Jarno Taipalus

Symfony framework

Opinnäytetyö

Kevät 2009

Tekniikan yksikkö

Tietotekniikka

Ohjelmistotekniikka

Seinäjoen ammattikorkeakoulu
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

OPINNÄYTETYÖN TIIVISTELMÄ

Koulutusyksikkö: Tekniikan yksikkö
Koulutusohjelma: Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto: Ohjelmistotekniikan suuntautumisvaihtoehto

Tekijä: Jarno Taipalus

Työn nimi: Symfony framework

Ohjaaja: Petteri Mäkelä

Vuosi: 2009 Sivumäärä: 59 Liitteiden lukumäärä: 0

Tämä opinnäytetyö esittelee PHP-ohjelmistokehykset yleisesti, Symfony-ohjelmistokehyksen tarkemmin ja erityisesti sen MVC-arkkitehtuurin, asennuksen, käyttöönoton ja konfiguroinnin. Tämän työn lopussa tehdään esimerkkisovellus ilman kehystä ja sen avulla, näin nähdään kehyksen antamat edut. Työn tavoitteena oli esitellä PHP-ohjelmistokehykset ja erityisesti Symfony-ohjelmistokehys sekä sen käyttämät tekniikat.

Ohjelmistokehys helpottaa ohjelmistokehitystä sisältämällä ratkaisut yleisimpiin useasti toistuviin tehtäviin. Kehys tarjoaa valmiin rakenteen sitä ympäröivälle koodille, mikä auttaa ohjelmoijaa kirjoittamaan parempaa, luettavampaa ja helpommin hallittavaa koodia. Kehyksen avulla suunnittelija voi suorittaa monimutkaisia tehtäviä yksinkertaisilla kutsuilla.

Symfony on isoja projekteja varten tehty ohjelmistokehys PHP-kielelle, joka on suunniteltu helpottamaan web-sovellusten kehitystä. Symfony käyttää MVC-arkkitehtuuria eli se erittelee web-sovelluksen koodin ulkoasun (view), logiikan (controller) ja tiedonkäsittelyn (model) kerroksien mukaan. Model layer -kerroksessa säilytetään tietokantaan liittyvä koodi. View layer -kerroksessa säilytetään käyttäjälle näytettävän sisällön koodi. Controller layer -kerroksessa säilytetään koodia, joka ohjaa kahden edellisen kerroksen toimintaa.

Jos PHP-suunnittelija on kehittämässä sivustoa, jossa tulee olemaan yli kymmenen eri sivua, suuri tietokanta, paljon logiikkaa ja tarve käyttäjähallinnalle, niin projekti kannattaa tehdä ohjelmistokehystä käyttämällä.

Asiasanat: PHP, ohjelmointi, ohjelmointiympäristö

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology
Degree programme: Information Technology
Specialisation: Software Engineering

Author: Jarno Taipalus

Title of the thesis: Symfony framework

Tutor: Petteri Mäkelä

Year: 2009 Number of pages: 59 Number of appendices: 0

This thesis presents PHP frameworks, more specifically Symfony framework and especially its MVC architecture, installation, initialization and configuration. There is an example application in the end of this thesis with and without framework so we can see the benefits that the framework gives. The aim of this thesis is to present PHP frameworks and especially Symfony framework and the techniques that it uses.

Framework makes it easier to develop web applications by offering solutions to most common tasks. Framework offers structure for code which helps the developer to write better, more reliable and more controllable code. Furthermore it wraps up complicated features into simple calls, making programming easier.

Symfony is a framework for big PHP projects, designed to help with the development of web applications. Symfony uses MVC architecture which means that it categorizes code into a View layer, a Controller layer and a Model layer. The Model layer contains code that is related to the database. The View layer contains code that is related to the visual style that is shown to a user. The Controller layer contains code which controls the actions that the two previous layers have.

If PHP developer is developing a web site that will contain more than 10 different pages, a big database, a lot of logic and need for user control, then that project should be done using a framework. If the web site would contain less than 10 pages and does not contain a lot of logic, then that project should be done without a framework.

Keywords: PHP, programming, programming environment

SISÄLLYS

KÄYTETYT TERMIT JA LYHENTEET

Symfony	Ohjelmistokehys PHP-kielelle.
PHP	Web-ohjelmointikieli, joka suoritetaan palvelimella.
Javascript	Web-ohjelmointikieli, joka suoritetaan web-sivun käyttäjän koneella.
LAMP	Palvelimen asennuspaketti Linux-käyttöjärjestelmälle.
WAMP	Palvelimen asennuspaketti Windows-käyttöjärjestelmälle.
Apache	Palvelinohjelma.
MySQL	SQL-tietokanta.
SQL	Tietokantakieli.
URL	Web-sivun osoite.
HTML	Web-ohjelmointikieli.

(Lieska tuotanto, 2009.)

1 JOHDANTO

1.1 Työn tausta

Tämän työn tekijä on ollut aloittamassa suurta PHP-kielellä toteutettua web-sovellusta, joka tuli espoolaisen yrityksen omaan käyttöön sekä myyntiin. Usein yrityksille tehtävät suuret sovellukset kasvavat niin laajoiksi, että niiden ylläpidettävyys ja jatkokehitys tulevat todella vaikeiksi. Ohjelmistokehykset on suunniteltu pitämään suurienkin sovellusten ylläpidettävyys ja jatkokehitys helppona. Sovellus tehtiin Symfony-ohjelmistokehyksellä.

1.2 Työn tavoitteet

Työn tavoitteena on tehdä tästä työstä mielenkiintoinen kuvaus Symfony-ohjelmistokehyksestä, jonka luettuaan lukija haluaisi aloittaa tekemään sen avulla omia web-sovelluksia.

1.3 Työn sisältö

Luvussa 2 kerrotaan PHP-ohjelmistokehyksistä yleisesti. Luvussa 3 kerrotaan Symfony-ohjelmistokehyksestä yleisesti. Luvussa 4 kerrotaan Symfony-ohjelmistokehyksen MVC-arkkitehtuurista. Luvussa 5 kerrotaan kehyksen asentamisesta. Luvussa 6 kerrotaan kehyksen käyttöönotosta. Luvussa 7 kerrotaan kehyksen konfiguroinnista. Luvussa 8 kerrotaan esimerkkisovelluksen tekemisestä ilman kehystä ja kehyksen avulla. Luvussa 9 kerrotaan tämän työn johtopäätökset.

2 PHP-OHJELMISTOKEHYKSET LYHYESTI

Aluksi www-sivut olivat vain HTML-kieltä sisältäviä sivuja, joihin vain sivujen tekijä pystyi lisäämään sisältöä koodin kautta. Web-sivujen suunnittelijat huomasivat kuitenkin tarpeen käyttäjien syöttämälle sisällölle. Dynaamisille web-sivuja, jotka mahdollistaisivat käyttäjien sisällön lisäämisen, kehitettiin ohjelmointikieliä kuten esimerkiksi PHP. Suurien sivustojen tekeminen PHP-kielellä oli kuitenkin hankalaa, koska koodia tuli niin paljon, että sivustojen ylläpito ja jatkokehitys hankaloitui. Tätä ongelmaa ratkaisemaan kehitettiin PHP-ohjelmistokehykset. (Wikipedia 2009.)

Ohjelmistokehys helpottaa sovellusten kehitystä sisältämällä ratkaisut moniin yleisesti toistuviin tehtäviin. Kehys sisältää valmiin rakenteen koodille, mikä auttaa suunnittelijaa kirjoittamaan parempaa, luettavampaa ja helpommin hallittavaa koodia. Kehyksen avulla suunnittelija voi suorittaa monimutkaisia tehtäviä yksinkertaisilla kutsuilla. (Potencier & Zaninotto 2005, 3.)

Yleensä ohjelmistokehykset vähentävät web-sovelluksen suunnittelijan tarvetta tehdä samalle operaatiolle koodia moneen kertaan. Kehykset helpottavat suunnittelijan työtä yksinkertaistamalla komentoja, joilla esimerkiksi otetaan yhteys tietokantaan. Usein kehykset helpottavat Javascript-kielellä tehtävien AJAX-toimintojen tekemistä avustajafunktioiden avulla. Monet PHP-ohjelmistokehykset sisältävät Javascript-ohjelmistokehyksen nimeltään Prototype avustajafunktioiden toteuttamiseen. (Potencier & Zaninotto 2005, 3.)

Seuraavissa luvuissa esitellään viisi suosituimpaa PHP-ohjelmistokehystä, jotka perustuvat Ruby-kielelle tehtyyn Ruby on Rails -ohjelmistokehykseen ja MVC-arkkitehtuuriin. Ruby on Rails -kehystä on käytetty näiden ohjelmistokehyksien mallina, koska käyttämä tekniikka on todettu hyväksi ja helpoksi.

Ruby on Rails -kehys perustuu ”Convention over Configuration” ja “Don’t Repeat Yourself” -ideoihin. “Convention over Configuration” -idealla tarkoitetaan sitä, että ohjelmistokehys tekee automaattisesti joitain tehtäviä, joten suunnittelijan ei tarvitse käyttää niin paljon aikaa kehityksessä. ”Don’t Repeat Yourself” -idealla tarkoitetaan sitä, että ohjelmistokehys vähentää suunnittelijan tarvetta koodata samaa asiaa moneen kertaan. (Ruby on Rails 2009.)

2.1 CakePHP

CakePHP-ohjelmistokehyksestä löytyy paljon ominaisuuksia, mutta se on silti helppo käyttää ja nopea. Kehys tukee PHP-kielen 4 ja 5 versioita. Se vaatii suunnittelijalta hyvin vähän konfigurointia. Kehys sisältää AJAX- ja Javascript -avustajafunktioita. Huonona puolena on se, että kehyksen dokumentoinnissa on puutteita ja tuki monelle eri kielelle puuttuu. (Cevasco 2006.)

