petri Heliniemi

ABSTRACT

The purpose of this thesis is to take a look into Zend Framework from a higher perspective than just describing the different components in detail. During the work, we will see how this web application framework actually works.

The components are easy to use and they are documented relatively well. To build a complete web application, however, a lot more knowledge is required about the architecture of the framework.

First I will introduce briefly what the framework is all about, and then dive into the real subject of this thesis. After introducing the framework, I will build a small example application where learned things are put into practice.

Keywords PHP, Zend Framework, software framework, architecture

Sisällys

Käsite- ja lyhenneluettelo

API

API on ohjelmointirajapinta, joka on kahden eri sovelluksen, kuten käyttöjärjestelmän ja jonkin muun sovelluksen välillä. Ohjelmointirajapinna tarkoituksena on helpottaa ohjelmoijan työtä. (Tietotekniikan termitalkoot 2001)

MVC

MVC tulee englanninkielisistä sanoista Model-View-Controller. Se on alunperin Trygve Reenskaugin kehittämä arkkitehtuurimalli, jonka tarkoituksena on eriyttää liiketoimintalogiikka käyttöliittymästä. Kun sovellus on tällä tavoin jaoteltu, näitä yksittäisiä osa-alueita voidaan kehittää, ylläpitää ja testata toisistaan riippumatta. (Wikipedia 2010a)

Nimiavaruus (engl. namespace)

Nimiavaruus on eräänlainen tapa kapseloida tietoa. Esimerkiksi tietokoneen kovalevyllä voi olla useita samannimisiä tiedostoja samanaikaisesti, mutta samassa hakemistossa taas ei. PHP-ohjelmointikielessä nimiavaruudet ovat olleet saatavilla vasta versiosta 5.3.0 lähtien. (PHP Manual 2010a)

Palvelukerros (engl. service layer)

Palvelukerros toimii rajapintana käyttöliittymän ja sovelluksen liiketoimintalogiikan välillä.(Stafford 2010)

Perintä (engl. extending)

Perinällä tarkoitetaan sitä, kun jokin luokka perii kantaluokan, niin perivä luokka sisältää kaikki metodit ja ominaisuudet, joita kantaluokka sisältää. PHP-ohjelmointikieli ei tue samanaikaista perimistä useammalta eri kantaluokalta. (PHP Manual 2010b)

SQL

SQL tulee englanninkielisistä sanoista Structured Qurey Langauge. Se on kieli, jolla voidaan tehdä kyselyitä tietokantaan. Näillä kyselyillä haetaan, muokataan tai lisätään tietoa tietokantaan. (Wikipedia 2010b)

1 Johdanto

Opinnäytetyön aihetta miettiessä oli jo erään ystävän kanssa ollut puhetta oman yrityksen perustamisesta. Ajatuksena yrityksen perustamisen taustalla oli hyödyntää omaa osaamista digitaalisen median palveluiden tuottajana. Yksi ideoinnin yhteydessä esiin tullut asia oli uudenlaisen asiakasjärjestelmän tarve. Se olisi kuitenkin ollut aivan liian suuri urakka, joten päädyin rajaamaan opinnäytetyöni aiheen koskemaan Zend Framework -ohjelmistokehystä, jota käyttäen kyseinen järjestelmä rakennetaan.

Ohjelmistokehyksen komponentit ovat suhteellisen helppoja käyttää, ja niistä löytyy melko kattava dokumentaatio. Ohjelmistokehys ei kuitenkaan ole vain kokoelma komponentteja, joita käyttämällä saa helposti hienon sovelluksen aikaan. Kehys sisältää erilaisia arkkitehtonisia ratkaisuja, joiden ymmärtäminen on välttämätöntä, jotta voidaan ohjelmoida toimivia, helposti laajennettavissa ja ylläpidettävissä olevia sovelluksia. Ennen varsinaisen sovelluksen rakentamista on siis opeteltava ohjelmistokehyksen toimintaperiaatteita.

Tämän opinnäytetyön tavoitteena on tutustua Zend Frameworkilla luotavaan WWW-sovellukseen kokonaisuutena, jossa ei mennä eri komponenttien yksityiskohtaiseen esittelyyn. Työssä tarkasteltaviin aiheisiin kuuluu mm. *etuohjaimen liitännäiset* sekä modulaarisen sovelluksen fyysinen ja looginen rakenne. Lopuksi rakennetaan esimerkkisovellus, jossa pyritään hyödyntämään näitä tietoja ja taitoja. Valmiin esimerkkisovelluksen lähdekoodi löytyy liitteenä olevalta CD-levyltä (Liite 3), ja sovellusta kuvataan ruudunkaappauksin liitteessä 1.

Työstä saan itselleni tarvittavia työkaluja tulevaisuuden työelämää ajatellen. Olen aiemmin ollut mukana vain yhdessä laajemmassa ohjelmistoprojektissa, ja siitäkin on aikaa jo yli 5 vuotta. Minulla ei siis ole kovin paljoa kokemusta todellisesta ohjelmointityöstä.

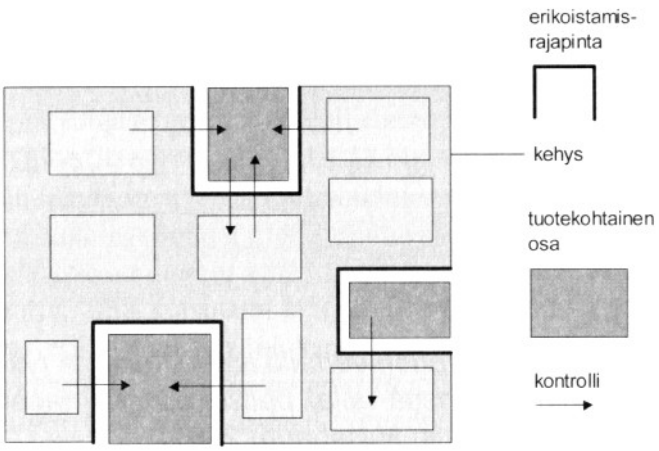
Työn ymmärtämiseksi lukijalta edellytetään ymmärtämystä olio-ohjelmoinnista, sekä hieman ohjelmistokehysten yleistä tuntemusta.

2 Ohjelmistokehys lyhyesti

Ohjelmistokehyksen idea ei ole riippuvainen mistään tietystä ohjelmointiparadigmasta. Zend Framework on täysin oliopohjainen ohjelmistokehys. (Koskimies & Mikkonen 2005, 187; Zend Framework 2010a.) Tästä syystä tässä luvussa tarkastellaankin ohjelmistokehystä oliopohjaisesta näkökulmasta.

Ohjelmistokehys on luokista, komponenteista ja/tai rajapinnoista koostuva ohjelmistorunko, jota täydentämällä luodaan uusia sovelluksia. Tällaista runkoa täydennetään omalla ohjelmakoodilla, joka liitetään ohjelmistokehykseen erityisten liitoskohtien kautta, joita kutsutaan *erikoistamisrajapinnoiksi*. Näissä rajapinnoissa käytetään yleensä tekniikkana periytymistä, jossa ohjelmoijan luoma luokka perii ohjelmistokehyksestä löytyvän kantaluokan. (Koskimies & Mikkonen 2005, 187, 189, 191.) Esimerkiksi tässä työssä esiteltävän Zend Frameworkin *toiminto-ohjaimet* perivät ohjelmistokehyksessä sijaitsevan Zend_Controller_Action-kantaluokan (Zend Framework 2010b).

Näitä ohjelmoijan luomia luokkia kutsutaan sovelluskokonaisuuden *tuotekohtaisiksi* osiksi (Koskimies & Mikkonen 2005, 189). Kuvassa 1 nämä osat ovat ilmaistu tummemmalla harmaalla kuin ohjelmistokehys. Kuvasta näkyy myös, miten nämä tuotekohtaiset osat ovat liitetyt ohjelmistokehykseen erikoistamisrajapintojen kautta.



Kuva 1: Ohjelmistokehyksen tarjoama sovellusrunko tuotekohtaisine osineen (Koskimies & Mikkonen 2005, 189)

Ohjelmistokehyksen päätarkoituksena on ohjelmakoodin uudelleenkäyttö. Sen lisäksi se tehostaa sovelluskehitystä ja tarjoaa sovellukselle valmiin arkkitehtuurin. (Koskimies & Mikkonen 2005, 187, Baker 2009.) Uudelleenkäytöllä tarkoitetaan yksinkertaisesti sitä,

että kun jokin tietty, yleinen osa sovelluksesta on jo kerran rakennettu, ei sitä tarvitse enää uudelleen rakentaa. Kun tällaisia yleisiä osia rakennetaan ja kootaan yhteen tarpeeksi, saadaan aikaiseksi *ohjelmistokirjasto*, jota voidaan käyttää monissa eri sovellusprojekteissa.

Ohjelmistokehystä ei kuitenkaan tule sekoittaa ohjelmistokirjastoon. Ohjelmistokehyksen ja ohjelmistokirjaston ero on siinä, että ohjelmistokehys ohjaa koko sovelluksen toimintaa: se huolehtii ohjelmoijan luoman koodin suorittamisesta automaattisesti. Ohjelmistokirjastoa käyttäessään ohjelmoija joutuu itse luomaan oliot ja kutsumaan niiden metodeja. Tällaista ominaisuutta kutsutaan käänteiseksi kontrolliksi (engl. *inversion of control*). (Baker, 2009.) Ohjelmistokirjastosta ei myöskään saada valmista sovellusta, vaikka sitä täydennettäisiin omalla ohjelmakoodilla.

Esimerkkinä siitä, miten ohjelmistokehys hallitsee sovelluksen suoritusprosessia, voidaan ajatella Zend Frameworkin toiminto-ohjainta. Vaikka se on ohjelmoijan luomaa ohjelmakoodia, ei siitä kuitenkaan tarvitse erikseen luoda oliota ja kutsua sen metodia, vaan ohjelmistokehys huolehtii näistä toimenpiteistä automaattisesti. Kun käyttäjä kertoo sovellukselle, minkä toiminnon hän haluaa suorittaa, etsii kehyksen ohjelmakoodi automaattisesti halutun luokan, luo siitä olion ja suorittaa pyydetyn toiminnon.

Myös sovelluksen perustoiminnallisuus on valmiina ohjelmistokehyksessä, jonka tähden voidaan keskittyä korkeamman tason ongelmien ratkaisemiseen (Baker 2009). Ohjelmoijan ei esimerkiksi tarvitse erikseen huolehtia tietokantayhteyden avaamisesta, SQL-kyselyn kirjoittamisesta ja tietokantayhteyden sulkemisesta, vaan ohjelmistokehykseen sijoitettu perustoiminnallisuus huolehtii tällaisista asioista.

Toisena esimerkkinä voidaan ajatella graafiseen sovelluksen käyttöliittymäkomponentteja sisältävää ohjelmistokehystä. Bakerin esimerkissään rakentama ”Hello World” -ohjelma mahtui muutamiin riveihin; kuinka paljon enemmän olisi tarvittu ohjelmakoodia, jos tuo sovellus olisi rakennettu aivan alusta alkaen itse? Ikkunan koon muuttaminen, ikkunan sulkeminen ja käyttäjän syötteisiin vastaaminen, joista ohjelmistokehys nyt huolehtii automaattisesti. (Baker 2009.)

Ohjelmistokehyksen rakentaminen on kuitenkin aikaa vievää ja kallista työtä (Koskimies & Mikkonen 2005, 188). Nykyään on tarjolla ilmaiseksi monia valmiita ohjelmistokehyksiä, joten on syytä tarkistaa, löytyykö omiin tarpeisiin jo valmista kehystä.

