SOVELLUSKEHITYS SYMFONY-OHJELMISTOKEHYKSELLÄ

Joni Rantala

Opinnäytetyö
Kesäkuu 2012
Tietojenkäsittelyn koulutusohjelma
Ohjelmistotuotannon suuntautumisvaihtoehto

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Ohjelmistotuotannon suuntautumisvaihtoehto

RANTALA, JONI:
Sovelluskehitys Symfony-ohjelmistokehyksellä

Opinnäytetyö 51 sivua, josta liitteitä 0 sivua
Kesäkuu 2012

Opinnäytetyö käsittelee sovelluskehitystä web-käyttöön suunnatulla Symfony-ohjelmistokehyksellä. Työn toimeksiantaja oli tamperelainen sähköalan yritys Wirtapiiri Oy. Tarve tälle opinnäytetyölle syntyi yrityksen aikomuksesta toteuttaa web-palveluja asiakkaidensa käyttöön. Työn tavoitteena oli tutkia Symfony-ohjelmistokehyksen toimintaa. Tutkimuksen pohjalta pyrittiin löytämään tehokkaat työkalut ja menetelmät toimeksiantajan tulevien web-projektien toteuttamiseen.

Opinnäytetyössä käydään läpi Symfony-ohjelmistokehyksen tärkeimmät ominaisuudet, toimintavat ja sen käyttämät ohjelmistosuunnittelumallit. Symfony-sovelluksen testaukseen perehdytään yksikkötestauksen ja funktionaalisen testauksen osalta. Työ sisältää useita koodiesimerkkejä, joiden tarkoitus on helpottaa käsiteltyjen asioiden ymmärtämistä.

Opinnäytetyön tekovaiheessa toteutettiin esimerkkisovellus, jonka toteuttamiseen sovellettiin työssä käsiteltyjä aiheita. Koko opinnäytetyö on kirjoitettu opasmuotoon, jotta toimeksiantaja voi käyttää sitä esimerkkisovelluksen ohella apukeinona uuden työntekijän perehdyttämiseen.

Asiasanat: sovelluskehitys, ohjelmistokehys, symfony

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Development

RANTALA, JONI:
Software Development with Symfony Framework

Bachelor's thesis 51 pages, appendices 0 pages
June 2012

This thesis covers aspects of web application development with Symfony PHP framework. The work was assigned by a company called Wirtapiiri. The goal of the thesis was to get to know Symfony framework and its components thoroughly. During the thesis process the framework's features and capabilities were explored in order to be able to use it efficiently in company's future projects.

The thesis describes Symfony framework's most notable features and its software design patterns. It also describes how to execute unit and functional testing in Symfony environment. The work contains several code examples so that it is easier for the reader to form an understanding of the subject being covered.

An example application was made in order to demonstrate how to adapt some of the subjects covered in the thesis. The thesis was written in a way so that it can be used as a base manual during the time when a person is learning how to use Symfony framework.

Key words: software development, software framework, symfony

SISÄLLYS

LYHENTEET JA TERMIT

Ohjelmistokehys	Ohjelmistorunko, jonka tarkoituksena on nopeuttaa ohjelmistokehitystä tarjoamalla valmiita osia
Symfony	Suosittu web-käyttöön suunnattu avoimen lähdekoodin ohjelmistokehys PHP-ohjelmointikielelle
MVC	Ohjelmistoarkkitehtuurimalli, jossa sovellus jaetaan kolmeen kerrokseen kehityksen ja ylläpidon helpottamiseksi
PHP	Suosittu, pääosin web-käyttöön tarkoitettu ohjelmointikieli

1 JOHDANTO

Tarve tälle opinnäytetyölle syntyi toimeksiantajan, tamperelaisen sähköalan yrityksen Wirtapiiri Oy:n aikomuksesta tarjota asiakkailleen web-palveluja. Tällä hetkellä suunnitteilla on jo useampi erillinen web-palvelu, joiden kehitystyöhön piti löytää yhteiset tehokkaat työkalut. Tulen jatkossa toimimaan toimeksiantajan web-projekteissa pääkehittäjänä. Ohjelmistokehykset olivat minulle jo entuudestaan tuttuja, mikä helpotti minua sopivan ohjelmistokehyksen valitsemisessa. Symfonyn valitsin käytettäväksi ohjelmistokehykseksi sen antaman ammattimaisen kokonaiskuvan johdosta.

Opinnäytetyön teon aikana tavoitteenani oli perehtyä Symfonyn suunnitteluun ja toimintatapoihin mahdollisimman syvällisesti, jotta osaan tulevissa projekteissa käyttää sitä tehokkaasti. Opinnäytetyön tarkoituksena on opastaa lukija Symfonyn käytön perusteista aina hieman vaativampiinkin aiheisiin. Työssä käytetään asioiden ymmärtämisen ja sisäistämisen helpottamiseksi useita koodiesimerkkejä. Opinnäytetyötä voidaan käyttää toimeksiantajan projekteissa uuden kehittäjän perehdyttämisen pohjana.

Opinnäytetyön sisältö etenee siten, että aluksi käsittelen Symfonya yleisemmällä tasolla, jonka jälkeen käsitellään sen toimintamalleja ja ohjelmistoarkkitehtuuria. Testausluvussa kuvaan kuinka Symfony-sovelluksen testaus tehdään tehokkaasti. Lopuksi vielä toteutan pankkisovelluksen, jonka tekoon sovellan opinnäytetyössä käsittelemiäni menetelmiä ja toimintatapoja. Valmiin pankkisovelluksen lähdekoodi on vapaasti ladattavissa.

Tämän opinnäytetyön lukijalta edellytetään perustason tuntemus PHP-ohjelmointikielestä ja sen olio-ohjelmointimenetelmistä.

2 SYMFONY-OHJELMISTOKEHYS

2.1 Symfony

Symfony on web-käyttöön suunnattu avoimen lähdekoodin ohjelmistokehys PHP-ohjelmointikielelle. Sen kehityksestä vastaa Fabien Potencier, apunaan laaja kehittäjäyhteisö. Symfonyn kehitystä sponsoroi ranskalainen yritys Sensio Labs (About n.d.). Symfony tarjoaa kehittäjälle yleisesti web-sovelluksissa käytettävän perustoiminnallisuuden, jolloin kehittäjä pystyy keskittymään pelkästään oman sovelluksensa koodin kirjoittamiseen. Symfony julkaistaan MIT-lisenssin alaisena.

Symfonyn kehitys aloitettiin vuonna 2005 ja sen ensimmäinen tuotantoversio 1.0 julkaistiin tammikuussa 2007 (Zaninotto 2007). Vanhan version 1 kehitys on korjauspäivityksiä lukuun ottamatta lopetettu ja nykyään kehitystyö keskittyy uuteen versioon 2, joka julkaistiin heinäkuussa 2011 (Potencier 2011). Symfony 2 on kirjoitettu lähes kokonaan uudelleen hyödyntäen PHP:n nimiavaruuksia. Uuden version suunnittelussa on panostettu entistä enemmän modulaariseen rakenteeseen ja sovellusosien uudelleenkäytön helpottamiseen. Tässä opinnäytetyössä, ellei toisin mainita, termillä Symfony viitataan aina Symfony-ohjelmistokehyksen versioon 2.

Symfony soveltuu niin pienten kuin suurtenkin web-sovelluksien tekoon, jopa vain sen yksittäisten komponenttien käyttö omassa projektissa on mahdollista. Symfonyn päälle rakennettuja suuria verkkopalveluita ovat mm. viihdesivusto Dailymotion ja tiedonhaku- ja -jakopalvelu Yahoo! Answers (Potencier 2009; Whittle 2008). Suosittu sisällönhallintajärjestelmä Drupal tulee käyttämään Symfonyn komponentteja tulevassa versiossaan 8 (Potencier 2012).

2.2 Vahvuudet

Sovelluskehityksen nopeus

Symfony tarjoaa monipuoliset olioperustaiset työkalut sovelluskehityksen nopeuttamiseksi. Sen omien komponenttien avulla mm. HTTP-pyyntöjen reititys sekä HTML-lomakkeiden ja sivupohjien teko on nopeaa. Tämän lisäksi Symfony käyttää useita kolmansien osapuolien kirjastoja, jotka se integroi osaksi itseään. Symfonyn mukana tulee kirjastot mm. tietokantojen hallintaan ja sähköpostien lähetykseen. Myös sovelluksen testaus on tehty helpoksi.

Turvallisuus

Oikein käytettynä Symfony tarjoaa suojan yleisimpiä haavoittuvuuksia, kuten SQL-injektioita sekä CSRF- ja XSS-hyökkäyksiä vastaan. Symfonyn avulla kehittäjän on helppo toteuttaa käyttäjän tunnistus ja käyttöoikeuksien hallinta.

Laaja dokumentaatio

Symfonyn virallisella kotisivulla osoitteessa symfony.com on tarjolla laaja dokumentaatio käyttöesimerkkeineen. Saatavilla on aloittelijan opas, laajempi 200-sivuinen Symfony-kirja sekä ns. keittokirja, joka sisältää esimerkkejä usein web-kehityksessä toistuvista tapauksista, ja kuinka ne tulisi toteuttaa Symfony-ympäristössä.

Yhteisö

Symfonylla on laaja ja aktiivinen kehittäjäyhteisö. Symfonyn lähdekoodi on vapaasti saatavilla suositussa Github-verkkopalvelussa. Palvelun avulla kuka tahansa voi osallistua esim. bugikorjauksien tekoon ja uusien ominaisuuksien ideointiin. Kehittäjille on tarjolla sähköpostilistoja, wiki, keskustelualue virallisella kotisivulla ja IRC-kanavia useille kielille (Community n.d.). Tähän mennessä Symfony 2:n kehitykseen on osallistunut yli 400 kehittäjää ympäri maailmaa (Contributors n.d.).