2.2 Symfony

Symfony-ohjelmistokehyksestä löytyy kaikista PHP-kehyksistä eniten ominaisuuksia. Kehyksellä on laaja dokumentaatio ja sillä on suuri yhteisö, josta suunnittelija voi kysyä apua. Kehyksessä on tuki monelle eri kielelle. Huonona puolena on, että kehys on hidas ja tukee ainoastaan PHP-kielen versiota 5. (Cevasco 2006.)

2.3 PHP on Trax

PHP on Trax -ohjelmistokehys on PHP-kehyksistä kaikkein lähinnä Ruby on Rails -ohjelmistokehystä. Kehys on yksinkertainen ja vaatii suunnittelijalta hyvin vähän konfigurointia. Huonona puolena on se, että kehys tukee ainoastaan PHP-kielen versiota 5 ja dokumentaatiossa on paljon puutteita. (Cevasco 2006.)

2.4 Code Igniter

Code Igniter -ohjelmistokehys sopii PHP-kehyksistä parhaiten pienempiin web-sovelluksiin helpon käytettävyyden ja vähäisen konfigurointinsa ansiosta. Kehys tukee PHP-kielen versioita 4 ja 5. Huonona puolena on se, että kehyksen dokumentaatio on heikkoa. (Cevasco 2006.)

2.5 Zend framework

Zend-ohjelmistokehys on PHP-kielen tekijöiden tekemä ohjelmistokehys. Kehyksellä on laaja dokumentaatio ja suuri yhteisö. Kehys sisältää paljon ominaisuuksia, mutta on silti helppo käyttää ja nopea. Kehys on PHP-kehyksistä kaikkein suosituin. Huonona puolena on se, että kehys tukee ainoastaan PHP-kielen versiota 5. (Cevasco 2006.)

3 SYMFONY LYHYESTI

3.1 Symfony-ohjelmistokehyksestä yleisesti

Symfony on ohjelmointikehys, joka on suunniteltu helpottamaan web-sovellusten kehitystä muutamien pääominaisuuksiensa avulla. Symfony erittelee web-sovelluksen ulkoasun (view), logiikan (controller) ja tiedonkäsittelyn (model) kolmeen eri kerrokseen. Kehys sisältää lukuisia työkaluja ja luokkia, jotka on tarkoitettu lyhentämään monimutkaisten ohjelmistojen kehitysaikaa. Lisäksi se automatisoi yleisiä toimintoja, jotta ohjelmoija voi keskittyä täydellä teholla oman ohjelmansa erityispiirteisiin. Lopputuloksena kehittäjän ei siis tarvitse keksiä pyörää uudelleen jokaista uutta sovellusta rakentaessaan. (Potencier & Zaninotto 2005, 3.)

Symfony on kirjoitettu kokonaan PHP 5 -kielellä. Se on testattu kokonaisuudessaan useissa oikeissa projekteissa, ja on käytössä monissa vaativissa yritysratkaisuissa. Symfony on yhteensopiva useimpien saatavilla olevien tietokantojen kanssa mukaanlukien MySQL, PostgreSQL, Oracle ja Microsoft SQL. Käyttöjärjestelmätuki on Linux- ja Windows-alustoille. (Potencier & Zaninotto 2005, 3.)

3.2 Symfony-ohjelmistokehyksen ominaisuudet

Symfony on helppo asentaa ja konfiguroida, se on riippumaton tietokannasta, yksinkertainen käyttää, konfigurointitarve on vähäinen, se sisältää helppolukuista koodia ja sitä on helppo laajentaa plugin-ohjelmilla. (Potencier & Zaninotto 2005, 3.)

Symfony-ohjelmistokehykseen on automatisoitu muutamia isoissa projekteissa käytettyjä ominaisuuksia, kuten sisäänrakennettu kansainvälisyystuki, avustajafuntiot, lomakkeet, jotka tukevat tiedontarkistusta ja uudelleentäyttöä, välimuistiominaisuudet, jotka madaltavat palvelimen kuormitusta, käyttäjätunnistus, URL-osoitteiden reititys, listat joissa on sivutus ja ajax-toimintojen avustajafunktiot. (Potencier & Zaninotto 2005, 3.)

Täyttääkseen yritysten kovat vaatimukset Symfony on täysin muokattavissa. Koodin generointi mahdollistaa backend-toimintojen luonnin vaivattomasti. Backend-toiminnoilla tarkoitetaan käyttäjähallintaa ja sivujen ylläpitoa helpottavia toimintoja. Sisäänrakennetut yksikkö- ja toiminnallisuustestaustyökalut auttavat sovelluksen testauksessa. Debug-paneeli auttaa kehittäjää näyttämällä tietoja sivun toiminnasta. Lokitiedostot näyttävät ylläpitäjälle tietoja ohjelman toiminnasta. (Potencier & Zaninotto 2005, 3.)

3.3 Kuka teki Symfony-ohjelmistokehyksen ja miksi?

Ensimmäinen versio Symfony-ohjelmistokehyksestä julkaistiin lokakuussa 2005 ja sen julkaisi kehyksen perustaja Fabien Potencier. Potencier on Sensio-nimisen yrityksen toimitusjohtaja. Sensio on ranskalainen yritys, joka tunnetaan web-sivujen kehityksestä. (Potencier & Zaninotto 2005, 4.)

Kun PHP5 julkaistiin Potencier päätti ryhtyä tekemään Symfony-ohjelmistokehyksen ydintä, joka tulisi perustumaan Model-View-Controller (MVC) -arkkitehtuuriin, Propel object-relational mapping (ORM) -tekniikkaan ja Ruby on Rails -ohjelmointikieleen. (Potencier & Zaninotto 2005, 4.)

Potencier kehitti Symfony-kehyksen alunperin Sension projekteja varten, koska se auttaa ohjelmistokehittäjää tekemään ohjelmia nopeammin ja tehokkaammin. Muutaman Symfony-kehyksellä tehdyn onnistuneen projektin jälkeen Potencier päätti julkaista kehittämänsä ohjelmistokehyksen avoimen lähdekoodin lisenssillä internetiin kaikkien saataville. (Potencier & Zaninotto 2005, 4.)

Tullakseen menestyväksi avoimen lähdekoodin projektiksi Symfony tarvitsi laajan englanninkielisen dokumentaation, joka herättäisi ihmisten mielenkiinnon. Potencier pyysi Sensiossa työskentelevää kollegaansa Francois Zaninottoa tarkastelemaan Symfonyn koodia ja kirjoittamaan siitä kirjan. Kirjan kirjoittamiseen meni aikaa, mutta kun projekti vihdoin julkaistiin, se oli dokumentoitu tarpeeksi hyvin ja muutkin kehittäjät innostuivat Symfony-kehyksestä. (Potencier & Zaninotto 2005, 4.)

3.4 Symfony-ohjelmistokehyksen yhteisö

Symfony-ohjelmistokehyksen kotisivun julkaisun jälkeen web-sovellusten kehittäjät ympäri maailmaa kiinnostuivat tästä kehyksestä ja aloittivat kehittämään sovelluksiaan kehyksen avulla. Internetissä viesti uudesta ohjelmistokehyksestä levisi nopeasti. (Potencier & Zaninotto 2005, 5.)

Ohjelmistokehykset sovelluksille alkoivat saada suurta suosiota ja tarve hyvälle ohjelmistokehykselle PHP:ssa oli suuri. Symfony tarjosi hyvän ratkaisun, koska sen koodi oli laadukasta ja dokumentointi runsasta. Varsinkin hyvä dokumentointi oli suuri etu muihin PHP-ohjelmistokehyksiin verrattuna. Symfony-kehyksen

kehittäjillä oli paljon avustajia, jotka ehdottivat korjauksia ja parannuksia. (Potencier & Zaninotto 2005, 5.)

Julkinen lähdekoodi ja projektinseurantajärjestelmä tarjoavat mahdollisuuden auttaa Symfony-kehyksen kehityksessä. Potencier toimii yhä kehyksen pääkehittäjänä ja takaa koodin hyvän laadun. Nykyään kehyksellä on oma foorumi, postituslista ja IRC-kanava, joissa käyttäjät voivat kysyä toisiltaan apua tai auttaa muita. (Potencier & Zaninotto 2005, 5.)

3.5 Kenelle Symfony-ohjelmistokehys on tehty?

Symfony-kehystä voivat käyttää kokeneet sekä aloittelevat PHP-suunnittelijat. Suurin valintakriteeri Symfony-kehyksen käyttämiseen on projektin koko. (Potencier & Zaninotto 2005, 5.)

Jos PHP-suunnittelija on tekemässä web-sivustoa, jossa tulee olemaan alle kymmenen eri sivua, pieni tietokanta tai ei tietokantaa ollenkaan ja ilman käyttäjähallintaa niin projekti kannattaa tehdä normaalia PHP:ta käyttämällä. Kehyksestä ei näin pienessä projektissa olisi mitään hyötyä ja se ainoastaan hidastaisi sivuston tekemistä. (Potencier & Zaninotto 2005, 5.)

Toisaalta jos PHP-suunnittelija on kehittämässä sivustoa, jossa tulee olemaan yli kymmenen eri sivua, suuri tietokanta, paljon logiikkaa ja tarve käyttäjähallinnalle niin projekti kannattaa tehdä ohjelmistokehystä käyttämällä. Usein tämän kokoiset sivustot tehdään jollekin yritykselle, joka myös luultavasti haluaa myöhemmin lisätä sivustolle uusia ominaisuuksia. (Potencier & Zaninotto 2005, 5.)