Toisaalta valmiin ohjelmistokehyksen käyttöönottokin vie oman aikansa (Koskimies & Mikkonen 2005, 188). Kehyksen tehokas käyttö vaatii sen hyvää tuntemusta, eikä oppiminen ole nopea toimenpide. Ohjelmistokehyksen käyttöä opetellessa kuitenkin oppii sen arkkitehtuurista ja erilaisista suunnittelumalleista. Näitä tietoja ja taitoja voi hyödyntää myöhemmin itse muissa projekteissa.

3 Zend Framework -ohjelmistokehys

PHP-ohjelmointikieltä käytettiin alun perin suoraan WWW-sivun muodostavan HTML-merkintäkielen seassa. Tällä tavoin pystyttiin kehittämään WWW-sovelluksia erittäin nopeasti, mutta sovellusten laajuuden kasvaessa niistä tuli kuitenkin vaikeasti ylläpidettäviä ja laajennettavia. Tähän ongelmaan kehitettiin ratkaisuksi mm. Zend Framework. (Allen, Lo & Brown 2008, 3-4.)

Zend Framework on PHP-ohjelmointikielellä toteutettu ohjelmistokehys. Se julkistettiin ensimmäistä kertaa vuoden 2005 Zend-konferenssissa, ja versio 1.0 julkaistiin lähes kaksi vuotta myöhemmin, heinäkuussa 2007 (Wikipedia 2010c). Tätä työtä kirjoitettaessa siirryttiin versiosta 1.9 versioon 1.10.

Zend Framework sisältää valtavan määrän erilaisia uudelleenkäytettäviä komponentteja. Näillä komponenteilla voidaan esimerkiksi lähettää sähköpostia sekä luoda PDF-tiedostoja ja hakea tietoja esimerkiksi Googlen palveluista. Lisäksi se sisältää MVC-arkkitehtuurimallin ohjain- ja näkymäkomponentit. Allen ym. väittävät myös, että Zend_Db ja Zend_Services-komponentit muodostavat malli-komponentin. (Allen ym. 2008, 14, 10.) Näin ei silti ole, vaan nämä mainitut Zend_-kompoentit tarjoavat vain työkalut mallin rakentamiseen (ks. perustelut luvusta 4.1).

Komponentit ovat löyhästi sidotut (engl. *loosely coupled*) toisiinsa, jolloin niitä voidaan käyttää erikseen myös muiden ohjelmistokehysten kanssa tai jopa liittää olemassa olevaan projektiin (Allen ym. 2008, 6).

Komponentit sisältävät kuitenkin jonkin verran eriasteisia riippuvuuksia sekä toisistaan että PHP-tulkin ominaisuuksista. Riippuvuudet PHP-tulkin ominaisuuksista ovat tyypiltään *pehmeitä* tai *kovia*. Pehmeällä riippuvuudella tarkoitetaan sitä, että vaikka jokin vaatimus ei täyttyisikään, kykenee toiminto kuitenkin tuottamaan halutun lopputuloksen. Kova riippuvuus taas tarkoittaa sitä, että ilman vaatimuksen täyttymistä ei kyseistä toimintoa voida käyttää. (Zend Framework 2010c.)

Puhuttaessa eri komponenttien välisistä riippuvuuksista, tyypiltään pehmeä riippuvuus saa erilaisen merkityksen: vaatimuksena olevaa komponenttia voidaan tarvita erityistilanteissa tai jonkin erityisen sovittimen kanssa. Lisäksi mukaan tulee vielä kaksi erityyppistä riippuvuutta: *fix* ja *sub*. Fix-tyyppinen riippuvuus tarkoittaa, että riippuvuu-

deksi ilmoitettua komponenttia käytetään kaikissa tilanteissa. Sub-tyyppinen riippuvuus taas tarkoittaa, että riippuvuudeksi ilmoitettua komponenttia käytetään vain erityistilanteissa tai erityisen sovittimen kanssa. (Zend Framework 2010c.)

3.1 Materiaalit

Internetistä löytyy monia eri tavoin aiheeseen liittyviä sivustoja ja blogeja. Paras paikka aloittaa tutustuminen on kuitenkin Zend Frameworkin omalta kotisivulta löytyvä opas. Tämä englanninkielinen opas on käännetty myös muutamalle muulle kielelle, kuten saksaksi ja ranskaksi. Lisäksi manuaalin kielivalikosta on löydettävissä venäjä, kiina ja japani, mutta nämä käännökset näyttivät kuitenkin olevan pahasti keskeneräisiä.

Oppaan ohessa on pika-aloitusopas, jonka yhteydessä rakennettavan sovelluksen lähdekoodi on vapaasti ladattavissa. Tämä sovellus on yksinkertainen vieraskirja, jossa merkintöjä luetaan ja tallennetaan tietokantaan WWW-käyttöliittymän kautta. Vieraskirjasovellus käyttää hyväkseen myös Zend_Layout-komponenttia, jonka avulla sovellukselle saadaan yhtenäinen sivupohja.

Tekstimuodossa oleva materiaali teknisine yksityiskohtineen ei ole aina mielenkiintoisinta luettavaa. Tähän tarjoaa apunsa Zendcasts-sivusto, josta löytyy kymmeniä videomuodossa olevia oppaita eri aihepiireistä. Videot ovat ilmaiseksi katsottavissa, ja sivustolla olevalla foorumilla voi keskustella videon sisällöstä, kysyä neuvoja tai jopa ehdottaa jotakin itseä kiinnostavaa aihepiiriä käsiteltäväksi tulevissa videoissa.

Zendcasts-sivusto on huomattu jopa Zend Frameworkin Internet-sivustolla, mikä antaa viitteitä sivuston luotettavuudesta ja käyttökelpoisuudesta. Kaikkein vanhimmat sivustolta löytyvät videot eivät kuitenkaan ole enää ajantasaisia. Esimerkiksi joulukuun 20. päivänä 2008 lisätty video esilataustiedoston luomisesta ei tänä päivänä enää ole käyttökelpoinen. On siis syytä tarkistaa vanhimmilta videoilta löytyvän materiaalin ajantasaisuus. Vaikka sivustosta on paljon apua, se vielä pieni, eikä niin aktiivinen.

Zend Frameworkista on myös API-dokumentaatio sitä tarvitseville. Tämä dokumentaatio on luettavissa verkossa, ja halutessaan sen voi myös ladata omalle työasemalleen. Nykyisen version lisäksi sekä API-dokumentaatio että opas ovat saatavilla myös ohjelmistokehyksen aiempiin versioihin.

Zend Frameworkista on painettu myös kirjoja. Painotuotteiden ongelma on kuitenkin kehityksen nopea eteneminen; teksti on kirjoitettu jostakin tietystä ohjelmistokehyksen versiosta. Uuden version sekä uusien ominaisuuksien tullessa teos on jo vanha. Kaikki kirjan tieto ei kuitenkaan vanhene heti, vaan lukijan on osattava poimia käyttökelpoinen tieto vanhan tiedon joukosta. Myös hyviä periaatteita voi painetusta tekstistä löytää. Aivan käyttökelvottomia ne eivät siis ole.

Monet muut Internetistä löytyvät blogit ja foorumit voivat olla hyvinkin asiallisia ja hyödyllisiä, mutta niiden luotettavuudesta ei silti ole takeita. Lukemalleen materiaalille kannattaakin etsiä vahvistusta myös muilta, luotettavammilta sivuilta.

Wikipediaa olen käyttänyt vain yleisluontoisiin tai historiallisiin tietoihin. Tämän lähteen vahvuus, sekä myös heikkous on maailmanlaajuinen joukko ylläpitäjiä. Wikipediaa ei pitäisi käyttää suurta tarkkuutta vaativiin tärkeisiin asioihin, vaan pikemminkin yleisluontoisena tietosanakirjana, josta on mahdollista löytää lisäluettavaa lähdeviitteiden kautta.

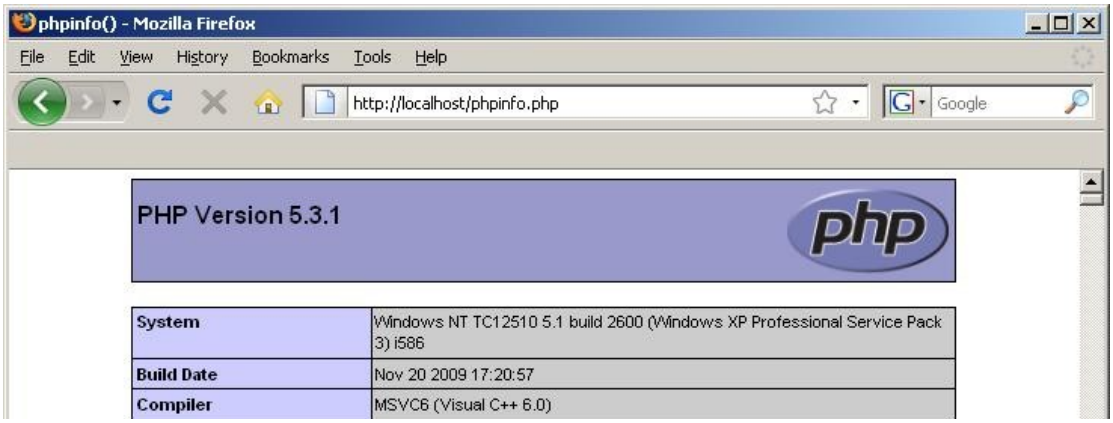
3.2 Järjestelmävaatimukset

WWW-sovellusta rakennettaessa tarvitaan PHP-tulkin lisäksi myös WWW-palvelin. PHP-tulkin tulee olla versio 5.2.4 tai sitä uudempi, mutta WWW-palvelimen vaatimuksia ei ole erikseen määritelty (Zend Framework 2010c). Palvelinsovellukseksi käy esimerkiksi laajalti tunnettu Apache tai Microsoftin IIS.

Muita pakollisia vaatimuksia ei varsinaisesti ole, mutta jotkut komponentit vaativat erilaisten PHP-tulkin liitännäisten tai WWW-palvelimen omaisuuksien olevan käytössä. Esimerkiksi jos halutaan hakukoneystävälliset osoitteet, tulee WWW-palvelimen tukea osoitteiden uudelleenkirjoitusta. Tähän tarkoitukseen soveltuu esimerkiksi Apachen *rewrite*-moduuli. (Zend Framework 2010c.)

Vaatimusten täyttyminen voidaan tarkistaa PHP-ohjelmointikielen *phpinfo*-funktiolla. Se muodostaa WWW-sivun, jossa tiedot esitetään selkeästi taulukkomuodossa. Kuvassa 2 on esimerkki tällaisesta sivusta. Sivulta käy ilmi myös muita käytössä olevan järjestelmän tietoja, kuten esimerkiksi käyttöjärjestelmä ja tietokoneen nimi.

Samalta sivulta selviää myös *php.ini*-tiedoston sijainti. Tähän tiedostoon tehdään tarvittavat asetusmuutokset, jonka jälkeen WWW-palvelin on käynnistettävä uudelleen. Uudelleenkäynnistys tarvitaan, jotta tehdyt asetusmuutokset tulevat voimaan.



Kuva 2: Osa phpinfo-funktion tuottamasta WWW-sivusta

3.3 Kehitysympäristön asetukset

Jotta ohjelmistokehystä ei tarvitsisi kopioida erikseen jokaiseen projektiin, tulee lähdekoodin sisältävä hakemisto kopioida sellaiseen paikkaan, josta PHP-tulkki sen tarvittaessa löytää. Hakemistopolku määritetään PHP-tulkin *include_path*-asetuksella, joka voidaan asettaa koko palvelimen laajuisesti php.ini-tiedostosta (PHP Manual 2010c). Include_path-muuttujan sisältämät hakemistopolut voi varmistaa myös kuvan 1 mukaiselta sivulta.