2.3 Heikkoudet

Aloittelijalle paljon opeteltavaa

Symfony on aluksi varmasti monelle aika suuri pala purtavaksi, varsinkin kehittäjille, joilla ei ole aikasempaa kokemusta PHP-ohjelmistokehyksistä. Sen hyödyntämien suunnittelumallien ja -menetelmien ymmärtämiseen ja sisäistämiseen saattaa kulua paljon aikaa.

Symfony 2 -aiheisen kirjallisuuden puute

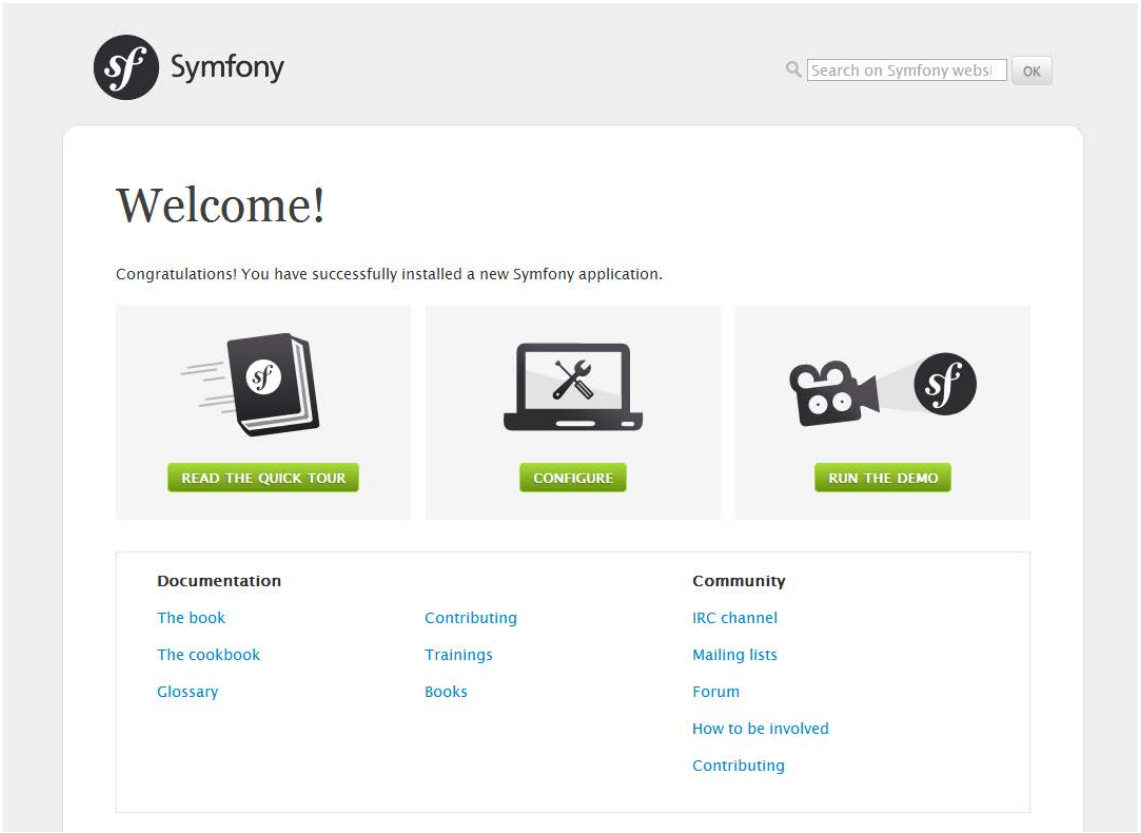
Symfony 2 -aiheista kirjallisuutta ei tällä hetkellä oikeastaan ole. Tämä johtuu varmasti osin siitä, että virallinen dokumentaatio on niin kattava ja että Symfony 2 on vasta vajaan vuoden ikäinen. Symfonyn versiosta 1 on saatavilla muutamia kirjoja, mutta ne eivät ole täysin sovellettavissa versioon 2.

3 SYMFONY-SOVELLUKSEN RAKENNE

3.1 Asennus

Symfonyn voi ladata sen virallisen kotisivun lataussivulta osoitteesta http://symfony.com/download. Symfonya käyttääkseen järjestelmässä tulee olla asennettuna PHP 5.3.2 tai sitä uudempi sekä PHP:n laajennokset Sqlite3, JSON ja ctype (Requirements for running Symfony2 n.d.). Symfonyn vaatimat PHP-laajennokset sisältyvät PHP:n oletusasennukseen.

Symfony asennetaan purkamalla ladatun asennuspaketin tiedostot web-palvelimen julkisesti verkossa jaetun hakemiston juureen. Tämän jälkeen mikäli minimivaatimukset on täytetty, sovelluksen tulisi näyttää juuri asennetun sovelluksen pääsivu osoitteessa http://localhost/Symfony/web/app_dev.php (kuva 1). Symfony tarjoaa pääsivulla käyttäjälle hyödyllisiä linkkejä mm. pikaoppaaseen ja Symfony-kirjoihin. Samasta näkymästä käyttäjä voi siirtyä konfigurointisivulle, jossa käyttäjä voi mm. määrittää käytettävän tietokannan asetukset helposti.



Kuva 1. Symfony onnittelee käyttäjää onnistuneesta asennuksesta.

Sovelluksen tietoturvaa voi parantaa konfiguroimalla web-palvelin siten, että vain sovelluksen web-hakemisto on jaettuna julkiseen verkkoon. Tällöin ulkopuoliset eivät pääse vahingossakaan tarkastelemaan sovelluksen konfiguraatiota tai muita salaisia tiedostoja verkon kautta, mikä edesauttaa sovelluksen tietoturvaa. Tässä opinnäytetyössä oletetaan, että web-palvelin on konfiguroitu edellä esitetyllä tavalla, ja tällöin sovellus vastaa pyyntöihin kehitysympäristön osalta osoitteessa http://localhost/app_dev.php.

3.2 Hakemistorakenne

Symfony-sovelluksen oletushakemistorakenteessa sovelluksen tiedostot on jaettu viiteen hakemistoon: app, bin, src, web ja vendor. Tätä hakemistorakennetta ei ole pakko käyttää, vaan kehittäjä voi halutessaan muuttaa sen vastaamaan omaa mieltymystään. Tämän opinnäytetyön myöhemmissä luvuissa noudatetaan Symfonyn oletushakemistorakennetta.

Jokaisella hakemistolla on oma käyttötarkoituksensa (koodiesimerkki 1):

- App-hakemistossa (1) säilytetään sovelluksen konfiguraatiot, lokit, välimuistin käyttämät tiedostot sekä koko sovellukselle yhteiset staattiset resurssitiedostot, kuten sivupohjat, kuvat ja tyylitiedostot.
- Bin-hakemistossa (2) säilytetään komentorivityökalut ja muut erilliset työkalut.
- Src-hakemistossa (3) säilytetään sovelluksen oma lähdekoodi.
- Web-hakemisto (4) on sovelluksen julkinen hakemisto, jonka sisältö on nähtävissä verkon kautta.
- Vendor-hakemistossa (5) säilytetään sovelluksen käyttämät kolmasien osapuolten tekemät lisäosat ja laajennukset.

```
app/                            # 1)
  cache/                          # Välimuisti
  config/                         # Konfiguraatiohakemisto
    config.yml                      # Pääkonfiguraatio
    config_dev.yml                  # Kehitysympäristön konfiguraatio
    config_prod.yml                 # Tuotantoympäristön konfiguraatio
    routing.yml                     # Reitityskonfiguraatio
  logs/                           # Lokitiedostot
    dev.log                         # Kehitysympäristön loki
    prod.log                        # Tuotantoympäristön loki
  Resources/                      # Yhteiset resurssitiedostot
    css/
      styles.css
    views/
      layout.html.twig
bin/                            # 2)
  vendors
src/                            # 3)
  Ont/
    DemoBundle/
      Controller/
        DefaultController.php
      Tests/
      Resources/
        views/
          Default/
            index.html.twig
web/                            # 4)
  app.php
  app_dev.php
vendor/                         # 5)
```

Koodiesimerkki 1. Esimerkki Symfony-sovelluksen hakemistorakenteesta.

3.3 Komponentit

Symfony tarjoaa kehittäjälle useita täysin itsenäisiä komponentteja, joita voi käyttää missä tahansa PHP-sovelluksessa, ilman että koko sovellus pitää rakentaa Symfonyn päälle (The Book 2012, 12). Komponentit käyttävät toteutuksissaan laajasti rajapintoja, mikä mahdollistaa niiden toiminnan helpon laajentamisen omalla koodilla. Kaikki komponentit ovat täysin olioperustaisia. Lisäksi suurin osa niistä kuuluu ns. stable API:in, joka tarkoittaa että komponentin käyttörajapinta on lopullinen, eikä sen luokkien tai metodien nimet enää muutu (The Book 2012, 220).

Tärkeimmät Symfonyn tarjoamista komponenteista ovat:

- HttpFoundation, joka tarjoaa työkalut mm. HTTP-pyyntöjen ja -vastausten käsittelyyn. Komponentin Request- ja Response-olioiden toimintaa käsitellään luvussa 4.1
- Routing-komponentti, joka tarjoaa tehokkaan reititysjärjestelmän käyttäjän tekemien HTTP-pyyntöjen välittämiseksi ohjaimille. Reititystä käsitellään luvussa 4.3
- Form-komponentti, joka tarjoaa työkalut HTML-lomakkeiden rakentamiseen ja lähetyksen valvontaan
- Validator-komponentti, jonka avulla voidaan helposti tarkastaa esim. käyttäjän syötteen oikeellisuus usein eri perustein
- ClassLoader-komponentti, joka mahdollistaa projektissa tarvittavien PHP-tiedostojen automaattiseen lataamisen tiedostojärjestelmästä. Luokkien latausta käsitellään luvussa 3.6.2
- Security-komponentti, joka tarjoaa monipuoliset työkalut sovelluksen turvallisuuden huomioimiseksi. Komponentin avulla voidaan toteuttaa mm. käyttäjän tunnistus ja käyttöoikeuksien jakaminen
- Translation-komponentti, jonka tehtävänä on helpottaa monikielisten sovelluksien tekoa
- Templating-komponentti, joka mahdollistaa sivupohjien kirjoituksen lähes missä muodossa tahansa
- EventDispatcher-komponentti, joka tarjoaa työkalut tapahtumakäsittelyyn. Symfony käyttää komponenttia laajasti omassa toteutuksessaan
- YAML-komponentti, jonka avulla voidaan käsitellä YAML-muotoista dataa

3.4 Bundlet

Symfony käyttää sovelluksen laajennoksista ja lisäosista nimitystä bundle. Virallisen määrityksen (The Book 2012, 36-37) mukaan bundle on strukturoitu joukko tiedostoja, jotka yhdessä toteuttavat sovelluksessa yhden toiminnon. Tällainen toiminto voi olla esim. blogi, uutispalsta tai kuvagalleria.