4 SYMFONY-OHJELMISTOKEHYKSEN MVC-ARKKITEHTUURI

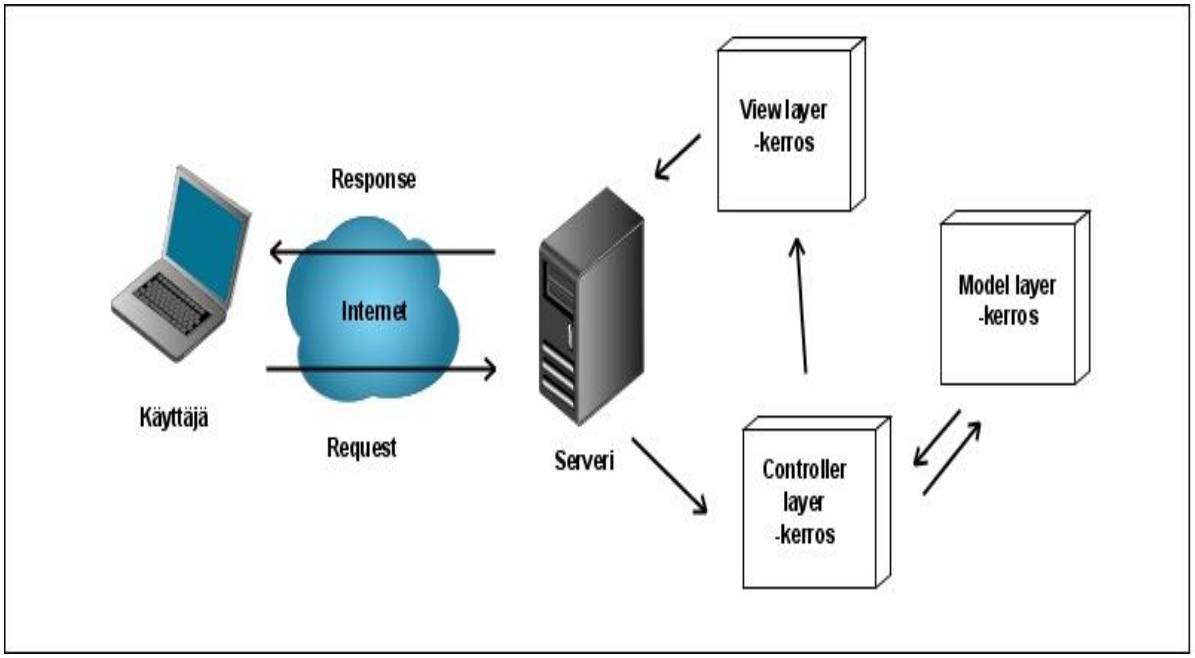
Symfony käyttää MVC-arkkitehtuuria, jossa koodi jaetaan kolmeen eri kerrokseen koodin tarkoituksen mukaisesti. Seuraavassa kerrotaan MVC-arkkitehtuurista ja tehdään koodiesimerkit ilman MVC-mallia ja sen avulla.

4.1 MVC-malli

Symfony perustuu web-suunnittelumalliin joka tunnetaan nimellä MVC-arkkitehtuuri. Se sisältää kolme kerrosta:

- Model layer (Tiedonkäsittely) -kerroksessa säilytetään tietokantaan liittyvä koodi.
- View layer (Ulkoasu) -kerroksessa säilytetään käyttäjälle näytettävän sisällön koodi.
- Controller layer (Logiikka) -kerroksessa säilytetään koodi, joka ohjaa kahden edellisen kerroksen toimintaa. (Potencier & Zaninotto 2005, 10.)

MVC-arkkitehtuuri erottelee nämä kolme kerrosta, mikä takaa koodin helpomman ylläpidettävyyden. Jos esimerkiksi web-sivu täytyy saada toimimaan pc:n selaimilla sekä puhelimien selaimilla, suunnittelijan täytyy vain tehdä erilliset ulkoasutiedostot pc:n selaimille ja puhelimien selaimille. Jos täytyy siirtyä käyttämään erilaista tietokantaa kuin aiemmin on käytetty, suunnittelijan tarvitsee vain vaihtaa tiedonkäsittelytiedostot. (Potencier & Zaninotto 2005, 10.)



Kuva 1. MVC-malli

4.2 Koodiesimerkki ilman MVC -arkkitehtuuria

Esimerkki normaalin web-sivun koodista, jossa otetaan yhteys tietokantaan, suoritetaan SQL-kysely, näytetään sivulla kyselyn tulokset ja suljetaan yhteys tietokantaan, ilman MVC -arkkitehtuuria.

Esimerkki 1. Koodiesimerkki ilman MVC -arkkitehtuuria

```
<?PHP

// Yhdistetään tietokantaan
$yhteys = mysql_connect('palvelin', 'kayttaja', 'salasana');
mysql_select_db('tietokanta', $yhteys);

// SQL kyselyn suorittaminen
$tulokset = mysql_query('SELECT id, nimi FROM henkilot',
$yhteys);

?>

<html>
```

```
  <head>
    <title>Lista henkilöistä</title>
  </head>
  <body>
   <h1>Lista henkilöistä</h1>
   <table>
     <tr><th>ID</th><th>Nimi</th></tr>
<?PHP
// Kyselyn tulosten tulostaminen
while ($rivi = mysql_fetch_array($tulokset, MYSQL_ASSOC))
{
echo "<tr>";
echo “<td> $rivi[‘id’] </td>”;
echo “<td> $rivi[‘nimi’] </td>”;
echo "</tr>";
}
?>
    </table>
  </body>
</html>

<?PHP

// Yhteyden sulkeminen
mysql_close($yhteys);

?>
```

Tällä tavalla koodia on aluksi nopea kirjoittaa ja suorittaa, mutta isommissa projekteissa koodia olisi vaikeaa ylläpitää ja päivittää, koska yhden web-sivun taakse saattaa tulla monta tuhatta riviä koodia tällä tavalla kirjoitettuna. (Potencier & Zaninotto 2005, 11.)

Suurimmat ongelmat näin kirjoitetussa koodissa ovat, että niissä ei ole minkäänlaista virheentarkistusta, HTML- ja PHP-koodi ovat sekaisin ja koodi on sidottu tiettyyn MySQL-tietokantaan. (Potencier & Zaninotto 2005, 11.)

4.3 Koodiesimerkki MVC -arkkitehtuurissa

Seuraavissa kappaleissa tehdään esimerkki koodista, jossa edellisen esimerkin ilman MVC-arkkitehtuuria tehty koodi jaetaan MVC-arkkitehtuurin mukaisesti kolmeen eri kerrokseen.

4.3.1 Model layer (Tiedonkäsittely) -kerros

Esimerkki koodista Model layer -kerroksessa, jolla otetaan yhteys tietokantaan, suoritetaan SQL-kysely, siirretään tulokset taulukkoon ja suljetaan yhteys tietokantaan.

Esimerkki 2. Koodi Model layer -kerroksessa

```
<?PHP

function getKaikkiHenkilot()
{
  // Otetaan yhteys tietokantaan
  $yhteys = mysql_connect('palvelin', 'kayttaja',
'salasana');
  mysql_select_db('tietokanta', $yhteys);

  // Suoritetaan SQL-kysely
  $tulokset = mysql_query('SELECT id, nimi FROM henkilot',
$yhteys);

  // Siirretään tulokset taulukkoon
  $henkilot = array();
  while ($rivi = mysql_fetch_array($tulokset, MYSQL_ASSOC))
  {
     $henkilot[] = $rivi;
  }

  // Suljetaan yhteys tietokantaan
  mysql_close($yhteys);

  // Palautetaan henkilot taulukko
  return $henkilot;
}
```

Tiedonkäsittelykerros koostuu yleensä tämän tapaisista metodeista, joissa suoritetaan tietty SQL-kysely, laitetaan kyselyn tulokset taulukkoon ja palautetaan tämä taulukko controllerille, joka tätä kyseistä metodia kutsui. Näitä metodeja voi kutsua missä tahansa controllerissa, joten tämä ominaisuus poistaa tarpeen kirjoittaa samaa kyselyä kahteen eri paikkaan. (Potencier & Zaninotto 2005, 13.)

4.3.2 Controller layer (Logiikka) -kerros

Esimerkki koodista Controller layer -kerroksessa, jolla otetaan yhteys tietokantaan, suoritetaan SQL-kysely, siirretään tulokset taulukkoon, suljetaan yhteys tietokantaan ja siirrytään View layer -kerrokseen.

Esimerkki 3. Koodi Controller layer -kerroksessa

```
<?PHP

 // Otetaan yhteys tietokantaan
 $yhteys = mysql_connect('palvelin', 'kayttaja', 'salasana');
 mysql_select_db('tietokanta', $yhteys);

 // Suoritetaan SQL-kysely
 $tulokset = mysql_query('SELECT id, nimi FROM henkilot',
$yhteys);

 // Siirretään tulokset taulukkoon
 $henkilot = array();
 while ($rivi = mysql_fetch_array($tulokset, MYSQL_ASSOC))
 {
    $henkilot[] = $rivi;
 }

 // Suljetaan yhteys tietokantaan
 mysql_close($yhteys);

 // Siirrytään View layer-kerrokseen
 require('view.php');
```

Model layer -kerroksen ansiosta tämä voidaan kuitenkin tiivistää hyvin vähäiseen koodirivien määrään.

Esimerkki 4. Koodi Controller layer -kerroksessa tiivistettynä

```
<?PHP

// Sisällytetään Model layer-kerros
require_once('model.php');

// Haetaan henkilot taulukko
$henkilot = getKaikkiHenkilot();

// Sisällytetään View layer-kerros
require('view.php');
```

Controller layer -kerros toimii siis Model layer -kerroksen ja View layer -kerroksen välissä, yleensä välittäen Model layer -kerroksessa tehtyjen tietokantakyselyjen tulokset View layer -kerrokselle. Suurimmissa projekteissa tulee juuri tähän kerrokseen paljon koodia, ja on tärkeää että tämän kerroksen koodi pysyy helposti luettavana ja muokattavana. (Potencier & Zaninotto 2005, 12.)

4.3.3 View layer (Ulkoasu) -kerros

Esimerkki koodista View layer -kerroksessa, jolla näytetään web-sivulla Model layer -kerroksessa tehdyn kyselyn tulokset.