Ohjelmoitaessa tulee usein erilaisia virheitä, jollaisia ovat esimerkiksi lopetussulkeen tai puolipisteen puuttuminen ja tietokantayhteyden epäonnistuminen. Virheitä on huomattavasti helpompi etsiä ja korjata, jos ne tulostuvat ohjelmoijan näyttöruudulle tyhjän sivun sijaan. Tätä kontrolloidaan php.ini-tiedostosta asetettavalla *display_errors*-asetuksella (PHP Manual 2010d). Tuotantoympäristössä tämä asetus otetaan kuitenkin pois päältä, jotta mahdollisen virhetilanteen sattuessa käyttäjälle ei paljasteta tietoa, joka voisi aiheuttaa tietoturvariskin.

Sovelluskehityksen helpottamiseksi Zend Frameworkissa on komentorivityökalu, jolla voidaan esimerkiksi luoda sovellusrunko tai lisätä sovellukseen ohjaimia. Tämä työkalu on nimeltään Zend CLI Tool (tästä eteenpäin Zend-työkalu). Se tarvitsee toimiakseen tiedon kahdesta eri hakemistopolusta: komentoriviltä ajettavan PHP-tulkin sijainnista ja varsinaisen ohjelmistokehyksen sijainnista.

PHP-tulkin sijainti määritetään sekä Windows- että Unix-ympäristössä *PATH*-ympäristömuuttujan avulla. Tähän samaan muuttujaan lisätään myös polku, jossa työkalu itse sijaitsee, jolloin se on käytettävissä mistä tahansa hakemistopolusta.

Ohjelmistokehyksen sijainti kerrotaan Zend-työkalulle ympäristömuuttujalla, jonka nimi on *ZEND_TOOL_INCLUDE_PATH*.

3.4 Ohjelmistokehyksen asennus

Zend Framework -ohjelmistokehyksen voi hankkia osoitteesta http://framework.zend.com/download/latest. Sivulta löytyy useita eri latausvaihtoehtoja, joista osa vaatii rekisteröitymisen. Kun tarvitaan vain pelkkä ohjelmistokehys, valitaan *minimal*-paketti, joka sisältää vain ohjelmistokehyksen perustiedostot. Puretusta Zip tai Gzip paketista löytyy *bin*- ja *library*-hakemistot. Komentorivityökalun tiedostot sijaitsevat bin-hakemistossa, ja ohjelmistokehyksen lähdekoodi sijaitsee library-hakemiston Zend-alihakemistossa.

Zend Framework asennetaan kopioimalla Zend-hakemisto sellaiseen paikkaan, joka löytyy PHP-tulkin include_path-asetuksesta. Kun hakemisto on kopioitu, asennusta voidaan testata esimerkiksi käyttämällä jotakin ohjelmistokehyksen tarjoamaa komponenttia. Lyhyen testiohjelman voi tehdä esimerkiksi koodiesimerkin 1 kuvaamalla tavalla.

```php
<?php
/**
 * Koodinpätkä asennuksen testaamiseksi
 */
require_once 'Zend/Feed/Rss.php';

$channel = new Zend_Feed_Rss(
    'http://feeds.feedburner.com/avoimettyopaikat?format=xml'
);

echo $channel->title();
```

Koodiesimerkki 1: Asennuksen testaaminen. Esimerkin pitäisi tulostaa ”Avoimet työpaikat – Vierityspalkki.fi”.

Kun ohjelmistokehys on asennettu, se on saatavilla myös kaikissa tulevissa projekteissa, eikä ohjelmistokehystä tarvitse erikseen kopioida jokaiseen uuteen projektiin.

4 Zend Framework -sovelluksen esittely

Aloitetaan Zend Framework -sovelluksen esittely käymällä sen rakennetta läpi. Rakenne koostuu sovellusrungosta, jota voidaan kuvata kahdella eri tasolla: fyysisellä tasolla, joka tarkoittaa hakemistoja ja tiedostoja, sekä loogisella tasolla, joka tarkoittaa sovelluksen arkkitehtuuria. Seuraavissa luvuissa käyn läpi rakenteen lisäksi myös sovelluksen muita osia, kuten liitännäisiä ja moduuleita, sekä sovelluksen suoritusprosessia.

4.1 Looginen rakenne

Zend Framework tarjoaa WWW-sovelluksen rakentamiseen MVC-arkkitehtuurimallia, ja seuraavaksi esittelenkin lyhyesti nämä kolme MVC-mallin komponenttia. Vaikka osa tiedoista päteekin yleisesti, tarkastelen silti näitä komponentteja Zend Frameworkin näkökulmastsa.

Malli (M, model)

Zend Framework ei sisällä erillistä mallikomponenttia. Syynä tähän on se, että malli sisältää mm. sovelluksen liiketoimintalogiikan, jota on lähes mahdotonta toteuttaa yleisellä tasolla. Ohjelmoijan tulee siis itse rakentaa oma malli, ja tähän ohjelmistokehys tarjoaa paljon erilaisia työkaluja. (Pope 2009, 115.)

Malli-käsitteen merkityksestä on ilmeisesti olemassa lukuisia eri käsityksiä. Useissa tapauksissa malli rajataan virheellisesti tarkoittamaan vain tietokantaoperaatioita suorittavaksi luokaksi. Tämä on erittäin paha erehdys, sillä se voi johtaa ohjelmoijan kirjoittamaan sovelluslogiikan ohjaimeen. Kun sovelluslogiikka sijaitsee ohjaimessa, samaa koodia voidaan joutua toistamaan useita kertoja ja myös testaaminen vaikeutuu. (Brady 2008.) Tällöin myös menetetään MVC-mallin tuomat edut.

Näkymä (V, view)

Näkymistä, kuten MVC-mallin muistakin komponenteista, on erilaisia mielipiteitä ja näkemyksiä, mitä ne tarkoittavat. Reenskaugin (1979) alkuperäisen teoksen mukaan näkymä on käyttäjälle näkyvä, visuaalinen esitys mallista. Näkymä voi siis olla muutakin kuin WWW-sivu, kuten esimerkiksi kuvia, tallennettava tiedosto tai RSS-syöte (O'Phinney 2007).

Näkymän tulisi hakea tietonsa suoraan mallilta, jolloin vältytään ohjelmakoodin toistamiselta ohjaimessa. Tietoja ei kuitenkaan tule hakea suoraan näkymästä käsin, vaan Zend Frameworkissa on toteutettu vuosia vanha idea näkymien apuluokista (engl. *view helpers*). Näiden apuluokkien avulla voidaan hakea tietoja malleilta ilman tarvetta käyttää ohjainta. (Brady 2008.) Apuluokat vähentävät myös HTML-merkkauksen sekaan kirjoitettavaa ohjelmakoodia, jolloin HTML-merkkauksesta tulee selkeämpi lukea ja helpompi muuttaa.

Ohjain (C, controller)

Zend Frameworkin ohjainjärjestelmä jakautuu kahteen osaan: etuohjaimeen (engl. *front controller*) ja toiminto-ohjaimeen (eng. *action controller*). Etuohjaimen tehtävänä on huolehtia yksittäisen ohjaimen toiminnon suorittamisesta alusta loppuun asti. Tähän kuuluu mm. *reitityksestä* huolehtiminen ja *liitännäisten* (ks. luku 4.4) suorittaminen. (Zend Framework 2010d.)

Toiminto-ohjaimet ovat käyttäjän luomia luokkia, joista jokainen perii ohjelmistokehyksestä löytyvän Zend_Controller_Action-luokan (Zend Framework 2010b). Hyvän toiminto-ohjaimen tunnistaa siitä, että se sisältää mahdollisimman vähän ohjelmakoodia (Buck 2010). Tällöin saadaan luotua helpommin luettavissa olevaa sekä helpommin ylläpidettävää ja uudelleenkäytettävää ohjelmakoodia (Pope 2009, 119).

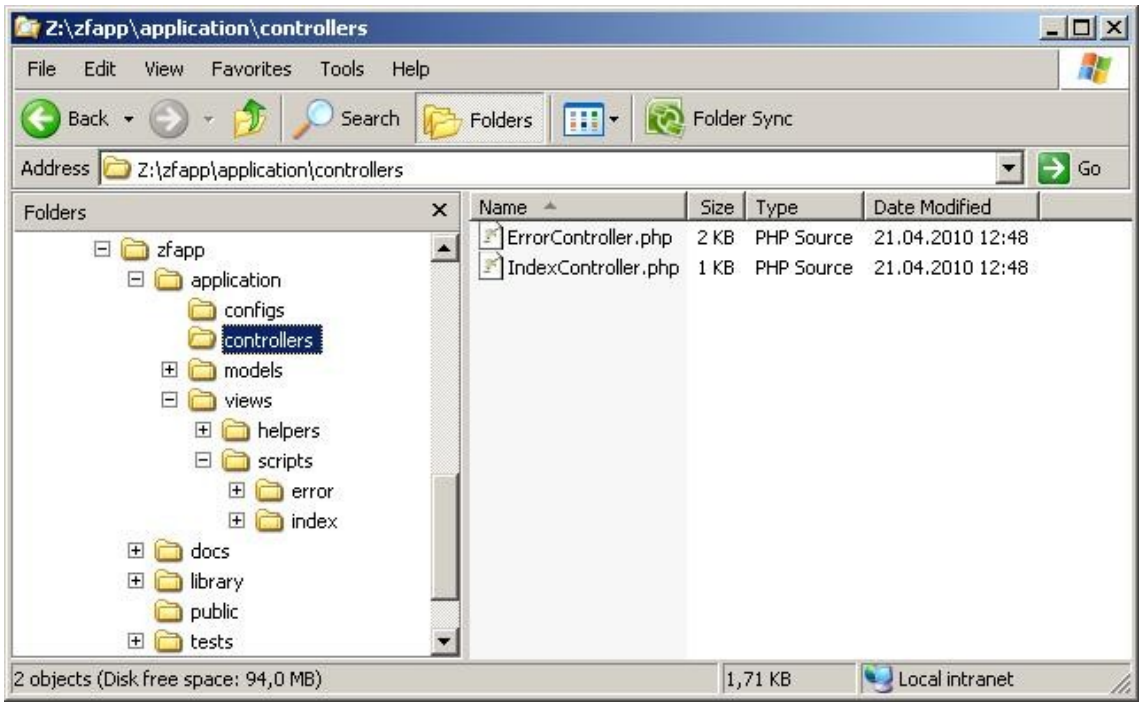
Toiminto-ohjaimen tulisi sisältää vain ohjauslogiikkaa. Ohjain on sidottu ohjelmistokehykseen, jolloin sen sitä on mm. hankala testata ja siirtää. Testattavuutta hankaloittaa mm. se, että ohjaimen suorittamiseksi tarvitaan koko MVC-ohjelmistokehyksen suorittaminen. (Brady 2008.)

4.2 Fyysinen rakenne

Sovelluksen eri osat jaetaan myös fyysisesti eri hakemistoihin, jotka muodostavat sovelluksen rungon. Yleisimmin hakemistorakenne jakautuu 4 eri päähakemistoon, mutta se voi olla myös muunlainen. Useimmiten Zend-työkalulla luotava hakemistorakenne kuitenkin riittää. (Pope 2009, 45.)