Bundlen omassa hakemistossa säilytetään kaikki siihen liittyvät tiedostot: PHP-, tyyli- ja JavaScript-tiedostot, kuvat, sivupohjat ja testit. Symfony käyttää bundle-järjestelmää

myös omassa toteutuksessaan: Symfonyn sisältäessä useita itsenäisiä komponentteja, on FrameworkBundle-bundlen tehtävänä saada komponentit toimimaan keskenään saumattomasti ja rakentaa ohjelmistokehyksen perustoiminnallisuus (The Book 2012, 201). Bundle-järjestelmä antaa kehittäjälle täyden päätösvallan käyttämistään bundleista, eikä edes FrameworkBundlea tarvitse käyttää, joskin sen käyttö on suositeltavaa. Bundle-järjestelmä mahdollistaa myös valmiin toiminnallisuuden jakamisen kehittäjien kesken.

Symfonyn mukana tulee useita bundleja, joista tärkeimmät ovat:

- FrameworkBundle, jonka tehtävänä on tarjota ohjelmistokehyksen perustoiminnallisuus hyödyntämällä Symfonyn komponentteja (The Book 2012, 212)
- SensioFrameworkExtraBundle, joka laajentaa FrameworkBundlen toimintaa mahdollistaen reittien määrityksen ohjaimessa annotaatioita käyttäen (SensioFrameworkExtraBundle n.d.)
- DoctrineBundle ja DoctrineAbstractBundle, jotka integroivat Doctrine-tietokantakirjaston Symfonyyn

Uuden bundlen luonti tapahtuu komentorivikomennolla sovelluksen juurihakemistossa:

```
php app/console generate:bundle --namespace=Ont/DemoBundle
```

Yllä mainittu komento luo bundlelle hakemiston src/Ont/DemoBundle. Lisäksi se luo bundlejen yleisesti käyttämät alihakemistot. Bundlen nimiavaruuden tulee päättyä aina sanaan Bundle.

3.5 Konfigurointi

Symfony-sovelluksen konfigurointi tehdään pääasiassa joko YAML-, XML- tai PHP-muodossa. Tämän lisäksi ohjelmistokehys tukee annotaatioita, joiden avulla kehittäjä voi tehdä konfiguraation suoraan ohjelmakoodista käsin. Oletuksena Symfony käyttää konfigurointiin YAML-muotoisia tiedostoja. Tässä opinnäytetyössä sovelluksen konfigurointi tehdään joko YAML-muodossa tai käyttäen annotaatioita.

Sovelluksen konfiguraatiotiedostot säilytetään hakemistossa app/config. Konfiguraation perustana käytetään tiedoston config.yml asetuksia, joita laajennetaan ajoympäristökohtaisissa konfiguraatioissa, esim. kehitysympäristön tiedostossa config_dev.yml.

Konfiguraatiotiedostossa kaikki asetukset määritetään bundlekohtaisesti. Jokaisen bundlen konfiguraatio voi sisältää useita avain-arvo -pareja. Mitä tahansa avaimia ei tule määrittää, vaan avaimen tulee olla sellainen, jota bundle todella käyttää oman toimintansa konfigurointiin. Virheellisen avaimen käyttö aiheuttaa poikkeuksen.

Koodiesimerkissä 2 nähdään ote sovelluksen oletuskonfiguraatiosta, jossa määritetään asetuksia bundleille FrameworkBundle (1) ja TwigBundle (2). Bundleista käsin konfiguraation arvoja luetaan käyttämällä pistenotaatiota, jolloin esim. FrameworkBundlen lukiessa istunnon oletusmerkistöä konfiguraatiosta (3), se lukee avaimen framework.session.default_locale arvon.

```
# app/config/config.yml
...
# 1)
framework:
  secret:           %secret%
  charset:          UTF-8
  form:             true
  csrf_protection:  true
  session:
    # 3)
    default_locale: %locale%
    auto_start:     true
# 2)
twig:
  debug:            %kernel.debug%
  strict_variables: %kernel.debug%
...
```

Koodiesimerkki 2. Ote Symfony-sovelluksen oletuskonfiguraatiosta.

YAML

YAML (YAML Ain't Markup Language) on virallisen määrityksen (Evans n.d.) mukaan ihmisystävällinen datan merkintä- ja serialiasointistandardi, joka soveltuu käytettäväksi millä ohjelmointikielellä tahansa. Serialisoinnilla tarkoitetaan sovelluksen tietorakenteiden, kuten taulukkojen ja olioiden muuntamista muotoon, josta tietorakenteet on helppo tallentaa esimerkiksi tiedostoon. Serialisoitu data voidaan muuntaa helposti takaisin alkuperäiseen muotoonsa. Koska PHP:ssa ei ole

sisäänrakennettua tukea YAML-datan käsittelyyn, Symfony käyttää omaa YAML-komponenttiaan. Komponentti muuntaa YAML-datan PHP-taulukoksi, mikä mahdollistaa datan helpon jätkokäytön (The YAML Component n.d.).

Annotaatiot

Symfony tukee Java-ohjelmointikielestä tuttuja annotaatioita, joiden tarkoituksena on mahdollistaa sovelluksen konfigurointi ohjelmakoodissa. Annotaatiot kirjoitetaan luokan tai metodin yläpuolelle kommenttien sisään (koodiesimerkki 3). Annotaation nimi alkaa @-merkillä ja niille voidaan usein määrittää parametreja. YAML:n tapaan myöskään annotaatioille ei PHP:ssa ole sisäänrakennettua tukea, vaan tuen annotaatioden käyttöön tarjoaa SensioFrameworkExtraBundle (SensioFrameworkExtraBundle n.d.). Koodiesimerkissä 3 määritetään reititysreitti @Route- (1) ja @Method-annotaatioita (2) käyttäen.

```
class DefaultController {
  /**
   * Esimerkki reitin määrityksestä ohjaimesta käsin.
   * @Route("/")     1)
   * @Method("GET")  2)
   */
  public function indexAction() {
    ...
  }
  ...
}
```

Koodiesimerkki 3. Reitin määritys ohjaimessa annotaatioita käyttäen.

3.5.1 Konfiguraation jakaminen useaan tiedostoon

Symfony tukee konfiguraation jakamista useisiin tiedostoihin. Tällöin myöhemmin määritetyt asetukset korvaavat aikaisemmat. Lataus ulkoisesta tiedostosta tapahtuu resource-avaimella, jolle annetaan arvoksi joko absoluuttinen tai relatiivinen tiedostosijainti (koodiesimerkki 4, 1) tai ns. looginen nimi (2).

```
imports:
# 1)
  - { resource: security.yml }
# 2)
  - { resource: "@OntBankBundle/Resources/config/bank_data.yml" }
```

Koodiesimerkki 4. Ulkoisten tiedostojen lataus konfiguraatioon.

3.5.2 Loogiset nimet

Koska bundlet voivat sijaita useissa eri hakemistoissa, on Symfonyyn kehitetty resurssien sijaintien määrittämistä helpottamaan loogiset nimet (engl. logical names). Looginen nimi alkaa @-merkillä, jota seuraa bundlen nimi. Loogista nimeä käytettäessä bundlen todellista hakemistosijaintia ei tarvitse tietää, koska ne on jo kerrottu Symfonylle bundlejen rekisteröinnin yhteydessä. Bundlejen rekisteröintiä käsitellään luvussa 3.6.1.

3.5.3 Ajoympäristöt

Ajoympäristöjen tarkoituksena on mahdollistaa saman sovelluksen suoritus eri konfiguraatioita käyttäen (The Book 2012, 40). Kehitysympäristössä sovelluksen suorituksesta halutaan mahdollisimman paljon tietoa, mukaan lukien poikkeusviestit, kun taas tuotantoympäristössä poikkeusviestit halutaan piilottaa loppukäyttäjältä. Oletuksena Symfony-sovellus sisältää kolme ympäristöä: prod (tuotanto), dev (kehitys) ja test (testaus). Testausympäristöä ei ole mahdollista käyttää itsenäisesti, vaan Symfony käyttää sitä automaattisesti automatisoituja testejä ajettaessa (The Book 2012, 41). Tarvittaessa kehittäjä voi luoda uusia ajoympäristöjä, joita varten hänen tarvitsee vain luoda uusi konfiguraatiotiedosto ja juuri luotua konfiguraatiota käyttävä etuohjain. Etuohjaimen toimintaa käsitellään syvällisemmin luvussa 3.6.4.

3.6 Sovelluksen lataus ja käynnistys

3.6.1 Kernel-luokka

Kernel-luokan tärkein tehtävä on kertoa Symfonylle mitä bundleja sovelluksessa halutaan käyttää (The Book 2012, 37). Tiedostossa app/AppKernel.php esitellään luokka AppKernel, joka sisältää metodin registerBundles(). Tämä metodi palauttaa taulukon bundle-olioista (koodiesimerkki 5).

```
// app/AppKernel.php

class AppKernel extends Kernel {
  public function registerBundles() {
    $bundles = array(
      new Symfony\Bundle\FrameworkBundle\FrameworkBundle(),
      new Symfony\Bundle\SecurityBundle\SecurityBundle(),
      new Ont\DemoBundle\DemoBundle(),
      ...
    );
    return $bundles;
  }
  ...
}
```

Koodiesimerkki 5. Kernel-luokan ja registerBundles()-metodin käyttö.