Esimerkki 5. Koodi View layer -kerroksessa

```
<html>
  <head>
    <title>Lista henkilöistä</title>
  </head>
  <body>
    <h1>Lista henkilöistä</h1>
    <table>
      <tr><th>ID</th><th>Nimi</th></tr>
    <?PHP foreach ($henkilot as $henkilo): ?>
      <tr>
        <td><?PHP echo $henkilo['id'] ?></td>
        <td><?PHP echo $henkilo['nimi'] ?></td>
      </tr>
    <?PHP endforeach; ?>
```

```
    </table>
  </body>
</html>
```

View layer -kerros sisältyy yleensä pelkästään html-koodista ja muutamista PHP-komennoista, joiden avulla tulostetaan taulukko, jossa on Model layer -kerroksen suorittaman tietokantakyselyn tulokset, jotka Controller layer -kerros välitti tähän kerrokseen. View layer-kerroksen sisällä pyritään pitämään PHP-koodin määrä minimissä. (Potencier & Zaninotto 2005, 12.)

4.4 Tarkemmin jaoteltu MVC -arkkitehtuuri

MVC-arkkitehtuurin perusidea on jakaa koodi kolmeen eri kerrokseen. Symfonyssä nämä kolme kerrosta jaetaan vielä eri osiin, mikä edelleen helpottavaa suunnittelijan työtä. (Potencier & Zaninotto 2005, 13.)

Model layer -kerros jaetaan kahteen eri kerrokseen. Ensimmäisessä kerroksessa suoritetaan yhteys tietokantaan (database abstraction layer) ja toisessa kerroksessa suoritetaan tietokantakyselyt (data access layer.) (Potencier & Zaninotto 2005, 13.)

4.4.1 Esimerkki Database abstraction layer -kerroksesta

Esimerkki koodista Database abstraction layer -kerroksessa, jolla otetaan tietokantaan yhteys ja tehdään sielä SQL-kysely dynaamisesti.

Esimerkki 6. Koodi Database abstraction layer -kerroksessa

```
<?PHP

function avaa_yhteys($palvelin, $kayttaja, $salasana)
{
  return mysql_connect($palvelin, $kayttaja, $salasana);
```

```
}

function sulje_yhteys($yhteys)
{
  mysql_close($yhteys);
}

function suorita_kysely($kysely, $tietokanta, $yhteys)
{
  mysql_select_db($tietokanta, $yhteys);

  return mysql_query($tietokanta, $yhteys);
}

function hae_tulokset($tulokset)
{
  return mysql_fetch_array($tulokset, MYSQL_ASSOC);
}
```

4.4.2 Esimerkki Data access layer -kerroksesta

Esimerkki koodista Data access layer -kerroksessa, jossa käytetään database abstraction layer -kerroksen metodeja ja laitetaan SQL-kyselyllä saadut tulokset taulukkoon.

Esimerkki 7. Koodi Data access layer -kerroksessa

```
function getKaikkiHenkilot()
{
  // Otetaan yhteys tietokantaan
  $yhteys = avaa_yhteys('localhost', 'jarno', 'kalakukko');

  // Suoritetaan SQL-kysely
  $tulokset = suorita_kysely('SELECT id, nimi FROM henkilot',
'tietokanta', $yhteys);

  // Siirretään saadut tulokset taulukkoon
  $henkilot = array();
  while ($rivi = hae_tulokset($tulokset))
  {
     $henkilot[] = $rivi;
  }
```

```
  // Suljetaan yhteys tietokantaan
  sulje_yhteys($yhteys);

  return $henkilot;
}
```

View layer -kerros jaetaan osaan, jossa tehdään sivulle yleinen ulkoasu (Layout part), osaan jossa annetaan sivulle sisältöä (Template part), ja osaan jossa määritellään edellisten osien logiikka (View logic part). (Potencier & Zaninotto 2005, 14.)

4.4.3 Esimerkki Layout part -osasta

Esimerkki koodista Layout part -osassa, jolla tehdään web-sivulle ulkoasu.

Esimerkki 8. Koodi Layout part -osassa

```
<html>
  <head>
    <title><?PHP echo $title ?></title>
  </head>
  <body>
    <?PHP echo $content ?>
  </body>
</html>
```

4.4.4 Esimerkki Template part -osasta

Esimerkki koodista Template part -osassa, jolla laitetaan web-sivulle sisältöä.

Esimerkki 9. Koodi Template part -osassa

```
<h1>Lista henkilöistä</h1>
<table>
<tr><th>ID</th><th>Nimi</th></tr>
<?PHP foreach ($henkilot as $henkilo): ?>
  <tr>
```

```
    <td><?PHP echo $henkilo['id']; ?></td>
    <td><?PHP echo $henkilo['nimi']; ?></td>
  </tr>
<?PHP endforeach; ?>
</table>
```

4.4.5 Esimerkki View logic part -osasta

Esimerkki koodista View logic part -osassa, joka toimii edellisten osien luurankona.

Esimerkki 10. Koodi View logic part -osassa

```
<?PHP

$title = 'Lista henkilöistä';
$content = include('mytemplate.php');
```

4.5 Action ja front controller

Controller layer -kerros ei tässä esimerkissä sisältänyt kovinkaan paljon koodia mutta isommissa projekteissa se kasvaa välillä isoksi. Normaalisti sielä suoritetaan käyttäjän toimintapyynnöt, tietoturva, web-sivun asetusten lataaminen ja muut vastaavat toiminnot. Näiden käsittelyä helpotetaan jakamalla Controller layer -kerros osaan, jossa käsitellään tietoturva sekä asetukset (front controller) ja osaan, jossa käsitellään yksilöllisen sivun logiikan koodi (action). Front controller tarjoaa helpon tavan muuttaa sivujen yleisimpiä asetuksia, koska asetukset täytyy muuttaa vain yhteen paikkaan. (Potencier & Zaninotto 2005, 15.)

4.6 Symfony-ohjelmistokehyksen MVC-arkkitehtuuri

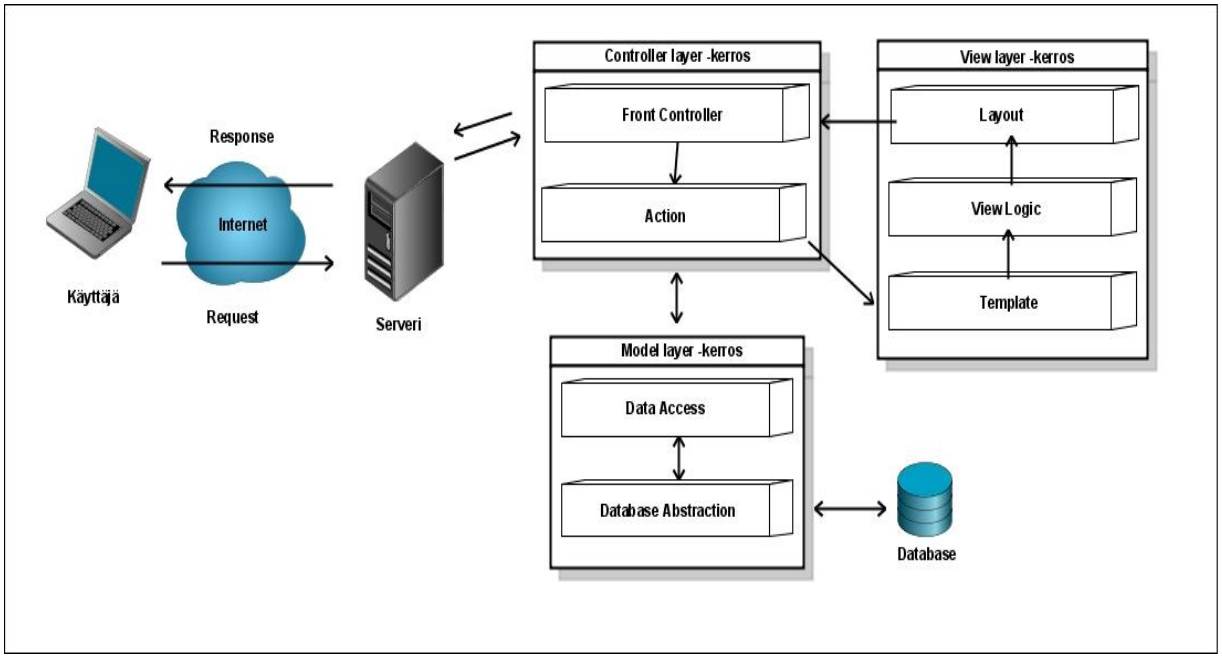
Symfonyssä koodi jaetaan tarkemmin jaotellun MVC-arkkitehtuurin mukaisesti seuraaviin osiin:

- Model layer
  - Database abstraction
  - Data access
- View layer
  - View
  - Template
  - Layout
- Controller layer
  - Front controller
  - Action

Seitsemän tiedostoa saattaa vaikuttaa aika suurelta määrältä, mutta näiden käyttö on nopeaa ja helppoa kun suunnittelija oppii niiden käytön. (Potencier & Zaninotto 2005, 16.)

Front controller ja Layout ovat samoja jokaiselle ohjelman action-osalle. Niitä voi tehdä enemmänkin, mutta vain yksi on pakollinen. Front controller -osaa ei tarvitse edes itse kirjoittaa, koska Symfony luo sen automaattisesti. (Potencier & Zaninotto 2005, 16.)

Symfony luo myös Model layer -kerroksen luokat automaattisesti käytettävän tietokantarakenteen mukaisesti. Tämän toiminnon suorittaa Propel-luokka, joka tekee luokkien koodipohjat. Jos Propel esimerkiksi löytää päivämäärämuotoisen kentän tietokantarakenteesta, se osaa tehdä siihen tarvittavat metodit. (Potencier & Zaninotto 2005, 16.)



Kuva 2. Symfony -ohjelmistokehyksen kierto.

Tehdäkseen ensimmäisen esimerkin kaltaisen sivun käyttäjän tarvitsee itse tehdä vain kolme PHP-tiedostoa:

Action-osa, jossa suoritetaan SQL-kysely, sijaitsee kansiopolussa myproject/apps/myapp/modules/weblog/actions/actions.class.php.