Hakemistorakenne voidaan luoda komentorivityökalun lisäksi myös käsin, mutta Zend-työkalu on tähän kätevä apuväline, sillä sen avulla monet toimenpiteet ovat nopeita suorittaa. Sen avulla vältytään myös erilaisilta ohjelmoijan tekemiltä virheiltä, kuten väärin-

kirjoitetuilta luokkien nimiltä. Komentorivityökalun käyttö ei rajoitu vain projektin luomiseen, vaan sillä voidaan lisätä moduuleita ja ohjaimia sovellukseen myös työn myöhemmässä vaiheessa. Kuvassa 3 on Zend-työkalun luoma hakemistorakenne ja sovelluksen suorittamiseksi tarvittavat ohjaimet.



Kuva 3: Zend-työkalun luoma WWW-sovelluksen ei-modulaarinen hakemistorakenne

Päähakemistot

Application-hakemisto sisältää hakemistot ohjaimille, näkymille, malleille ja myös asetustiedostoille. Modulaarisessa sovelluksessa tilanne on kuitenkin hieman monimutkaisempi. Tällöin application-hakemisto sisältää modules-hakemiston, joka puolestaan sisältää moduulin nimen mukaan nimetyn hakemiston. Vasta tällaisessa hakemistossa sijaitsevat moduulin omat malli-, näkymä- ja ohjain-hakemistot.

Public-hakemistossa sijaitsee sovelluksen *index.php*-tiedosto, jonka kautta kulkee jokainen sovellukselle tullut pyyntö. Tässä hakemistossa sijaitsevat myös sovelluksen käyttöliittymän tiedostot, kuten kuvat ja CSS-tiedostot. Lisäksi osoitteiden uudelleenkirjoittamiseen tarvittava *.htaccess*-tiedosto sijaitsee tässä hakemistossa.

Library-hakemisto sisältää käyttäjän omia komponentteja, jotka eivät sisälly malleihin (Pope 2009, 13). Tällaisia voivat olla esimerkiksi palvelukerroksen luokat ja erillaiset lomakkeet.

Tests-hakemisto sisältää kaikki ohjelmakoodin testaamiseen tarvittavat tiedostot. Zend-työkalulla luotavassa uudessa projektissa, tests-hakemisto sisältää vain kaksi hakemistoa. Testejä organisoidessa voidaan tämän hakemiston sisältö rakentaa siten, että se vastaa rakenteeltaan varsinaista sovellusta (PHPUnit 2010). Esimerkiksi modulaarisessa sovelluksessa application-hakemisto sisältää modules-hakemiston. Tällöin myös tests-hakemiston application-alihakemisto sisältää modules-hakemiston.

Moduulit

Sovelluksen koon kasvaessa on hyvä pohtia toiminnallisuuden jakamista ryhmiin. Jako tapahtuu aihepiirin mukaan, eli esimerkiksi uutis-osion muodostavat osat voidaan erottaa uutiset-moduuliksi ja hallintajärjestelmän muodostavat osat admin-moduuliksi.

Moduulit ovat itsenäisiä, pieniä MVC-sovelluksia, joiden tarkoituksena on edistää ohjelmakoodin uudelleenkäyttöä. Jokainen moduuli sisältää omat malli-, näkymä- ja ohjain-hakemistonsa. (Allen ym. 2008, 368-369.) Moduuli voi sisältää myös oman esilatausohjelman.

Moduulille annetaan luonnin yhteydessä nimi, joka on samalla myös hakemiston nimi. Sovelluksella on oletusmoduuli, jota käytetään silloin, kun tulevassa pyynnössä (engl. *request*) ei ole erikseen määritelty, mitä moduulia tarkoitetaan. Tämä oletusmoduuli on nimeltään default. (Zend Framework 2010e.) Myös jokin toisin nimetty moduuli voidaan asettaa sovelluksen oletusmoduuliksi application.ini-tiedostossa.

Moduuleja käytettäessä tulee huomioida erityisesti se, että moduulien ohjain-luokat tarvitsevat etuliitteen, joka on sama kuin moduulin nimi (Allen ym. 2008, 368-369). Oletusmoduulin kohdalla on kuitenkin mahdollista poiketa tästä. Application.ini-tiedostossa voidaan määritellä asetus, joka kertoo sovellukselle, tarvitaanko oletusmoduulin luokkiin etuliitettä.

Koodiesimerkissä 2 on oletusmoduulin ja admin-moduulin Index-ohjaimen luokkien esittelyt, jotka selventävät modulaarisen sovelluksen luokkien nimeämiskäytäntöä. Esimerkin viimeisellä rivillä kerrotaan sovellukselle, että oletusmoduulin luokat eivät tarvitse etuliitettä.

```
<?php
/**
 * Oletusmoduulin index-ohjain
 */
class IndexController
    extends Zend_Controller_Action { ...

<?php
/**
 * Admin-moduulin index-ohjain
 */
class Admin_IndexController
    extends Zend_Controller_Action { ...


Application.ini
...
resources.frontController.prefixDefaultModule = 0
```

Koodiesimerkki 2: Ohjain-luokkien nimeämiskäytäntö modulaarisessa sovelluksessa

4.3 Sovelluksen asetukset

Sovelluksen toimintaa voidaan mukauttaa asetustiedoston kautta, joka Zend-työkalun luomassa sovellusrungossa on oletusarvoisesti *ini*-muotoinen. Zend Framework tukee myös XML- ja PHP-muotoisia asetustiedostoja (Zend Framework 2010f).

Asetukset on ryhmitelty sovelluksen tilan mukaan. Nämä ryhmät voivat periä asetukset toisiltaan, jolloin vältytään samojen asetusten toistamiselta ja sovelluksen toimintaa voidaan muuttaa vaihtamalla vain sovelluksen tila. Asetusten perinnässä tulee kuitenkin konfliktitilanteita, joissa käytössä olevan tilan asetukset voittavat. (Zend Framework 2010g.) Esimerkiksi tuotantotilassa ei tule näyttää virheilmoituksia käyttäjälle, ja siksi ne laitetaan pois päältä. Sovellusta kehitettäessä virheilmoitusten tulee kuitenkin olla nähtävillä. Kun kehitystila perii tuotantotilan asetukset, tulee virheilmoitusten osalta konfliktitilanne. Syntyneessä konfliktissa vallitsevan tilan, eli tässä tapauksessa kehitystilan asetukset tulevat voimaan.

Koodiesimerkissä 3 näkyvä PHP-tulkin aikavyöhykeasetus periytyy myös; kehitystilassa ei ole aikavyöhykkeelle asetusta, joten tuotantotilasta peritty asetus on voimassa myös siellä. Esimerkin tuotantotilassa taas konfliktitilannetta ei voi syntyä, sillä se ei peri asetuksiaan miltään muulta ryhmältä, vaan sillä on täysin itsenäiset asetukset.

```
[production]

;; Tuotantotila, ei perintää
phpSettings.display_startup_errors = 0
phpSettings.display_errors = 0

;; Periytyvä asetus, jonka kanssa ei tule konfliktia
phpSettings.date.timezone = "Europe/Helsinki"


[development : production]

;; Kehitystila, perii tuotantotilan asetukset
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1
```

Koodiesimerkki 3: Kehitystilan ollessa käytössä tuotantotilan ja kehitystilan virheennäyttöasetuksissa on konflikti

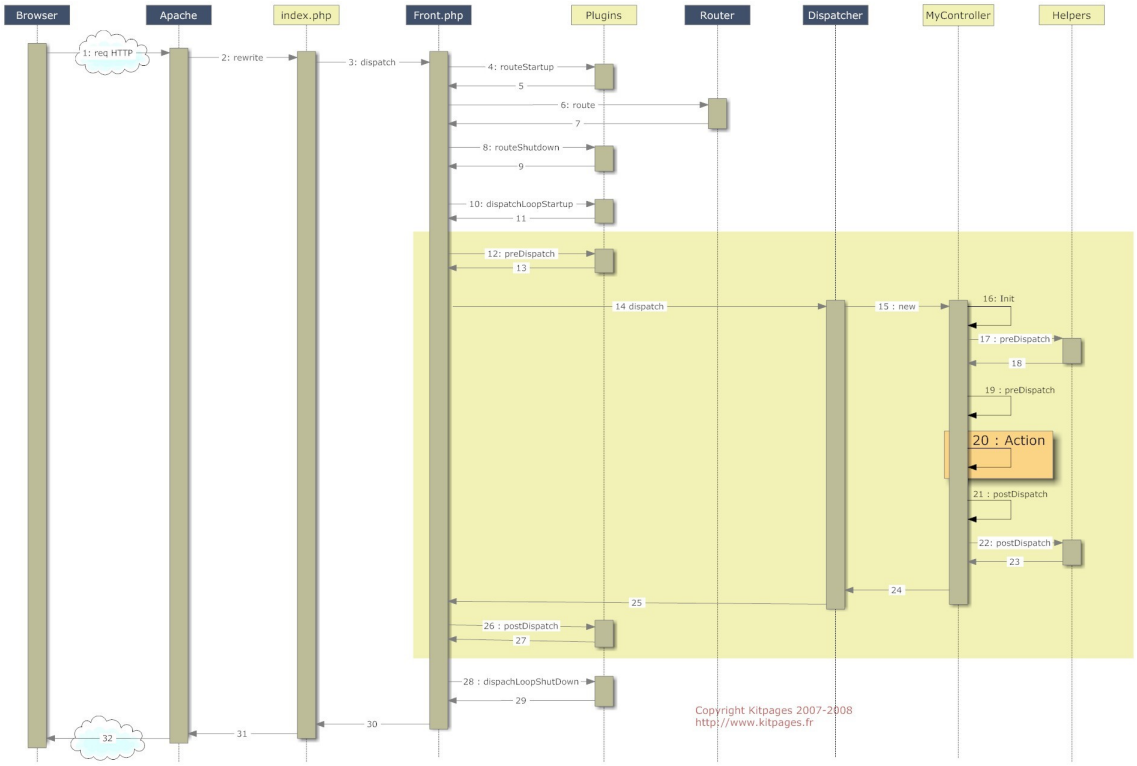
Erilaisten PHP-tulkin asetusten lisäksi application.ini-tiedostoon voidaan määritellä myös muita sovelluksen asetuksia, kuten sivupohjan sijainti ja tiedostonimi, sekä tietokannan asetukset. Näin asetukset saadaan koottua yhteen paikkaan, ja niiden muuttaminen on huomattavasti yksinkertaisempaa. Samalla asetusten tekeminen varsinaisessa ohjelmakoodissa vähenee.

Asetuksia voidaan tehdä myös Zend-työkalulla, mutta yksittäisten asetusten tekeminen käsin näyttäisi hieman helpommalta.

4.4 Sovelluksen suoritusprosessi

Sovelluksen suoritusprosessiin kuuluu monia erilaisia asioita, joita on syytä käydä läpi ainakin jonkin verran, sillä näitä tietoja tarvitaan mm. liitännäisten toiminnan selventämiseen. Kuvassa 4 on sekvenssikaavio, josta nähdään sovelluksen suoritusprosessi kokonaisuudessaan. Sama kaavio löytyy hieman suurempana versiona liitteestä 2.

Suoritusprosessi alkaa pyynnöstä, jonka selain lähettää palvelimelle. Jokainen sovellukseen tuleva pyyntö kulkee index.php-tiedoston kautta, jossa käynnistetään varsinainen sovellus. Pääosan vastuusta prosessissa kantaa etuohjain, jonka tehtäviin kuuluu mm. reitityksestä ja ohjain-toimintoparin suorituksesta huolehtiminen. Etuohjaimeen kuuluu myös luvussa 4.1 esitelty liitännäisjärjestelmä, jota käsittelen enemmän hieman edempänä.