3.6.2 PHP-tiedostojen ajonaikainen lataus

Useita PHP-tiedostoja sisältävissä sovelluksissa toisten PHP-tiedostojen lataus on totuttu tekemään include- tai require-lausekkeilla. Symfonyn käytäntöihin kuuluu tapa, jossa jokainen luokka tallennetaan omaan tiedostoonsa mahdollisimman modulaarisen rakenteen saavuttamiseksi. Tämän johdosta jo suppeatkin Symfony-sovellukset saattavat käyttää apunaan kymmeniä ellei satoja PHP-tiedostoja. Symfony tarjoaa kehittäjälle tiedostojen latauksen helpottamiseksi UniversalClassLoader-luokan, sillä käsin satojen require-lausekkeiden kirjoittaminen olisi erittäin työlästä ja aikaavievää.

UniversalClassLoader-luokka tarjoaa luokkien lataukseen useita metodeita, joista tärkein metodi on registerNamespaces(). Metodille annetaan parametrina taulukko, joka sisältää alkioiden avaimina nimiavaruuksien nimet (koodiesimerkki 6, 1) ja alkioiden

arvoina hakemistopolut, joissa nimiavaruuksien luokkia säilytetään tiedostojärjestelmässä (2). Kaikkia nimiavaruuden luokkia ei tarvitse säilyttää samassa hakemistossa, vaan ne voidaan jakaa useampaan hakemistoon. Tällöin kyseisen nimiavaruuden hakemistopolut annetaan registerNamespaces()-metodille taulukossa (3). Määritykset UniversalClassLoader-luokan käyttöön sijaitsevat tiedostossa app/autoload.php, jonka Symfony lataa automaattisesti sovelluksen käynnistyksessä.

```
// app/autoload.php

$loader = new UniversalClassLoader();
$loader->registerNamespaces(array(
  // 1)        2) & 3)
  'Symfony' => array(
    __DIR__.'/../vendor/symfony/src',
    __DIR__.'/../vendor/bundles'
  ),
  'Sensio'  => __DIR__.'/../vendor/bundles',
  'JMS'     => __DIR__.'/../vendor/bundles',
  ..
);
```

Koodiesimerkki 6. UniversalClassLoader-luokan käyttö.

Nimiavaruuksien käyttö PHP:ssa

Nimiavaruudet tulivat PHP-ohjelmointikieleen version 5.3 mukana. Nimiavaruudet mahdollistavat koodin enkapsuloinnin; sen jakamisen pienempiin loogisiin osiin. (Namespaces overview 2012) Käytettävä nimiavaruus määritetään avainsanalla namespace, jolloin kaikki määrityksen jälkeen esitellyt funktiot, vakiot ja luokat liitetään kyseiseen nimiavaruuteen. Nimiavaruudet voidaan jakaa aliavaruuksiin erotinmerkillä, joka PHP:ssa on kenoviiva (\). Nimiavaruudet mahdollistavat samannimisten luokkien käytön eri nimiavaruuksissa.

Nimiavaruuksien sisäisiä resursseja käytetään joko relatiivisin tai absoluuttisin nimin, tai vaihtoehtoisesti use-lausekkeilla, jotka mahdollistavat lyhyemmät nimet ja aliakset, ja täten parantavat koodin luettavuutta. Use-lauseketta käytettäessä sille annettava nimiavaruus on aina absoluuttinen. PHP:n omat funktiot, vakiot ja luokat sijaitsevat oletusnimiavaruudessa \, jolloin toisesta nimiavaruudesta niitä kutsuttaessa käytetään absoluuttista nimeä, esim. strpos()-funktiota kutsutaan \strpos(). (Defining Namespaces 2012)

Koodiesimerkissä 7 esitellään nimiavaruuksien käyttöä:

1. Otetaan luokka \OtherProject\Foo käyttöön use-lausekkeella.
2. Otetaan luokka \OtherProject\Bar käyttöön use-lausekkeella ja annetaan sille alias OtherBar.
3. Määritetään käytettäväksi nimiavaruudeksi \MyProject.
4. Esitellään luokka \MyProject\Bar.
5. Luodaan olioita use-lausekkeella käyttöön otetuista luokista.
6. Luodaan luokasta \MyProject\Model\Goo olio relatiivista nimeä käyttäen.
7. Luodaan luokasta \OtherProject\Baz olio absoluuttista nimeä käyttäen.
8. Heitetään PHP:n oletuspoikkeus.
9. Esitellään luokka \MyProject\Exception, joka perii PHP:n poikkeusluokan.
10. Heitetään poikkeus \MyProject\Exception.

```
<?php

...

// 1) use-lausekkeen käyttö,
//    tästedes Foo viittaa luokkaan \OtherProject\Foo
use OtherProject\Foo;

// 2) Aliaksen käyttö,
//    tästedes alias OtherBar viittaa luokkaan \OtherProject\Bar
use OtherProject\Bar as OtherBar;

// 3) Määritetään käytettävä nimiavaruus
namespace MyProject;

// 4) Esitellään luokka \MyProject\Bar
class Bar { }

// 5) Luodaan oliot use-lausekkeella esitellyistä luokista
$bar      = new Bar;
$otherBar = new OtherBar;

// 6) Luodaan olio relatiivista nimeä käyttäen
$goo = new Model\Goo;

// 7) Luodaan olio absoluuttista nimeä käyttäen
$baz = new \OtherProject\Baz;

// 8) Heitetään poikkeus \Exception
throw new \Exception;

// 9) Esitellään \MyProject\Exception-luokka
class Exception extends \Exception { }

// 10) Heitetään poikkeus \MyProject\Exception
throw new Exception;
```

Koodiesimerkki 7. PHP:n nimiavaruuksien käyttö.

Luokkien nimeämiskäytäntö

Jotta luokkien lataus tietämällä pelkästään nimiavaruuden hakemistosijainti olisi mahdollista, on luokkien nimeämisen seurattava tiettyä käytäntöä. UniversalClassLoader-luokan toteutus seuraa PSR-0 -standardia, jonka tarkoituksena on helpottaa eri PHP-kirjastojen keskinäistä käyttöä. Standardi perustuu siihen, että luokkien nimeäminen seuraa oheista käytäntöä (O'Phinney 2010):

- Luokat nimetään noudattaen seuraavaa kaavaa:
  `\<Tekijä>\(Nimiavaruus\)*<Luokka>`,
  esim. Symfony\Component\Yaml\Dumper.
- Nimiavaruuden alkuosa on aina tekijän nimi.
- Nimiavaruus voi sisältää useita aliavaruuksia.

PSR-0:n käytäntöä seurattaessa luokan tiedostopolku saadaan selville, kun nimiavaruuden erotinmerkit korvataan käytettävän käyttöjärjestelmän hakemistoerottimella ja perään lisätään .php-pääte. Tällöin jos lataajalle on kerrottu \Ont-nimiavaruuden tiedostojen sijaitsevan src-hakemistossa, nimiavaruudessa \Ont\DemoBundle\Controller\ (koodiesimerkki 8, 1) määritetyn luokan DefaultController (2) tulee sijaita tiedostojärjestelmässä polussa src/Ont/DemoBundle/Controller/DefaultController.php (3).

```
// src/Ont/DemoBundle/Controller/DefaultController.php (3

namespace \Ont\DemoBundle\Controller; // 1)

class DefaultController { // 2)
  ...
}
```

Koodiesimerkki 8. PSR-0 -käytännön mukainen tiedoston nimeäminen.

Symfony on oletuksena konfiguroitu etsimään kehittäjän omia luokkia src-hakemistosta. Tällöin kehittäjän säilöessä sovelluksensa lähdekoodin src-hakemistoon, ei hänen tarvitse muokata app/autoload.php-tiedoston sisältöä.

3.6.3 Ulkoisten kirjastojen käyttö

Symfonyn kanssa on mahdollista käyttää lähes mitä tahansa kolmannen osapuolen kirjastoa. Tällöin ohjelmoijan täytyy kuitenkin huolehtia, että kirjasto ladataan app/autoload.php-tiedostossa kirjaston suosittelemalla tavalla.

Symfonyn mukana tulee useita kolmansien osapuolten tekemiä kirjastoja, joista tärkeimpiä ovat Doctrine ja SwiftMailer. Doctrinen toiminnallisuus keskittyy tietokantojen hallintaan ja SwiftMailerin sähköpostien lähettämiseen. Kirjastojen käytön helpottamiseksi Doctrine ja SwiftMailer on integroitu Symfonyyn bundlejen avulla.

Doctrine

Doctrine tarjoaa ohjelmoijalle monipuoliset tietokantariippumattomat työkalut tietokantojen hallintaan. Kehittäjän kannalta tärkein ominaisuus on tuki olioperustaisten tietokantakyselyjen suoritukselle perinteisen SQL:n sijasta, jolloin kyselyt palauttavat tulostietueet automaattisesti olioina, eikä kehittäjän tarvitse itse käsin luoda olioita. (The Book 2012, 81-84.)

SwiftMailer

SwiftMailer tarjoaa selkeän olioperustaisen tavan sähköpostien lähettämiseksi. Sen ominaisuuksiin kuuluu tuki useille eri välitystavoille (mm. sendmail ja SMTP), moniosaisten viestien lähetys sekä liitetiedostojen lähetys. (Swiftmailer n.d.)

3.6.4 Etuohjain

Perinteisesti PHP-sovellukset on rakennettu niin, että jokaista sovelluksessa olevaa sivua varten luodaa oma PHP-tiedosto, joka hoitaa itsenäisesti käyttäjän lähettämän HTTP-pyynnön vastaanoton ja käsittelyn. Tämä lähestymistapa johtaa usein koodin duplikointiin, koska sovelluksella on lähes aina alustustoimenpiteitä, jotka se suorittaa jokaisella suorituskerralla ja nämä toimenpiteet tulee kopioida jokaiseen tiedostoon (Zandstra 2010, 235-236). MVC-arkkitehtuurin sovelluksissa, joita myös Symfonylla tehdyt sovellukset pääasiassa ovat, kaikki loppukäyttäjän tekemät pyynnöt vastaanottaa keskitetysti yksi PHP-tiedosto, etuohjain (engl. front controller) (taulukko 1). Tämä

mahdollistaa alustustoimien suorittamisen keskitetysti. MVC-arkkitehtuuria käsitellään luvussa 4.2.