Esimerkki 11. Koodi Action-osassa

```
<?PHP

class weblogActions extends sfActions
{
  public function executeList()
  {
    $this->posts = PostPeer::doSelect(new Criteria());
  }
}
```

Template-osa, jossa käydään läpi kyselystä saadut tulokset ja tulostetaan ne, sijaitsee kansiopolussa modules/weblog/templates/listSuccess.php.

Esimerkki 12. Koodi Template -osassa

```
<?PHP slot('title', 'Lista henkilöistä') ?>

<h1>Lista henkilöistä</h1>
<table>
<tr><th>ID</th><th>Nimi</th></tr>
<?PHP foreach ($henkilot as $henkilo): ?>
  <tr>
    <td><?PHP echo $henkilo->getID() ?></td>
    <td><?PHP echo $henkilo->getNimi() ?></td>
  </tr>
<?PHP endforeach; ?>
</table>
```

Layout-osa, jossa tehdään luuranko edellisille osille, sijaitsee kansiopolussa myproject/apps/myapp/templates/layout.php.

Esimerkki 13. Koodi Layout-osassa

```
<html>
  <head>
    <title><?PHP include_slot('title') ?></title>
  </head>
  <body>
    <?PHP echo $sf_content ?>
  </body>
</html>
```

Layout-tiedostoa ei tarvitse tehdä kuin kerran, koska samaa tiedostoa voidaan käyttää jokaisella sivulla. Muut tarpeelliset tiedostot, jotka saavat nämä tiedostot toimimaan yhdessä, Symfony tekee automaattisesti. Tarvittavien koodirivien määrästä näkee, että Symfonyn MVC-arkkitehtuuri ei vaadi yhtään sen enempää rivejä kuin normaalisti kirjoitettu PHP-koodi. Sitä käyttämällä saa suuria hyötyjä, kuten selkeän koodirakenteen, se poistaa tarpeen koodata samaa asiaa kahteen eri paikkaan, se antaa joustavuutta, tarjoaa debug-ympäristön, se on tietokantariippumaton ja sisältää monia muita etuja. (Potencier & Zaninotto 2005, 17.)

4.7 Symfonyn ydinluokat

MVC-arkkitehtuurin toteutus Symfonyssä käyttää muutamia luokkia, jotka on hyvä tietää:

- sfController tulkitsee käyttäjältä tulleet pyynnöt ja välittää ne action-osille.
- sfRequest säilyttää kaikki request-elementit.
- sfResponse sisältää vastauksen, joka lähetetään käyttäjälle.
- sfContext sisältää asetukset. (Potencier & Zaninotto 2005, 17.)

Kaikki Symfonyn omat luokat käyttävät kirjaimia sf nimensä edessä, joten ne on helppo tunnistaa. (Potencier & Zaninotto 2005, 17.)

4.8 Koodirakenne

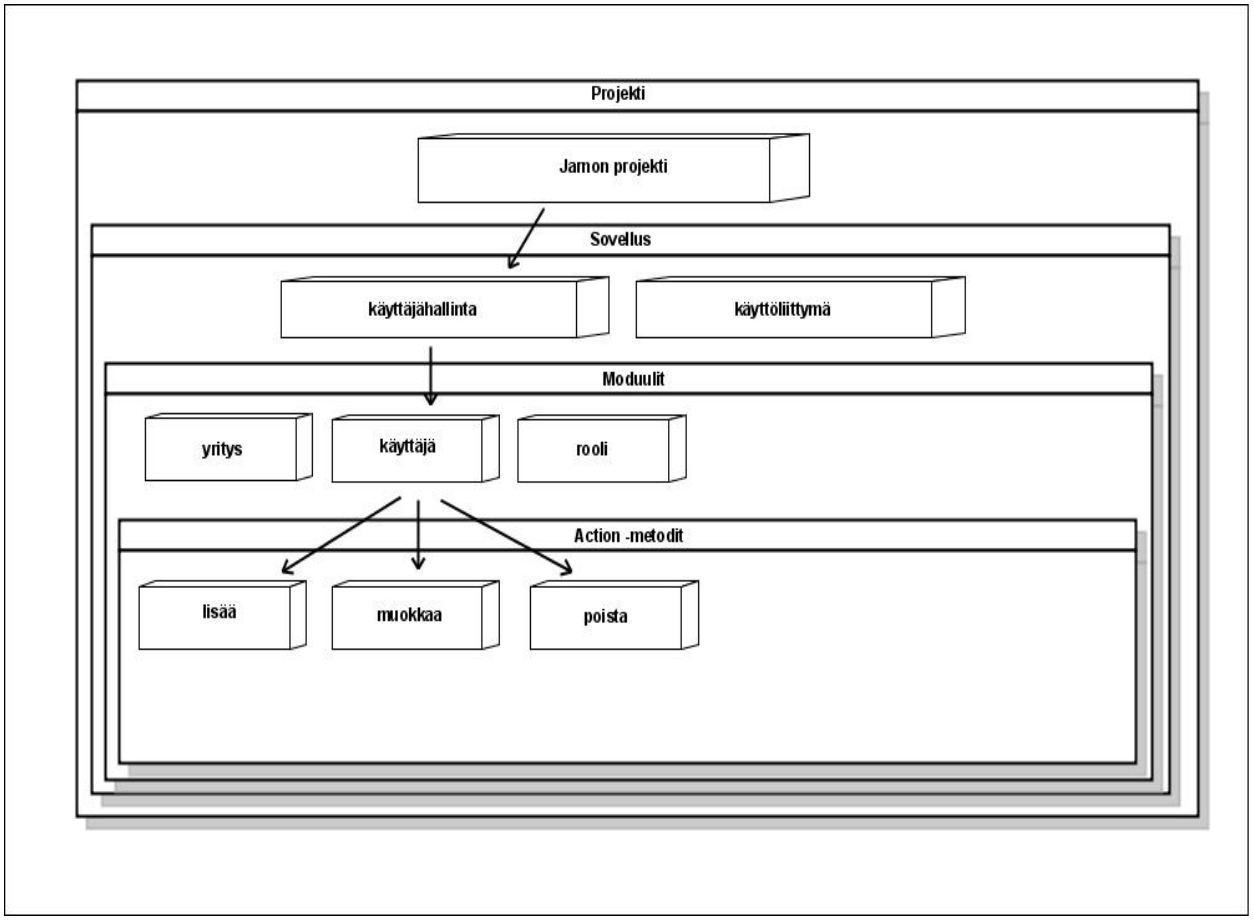
Symfonyssä koodi järjestetään projektin mukaan ja tiedostot projektin alla järjestetään puurakenteella. Tässä luvussa käydään läpi projektin, tiedostojen, juurihakemiston, sovelluksen, moduulin ja web-kansion rakenteet.

4.8.1 Projektin rakenne

Projekti on kokoelma palveluja tietyn domain-nimen alla. Projektin sisällä palvelut on ryhmitelty sovelluksiin. Normaalisti sovellus toimii itsenäisesti, välittämättä muista sovelluksista projektin sisällä. Isommissa projekteissa on yleensä kaksi sovellusta, toinen on itse sovellus ja toinen on sovellukseen liittyvä käyttäjähallinta. (Potencier & Zaninotto 2005, 18.)

Jokaisessa sovelluksessa on yksi tai useampia moduuleita. Moduuli on kokoelma web-sivuja, jotka on tehty jotain tiettyä toimintoa varten. Moduuli sisältää action-osia, joissa on toiminnot, joita käyttäjä voi sivulla tehdä. Esimerkiksi user-nimisellä

moduulilla voisi olla add-, view-, update- ja delete action -osat. (Potencier & Zaninotto 2005, 18.)



Kuva 3. Esimerkki projektin rakenteesta.

4.8.2 Tiedostojen rakenne

Kaikki projektit muodostuvat yleensä samanlaisesta sisällöstä, kuten tietokanta, PHP-tiedostot, loki-tiedostot, asetus-tiedostot, muut tiedostot ja kirjastot. Symfony tarjoaa puurakenteen, jolla nämä tiedostot saadaan loogiseen järjestykseen. Symfony luo puurakenteen automaattisesti jokaiselle projektille. Tämä rakenne on muokattavissa, jos käyttäjä haluaa sitä muuttaa. (Potencier & Zaninotto 2005, 19.)

4.8.3 Juurihakemiston rakenne

Seuraavat kansiot löytyvät normaalin Symfony-projektin juuresta.

```
apps/
  frontend/
  backend/
cache/
config/
data/
  sql/
doc/
lib/
  model/
log/
plugins/
test/
  bootstrap/
  unit/
  functional/
web/
  css/
  images/
  js/
  uploads/
```

Seuraavassa esitellään Symfony-projektin juurihakemiston kansioita ja mitä kansiot sisältävät.

apps/	Sisältää sovellusten kansiot
cache/	Sisältää välimuisti-toiminnossa käytettävät tiedostot. Jokaisella sovelluksella on oma alikansionsa.
config/	Sisältää asetustiedostot.
data/	Sisältää tietokannan schema-tiedoston sekä sql-skriptin joka luo taulut.
doc/	Sisältää projektin dokumentaation.

lib/	Sisältää kolmannen osapuolen luokat ja kirjastot. Tänne voi tehdä luokkia, joita voi käyttää jokaisessa sovelluksessa.
log/	Sisältää lokitiedostot. Symfony luo yhden lokitiedoston jokaista sovellusta ja jokaista ympäristöä varten.
plugins/	Sisältää asennetut pluginit.
test/	Sisältää yksikkö- ja toiminnallisuus-testit. Symfony luo projektia luotaessa muutaman esimerkkitestin.
web/	Sisältää tiedostot, jotka ovat avoimia internettiin.

(Potencier & Zaninotto 2005, 19.)