Kuva 4: Sekvenssikaaviosovelluksen suoritusprosessista (Le Van 2008)

Suoritusprosessiin kuuluu 7 pääkohtaa, joista 1 on toiminto-ohjaimen toiminnon suorittaminen. Loput kuusi ovat tapahtumia, joihin voidaan kytkeä liitännäisiä. Ensimmäinen tapahtuma on route startup, joka sijaitsee suoritusprosessin alkupäässä, juuri ennen reititystä. Reitityksellä tarkoitetaan sovellukseen tulevan pyynnön tulkintaa, jossa päätetään mitä moduulia ja ohjain-toimintoparia on pyydetty. Löydettyään nämä tiedot pyynnöstä, reititin lisää kyseiset tiedot pyyntöolioon. (Pope 2009, 53-54.)

Reititystä seuraa kolme muuta tapahtumaa ennen varsinaisen ohjain-toimintoparin suorittamista. Viimeinen tapahtuma on dispatch loop shutdown, jonka jälkeen toiminto-ohjaimen tuottama vastaus lähetetään selaimelle.

Liitännäiset

Edellisissä kappaleissa on jo hieman sivuttu liitännäisistä. Liitännäiset ovat siis erityisiä luokkia, joilla voidaan laajentaa etuohjaimen suoritusprosessia. Jokainen ohjelmoijan luoma liitännäis-luokka perii Zend_Controller_Plugin_Abstract-luokan. Tähän omaan-luokkaan lisätään mainitun abstraktin luokan määrittelemiä metodeja. Yhden liitännäisen voi liittää myös useampaan eri tapahtumaan lisäämällä liitännäis-luokkaan kyseisen tapahtuman metodin. (Pope 2009, 52.) Koodiesmerkissä 4 on esimerkkiliitännäinen, joka on liitetty kahteen eri tapahtumaan.

```php
<?php

class HM_Controller_Plugin_Example
    extends Zend_Controller_Plugin_Abstract
{
    public function preDispatch(Zend_Controller_Request_Abstract $request)
    {
        // route shutdown -tapahtuma...
    }

    public function postDispatch(Zend_Controller_Request_Abstract $request)
    {
        // post dispatch -tapahtuma...
    }
}
```

Koodiesimerkki 4: Kahteen eri tapahtumaan liitetty esimerkkiliitännäinen

Liitännäisten suorittamiseen käytetään erityistä välittäjäjärjestelmää. Se toimii myös rekisterinä, jossa säilytetään rekisteröityjä liitännäisiä (Zend Framework 2010h). Etuohjain käyttää tätä välittäjää jokaisen liitännäisen metodien kutsumiseen. Jokaisen tapahtuman kohdalla liitännäiset käydään läpi *foreach*-silmukassa, jossa kutsutaan niiden kyseiseen tapahtuman metodia. Vaikka metodia ei olisikaan määritelty varsinaisessa liitännäisessä, voidaan sitä kutsua, sillä metodin tyhjä runko on kuitenkin määritelty abstraktissa Zend_Controller_Plugin_Abstract-luokassa. Koodiesimerkissä 5 nähdään route shutdown -tapahtuman silmukka, jossa kutsutaan jokaisen liitännäisen routeShutdown-metodia.

```php
/**
 * Called before Zend_Controller_Front exits its iterations over
 * the route set.
 *
 * @param  Zend_Controller_Request_Abstract $request
 * @return void
 */
public function routeShutdown(Zend_Controller_Request_Abstract $request)
{
    foreach ($this->_plugins as $plugin) {
        try {
            $plugin->routeShutdown($request);
        } catch (Exception $e) {
            if (Zend_Controller_Front::getInstance()->throwExceptions()) {
                throw $e;
            } else {
                $this->getResponse()->setException($e);
            }
        }
    }
}
```

Koodiesimerkki 5: Route shutdown -tapahtuman suoritus välittäjäluokassa

Liitännäiset suoritetaan oletusarvoisesti siinä järjestyksessä kuin ne on rekisteröity. Suoritusjärjestykseen voidaan kuitenkin vaikuttaa rekisteröinnin yhteydessä annettavalla *stackIndex*-paramterilla. Liitännäinen, jolla on matalampi stackIndex-arvo, suoritetaan

aiemmin. Esimerkiksi ErrorHandler-liitännäinen, jonka Zend Framework automaattisesti rekisteröi arvolla 100, suoritetaan viimeisenä. (Pope 2009, 52.)

Liitännäisiä tulisi käyttää vain sellaiseen tarkoitukseen, jossa pyritään vaikuttamaan koko sovelluksen laajuisesti. Tällaisia tilanteita voivat olla esimerkiksi välimuistiin tallentaminen (engl. *caching*) tai käyttäjän oikeuksien tarkistaminen. Haluttaessa vaikuttaa pienempään kokonaisuuteen, kuten moduuliin, käytetään toimintojen apuluokkia (engl. *action helpers*). (Pope 2009, 53.)

Eräs erinomainen liitännäisen käyttökohde onkin edellä mainittu oikeuksien tarkistaminen. Ajatellaanpa sovellusta, jossa on 5 moduulia ja jokaisessa moduulissa 10 ohjainta. Nyt oikeuksien tarkistamiseen liittyvä ohjelmakoodi pitäisi kirjoittaa pahimmassa tapauksessa 50 kertaa. Lisäksi vain yhden ohjaimen käsittävät oikeudet voitaisiin tarkistaa kerrallaan. Kun tämä ohjelmakoodi siirretään liitännäiseen, kirjoitetaan sama ohjelmakoodi vain kerran. Nyt yhdestä paikasta voidaan myös hallita koko sovelluksen laajuisesti oikeuksia; samaa liitännäistä voidaan käyttää myös myöhemmässä vaiheessa lisättävän toiminnallisuuden yhteydessä.

5 Esimerkkisovellus

Ennen sovelluksen rakentamista on hyvä miettiä, mitä sovelluksella tehdään, ja millaisia vaatimuksia sille asetetaan. Tämä esimerkkisovellus on eräänlainen käyttäjähallinta, jonka avulla voidaan hallita käyttäjätunnuksia ja rooleja. Hallintatoimenpiteisiin kuuluvat lisäys, muokkaus ja poisto, sekä käyttäjän asettaminen ei-aktiiviseen tilaan.

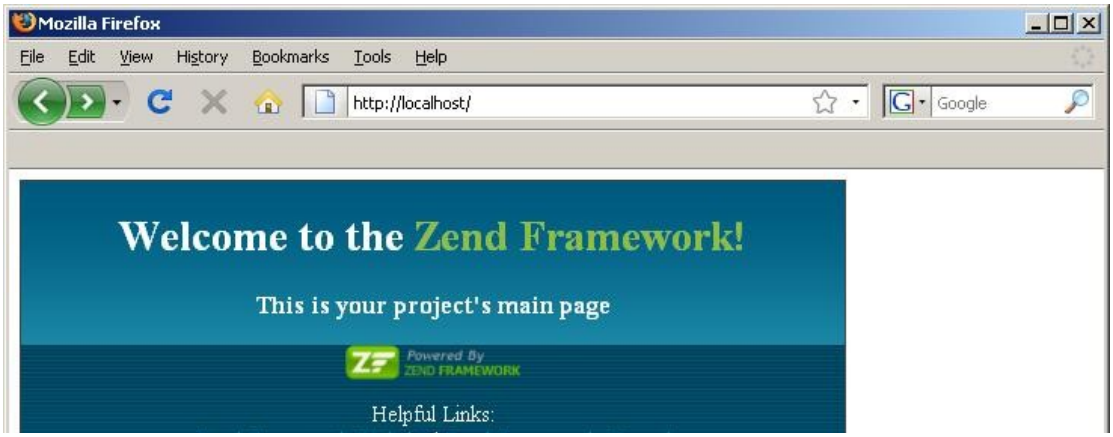
Sovelluksessa pyritään käyttämään hyväksi aiemmissa luvuissa opittuja asioita: sovellus on modulaarinen, oikeuksien valvonta tapahtuu liitännäistekniikan avulla ja asetukset tehdään application.ini-tiedostoon. Lisäksi sovelluksessa sijoitetaan osa toiminnallisuudesta palvelukerrokseen.

Käyttäjät ja roolit

Sovelluksessa on kaksi eri roolia: user ja admin. User-rooli on tavallisen käyttäjän rooli, joka oikeuttaa muokkaamaan vain omia tietoja. Tavallinen käyttäjä ei voi muuttaa rooliaan, vaan ainoastaan hallintoroolissa oleva käyttäjä voi sen tehdä. Admin on tällainen hallintorooli. Se oikeuttaa muokkaamaan kaikkien käyttäjien tietoja, sekä poistamaan käyttäjiä. Käyttäjätunnuksen voi myös asettaa myös ei-aktiiviseen tilaan, jolloin tunnus on kyllä olemassa, mutta käyttäjä ei voi kirjautua sovellukseen.

5.1 Modulaarinen sovellus

Uudelle sovellukselle luodaan fyysinen runko komennolla *zf create project zfapp*. Sovellusrunko, joka nyt luodaan, on jo sellaisenaan suoritettava sovellus. Sitä voi testata selaimella, kuten kuvassa 5 on tehty.



Kuva 5: Zend-työkalun luoma WWW-sovelluksen runko selaimessa

Sovellukseen luodaan myös moduuli komennolla *zf create module user*, jonka jälkeen *application*-hakemiston sisällä olevat models-, views- ja controllers-hakemistot siirretään juuri luodun moduulin sisälle.

Modulaarinen sovellus vaatii joitakin lisäasetuksia toimiakseen. Koodiesimerkissä 6 listataan asetustiedostoon lisättävät rivit. Tässä tulee huomioida, että *moduleDirectory*-asetus **korvaa** *controllerDirectory*-asetuksen, sillä kaikki ohjaimet sijaitsevat nyt moduulihakemiston sisällä.

Kahdella viimeisellä rivillä asetetaan user-moduuli oletusmoduuliksi, ja kerrotaan sovellukselle, että oletusmoduulin ohjain-luokkien nimet eivät tarvitse etuliitettä. Kun sovellusta katsellaan uudestaan selaimella, näkyvä WWW-sivu on samanlainen, kuin kuvassa 5, vain sovelluksen rakenne on muuttunut.

```
[production]

;; ...

;; Moduulien asetukset
resources.frontController.moduleDirectory = APPLICATION_PATH "/modules"
resources.modules[] =

;; Oletusmoduulin asetukset
resources.frontController.defaultModule = "user"
resources.frontController.prefixDefaultModule = 0

;; ...
```

Koodiesimerkki 6: Modulaarisen WWW-sovelluksen asetukset

5.2 Esimerkkisovelluksen ohjaimet

Sovellukseen tarvitaan 3 eri ohjainta: yksi etusivua varten, yksi kirjautumistoimintoja varten ja yksi käyttäjien hallintaa varten. Koodiesimerkissä 7 luodaan auth-ohjain, ja se sijoitetaan user-moduuliin. Lisäksi ohjaimeen sisällytetään myös index-toiminto.

Samalla tavalla luodaan myös kirjautumistoimintoihin käytettävä ohjain; vain luotavan ohjaimen nimi muutetaan esimerkin komentoon. Etusivun ohjainta ei tarvitse erikseen luoda, sillä se on jo valmiina controllers-hakemistossa, joka aiemmin (ks. luku 5.1) siirrettiin moduuli-hakemiston alle. Samoin myös virhesivujen ohjain, ErrorController, on jo valmiina sovelluksessa.

```
Zend-työkalu:

        Luodaanko ohjaimeen index-toiminto?
                                |
$ zf create controller auth 1 user
                          |     |
      Luotavan ohjaimen nimi  Moduulin nimi, johon ohjain luodaan
```

Koodiesimerkki 7: Ohjaimen lisääminen sovellukseen Zend-työkalun avulla

5.2.1 Etusivun ohjain

Index-ohjain ja index-toiminto ovat oletusarvoisesti suoritettava ohjain-toimintopari, jonka tähden niiden olemassaolo on välttämätöntä. Tämä pari suoritetaan, jollei mitään muuta toimintoa tai ohjain-toimintoparia ole pyydetty.