Taulukko 1. Sivujen osoitteet perinteistä tapaa ja etuohjainta käytettäessä.

Perinteinen tapa	**Etuohjain**
/index.php	/index.php
/gallery.php	/index.php/gallery
/blog.php	/index.php/blog

Etuohjaimet tallennetaan sovelluksen web-hakemistoon. Oletuksena Symfony-sovelluksessa on kaksi etuohjainta, kehitysympäristön etuohjain app_dev.php ja tuotantoympäristön etuohjain app.php. Kehitysympäristön etuohjain hyväksyy vain pyynnöt, jotka tulevat paikallisesta IP-osoitteesta, eikä näin sovellusta pääse suorittamaan kehitysympäristössä lähiverkon ulkopuolelta. Symfony-ympäristössä etuohjaimet ovat melko yksinkertaisia, sillä ne sisältävät yleensä vain noin kymmenen riviä koodia.

Koodiesimerkissä 9 tarkastellaan kehitysympäristön etuohjaimen toteutusta:

1. Ladataan ohjelmistokehyksen sovelluksen käynnistykseen käyttämät luokat välimuistista.
2. Ladataan AppKernel-luokka.
3. Luodaan AppKernel-olio, jolle annetaan parametrina ajoympäristön nimi (dev) ja tieto, halutaanko sovellus suorittaa debug-moodissa (true).
4. Käynnistetään sovellus kutsumalla AppKernel-olion handle()-metodia. Metodille annetaan parametrina HTTP-pyyntöä mallintava Request-olio, joka luodaan staattisella metodilla createFromGlobals(). Sovelluksen suorituksen päätyttyä handle()-metodin palauttama Response-olio tulostetaan käyttäjälle kutsumalla metodia send(). Request- ja Response-olioita käsitellään laajemmin luvussa 4.1.

```
// web/app_dev.php

// 1)
require_once __DIR__.'/../app/bootstrap.php.cache';
// 2)
require_once __DIR__.'/../app/AppKernel.php';

use Symfony\Component\HttpFoundation\Request;

// 3)
$kernel = new AppKernel('dev', true);
$kernel->loadClassCache();
// 4)
$kernel->handle(Request::createFromGlobals())->send();
```

Koodiesimerkki 9. Etuohjaimen toteutus Symfony-sovelluksessa.

4 SYMFONY-SOVELLUKSEN ARKKITEHTUURI

4.1 Request- ja Response-luokat

HTTP-protokolla on yksinkertainen tiedonsiirtoon suunniteltu tekstipohjainen protokolla, jota käytetään kaikkialla internetissä. Yksinkertaistettuna HTTP toimii seuraavasti: käyttäjän selain lähettää palvelimelle pyynnön tietyn resurssin latauksesta, johon palvelin vastaa lähettämällä takaisin resurssin sisällön. (The Book 2012, 4-8.)

Koodiesimerkissä 10 on esitettynä käyttäjän selaimen palvelimelle lähettämä pyyntö ladattaessa osoitetta http://localhost/app_dev.php. Pyynnön ensimmäiseltä riviltä käy aina ilmi käytettävä HTTP-metodi (esimerkissä GET), polku haluttuun resurssiin (/app_dev.php) ja yhteydessä käytettävän protokollan versio (HTTP/1.1). Lisäksi pyyntö saattaa sisältää erinäisiä lisätietokenttiä, otsakkeita (engl. header), jotka sisällytetään pyyntöön avain-arvo -pareina.

```
GET /app_dev.php HTTP/1.1
Host: localhost
Accept: text/html
User-Agent: Mozilla/5.0 (Windows NT 6.1)
```

Koodiesimerkki 10. Esimerkki HTTP-pyynnöstä.

HTTP:n määrityksessä on esitelty useita eri metodeja, joista tärkeimmät ovat GET ja POST. GET-metodilla käyttäjä pyytää palvelimelta jotakin resurssia. POST-metodilla käyttäjä lähettää palvelimelle dataa käsiteltäväksi, kuten HTML-lomakkeelle syötetyt tiedot. PHP:ssa kehittäjän ei itse tarvitse huolehtia pyynnön parsimisesta, vaan PHP parsii pyynnön automaattisesti ja säilöö sen tiedot taulukkomuodossa ns. superglobaaleihin muuttujiin, joita ovat mm. $_SERVER, $_GET ja $_POST.

Koodiesimerkissä 11 on esitettynä web-palvelimen mahdollinen vastaus aiemmin esiteltyyn pyyntöön. Vastauksen ensimmäiseltä riviltä käy ilmi käytettävän protokollan versio (esimerkissä HTTP/1.1), vastauskoodi (200) ja vastausviesti (OK). Tämän lisäksi myös vastauksessa on otsakkaita, joista esim. Content-Type -otsake kertoo vastauksen olevan html-muodoista dataa. Viimeistä otsaketta seuraa tyhjä rivi, jonka jälkeen alkaa vastauksen varsinainen sisältöosa.

```
HTTP/1.1 200 OK
Date: Sun, 27 May 2012 15:53:33 GMT
Content-Type: text/html; charset=UTF-8
Server: Apache/2.2.21 (Win32) PHP/5.3.8

<html>
...
</html>
```

Koodiesimerkki 11. Esimerkki HTTP-vastauksesta.

Symfonyssa HTTP-pyynnöt ja -vastaukset käsitellään olioperustaisesti. Helpoin tapa uuden pyynnön, Request-olion, luontiin on käyttää luokan staattista metodia createFromGlobals() (koodiesimerkki 12, 1), joka luo olion superglobaalien muuttujien arvojen perusteella. Tällöin Request-olio sisältää $_GET- (2), $_POST- (3) ja $_SERVER-muuttujat (4). Lisäksi olio sisältää otsaketiedot (5), metodin nimen (6) sekä URI-polun, johon pyyntö tehtiin (7).

```
use Symfony\Component\HttpFoundation\Request;

// 1)
$request = Request::createFromGlobals();

// 2) - sama kuin $_GET["muuttuja"]
$request->query->get("muuttuja");

// 3) - sama kuin $_POST["muuttuja"]
$request->request->get("muuttuja")

// 4) - sama kuin $_SERVER["REMOTE_ADDR"]
$request->server->get("REMOTE_ADDR");

// 5)
$request->headers->get("host");

// 6)
$request->getMethod();

// 7)
$request->getPathInfo();
```

Koodiesimerkki 12. Request-luokan käyttö.

HTTP-vastausta mallintaa Response-luokan oliot. Vastauksen sisältö ja vastauskoodi määritetään metodeilla setContent() (koodiesimerkki 13, 1) ja setStatusCode() (2). Otsakkeita voi määrittää samaan tapaan kuin Request-olioiden kanssa (3). Vastaus lähetetään käyttäjälle metodilla send() (4).

```
use Symfony\Component\HttpFoundation\Response;

$response = new Response();
// 1)
$response->setContent("Hello, World!");
// 2)
$response->setStatusCode(200);
// 3)
$response->headers->set("Content-Type", "text/plain");
// 4)
$response->send();
```

Koodiesimerkki 13. Response-luokan käyttö.

4.2 Model-View-Controller

Model-View-Controller (MVC) on ohjelmistoarkkitehtuurimalli, jonka tarkoituksena on helpottaa ja nopeuttaa sovelluksen kehitystyötä ja ylläpitoa jakamalla sovelluksen toiminta kolmeen kerrokseen (McArthur 2008, 201; Jansch 2008, 100-102):

- malliin (Model), joka huolehtii sovelluksen liiketoimintalogiikasta ja datan säilömisestä
- näkymään (View), jonka tehtävänä on tarjota loppukäyttäjälle käyttöliittymä sovelluksen käyttämiseksi
- ohjaimeen (Controller), jonka tehtävänä on käsitellä käyttäjän tekemät pyynnöt sekä huolehtia tarvittavien mallien ja näkymien käytöstä

Sovelluksen toiminnan jakaminen useaan kerrokseen mahdollistaa usean kehittäjän samanaikaisen työskentelyn. Tällöin yksi kehittäjä voi kehittää käyttöliittymää, toisen toteuttaessa liiketoimintalogiikkaa. Lopuksi näkymän ja mallien yhteistoiminta toteutetaan ohjaimessa. (Jansch 2008, 103.)

4.2.1 Mallit

Symfony ei ota kantaa millä tavalla mallit tulisi toteuttaa, eikä siksi tarjoa itse valmiita rajapintoja tai luokkia mallien toteuttamiseen. Yleinen tapa Symfony-kehittäjien keskuudessa on käyttää Doctrine-tietokantakirjastoa mallien toteuttamiseen. Doctrinen avulla mallit voidaan helposti liittää tiettyyn tietokannan tauluun ja sen kenttiin annotaatioita käyttäen (koodiesimerkki 14). Koodiesimerkissä luodaan luokka Book,

jonka tietoihin kuuluu yksilöivä tunnus ($id), otsikko ($title) ja kirjoittajan nimi ($author). Tietokantataulun nimeksi määritetään book, id-kenttä määritetään kokonaislukumuotoiseksi pääavaimeksi, jonka arvo luodaan automaattisesti. Tämän lisäksi title- ja author-kentät määritetään merkkijonomuotoisiksi kentiksi, joiden maksimipituus on 100 merkkiä.

```
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 * @ORM\Table(name="book")
 */
class Book {
  /**
   * @ORM\Id
   * @ORM\Column(type="integer")
   * @ORM\GeneratedValue(strategy="AUTO")
   */
  protected $id;

  /**
   * @ORM\Column(type="string", length=100)
   */
  protected $title;

  /**
   * @ORM\Column(type="string", length=100)
   */
  protected $author;
  ...
}
```

Koodiesimerkki 14. Mallin toteuttaminen Doctrinen avulla annotaatioita käyttäen.