4.8.4 Sovelluksen hakemistorakenne

Seuraavassa esitellään sovelluksen hakemiston alihakemistot.

```
apps/
  [application name]/
    config/
    i18n/
    lib/
    modules/
    templates/
      layout.php
```

Seuraavassa esitellään sovelluksen hakemiston alihakemistoja ja mitä ne sisältävät.

config/	Sisältää sovelluksen asetustiedostot.

i18n/	Sisältää tiedostot, joita käytetään sovelluksen kansainvälistämiseen.
lib/	Sisältää sovelluksen sisäiset luokat ja kirjastot.
modules/	Sisältää sovelluksen sisäiset moduulit.
templates/	Sisältää graafisen ulkoasun, jota sovelluksen web-sivuilla käytetään.

(Potencier & Zaninotto 2005, 19.)

Sovelluksen luokat eivät pääse käyttämään toisen sovelluksen luokkia ja kirjastoja. Linkit sovelluksesta toiseen pitää kirjoittaa juuresta asti. (Potencier & Zaninotto 2005, 20.)

4.8.5 Moduulin hakemistorakenne

Jokainen sovellus sisältää yhden tai useamman moduulin. Jokaisella moduulilla on oma alihakemistonsa modules-kansiossa.

Seuraavassa esitellään moduulin kansion hakemistorakenne.

```
apps/
  [application name]/
    modules/
      [module name]/
          actions/
            actions.class.php
          config/
          lib/
          templates/
            indexSuccess.php
```

Seuraavassa esitellään moduulin hakemistoja ja mitä ne sisältävät.

actions/	Sisältää yleensä vain yhden tiedoston nimeltä actions.class.php, jossa on kaikki moduulin action-metodien koodit. Käyttäjä voi halutessaan tehdä useamman tiedoston, joihin jakaa moduulin action-metodien koodit.
congfig/	Sisältää moduulin asetustiedostoja.
lib/	Sisältää moduulin luokat ja kirjastot.
templates/	Sisältää moduulin web-sivut.

(Potencier & Zaninotto 2005, 21.)

4.8.6 Web-kansion hakemistorakenne

Web-kansion sisältö on avoin internetiin.

Seuraavassa esitellään web-kansion hakemistorakenne.

```
web/
  css/
  images/
  js/
  uploads/
```

Seuraavassa esitellään web-kansion hakemistoja ja mitä ne sisältävät.

css/	Sisältää .css-päätteiset tyylitiedostot.
images/	Sisältää kuvatiedostot, joita web-sivuilla käytetään.

js/	Sisältää .js-päätteiset javascript-tiedostot.
uploads/	Sisältää web-sivun käyttäjien lähettämiä tiedostoja.

(Potencier & Zaninotto 2005, 19.)

Näiden hakemistorakenteiden pitäminen on suositeltavaa, mutta web-sivujen suunnittelija voi halutessaan muokata näitä rakenteita. (Potencier & Zaninotto 2005, 21.)

5 SYMFONY-JÄRJESTELMÄN ASENNUS

Tässä esimerkissä asennetaan kaikki tarvittavat ohjelmat Symfony-ohjelmistokehyksen käyttämiseen Opensuse 11.0 -ympäristössä.

5.1 Vaatimukset

Synfonyn asennus vaatii muutaman ohjelman, jotka tulee olla asennettuna ennen Symfonyn asennusta:

- Apache2
- MySQL tai joku muu tietokanta
- PHP5
- PEAR helpottaa asennusta, mutta ei ole pakollinen
- PHPMyAdmin-ohjelmalla pystyy hallitsemaan mysql-tietokantaa.

(Potencier & Zaninotto 2005, 19.)

5.2 Apache2-asennus

Asennetaan Apache2 web-palvelin:

```
opensuse11:~ # yast2 -install apache2
```

Tämä komento kertoo että Apache2 on asennettu, mutta ei vielä käytössä:

```
opensuse11:~ # rcapache2 status
Checking for httpd2: unused
```

Tehdään yksinkertainen web-sivu, jolla voidaan testata palvelimen toimintaa:

```
opensuse11:~ # cd /srv/www/htdocs
opensuse11:~ # vi index.html
```

Käynnistetään Apache2 web-palvelin:

```
opensuse11:~ # rcapache2 start
Starting httpd2 (prefork) done
```

Mennään selaimella osoitteeseen http://localhost, siellä pitäisi näkyä tehty web-sivu.

5.3 PHP5-asennus

Asennetaan PHP5-, PHP5-MySQL- ja Apache2-PHP5-paketit, jotka tarvitaan PHP-koodin suorittamiseen palvelimella:

```
opensuse11:~ # yast2 -install PHP5 PHP5-mysql apache2-
mod_PHP5
```

Apache2 täytyy käynnistää uudelleen, jotta muutokset tulevat voimaan:

```
opensuse11:~ # rcapache2 restart
Syntax OK
Shutting down httpd2 (waiting for all children to termi-
nate)          done
Starting httpd2 (prefork) done
```

5.4 MySQL-asennus

Asennetaan MySQL-tietokanta:

```
opensuse11:~ # yast2 -install mysql mysql-tools
```

Suoritetaan komento, jolla nähdään että MySQL on asennettu:

```
opensuse11:~ # rcmysql status
Checking for service
MySQL:                                                  unused
```

Käynnistetään MySQL:

```
opensuse11:~ # rcmysql start
Starting service
MySQL                                                     done
```

Laitetaan MySQL:n root-käyttäjälle salasana:

```
opensuse11:~ # mysqladmin-u root-p rootpassword
```

MySQL:n asennus luo automaattisesti käyttäjän nimeltä root, jolla on oikeudet muokata tietokantaa.

5.5 PHPMyAdmin-asennus

Asennetaan PHPMyAdmin:

```
opensuse11:~ # yast2 -install PHPMyAdmin
```

Mennään selaimella osoitteeseen http://localhost/PHPmyadmin, siellä päästään muokkaamaan MySQL-tietokantaa tämän ohjelman avulla.

5.6 PEAR-asennus

Haetaan go-pear.php-skripti osoitteesta http://go-pear.php.

Ajetaan haettu skripti:

```
PHP go-pear.php
```

5.7 Symfony-asennus

Lisätään Symfonyn sivut PEAR-ohjelmalle:

```
> pear channel-discover pear.symfony-project.com
```

Tällä komennolla näkee kaikki paketit, jotka ovat ladattavissa Symfonyn sivuilta:

```
> pear remote-list-c symfony
```

Asennetaan viimeisin vakaa versio Symfonystä:

```
> pear install symfony/symfony

downloading symfony-1.2.0.tgz ...

Starting to download symfony-1.2.0.tgz (1,283,270 bytes)
```

```
.....................................................................
....

.....................................................................
....

.............done: 1,283,270 bytes

install ok: channel://pear.symfony-project.com/symfony-1.2.0
```

Ajetaan komento, jolla nähdään onko Symfony asentunut:

```
> symfony-V

symfony version 1.2.0 (/path/to/the/pear/symfony/lib/dir)
```

5.8 Apache web-palvelimen konfigurointi

Lisätään uusi virtuaalinen palvelin apache/conf/httpd.conf-tiedostoon:

```
<VirtualHost *:80>
  ServerName myapp.example.com
  DocumentRoot "/home/jta/myproject/web"
  DirectoryIndex index.php
  Alias /sf /$sf_symfony_data_dir/web/sf
  <Directory "/$sf_symfony_data_dir/web/sf">
    AllowOverride All
    Allow from All
  </Directory>
  <Directory "/home/jta/myproject/web">
    AllowOverride All
    Allow from All
  </Directory>
</VirtualHost>
```

Käynnistetään Apache uudelleen ja mennään selaimella osoitteeseen http://localhost/frontend_dev.php/ josta pitäisi nyt näkyä Symfony-sovelluksen index.php-nimisen tiedoston sisältö.

6 SYMFONY-JÄRJESTELMÄN KÄYTTÖÖNOTTO

Seuraavassa eritellään, kuinka Symfony-ohjelmistokehyksessä luodaan projekti, projektille moduuli ja moduulille action ja template. Näytetään myös esimerkki kuinka linkitys toisiin action-metodeihin toimii ja kuinka request-toimintoa käytetään.

6.1 Projektin luominen

Suoritetaan komento, joka luo projektin ja sille kaikki tarvittavat tiedostot ja hakemistot:

```
> mkdir ~/myproject

> cd ~/myproject

> symfony generate:project myproject
```

(Potencier & Zaninotto 2005, 28.)

6.2 Sovelluksen luominen

Projekti tarvitsee vähintään yhden sovelluksen. Suoritetaan komento, jolla luodaan sovellus ja kaikki sille tarvittavat tiedostot ja hakemistot:

```
> PHP symfony generate:app frontend
```

Tällä komennolla luotiin frontend-niminen sovellus. (Potencier & Zaninotto 2005, 28.)