Koodiesimerkki 8 selventää ohjaimen toiminnon, ja siihen liittyvän näkymän yhteistyötä. Index-ohjaimen index-toiminto syöttää ”Hei, Maailma!” -viestin varsinaiselle näkymälle Zend_View-komponentin kautta.

```php
<?php
/**
 * Ohjain: application/modules/user/controllers/IndexController.php
 */
class IndexController
    extends Zend_Controller_Action
{
    pulic function indexAction()
    {
        $this->view->message = "Hei, Maailma!";
    }
}

/**
 * Näkymä: application/modules/uesr/views/scripts/index.phtml
 */
<h1>Näkymä</h1>
<p>
    <?php echo $this->message; ?>
</p>
```

Koodiesimerkki 8: Ohjain antaa tiedon näkymälle, ja näkymä muotoilee ja lähettää sen käyttäjän selaimelle.

Tämä toimintaperiaate ei koske ainoastaan index-ohjainta, vaan myös kaikkia muita ohjaimia. Tästä ryhmästä kuitenkin jäävät pois ne ohjaimen toiminnot, joiden tarkoitus on tuottaa jotakin muuta kuin WWW-sivu.

5.2.2 Kirjautumistoimintojen ohjain

Kirjautumistoimintojen ohjain sisältää käyttäjän sisään- ja uloskirjaukseen tarvittavat toiminnot. Koska kirjautumistoimintojen logiikka ei kuulu ohjaimeen tai malliin, se täytyy sijoittaa muualle. Tämä koodi sijoitetaankin *palvelukerrokseen* (engl. s*ervice layer*), jonka tiedostot sijaitsevat library-hakemiston alihakemistoissa.

Sovellukseen kirjauduttaessa syötetty sähköpostiosoite ja salasana tarkistetaan tietokannasta. Jos kirjautuminen onnistui, tietokannasta löytyvät käyttäjän tiedot salasanaa lukuun ottamatta tallennetaan istuntoon eli *sessioon*. Istunto on PHP-tulkin ominaisuus, jonka avulla käyttäjän tila voidaan säilyttää HTTP-protokollan tilattomuudesta huolimatta useiden eri sivupyyntöjen ajan. Käyttäjä saa itselleen yleensä keksiin (engl. c*ookie*) tallennettavan istuntotunnisteen, joka lähetetään palvelimelle jokaisen sivupyynnön yhteydessä (PHP Manual 2010e.). Kirjautumislogiikkaan käytettävä Zend_Auth-komponentti hyödyntää istuntoja käyttäjän tietojen säilyttämiseen sivunlatausten välillä.

Kirjautumistoimintoihin käytetään edellä mainittua Zend_Auth-komponenttia. Se tarjoaa eräänlaisen rajapinnan kirjautumiseen tarvittaville toiminnoille. Aivan sellaisenaan se ei kuitenkaan ole käytettävissä, vaan Zend Auth -komponentti tarvitsee sovittimen. (Zend Framework 2010i.) Sovitinta vaihtamalla voidaan vaihtaa kirjautumispalvelua esimerkiksi oman sovelluksen tietokannasta LDAP-palvelimeen ilman oleellisia muutoksia ohjelmakoodiin.

Koodiesimerkissä 9 on Auth-ohjaimen login-toiminto, jossa tapahtuu käyttäjän kirjautuminen sovellukseen. Login-toiminnon ensimmäisellä rivillä luodaan kirjautumislomake-olio. Sen jälkeen tarkistetaan, onko lomake lähetetty. Jos lomaketta ei ole lähetetty, hypätään ehtolauseen ylitse ja syötetään lomake-olio näkymälle. Lomake-olio sisältää erityisen *__toString*-metodin, jolloin lomake olio voidaan tulostaa näytölle PHP:n *echo*-komennolla. Tämä metodi antaa ohjelmoijalle mahdollisuuden kontrolloida sitä, miten olio reagoi, kun sitä yritetään esimerkiksi tulostaa (PHP Manual 2010f).

Lomake sisältää *isValid*-metodin, joka on toteutettu Zend_Form-luokkaan. Tälle metodille annetaan parametriksi lähetetyn lomakkeen tiedot. (Zendcasts 2009.) Jos lomakkeen tiedot ovat kunnossa, tallennetaan sähköposti ja salasana omiin muuttujiinsa. Lopuksi käytetään palvelukerroksen *HM_Service_Auth*-luokkaa kirjautumiseen. Onnistuneen kirjautumisen jälkeen käyttäjä ohjataan user-moduulin etusivulle.

```php
<?php
/**
 * Kirjautumistoimintojen ohjain
 */
class AuthController
    extends Zend_Controller_Action {
    // ... tarvittavat resurssit ladataan ensin init-funktiossa

    public function loginAction() {
        $loginForm = new HM_Form_Login();
        if($this->getRequest()->isPost())
        {
            $data = $this->getRequest()->getPost();
            if($loginForm->isValid($data)) {
                $identity = $loginForm->getValue('email');
                $credential = $loginForm->getValue('password');
                if($this->_authService->login($identity, $credential))
                {
                    $this->_helper->Redirector->gotoUrl('user');
                }
            }
        }
        $this->view->loginForm = $loginForm;
    }
}
```

Koodiesimerkki 9: Käyttäjän sisäänkirjaus tapahtuu palvelukerroksen luokan avulla

Logout-toiminnossa on huomattavasti vähemmän ohjelmakoodia, sillä se mahtuu 4 koodiriviin. Logout-toiminto ei tarvitse näkymätiedostoa, sillä toiminnon ei ole tarkoitus näyttää käyttäjälle mitään. Toiminnossa estetään näkymätiedoston ja sivupohjan käyttö, sekä tyhjennetään istunto. Lopuksi käyttäjä ohjataan sovelluksen kirjautumissivulle.

5.2.3 Käyttäjähallinnan ohjain

Käyttäjätilejä hallitaan AccountController-ohjaimen kautta. Se sisältää toimintoja, joilla voidaan mm. lisätä, poistaa ja muokata käyttäjiä. Tässä ohjaimessa käytetään hyväksi Users-mallia, jonka avulla tietokannassa sijaitsevien tietojen muokkaaminen on helppoa.

Tämän ohjaimen toiminnot käyttäytyvät pitkälti samalla tavalla kuin edellisessä luvussa kuvatut Auth-ohjaimen toiminnot: Ensin luodaan lomakeolio. Sitten tarkistetaan, onko lomake lähetetty. Jos lomake on vielä lähettämättä, näytetään se käyttäjälle. Lomakkeen ollessa jo lähetetty, tarkistetaan syötteet ja tallennetaan tiedot tietokantaan.

Koodiesimerkkissä 10 on User-mallin *updateUser*-funktio, jolla päivitetään käyttäjän tietoja tietokannassa. Huomionarvoista esimerkissä on *hash*-metodin käyttö. Koska lomakkeella on mahdollista muuttaa käyttäjän salasana, tulee huomioida se, ettei salasa-

noja koskaan tallenneta selkokielisenä. Ainoastaan salasanasta luotu tiiviste tallennetaan tietokantaan. Kun salasanaa ei haluta päivittää, lomakkeen salasanakenttä jätetään tyhjäksi, jolloin hash-metodi osaa ottaa tämän huomioon.

```
public function updateUser(Zend_Controller_Request_Abstract $request)
{
    $form = $this->getUserEditor($request);

    try {

        $post = $request->getPost();

        if($form->isValid($post))
        {
            $values = $form->getValues();
            $values = $this->hash($values);

            $where = $this->getAdapter()->quoteInto('id=?', $values[id]);
            $this->update($values, $where);
            return true;
        }
        else
        {
            return $form;
        }

    } catch(Exception $e)
    {
        return false;
    }
}
```

Koodiesimerkki 10: Käyttäjän tiedot päivitetään tietokantaan palvelukerroksen luokan avulla.

Kuten luvussa 5 kerrottiin, vain hallintoroolissa oleva käyttäjä voi muuttaa rooleja. Tämän vuoksi syntyy ongelma, johon on kaksi ratkaisua. Uuden lomakkeen luominen ei kuitenkaan ole järkevää, sillä se olisi lähes samanlainen kuin alkuperäinen käyttäjän muokkaamiseen ja lisäämiseen tarkoitettu lomake. Niinpä jäljelle jää vain yksi oikea vaihtoehto. Koodiesimerkissä 11 on osa User-mallin *getUserEditor*-metodia, jonka avulla luodaan kirjautuneen käyttäjän roolia vastaava lomake. Tämä tapahtuu lisäämällä roolin valintaan tarkoitettu pudotusvalikko lomakkeelle.

Samalla lomakkeella siis lisätään ja muokataan käyttäjiä. Kun jonkin käyttäjän tietoja muokataan, tulee tietysti vanhat tiedot hakea lomakkeelle. GetUserEditor-metodi osaa huolehtia tämän lisäksi myös siitä, että tavallinen käyttäjä voi muokata vain omia tietojaan.

```
public function getUserEditor(Zend_Controller_Request_Abstract $request = null)
{
    $form = new HM_Form_User();
    $currentIdentity = $this->_auth->getIdentity();

    /**
     * Check the requesting role: Admin can change roles
     */

    $role = $currentIdentity->role;
    if($role === 'Admin')
        $form->addRoleElement();
```

Koodiesimerkki 11: Rooli-valikon lisääminen lomakkeelle

5.3 Oikeuksien tarkistaminen liitännäisen avulla

Käyttäjän oikeuksien tarkistaminen toteutetaan liitännäistekniikalla. Sovellukseen toteutetaan siis liitännäinen, joka valvoo käyttäjän tilaa: onko käyttäjä kirjautunut sovellukseen vai ei. Lisäksi se huolehtii pääsylistoista, joiden avulla valvotaan kirjautuneen käyttäjän oikeuksia pyydettyyn sivuun. Koodiesimerkissä 12 on osa Auth-liitännäistä. Sen Dispatch Loop Startup -tapahtumassa luodaan pääsylista ja tarkistetaan, onko käyttäjällä oikeutta pyydettyyn sivuun.

```
// ...

$acl->allow($guest, array($error, $login, $auth));
$acl->allow($user, array($siteIndex));

if($id === $param)
{
    $acl->allow($user, $edit);
}

$acl->allow($admin);


if(!$acl->isAllowed($currentRole, $requestedResource))
{
    $request->setModuleName('user')
            ->setControllerName('index')
            ->setActionName('deny');
}

// ...
```

Koodiesimerkki 12: Käyttäjän kirjautumista valvova Auth-liitännäinen

Esimerkin pääsylista sallii kirjautumattomalle käyttäjälle pääsyn vain virhe- ja kirjautumissivulle. Ensimmäisessä ehtolauseessa sallitaan tavalliselle käyttäjälle omien tietojen muokkausoikeudet. Ehtolauseen jälkeen sallitaan hallintoroolille kaikki mahdolliset toimenpiteet, ja viimeiseksi testataan, onko käyttäjällä oikeudet pyydettyyn sivuun.

6 Yhteenveto

Sovelluksen arkkitehtuuri on paljon suunnittelutyötä vaativa urakka. Ohjelmistokehyksen käyttö kuitenkin auttaa tässä, sillä se tarjoaa lähes valmiin arkkitehtuurin WWW-sovellukselle. Kokonaan arkkitehtonisia kysymyksiä pakoon ohjelmistokehyksenkään kanssa ei silti pääse. Huonolla suunnittelulla voi helpostikin mitätöidä ohjelmistokehyksen tuomat edut.