4.2.2 Ohjaimet

Ohjaimen tärkeimpänä tehtävä on tulkita käyttäjän lähettämää HTTP-pyyntöä mallintava Request-olio, käsitellä se ja lopuksi palauttaa vastaus, eli Response-olio (The Book 2012, 43).

Symfonyssa ohjaimena voi toimia funktio, luokan metodi tai anonyymi funktio (The Book 2012, 44). Kuitenkin yleensä ohjaimet perivät FrameworkBundlen Controller-luokan, koska se tarjoaa useita apumetodeita yleisimpiin käyttötapauksiin. Metodit, joita halutaan käyttää toiminnallisuuksiin tulee nimetä siten, että nimi päättyy sanaan Action, eli esim. indexAction(). Metodin näkyvyysmääreenä tulee olla public.

4.2.3 Näkymät

Symfonyssa näkymät rakennetaan Templating-komponenttia hyödyntäen. Sen avulla kehittäjä voi toteuttaa sivupohjat itselleen mielekkäimmällä sivupohjamoottorilla (engl. template engine). Oletuksena Symfony tukee PHP- ja Twig-muotoisia sivupohjia. (The Book 2012, 65-66.)

Sivupohjat tallennetaan bundlen sisällä hakemistoon Resources/views ja niiden nimeäminen seuraa tiettyä käytäntöä. Ne tulee nimetä muodossa <nimi>.<tiedostomuoto>.<sivupohjamoottori>, esim. index.html.twig. Nimen tiedostomuoto-osa määrittää, minkä muotoista sisältöä renderöity sivupohja sisältää, ja sivupohjamoottori-osa määrittää, mitä moottoria Symfonyn tulee käyttää sivupohjan renderöintiin.

Sivupohjan renderöinti tapahtuu ohjaimen metodilla render(), jolle annetaan parametrina sivupohjan looginen nimi sekä taulukko muuttujista, joita sivupohjassa halutaan käyttää. Render()-metodi palauttaa renderöidyn sivupohjan Response-oliona.

```
// src/Ont/DemoBundle/Resources/views/Default/hello.html.php

<h1>Hello, <?php echo $name; ?></h1>


// src/Ont/DemoBundle/Controller/DefaultController.php

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Response;

class DefaultController extends Controller {
  public function helloAction() {
    $response = $this->render(
      "@OntDemoBundle:Default:hello.html.php",
      array("name" => "World")
    );
    return $response;
  }
}
```

Koodiesimerkki 15. Sivupohjan renderöinti.

4.3 Reititys

Reitityksen tehtävänä on päätellä käyttäjän lähettämän HTTP-pyynnön perusteella, missä kyseinen pyyntö tulisi käsitellä. Päättelyn perustana reititin käyttää reitityskonfiguraatiota, johon määritetään reitit kaikkiin sovelluksen toimintoihin. Jos pyyntöä vastaavaa reittiä ei löydy, kerrotaan käyttäjälle tästä virhesivulla. Reititys toteutetaan Routing-komponentin tarjoamilla työkaluilla. (The Book 2012, 54-57).

Sovelluksen sisäinen reititys määritetään pääreitityskonfiguraatiossa, tiedostossa app/config/routing.yml. Yksittäisiä reittejä ei yleensä määritetä pääreitityksessä, vaan jokainen bundle määrittää reittinsä bundlen sisäiseen konfiguraatioon, josta ne ladataan sovelluksen käyttöön pääkonfiguraatiossa (koodiesimerkki 16).

```
# app/config/routing.yml

OntDemoBundle:
  resource: "@OntDemoBundle/Resources/config/routing.yml"
  prefix:   /


# src/Ont/DemoBundle/Resources/config/routing.yml

index:
  pattern:  /
  defaults: { _controller: OntDemoBundle:Default:index }

contact:
  pattern:  /contact
  defaults: { _controller: OntDemoBundle:Default:contact }
```

Koodiesimerkki 16. Bundlen reittien liittäminen reitityskonfiguraatioon.

5 SYMFONY-SOVELLUKSEN TESTAUS

5.1 PHPUnit

PHPUnit on Sebastien Bergmannin kehittämä ja ylläpitämä ohjelmistokehys automatisoidun testauksen toteuttamiseen PHP-ohjelmointikielellä. PHPUnit on noussut kehittäjien keskuudessa standardityökaluksi yksikkötestauksen toteuttamiseksi (Jansch 2008, 179). Se kuuluu xUnit-testaustyökaluperheeseen, jonka tuotteet tarjoavat samankaltaisen käyttörajapinnan suosituimmille ohjelmointikielille. Näitä kieliä ovat mm. Java (JUnit), C++ (CppUnit) ja .NET (NUnit).

5.1.1 Ensimmäisen testin kirjoitus

PHPUnitissa testaus koostuu testitapauksista ja yksittäisistä testeistä. Testitapaus on PHP-luokka, joka sisältää testejä yhden yksikön toiminnan testaamiseksi. Yksittäinen testi on vain tavallinen metodi testitapauksen luokassa, joka keskittyy yksikön yhden metodin testaukseen.

Testiluokat peritään luokasta PHPUnit_Framework_TestCase, joka tarjoaa useita apumetoideita testauksen toteuttamiseksi. Testiluokka tulee nimetä siten, että sen nimi loppuu aina sanaan Test, esimerkiksi laskimen toimintaa testaavan luokan nimi voisi olla CalculatorTest (koodiesimerkki 17, 1). Testimetodin nimen tulee alkaa sanalla test, joten laskimen yhteenlaskutoimintoa testaavan metodin nimi voisi olla esimerkiksi testAdd() (2). Testimetodin näkyvyysmääre tulee olla public. (Bergmann 2012a, 7.)

```
// Calculator.php
class Calculator {
  public function add($num1, $num2) {
    return $num1 + $num2;
  }
}

// CalculatorTest.php
// 1)
class CalculatorTest extends \PHPUnit_Framework_TestCase {
  // 2)
  public function testAdd() {
    $calculator = new Calculator;
    $this->assertEquals(3,  $calculator->add(1, 2));
    $this->assertEquals(10, $calculator->add(6, 4));
    $this->assertEquals(12, $calculator->add(3, 9));
  }
}
```

Koodiesimerkki 17. Esimerkki PHPUnit-testitapauksesta.

Testitapauksen saa ajettua komentoriviltä komennolla:
`phpunit CalculatorTest.php`

PHPUnit ilmoittaa testien tuloksista omin merkinnöin (koodiesimerkki 18). Onnistuneen testin kohdalla PHPUnit tulostaa pisteen (.). Muita mahdollisia merkintöjä ovat: F (failed) mikäli testi ei onnistu, S (skipped) mikäli testi jätetiin suorittamatta tai I (incomplete) mikäli testi on merkitty keskeneräiseksi.

```
F:\ont>phpunit CalculatorTest.php
PHPUnit 3.6.10 by Sebastian Bergmann.

.

Time: 0 seconds, Memory: 3.50Mb

OK (1 test, 3 assertions)
```

Koodiesimerkki 18. Testin ajaminen PHPUnitilla.

5.1.2 Testaukseen käytettävät metodit

PHPUnitilla testattaessa yleisin menetelmä testauksen toteuttamiseksi on verrata testattavan metodin palautusarvoa odotettuun arvoon. PHPUnit tarjoaa useita assert-alkuisia metodeja, jotka mahdollistavat palautusarvojen helpon vertailun. Näistä tärkeimmät metodit ovat assertFalse(), assertTrue(), assertNull(), assertEquals() ja

assertSame() (koodiesimerkki 19). Jo pelkällä assertEquals()-metodilla voisi periaatteessa toteuttaa koko testauksen, mutta luettavuuden kannalta kannattaa käyttää aina tarkoitukseen sopivinta metodia.

```
class AssertTest extends \PHPUnit_Framework_TestCase {
  public function testAssert() {
    $this->assertFalse(false);   // false ==  false
    $this->assertTrue(true);     // true  ==  true
    $this->assertNull(null);     // null  ==  null
    $this->assertEquals(1, "1"); // 1     ==  "1"
    $this->assertSame(1, 1);     // 1     === 1
  }
}
```

Koodiesimerkki 19. PHPUnitin yleisimmin käytetyt assert-alkuiset metodit.

5.1.3 Poikkeuksien testaus

Poikkeuksien testaus tapahtuu @expected-alkuisilla annotaatioilla tai metodilla setExpectedException(). Annotaatioita käytettäessä @expectedException (koodiesimerkki 20, 1) määrittää odotetun poikkeuksen nimen, ja odotetun poikkeuksen viestin ja koodin voi määrittää annotaatioilla @expectedExceptionMessage (2) ja @expectedExceptionCode (3). Ennen PHPUnitin versiota 3.7 tätä menetelmää ei ole mahdollista käyttää geneeristen poikkeuksien testaukseen, vaan testattavan poikkeuksen tulee olla aina oma alaluokka (Bergmann 2012b).

```
class ExceptionTest extends PHPUnit_Framework_TestCase {
  /* 1)
   * @expectedException        MyException
   * 2)
   * @expectedExceptionMessage Right message
   * 3)
   * @expectedExceptionCode 1
   */
  public function testExceptionMessage() {
    throw new MyException("Wrong message", 1);
  }

  public function testExceptionMessageAndCode() {
    $this->setExpectedException("MyException", "Message", 1);
    throw new MyException("Message", 1);
  }
}
```

Koodiesimerkki 20. Poikkeusten testaus.

5.1.4 Testiasetelmat

Testiasetelmaksi (engl. test fixture) kutsutaan tilaa, joka vallitsee testitapauksen yksittäisen testin suorituksen alkaessa. Usein jokaista testiä varten saatetaan joutua luomaan jokin monimutkainen olio. Saman koodin kopioiminen useaan paikkaan ei ole hyvien tapojen mukaista, ja juuri tätä varten on olemassa testiasetelmat.