6.3 Moduulin luominen

Sovellus tarvitsee vähintään yhden moduulin, johon action-metodit ja template-sivut laitetaan. Suoritetaan komento, jolla luodaan moduuli ja kaikki sille tarvittavat tiedostot ja hakemistot:

```
> PHP symfony generate:module frontend content
```

Tällä komennolla luotiin content-niminen moduuli frontend-nimiseen sovellukseen. Automaattisesti luodun actions/actions.class.php tiedoston koodi on:

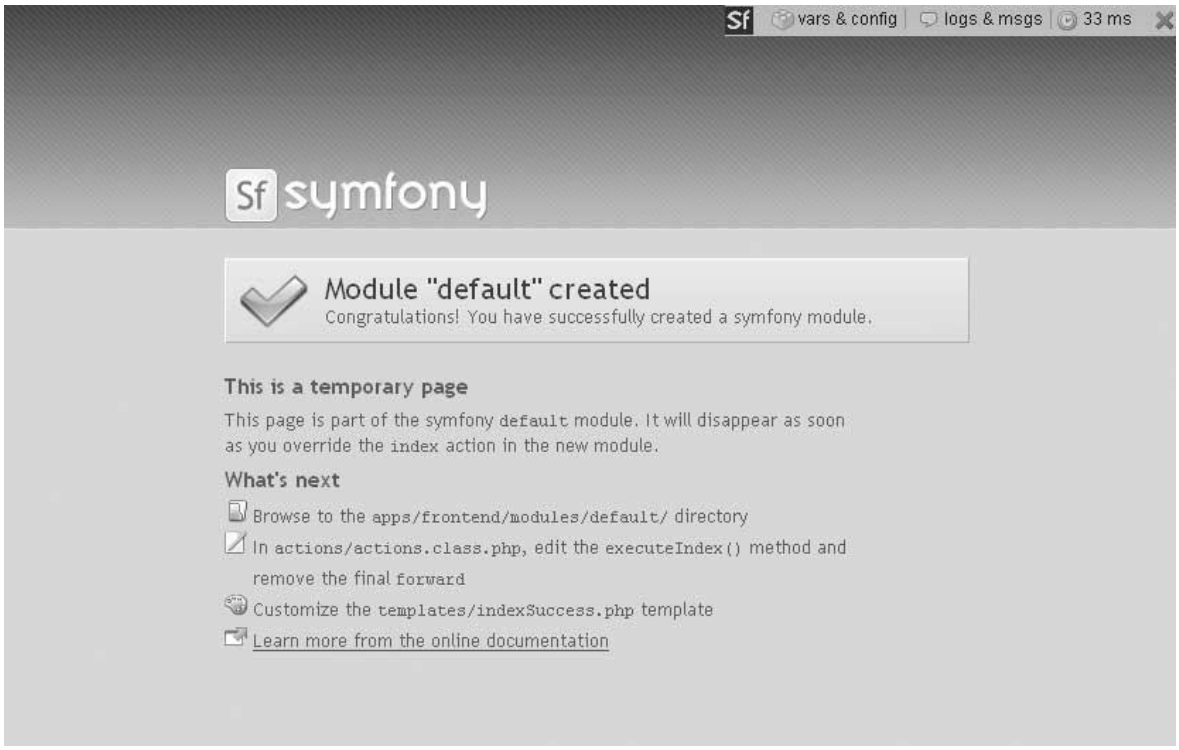
Esimerkki 14. Moduulin koodi

```
<?PHP

class contentActions extends sfActions
{
  public function executeIndex()
  {
    $this->forward('default', 'module');
  }
}
```

Symfony luo jokaiselle uudelle moduulille automaattisesti executeIndex-nimisen action-metodin ja indexSuccess.php-nimisen template-tiedoston. Jokaisen action-metodin nimen eteen tulee kirjoittaa execute, jotta Symfony osaa yhdistää sen metodin template-sivuun. Jokaisen template-tiedoston perään tulee kirjoittaa Success, jotta Symfony osaa yhdistää sen template-sivun metodiin. (Potencier & Zaninotto 2005, 34.)

Kun selaimella menee osoitteeseen http://localhost/frontend_dev.php/content/index, ja jos kaikki on mennyt ongelmitta, niin esimerkkisivun pitäisi näkyä.



Kuva 4. Esimerkkisivu.

Sivun yläkulmassa on debug-palkki, jonka avulla näkee Symfony-projektin asetuksia ja toimintoja, joita Symfony on suorittanut, ja kuinka monta millisekuntia sivun suoritus kesti. (Potencier & Zaninotto 2005, 34.)

6.4 Sivun lisääminen

Tehdään uusi sivu, jossa tulostetaan sivulle tervehdys maailmalle. Ensin luodaan uudelle sivulle action-metodi. $this->muuttujatoiminto tekee muuttujasta yhteisen metodille ja template-sivulle.

Esimerkki 15. Uuden sivun action-metodin koodi

```
<?PHP

class contentActions extends sfActions
{
  public function executeIndex()
  {
    $this->forward('default', 'module');
  }

  public function executeTervehdi()
  {
    $this->tervehdittava = ”maailma”;
  }
}
```

Sitten tehdään action-metodille template-sivu. Luodaan tiedosto nimeltä content/templates/tervehdiSuccess.php ja kirjoitetaan sinne seuraava PHP-koodi, joka tulostaa sivulle tervehdyksen tervehdittava-nimisessä muuttujassa viedylle nimelle.

Esimerkki 16. Uuden sivun koodi

```
<?PHP
  echo “Terve ”.$tervehdittava.”!”;
?>
```

Kaikki logiikka, jota sivustoon liittyy, pyritään tekemään action-metodeissa ja viemällä informaatio niiden kautta template-sivuille. (Potencier & Zaninotto 2005, 35.)

6.5 Toiseen action-metodiin linkittäminen

Seuraavassa esimerkki vanhasta tavasta linkittää sivulta toiselle.

Esimerkki 17. Vanha tapa linkittää sivulta toiselle

```
<a href="/linkin/polku">
  Linkin nimi
</a>
```

Seuraavasssa esimerkki Symfony-ohjelmistokehyksen tarjoamasta lyhyemmästä toiminnosta linkitykseen.

Esimerkki 18. Lyhyempi tapa linkitykseen

```
<?PHP echo link_to('Linkin nimi,'linkin/polku')
?>
```

Seuraavassa esimerkki link_to-toiminnosta, jolle saa annettua erilaisia toimintoja, esimerkiksi vahvistuksen haluaako käyttäjä todella tehdä toiminnon.

Esimerkki 19. Vahvistuksen tekeminen

```
<?PHP echo link_to('Linkin nimi', 'content/update?name=anonymous',
  array(
    'class'    => 'special_link',
    'confirm'  => 'Oletko varma?',
    'absolute' => true
)) ?>
```

Tämä toiminto kysyy käyttäjältä linkin painalluksen jälkeen, onko käyttäjä varma ja jatkaa eteepäin, jos käyttäjä vastaa kyllä. (Potencier & Zaninotto 2005, 37.)

6.6 Request-elementin käyttö

Käyttäjä voi lähettää informaatiota sivuilta action-metodeille joko form (POST request-elementti) -toiminnolla tai URL (GET request-elementti) -toiminnolla. (Potencier & Zaninotto 2005, 39.)

Kummatkin saadaan sfRequest-oliosta samalla getParameter()-metodilla. Kaikki executeXxx()-metodit ottavat sfRequest-olion ensimmäisenä parametrina. (Potencier & Zaninotto 2005, 39.)

Seuraavassa esimerkki getParameter()-metodin käytöstä action-metodissa.

Esimerkki 20. Metodin käyttö

```
<?PHP

class contentActions extends sfActions
{
  // ...

  public function executeUpdate($request)
  {
    $this->name = $request->getParameter('nimi');
  }
}
```

Seuraavassa esimerkki sfRequest-oliosta, jolla saa haettua informaatiota myös template-sivujen puolella sf_params-olion avulla.

Esimerkki 21. Olion käyttö

```
<p>Hello, <?PHP echo $sf_params->get('nimi') ?>!</p>
```

Komennolle saa myös annettua default-arvon, jota käytetään jos arvoa ei ole asetettu olioon. Seuraavassa esimerkki default-arvon antamisesta sf_params-oliolle. (Potencier & Zaninotto 2005, 39.)

Esimerkki 22. Default-arvon antaminen oliolle

```
<p>Terve, <?PHP echo $sf_params->get('nimi', 'John Doe') ?>!</p>
```

7 SYMFONY-JÄRJESTELMÄN KONFIGUROINTI

Symfony käyttää konfiguraatioiden säilyttämiseen oletuksena YAML-tiedostoja. Suunnittelija voi käyttää jotain muuta tapaa, jos haluaa. Jokaiselle projektille, sovellukselle ja moduulille on omat konfiguraatiotiedostonsa. Konfiguraatiotiedostoissa olevat arvot ovat saatavissa PHP-koodilla. Autorisointi on mahdollista toteuttaa YAML-tiedostojen avulla. (Potencier & Zaninotto 2005, 41.)

7.1 YAML

YAML on yksinkertainen kieli, jota käytetään pääasiassa konfiguraatiotiedostojen tekemiseen. Vastaavanlainen tekniikka on esimerkiksi XML ja INI. YAML on hiukan XML-kieltä yksinkertaisempaa, mutta YAML osaa laittaa tietoja taulukoihin toisin kuin INI-tiedostot. (Potencier & Zaninotto 2005, 8.)

Seuraavassa esimerkki taulukon tekemisestä PHP-koodissa.

Esimerkki 23. Taulukon tekeminen PHP-kielellä

```
$house = array(
  'family' => array(
    'name'     => 'Doe',
    'parents'  => array('John', 'Jane'),
    'children' => array('Paul', 'Mark', 'Simone')
  ),
  'address' => array(
    'number'   => 34,
    'street'   => 'Main Street',
    'city'     => 'Nowheretown',
    'zipcode'  => '12345'
  )
);
```

Seuraavassa esimerkki saman taulukon tekemisestä YAML-kielellä:

Esimerkki 24. Taulukon tekeminen YAML-kielellä

```
house:
  family:
    name:     Doe
    parents:
     - John
     - Jane
    children:
     - Paul
     - Mark
     - Simone
  address:
    number: 34
    street: Main Street
    city: Nowheretown
    zipcode: "12345"
```

Seuraavassa esimerkki saman taulukon tekemisestä YAML-kielellä lyhyemmällä, mutta epäselvemmällä tavalla.

Esimerkki 25. Taulukon tekeminen YAML-kielellä lyhyemmin

```
house:
  family: { name: Doe, parents: [John, Jane], children: [Paul, Mark, Si-
mone] }
  address: { number: 34, street: Main Street, city: Nowheretown, zipcode:
"12345" }
```

Arvojen kirjoituksessa ei saa käyttää sarkainta, vaan välit täytyy tehdä välilyönneillä. Ylätason taulukot tehdään kahdella välilyönnillä tai {}-merkeillä. Taulukot tehdään - tai []-merkeillä. (Potencier & Zaninotto 2005, 8.)

YAML tukee myös boolean-muuttujia. Niiden arvoina voivat olla on, 1 tai true positiivisena tai off, 0 tai false negatiivisena. (Potencier & Zaninotto 2005, 8.)