Zend Framework on yksi monista tarjolla olevista PHP-ohjelmistokehyksistä. Sen rakenteen vuoksi sitä voidaan kutsua myös komponenttikirjastoksi, sillä sen komponentit ovat käytettävissä myös muiden ohjelmistokehysten kanssa tai omassa projektissa. Opetellessa sen käyttöä törmää väistämättä erilaisiin suunnittelumalleihin ja arkkitehtonisiin ratkaisuihin, joita tutkiessa oppii myös yleisemmin ohjelmistojen suunnittelusta.

Eräs työn yhteydessä esille tullut ongelma on käsitteiden määrittely. Useat ihmiset tuntuvat käsittävän samat termit eri tavoin. Esimerkiksi kysyttäessä, mitä palvelukerrokseen kuuluu Zend Framework -sovelluksessa, saadaan joitakin eri vaihtoehtoja. Järjestin asiasta pienimuotoisen kyselyn, jonka tuloksista nähdään sama asia. Jotkut olivat sitä mieltä, että palvelukerros tarkoittaa ohjainta. Toiset taas ajattelivat sen tarkoittavan mallia, ja kolmannet vielä jotakin muuta. Itse päädyin pitämään Staffordin esitystä oikeimpana: palvelukerros kapseloi liiketoimintalogiikan ja toimii rajapintana liiketoimintalogiikan ja käyttöliittymän välillä.

Tutustuminen Zend Framework -ohjelmistokehykseen oli huomattavasti haastavampaa kuin olin odottanut. Alunperin ehkä hieman liian helppona pitämäni aihe muuttui työn aikana oppitunniksi ohjelmistokehyksen arkkitehtuurista ja erilaisista suunnittelumalleista. Ehkä nämä ovat syitä siihen, miksi teksti jäi hieman suppeaksi esitykseksi ohjelmistokehyksestä. Jos nyt tekisin työn uudelleen, käyttäisin huomattavasti enemmän aikaa materiaalin sekä ohjelmistokehyksen lähdekoodin läpikäymiseen. Lähdekoodin läpi käyminen on hieman hankalaa, mutta sieltä voi löytää asioita, joita ei mistään materiaaleista löydä. Esimerkiksi liitännäisten toimintaa tutkiessani kävin läpi useita eri lähdekoodia sisältäviä tiedostoja.

Esimerkkisovellus

Esimerkkisovelluksen rakenne muuttui myös työn aikana: eri osia tuli ja meni, ja jotkut muuttuivat. Lopputuloksena oli sovellus, jossa tavoitellut asiat toteutuivat. Se on modu-

laarinen, ja liitännäistä hyödynnetään käyttäjän toimien valvonnassa. Lisäksi sovellukseen luotiin ohjaimia, yksi malli, ja useita näkymätiedostoja.

Pääsylistasta ei kuitenkaan muotoutunut sellaista kuin toivoin. Ajatus oli, että sitä voisi käyttää yleisemmin. Nyt roolit ja resurssit on määritelty suoraan ohjelmakoodiin, josta niiden lisääminen tai muuttaminen on hankalaa.

Esimerkkisovelluksen käyttöliittymä on englanniksi, sillä lomakkeiden oletuskielenä on englanti. Tyhjistä kentistä aiheutuneet virheilmoitukset olisivat olleet hölmö yhdistelmä muuten suomenkielisen tekstin yhteydessä. Zend Framework sisältää Zend_Translate-komponentin, jonka avulla lomakkeiden virheilmoitusten kääntäminen olisi ollut helppo työ. Rajasin tämän kuitenkin tarkoituksellisesti pois tämän työn yhteydestä.

Lomakkeiden käsittely on sovelluksessa toteutettu kahdella eri tavalla: kirjautumistoimintojen lomake käsitellään ohjaimessa, kun taas käyttäjähallinnon lomake käsitellään mallissa. Kumpi näistä on oikeampi tapa, sitä en osaa sanoa. Jälkimmäisessä vaihtoehdossa ohjaimeen jäi pelkästään ohjaamiseen liittyvää logiikkaa, mikä lienee tarkoituksenmukaista.

Lopuksi

Opin työn aikana paljon uusia asioita Zend Frameworkista, mutta myös jotakin ohjelmistojen suunnittelusta yleensä. Zend Frameworkin toiminnan ymmärtäminen korkeammalla tasolla auttaa myös muiden samankaltaisten MVC-mallia hyödyntävien WWW-ohjelmistokehysten oppimisessa ja käyttöönotossa.

Kuten tämän työn johdannossa totesin, olen aiemmin ollut mukana vain yhdessä laajassa sovelluskehitysprojektissa. Nyt lähtökohdat oman osaamisen kannalta ovat kuitenkin huomattavasti paremmat kuin koskaan aikaisemmin. Tuotantokäyttöön tulevassa asiakasjärjestelmässä on haastetta riittämiin, ja näillä näkymin sen rakentaminen olisi tarkoitus aloittaa lähitulevaisuudessa.

Lähteet

Allen, Rob, Lo, Nick, Brown, Steven 2008 Zend Framework in Action
[online] [viitattu 2010]
Saatavissa: http://proquest.safaribooksonline.com.elib.tamk.fi/...

Baker, Mike 2009 What is Software Framework? And why should you like ‘em?
[online] [viitattu 4.5.2010]
Saatavissa: http://info.cimetrix.com/blog/bid/22339/What-is-a-Software-Framework-And-why-should-you-like-em

Buck, Jamis 2006 Skinny Controller, Fat Model
[online] [viitattu 8.3.2010]
Saatavissa: http://weblog.jamisbuck.org/2006/10/18/skinny-controller-fat-model

Brady, Pádraic 2008 The M in MVC: Why Models are Misunderstood and Unappreciated
[online] [viitattu 24.5.2010]
Saatavissa: http://blog.astrumfutura.com/archives/373-The-M-in-MVC-Why-Models-are-Misunderstood-and-Unappreciated.html

Koskimies, Kai, Mikkonen, Tommi 2005 Ohjelmistoarkkitehtuurit. Helsinki: Talentum.

Le Van, Philippe 2008 Plugins et helpers du MVC du Zend Framework
[online] [viitattu 3/2010]
Saatavissa: http://www.kitpages.fr/cms/site/tutoriaux/sequence_globale.jpg

O’Phinney Weir, Matthew 2007, Zend Framework MVC Quick Start
[online] [viitattu 26.4.2010]
Saatavissa: http://devzone.zend.com/content/zendcon_07_slides/Ophinney_Matthew_2007-ZendCon-MVC.pdf

PHP Manual 2010a Namespaces overview

[online] [viitattu 12.5.2010]
Saatavissa: http://www.php.net/manual/en/language.namespaces.rationale
.php

PHP Manual 2010b Extends
[online] [viitattu 24.5.2010]
Saatavissa: http://php.net/manual/en/keyword.extends.php

PHP Manual 2010c Description of core php.ini directives
[online] [viitattu 12.5.2010]
Saatavissa: http://www.php.net/manual/en/ini.core.php#ini.include-path

PHP Manual 2010d Runtime Configuration
[online] [viitattu 12.5.2010]
Saatavissa: http://www.php.net/manual/en/errorfunc.configuration.php
#ini.display-errors

PHP Manual 2010e Sessions: Introduction
[online] [viitattu 11.5.2010]
Saatavissa: http://www.php.net/manual/en/intro.session.php

PHP Manual 2010f Magic Methods
[online] [viitattu 25.5.2010]
Saatavissa: http://www.php.net/manual/en/language.oop5.magic.php

PHPUnit 2010 Organizing Tests
[online] [viitattu 14.5.2010]
Saatavissa: http://www.phpunit.de/manual/current/en/organizing
-tests.html

Pope, Keith 2009 Zend Framework 1.8 Web Application Development
[online] [viitattu 6.5.2010]
Saatavissa: http://http://proquest.safaribooksonline.com/...

Reenskaug, Trygve 1979 Models-Views-Controllers
[online] [viitattu 9.5.2010]

Saatavissa: http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf

Stafford, Randy 2010 Service Layer
[online] [viitattu 12.5.2010]
Saatavissa: http://martinfowler.com/eaaCatalog/serviceLayer.html

Tietotekniikan termitalkoot 2001
[online] [viitattu 20.5.2010]
Saatavissa: http://www.tsk.fi/tsk/termitalkoot/haku-266.html

Wikipedia 2010a Model-View-Controller
[online] [viitattu 2010]
Saatavissa: http://en.wikipedia.org/wiki/Model-View-Controller

Wikipedia 2010b SQL
[online] [viitattu 24.5.2010]
Saatavissa: http://en.wikipedia.org/wiki/SQL

Wikipedia 2010c Zend Framework
[online] [viitattu 2010]
Saatavissa: http://en.wikipedia.org/wiki/Zend_framework

Zendcasts 2009 Zend Form introduction
[online][viitattu 2010]
Saatavissa: http://www.zendcasts.com/zend_form-introduction-part-1/2009/02/

Zend Framework 2010a Overview
[online] [viitattu 2010]
Saatavissa:http://framework.zend.com/manual/en/introduction.overview.html

Zend Framework 2010b Action Controllers
[online] [viitattu 24.5.2010]
Saatavissa: http://framework.zend.com/manual/en/zend.controller.action.html

Zend Framework 2010c Requirements Introduction
[online] [viitattu 2010]
Saatavissa: http://framework.zend.com/manual/1.10/en/requirements.introduction.html

Zend Framework 2010d Zend_Controller Quick Start
[online] [viittattu 26.4.2010]
Saatavissa: http://framework.zend.com/manual/en/zend.controller.quickstart.html

Zend Framework 2010e Using a Modular Conventional Directory Structure
[online] [viitattu 27.4.2010]
Saatavissa: http://framework.zend.com/manual/en/zend.controller.modular.html

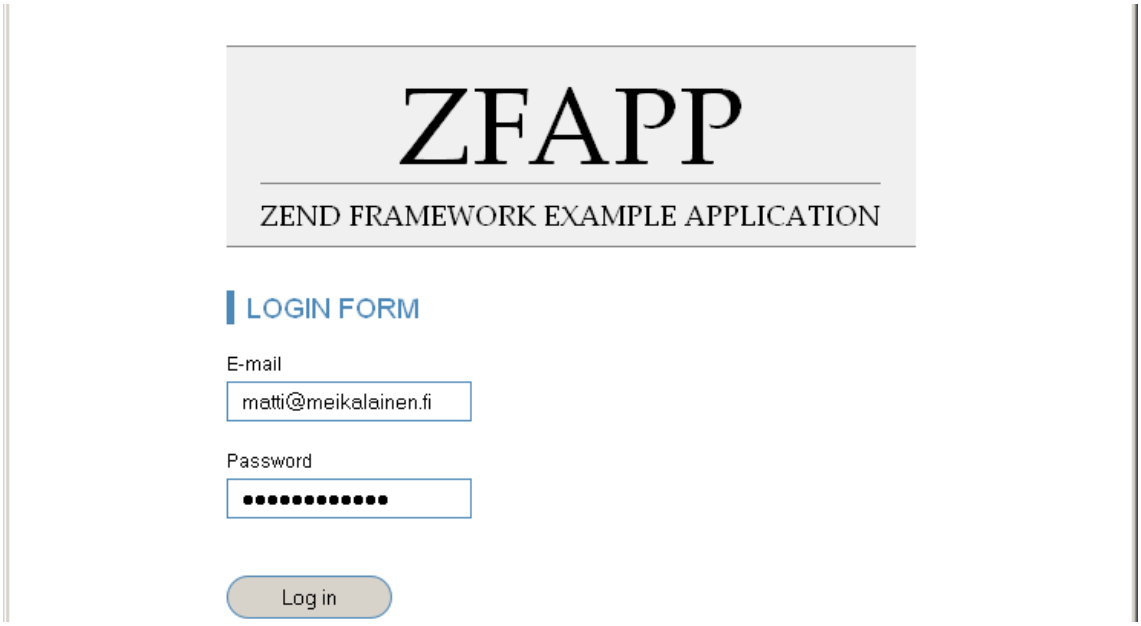
Zend Framework 2010f Zend Application Quick Start
[online] [viitattu 26.3.2010]
Saatavissa: http://framework.zend.com/manual/en/zend.application.quickstart.html