PHPUnit tarjoaa testiasetelmien määrittämiseksi metodit setUp() ja tearDown(), joista setUp() suoritetaan ennen jokaista testiä, ja tearDown() jokaisen testin jälkeen. Tämän lisäksi tarjotaan staattiset metodit setUpBeforeClass() ja tearDownAfterClass(), jotka suoritetaan vain kerran testitapauksen aikana. (Bergmann 2012a, 71.)

Koodiesimerkissä 21 esitellään setUp()-metodin käyttöä testiasetelman määritykseen. Luokkaan määritetään muuttuja $object (1), joka alustetaan ennen jokaista testiä setUp()-metodissa. Tämän jälkeen alustettu muuttuja $object on käytettävissä kaikissa testimetodeissa.

```
class SetUpTest extends \PHPUnit_Framework_TestCase {
  protected $object;

  public function setUp() {
    $this->object = new stdClass;
    $this->object->foo = "foo";
    $this->object->bar = "bar";
  }

  public function testObject() {
    $this->assertEquals("foo", $this->object->foo);
    $this->assertEquals("bar", $this->object->bar);
  }
}
```

Koodiesimerkki 21. Testiasetelma määritys setUp()-metodia käyttäen.

5.2 Web-testit

Yksikkötestin keskittyessä pääasiassa vain yhden luokan testaamiseen, web-testeissä testataan sovelluksen suuremman osan toimivuutta suorittamalla testi sen oman käyttöliittymän kautta. Web-testit kuuluvat funktionaaliseen testaukseen, jonka tarkoituksena on todentaa, että sovellus vastaa sille määritysvaiheessa asetettuja odotuksia (Jansch 2008, 169). Web-testeissä testaus etenee siten, että ensiksi tehdään HTTP-pyyntö testattavaan osoitteeseen, jonka jälkeen web-palvelimelta saatua vastausta tutkitaan ja varmistetaan, että se on odotetun mukainen (The Book 2012, 112).

Ensimmäisen testin kirjoitus

Symfony tarjoaa FrameworkBundlen mukana luokan WebTestCase, jonka testiluokat perivät. WebTestCase on luokan PHPUnit_Framework_TestCase aliluokka, joka mahdollistaa testien suorituksen PHPUnitilla.

Koodiesimerkissä 22 esitellään melko yksinkertainen web-testi. Siinä oletetaan, että sovellukseen on määritetty reitti /hello, joka tulostaa yksinkertaisen HTML-sivun. Sivun testaaminen aloitetaan luomalla HTTP-asiakas -olio createClient()-metodilla (1), jonka jälkeen asiakkaalla haetaan testattavan sivun sisältö request()-metodilla (2). Kyseinen metodi palauttaa DomCrawler-olion, jonka avulla voidaan tehdä hakuja juuri haetun HTML-sivun DOM-puuhun XPath- tai CSS-kyselyin. Lopuksi testataan, sisältääkö sivu tekstin "Hello, World!" (3).

```
<!-- http://localhost/app_dev.php/hello -->
<html>
<body>
Hello, World!
</body>
</html>


// src/Ont/DemoBundle/Tests/Controller/DemoWebTest.php
namespace Ont\BankBundle\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class DemoWebTest extends WebTestCase {
  public function testHello() {
    // 1)
    $client = static::createClient();

    // 2)
    $crawler = $client->request("GET", "/hello");

    // 3)
    $this->assertTrue(
      $crawler->filter("html:contains('Hello, World!')")->count() > 0
    );
  }
}
```

Koodiesimerkki 22. Yksinkertaisen web-testin toteutus.

Testin suoritus tapahtuu sovelluksen juurihakemistosta komennolla:
`phpunit -c app src/Ont/DemoBundle/Tests/Controller/DemoWebTest.php`

Web-testien suoritus testin hakemistosta ei onnistu, koska PHPUnit ei osaa automaattisesti ladata Symfonyn tiedostoja, esim. WebTestCase-luokkaa. Tämän takia komennolle annetaan c-optiona hakemisto (app), josta PHPUnit etsii testisarjan konfiguraation (app/phpunit.xml.dist). Web-testejä suoritettaessa täytyy muistaa, että jokaista testiä varten lähetään oikea HTTP-pyyntö, minkä takia testitapauksen suorittaminen saattaa olla melko hidasta.

6 PANKKISOVELLUKSEN SUUNNITTELU, TOTEUTUS JA TESTAUS

6.1 Tehtävänanto

Tarkoituksenani on suunnitella ja toteuttaa pankkisovellus Symfony-ympäristössä testivetoisen sovelluskehityksen keinoja apuna käyttäen. Tarkoituksenani ei ole toteuttaa täyslaajuista pankkijärjestelmää, vaan pankkisovelluksen ja tämän pääluvun tehtävänä on demonstroida ja soveltaa tässä opinnäytetyössä käytyjä asioita käytännön tasolla.

Valmiin pankkisovelluksen lähdekoodi on ladattavissa osoitteessa https://github.com/jomppapomppa/OntBankBundle.

6.2 Toiminnallisuuden määritys

Pankkisovellus koostuu pankin asiakkaille tarkoitetusta web-käyttöliittymästä sekä pankkien välisen rahaliikenteen ohjaukseen tarkoitetusta sovellusrajapinnasta (engl. Application Programming Interface, API). Web-käyttöliittymän avulla asiakkaat voivat tarkastaa oman pankkitilinsä saldon ja siirtää rahaa pankkitililtä toiselle, myös toisen pankin asiakkaan tilille. Toimintoja suorittaakseen asiakkaan tulee olla kirjautuneena pankin järjestelmään. Kirjautumista varten asiakkaalle on annettu käyttäjätunnus ja salasana.

Jokaisella pankilla ja pankkitilillä on yksilöivä tunniste. Pankeillä tunniste on kolmen numeron ja pankkitileillä viiden numeron pituinen. Pankkitilin lopullinen tilinumero saadaan yhdistettäessä nämä kaksi tunnistetta väliviivalla, esim. 123-12345.

Pankkien ja pankkitilien tiedot säilytetään sovelluksen konfiguraatiossa, josta sovellus lukee ja muuntaa ne olioiksi. Rahaliikenne ja muutokset pankkitilien saldoihin tallennetaan tiedostoon. Käyttäjän kirjautumistiedot säilytetään istunnossa.

6.3 Testivetoinen ohjelmistokehitys

Testivetoinen ohjelmistokehitys (engl. Test-Driven Development, TDD) on ohjelmistokehitysmenetelmä, jossa automatisoidut testit kirjoitetaan ennen varsinaisen sovelluskoodin kirjoittamista. Menetelmässä kehittäjä työskentelee lyhyissä sykleissä, joiden aikana hän kirjoittaa ensin testit, sen jälkeen testin läpäisevän koodin ja aina välillä refaktoroi. Refaktoroinnilla tarkoitetaan koodin selkeyttämistä ja siivousta, jota on tärkeää suorittaa aika ajoin, sillä testivetoisessa lähestymistavassa pyritään mahdollisimman nopeisiin sykleihin, jolloin tuotettavan koodin taso usein laskee. Uuden toiminnallisuuden toteuttamiseksi tarvitaan usein kymmeniä syklejä. (Jansch 2008, 181-182).

Testivetoisen kehitystavan vahvuuksiin kuuluu se, että menetelmää asianmukaisesti seurattaessa valmiista sovelluksista tulee kattavasti testattuja. Tämän lisäksi kehittäjien kirjoittaessa paljon testejä, he myös samalla kehittävät omia testaustaitojaan ja tulevaisuudessa testien kirjoittaminen on aina vain helpompaa ja nopeampaa.

6.4 Suunnittelu

Bundle

Sovellusta varten luodaan bundle Ont\BankBundle. Bundlen juurihakemisto on src/Ont/BankBundle.

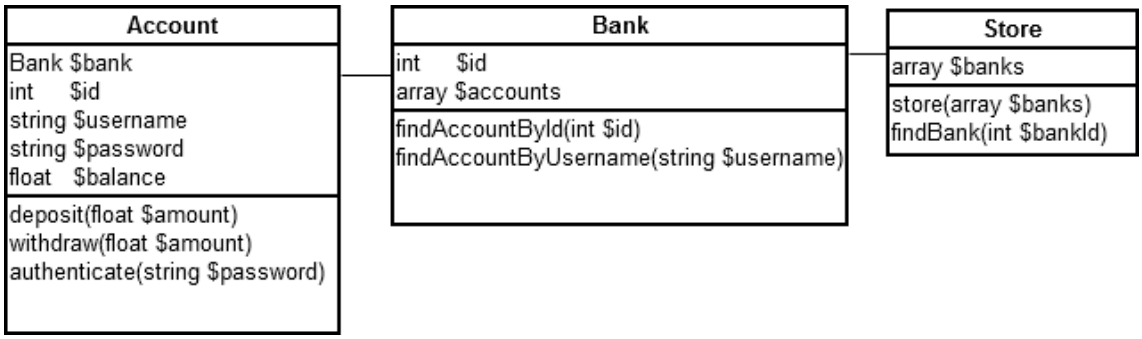
Mallit

Koska halusin pitää sovelluksen mahdollisimman yksinkertaisena, päätin toteuttaa liiketoimintalogiikan kolmen luokan avulla:

- Account-luokan olio mallintaa yksittäistä asiakasta sekä hänen pankkitiliään. Tässä sovelluksessa asiakkaalla voi olla vain yksi tili.
- Bank-luokan olio mallintaa pankkia, jolla voi olla useita tilejä.
- Store-luokan olio säilöö Bank-oliot.

Kun toteutettavien luokkien nimet ja tehtävät oli selvillä, aloin suunnitella niiden käyttörajapintoja (kaavio 1) paperia apuna käyttäen. Listasin paperille kunkin luokan tärkeimmät käyttötapaukset. Account-luokalla ne olivat rahan talletus ja nosto sekä

käyttäjän tunnistus, Bank-luokalla pankkitilien etsiminen tilin käyttäjänimen tai tunnuksen perusteella, jotta asiakkaan tunnistus ja tilien hallinta olisi mahdollista, ja Store-luokalla pankkien etsiminen pankin tunnuksen perusteella.