Kommentteja koodiin voi lisätä laittamalla #-merkin rivin eteen. Jos haluaa tehdä numeroista string-muotoisen arvon niin pitää muistaa käyttää ”-merkkejä. (Potencier & Zaninotto 2005, 8.)

YAML-tiedostoon laitettuja arvoja voi asettaa ja hakea PHP-koodissa sfConfig-luokan avulla. Seuraavassa esimerkki arvon asettamisesta YAML-tiedostoon. (Potencier & Zaninotto 2005, 8.)

Esimerkki 26. Arvon asettaminen

```
// Asetetaan asetus
sfConfig::set('param_name', $value);
```

Seuraavassa esimerkki YAML-tiedostoon laitetun arvon hakemisesta.

Esimerkki 27. Arvon hakeminen

```
// Haetaan asetus
parameter = sfConfig::get('param_name', $default_value);
```

param_name-elementin param-osa tarkoittaa tiedostoa, josta asetus halutaan hakea ja name-osa tarkoittaa nimeä, joka asetuksella tiedostossa on. (Potencier & Zaninotto 2005, 8.)

8 ESIMERKKISOVELLUS

Seuraavassa tehdään esimerkkisovellus, jossa käyttäjä voi hakea tietokannasta Henkilot-nimisestä taulusta kaikki henkilöt, jotka asuvat käyttäjän syöttämässä kaupungissa. Sovellukseen tulee kaksi web-sivua, joissa toisessa käyttäjä voi suorittaa haun ja toisessa näytetään haun tulokset. Esimerkkisovellus tehdään ilman Symfony-ohjelmistokehystä ja Symfony-ohjelmistokehyksen avulla, jotta nähdään Symfony-ohjelmistokehyksen antama hyöty.

8.1 Esimerkkisovellus ilman Symfony-ohjelmistokehystä

Seuraavassa esimerkki ilman Symfony-ohjelmistokehystä tehdystä koodista, jossa kysytään käyttäjältä kaupunki web-sivulla.

Esimerkki 28. Tiedon syöttäminen

```
<html>
<head>
<title>Esimerkkisovellus</title>
</head>
<body>
<form method=”post” action=”tulokset.php”>
<input type=”text” name=”kaupunki”>
<input type=”submit” value=”hae”>
</form>
</body>
</html>
```

Seuraavassa esimerkki ilman Symfony-ohjelmistokehystä tehdystä koodista, jossa haetaan Henkilot-nimisestä taulusta kaikki henkilöt, jotka asuvat käyttäjän syöttämässä kaupungissa, ja näytetään tulokset käyttäjälle web-sivulla.

Esimerkki 29. Tiedon hakeminen ja tulostaminen

```
<html>
<head>
<title>Esimerkkisovellus</title>
</head>
<body>

<?PHP

$kaupunki = $_POST[‘kaupunki’];

$yhteys = mysql_connect('localhost', 'jarno', 'kalakukko');
mysql_select_db('tietokanta', $yhteys);

$tulokset = mysql_query(”SELECT * FROM henkilot WHERE
kaupunki=’$kaupunki’;”, $yhteys);

mysql_close($yhteys);

?>

<table>
<tr>
<th>Etunimi</th><th>Sukunimi</th>
</tr>

<?PHP

while ($rivi = mysql_fetch_array($tulokset, MYSQL_ASSOC))
{
      echo “<tr>”;
      echo “<td> $rivi[‘etunimi’] </td>”;
      echo “<td> $rivi[‘sukunimi’] </td>”;
      echo “</tr>”;
}

?>

</table>
</body>
</html>
```

8.2 Esimerkkisovellus Symfony-ohjelmistokehyksen avulla tehtynä

Seuraavassa on esimerkki Symfony-ohjelmistokehyksen avulla tehdystä koodista, jossa kysytään käyttäjältä kaupunki web-sivulla. Koodista voidaan jättää sivun yleiseen ulkoasuun vaikuttavat rivit, koska ne on yleensä jo tehty layout.php-tiedostoon.

Esimerkki 30. Tiedon syöttäminen

```
<form method=”post” action=”<? echo
url_for(esimerkki/tulokset); ?>”>
<input type=”text” name=”kaupunki”>
<input type=”submit” value=”hae”>
</form>
```

Seuraavaksi tehdään web-sivu Symfony-ohjelmistokehyksen avulla, missä haetaan Henkilot-nimisestä taulusta kaikki henkilöt, jotka asuvat käyttäjän syöttämässä kaupungissa ja näytetään tulokset.

Seuraavassa esimerkki Model layer -kerroksesta, jossa suoritetaan SQL-kysely kaupungin mukaan.

Esimerkki 31. Tiedon hakeminen

```
<?PHP

class Henkilot extends BaseHenkilot
{
	function getHenkilotByKaupunki($kaupunki)
	{
		$connection = Propel::getConnection();
		$query = ”SELECT * FROM Henkilot WHERE kaupun-
		ki=’$kaupunki’;”;
		$statement = $connection-
		>prepareStatement($query);
		$resultset = $statement->executeQuery();
		$result = array();
		foreach($resultset as $row)
		{
			$result[] = $row;
```

```
            }
            return $result;
      }
}

?>
```

Seuraavassa esimerkki Controller layer -kerroksesta, jossa Model layer-kerroksessa oleva getHenkilotByKaupunki-niminen metodi suoritetaan.

Esimerkki 32. Tiedon hakuun käytettävän metodin koodi

```
<?PHP

class esimerkkiActions extends sfActions
{
  public function executeTulokset($request)
  {
       $kaupunki = $request->getParameter('kaupunki');
       $Henkilot = new Henkilot();
       $this->henkilot = $Henkilot-
       >getHenkilotByKaupunki($kaupunki);
  }
}
```

Seuraavassa esimerkki View layer -kerroksesta, jossa Controller layer-kerroksessa muuttujaan haetut kyselyn tulokset tulostetaan.

Esimerkki 33. Tiedon tulostaminen

```
<table>
<tr>
<th>Etunimi</th><th>Sukunimi</th>
</tr>

<?PHP

foreach($henkilot as $henkilo)
{
      echo "<tr>";
      echo "<td> $henkilo['etunimi'] </td>";
      echo "<td> $henkilo['sukunimi'] </td>";
      echo "</tr>";
}
```

```
?>

</table>
```

9 JOHTOPÄÄTÖKSET

Tämän opinnäytetyön kahdeksannessa luvussa tehdystä esimerkkisovelluksesta voidaan tehdä johtopäätöksiä vertaamalla ilman Symfony-ohjelmistokehystä tehtyä koodia ja Symfony-ohjelmistokehyksen avulla tehtyä koodia.

Ilman Symfony-ohjelmistokehystä tehdystä koodista voidaan todeta, että koodi on epäselvää, koska koodissa on HTML-, PHP- ja SQL-kieltä sekaisin. Tämä hankaloittaa varsinkin ohjelman ylläpitoa ja jatkokehitystä. Koodissa käytetään POST-toiminnallisuutta ilman suojauksia, joka on suuri tietoturvariski. Koodissa otetaan yhteys suoraan tietokantaan ilman suojauksia, joka on suuri tietoturvariski. Koodi on sidottu MySQL-tietokantaan, joka vaikeuttaa ohjelman ylläpitoa ja jatkokehitystä, jos tietokanta pitää joskus vaihtaa toiseen.

Symfony-ohjelmistokehyksen avulla tehdystä koodista voidaan todeta, että koodi on selkeätä, koska koodissa on html-, PHP- ja sql-kieltä sekaisin hyvin minimaalisesti ja koodi on jaettu eri kerroksiin. Tämä helpottaa ohjelman ylläpitoa ja jatkokehitystä todella paljon. Symfony-ohjelmistokehyksessä on suojaukset, joten tietoturvariskit eivät ole niin suuret. Koodai ei ole sidottu mihinkään tiettyyn tietokantaan, mikä helpottaa ohjelman ylläpitoa ja jatkokehitystä. Koodin jatkokehitystä helpottaa kaikki Symfony-ohjelmistokehyksen tarjoamat suunnittelijan työtä helpottavat ominaisuudet.

Pienessä ja yksinkertaisessa sovelluksessa, kuten seitsemännessä luvussa tehdyssä, ei ero ilman Symfony-ohjelmistokehystä tehdyssä koodissa ja Symfony-ohjelmistokehyksen avulla tehdyssä koodissa ole vielä läheskään niin suuri kuin se olisi, jos koodia olisi esimerkiksi sata kertaa enemmän. Näin yksinkertaista sovellusta ei suunnittelijan vielä kannata tehdä Symfony-ohjelmistokehyksen avulla, koska kehyksen asentamiseen menee aikaa eikä siitä saada vielä kovin paljon hyötyä. Silti näinkin yksinkertaisesta sovelluksesta erottaa hyötyjä kehyksen

avulla tehdyssä koodissa, josta voimme päätellä, että jos suunnittelija on tekemässä isompaa projektia niin hänen kannattaa ehdottomasti tutustua Symfony-ohjelmistokehykseen.

LÄHTEET

Cevasco . 2006. Rails-inspired PHP frameworks. [www-dokumentti]. [Viitattu 10.4.2009]. Saatavissa: http://www.h3rald.com/articles/rails-inspired-PHP-frameworks

Potencier & Zaninotto. 2005. The Definitive Guide to symfony. Kustantaja: Apress.

Potencier & Zaninotto. 2005. Symfony project. [www-dokumentti]. Sensio. [Viitattu 10.4.2009]. Saatavissa: http://www.symfony-project.org/

Ruby on Rails. 2009. Ruby on Rails. [www-dokumentti]. 37signals. [Viitattu 10.4.2009]. Saatavissa: http://rubyonrails.org/

Wikipedia. 2009. Web application framework. [www-dokumentti]. Wikipedia. Saatavissa: http://en.wikipedia.org/wiki/Web_application_framework

Lieska tuotanto. 2009. [Verkkosivusto]. [Viitattu 20.8.2009]. Saatavissa: http://www.lieska.net/faq.html

LIITTEET