Zend Framework 2010g Zend_Config
[online] [viitattu 4.3.2010]
Saatavissa: http://framework.zend.com/manual/en/zend.config.html

Zend Framework 2010h Plugins
[online] [viitattu 23.2.2010]
Saatavissa: http://framework.zend.com/manual/1.10/en/zend.controller.plugins.html

Zend Framework 2010i Zend Auth
[online] [viitattu 2010]
Saatavissa: http://framework.zend.com/manual/en/zend.auth.introduction.html

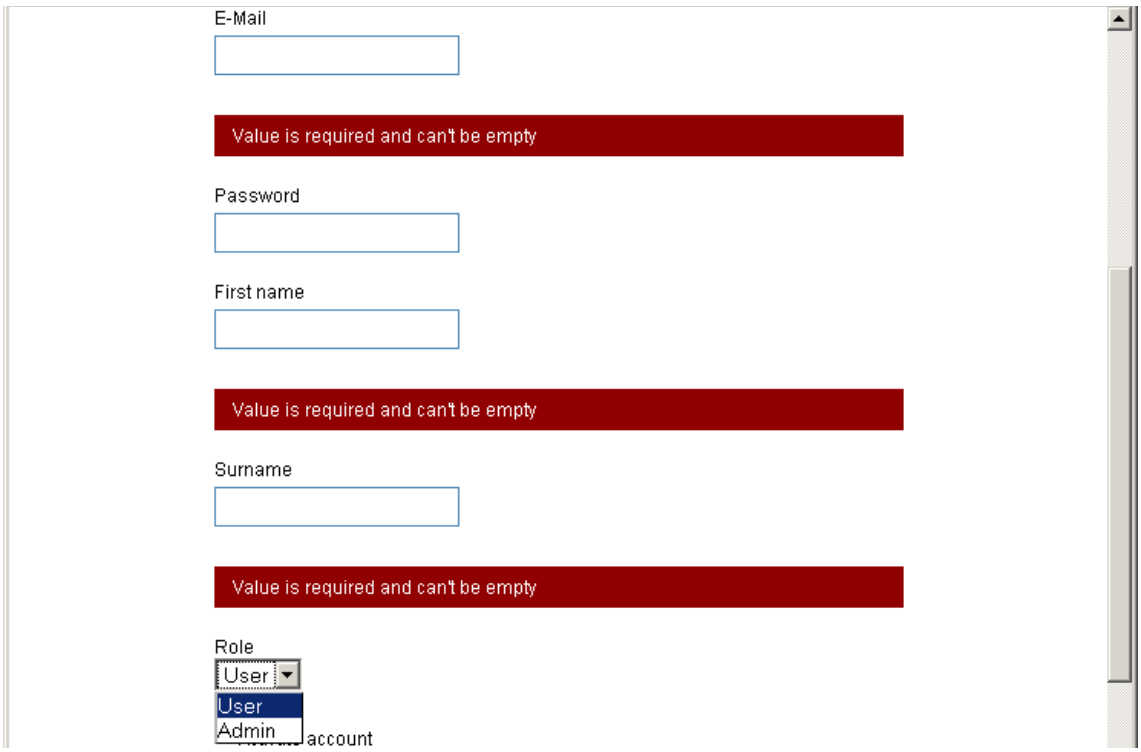
Liite 1: Valmis esimerkkisovellus kuvina



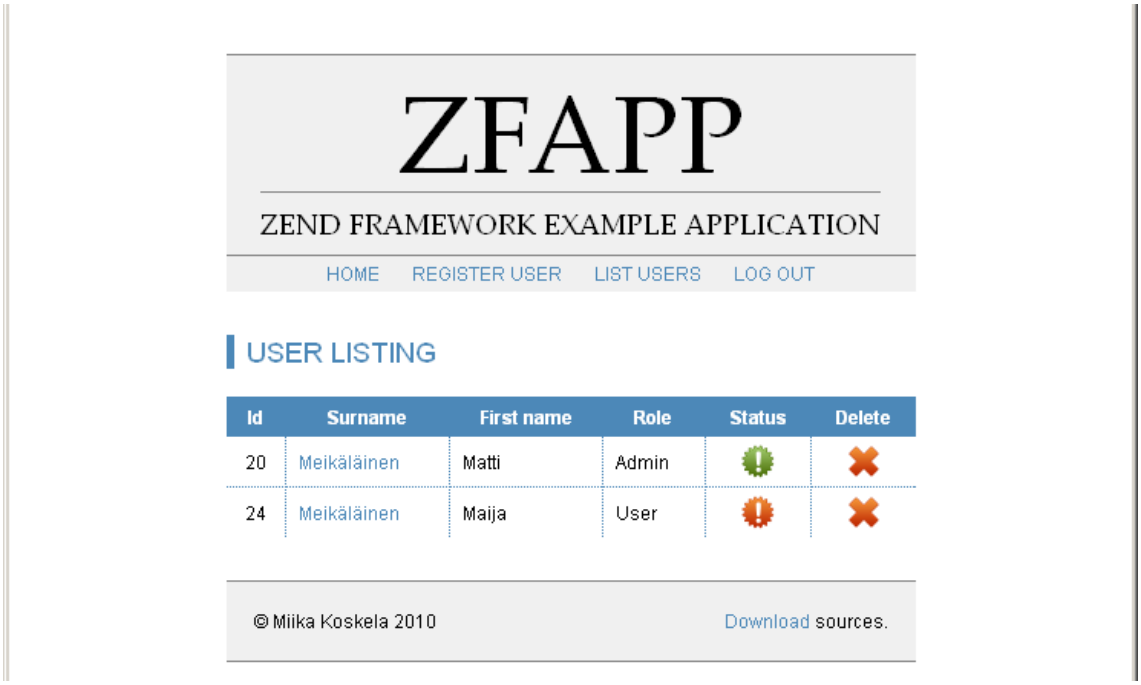
Kuva a: Esimerkkisovelluksen pääsivu. Sovellukseen kirjaudutaan sähköpostiosoite – salasanaparin avulla

Kuva b: Profiilisivu, jonne käyttäjä ohjataan onnistuneen kirjautumisen jälkeen

Käyttäjä voi muokata omia tietojaan. Käyttäjien listaus- ja rekisteröintilinkit näkyvät myös tavallisille käyttäjille, mutta he eivät pääse niistä eteenpäin, vaan heille näytetään ”Access Denied”-viesti.



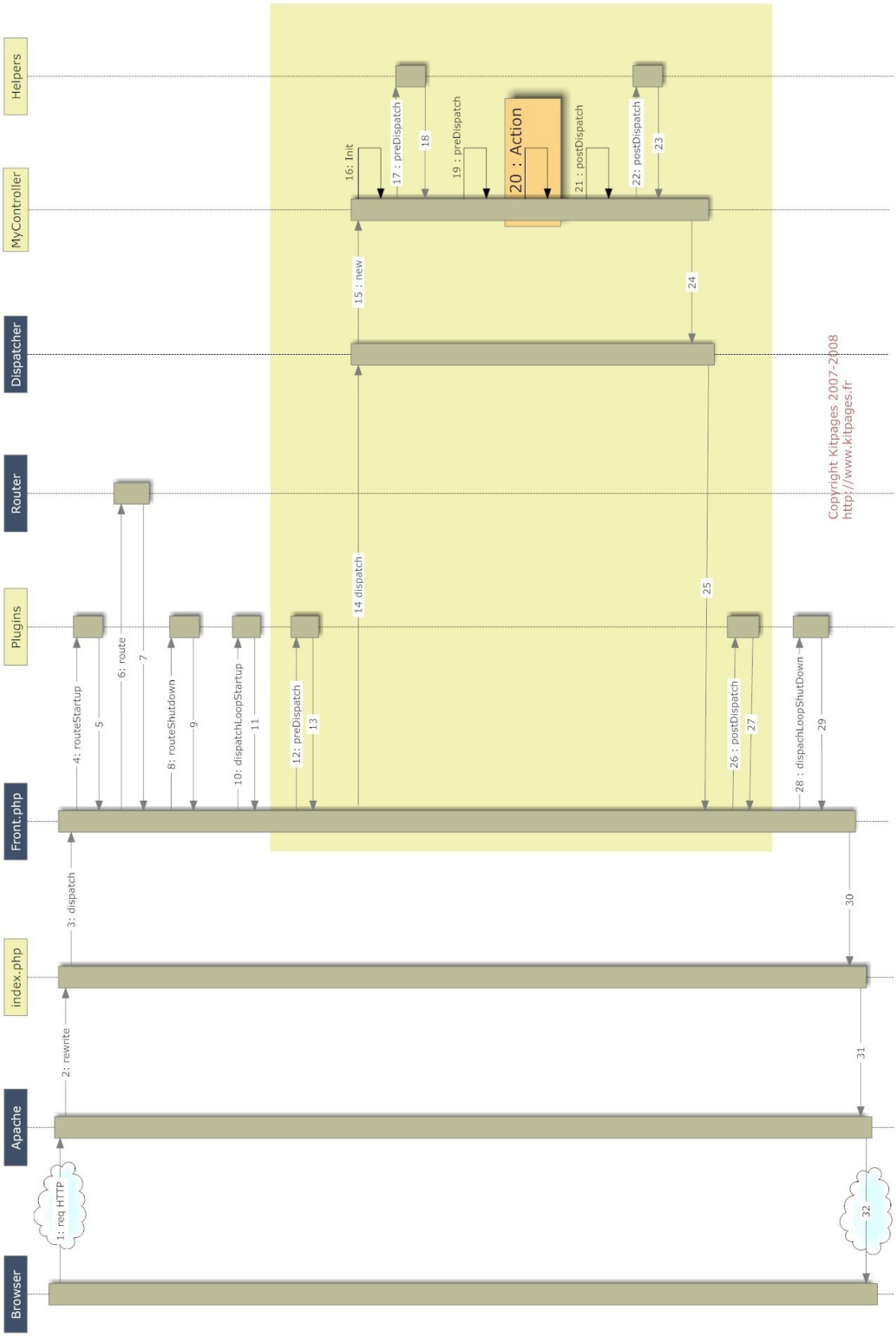
Kuva c: Käyttäjän hallinnointiin tarkoitettu lomake tyhjistä kentistä johtuvien virheilmoitusten kera



Kuva d: Käyttäjien listaus.

Status-kentän vihreä huutomerkki kertoo käyttäjän olevan aktiivinen, kun taas punainen kertoo ei-aktiivisesta tilasta. Tilaa voi vaihtaa klikkaamalla huutomerkkiä, ja käyttäjän voi poistaa klikkaamalla punaista X-kuvaketta.

Liite 2: Sekvenssikaavio



Kuva e: Zend Framework -sovelluksen sekvenssikaavio (Le Van 2008)

Liite 3: CD

Liitteenä olevalla CD-levyllä on tämä opinnäytetyö PDF-muodossa sekä esimerkkisovelluksen lähdekoodit.