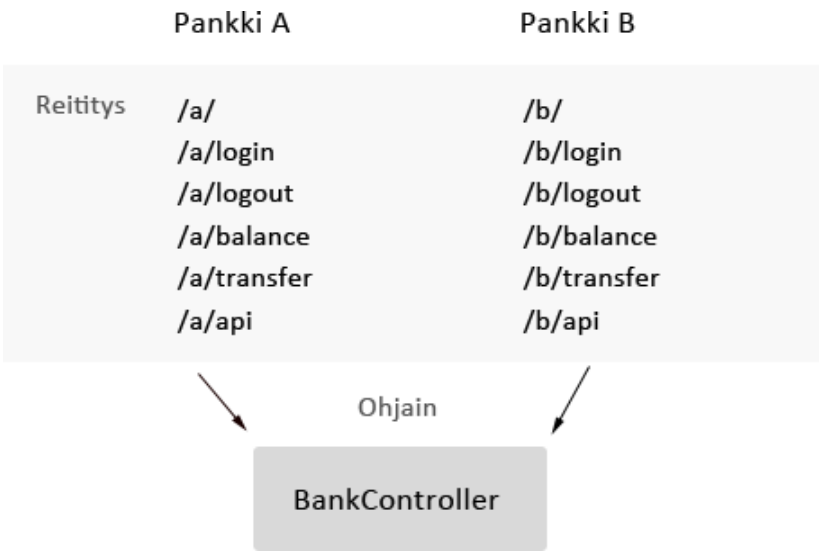
Kaavio 1. Mallien käyttörajapinnat.

Konfiguraatio

Koska sovellus ei käytä apunaan tietokantaa, säilytetään pankkien ja pankkitilien alkutiedot bundlen sisällä tiedostossa Resources/config/bank_data.yml, josta ne ladataan sovelluksen konfiguraatioon. Näiden tietojen perusteella ohjain lukee tiedot ja alustaa tarvittavat oliot. Sovellukseen määritetään kahden pankin (A ja B) ja näiden asiakkaiden tiedot. A-pankin tunnus (id) on 123 ja B-pankin 987.

Ohjain ja reititys

Sovellukseen luodaan ohjain nimeltä BankController, joka käsittelee niin web-käyttöliittymän, kuin sovellusrajapinnankin kautta tulevat pyynnöt. Reititystä varten kummallekin pankille luodaan reitit reitittimeen, joiden avulla pyynnöt välitetään yhteiselle ohjaimelle (kaavio 2).



Kaavio 2. Pankkisovelluksen reititys.

Taulukossa 2 on listattu pankkisovelluksen eri toiminnot. Käyttääkseen saldokysely- tai tilisiirtosivua käyttäjän tulee olla kirjautuneena, muuten käyttäjän selain ohjataan kirjautumissivulle.

Taulukko 2. Pankkisovelluksen eri toiminnot.

Esimerkkiosoite	**Ohjaimen metodi**	**Toiminnon kuvaus**
/a/	indexAction()	Pankin etusivu
/a/login	loginAction()	Kirjautumistoiminto
/a/logout	logoutAction()	Uloskirjautumistoiminto, joka tuhoaa olemassa olevan istunnon
/a/balance	balanceAction()	Saldokyselytoiminto
/a/transfer	transferAction()	Tilisiirtotoiminto, jonka avulla käyttäjä voi siirtää rahaa toisille pankkitileille
/a/api	apiAction()	Sovellusrajapinta

Näkymät

Sovellusta varten luodaan yksinkertainen ulkoasu, jota kaikki muut sivupohjat laajentavat. Kumpikin pankki käyttää samaa ulkoasua, ainoa keskinäinen ero on yläpalkin pankkikohtainen väritys. Sivupohjat toteutetaan Twig-sivupohjamoottoria käyttäen.

6.5 Toteutus

6.5.1 Mallit

Aloitin sovelluksen rakentamisen malleista. Loin jokaiselle mallille toteutusluokan nimiavaruuteen \Ont\BankBundle\Model\ ja testiluokan nimiavaruuteen \Ont\BankBundle\Tests\Model\. Olioiden ominaisuudet määritetään pääasiassa niitä luotaessa antamalla tarvittavat parametrit luokan muodostimelle. Jokaiselle luokassa määritellylle yksityiselle muuttujalle luotiin get-alkuinen getterimetodi, esim. getId().

Account- ja AccountTest-luokat
Aloitin mallien toteuttamisvaiheen kirjoittamalla AccountTest-luokkaan seuraavat testit:

- testConstructor(), joka testaa muodostimen toiminnan luokan gettereitä käyttäen (koodiesimerkki 23)
- testConstructorException(), joka antaa muodostimelle oliota luodessa parametrina virheellisen tunnuksen. Tämän pitäisi aiheuttaa poikkeus AccountException
- testWithdraw(), jonka nostaa rahaa olion withdraw()-metodilla, ja tämän jälkeen todentaa toimivuuden vertaamalla odotettua arvoa getBalance()-metodin palauttamaan arvoon
- testWithdrawException(), joka nostaa pankkitililtä enemmän rahaa, kuin mitä tilillä on. Tämän pitäisi aiheuttaa poikkeus AccountException
- testDeposit(), joka testaa rahan talletuksen toimivuuden olion getBalance()-metodia käyttäen
- testAuthenticate(), joka testaa käyttäjän tunnistuksen toimivuuden

```
// src/Ont/BankBundle/Tests/Model/AccountTest.php

class AccountTest extends \PHPUnit_Framework_TestCase {
  protected $account;

  public function setUp() {
    $this->account = new Account(12345, "tunnus", "salasana", 100.5);
  }

  public function testConstructor() {
    $this->assertEquals(12345,      $this->account->getId());
    $this->assertEquals("tunnus",   $this->account->getUsername());
    $this->assertEquals("salasana", $this->account->getPassword());
    $this->assertEquals(100.5,      $this->account->getBalance());
  }
  ...
}
```

Koodiesimerkki 23. Account-luokan muodostimen testaus.

Saadessani testiluokan valmiiksi, aloitin Account-luokan toteutuksen. Toteutin luokan metodit yksi kerrallaan, kunnes kaikki oli toteutettu ja testitapauksen suoritus onnistui.

Bank- ja BankTest-luokat

BankTest-luokkaan kirjoitin seuraavat testimetodit:

- testConstructor(), joka testaa muodostimen toiminnan samaan tapaan kuin AccountTest-luokassa
- testFindAccountById(), joka testaa tilien etsimistä tilin tunnuksen perusteella
- testFindAccountByUsername(), joka testaa tilien etsimistä tilin käyttäjätunnuksen perusteella. Koodiesimerkissä 24 on esitetty testin toteutus. Testin alussa luodaan kaksi Account-oliota (1), jotka syötetään setUp()-metodissa luodulle Bank-oliolle. Tämän jälkeen testataan, löytääkö metodi findAccountByUsername() oliot käyttäjätunnuksen perusteella (3). Lopuksi vielä testataan tapausta, jossa pankkitiliä ei löydy, jolloin testattavan metodin tulisi palauttaa false (4)

```
// src/Ont/BankBundle/Tests/Model/BankTest.php
...

public function testFindAccountByUsername() {
  // 1)
  $olli = new Account($this->bank, "31214", "olli", "passu", 4230);
  $jake = new Account($this->bank, "77772", "jake", "EFfxx", 1234);

  // 2)
  $this->bank->setAccounts(array(
    $olli, $jake
  ));

  // 3)
  $this->assertSame(
    $olli,
    $this->bank->findAccountByUsername("olli")
  );
  $this->assertSame(
    $jake,
    $this->bank->findAccountByUsername("jake")
  );

  // 4)
  $this->assertFalse(
    $this->bank->findAccountByUsername("jaKE"));
}
...
```

Esimerkki 24. BankTest-luokan testFindAccountByUsername()-metodin toteutus.

Store- ja StoreTest-luokat

Viimeisenä toteutusvuorossa oli Store-luokka. Koska luokka on toisia luokkia yksinkertaisempi, ei testimetodeja tarvinnut kirjoittaa kuin kaksi:

- testStoreException(), joka tarkastaa ettei store()-metodi hyväksy muita kuin Bank-oliota
- findBank, joka testaa pankkien hakua

6.5.2 Ohjain

Olioden alustus

Sovelluksen ensimmäisellä käynnistyskerralla ohjain lukee pankkien tiedot konfiguraatiosta. Näiden tietojen perusteella se luo tarvittavat Account- ja Bank-oliot. Luodut oliot säilötään Store-olioon, joka tallennetaan serialisoituna tiedostoon sivulatauksien välillä.

Reititys ja aktiivisen pankin tunnistaminen

Koska kumpikin pankki käyttää samaa ohjainta pyyntöjen käsittelyyn, on sovelluksen pystyttävä tunnistamaan kummalle pankille pyyntö on suunnattu (koodiesimerkki 24). Tästä johtuen reitit eri toimintoihin määritetään molemmille pankeille erikseen. Jokaisen reitin määritykseen liitetään pankkikohtainen bankId-parametri, jonka avulla pankin tunnistus tehdään. Symfony välittää parametrin arvon automaattisesti ohjaimen metodille (1).

```
# src/Ont/BankBundle/Resources/config/routing.yml

OntBankBundle_123_index:
    pattern:  /a/
    defaults: { _controller: OntBankBundle:Bank:index, bankId: 123 }
OntBankBundle_123_login:
    pattern:  /a/login
    defaults: { _controller: OntBankBundle:Bank:login, bankId: 123 }
...
OntBankBundle_987_index:
    pattern:  /b/
    defaults: { _controller: OntBankBundle:Bank:index, bankId: 987 }
OntBankBundle_987_login:
    pattern:  /b/login
    defaults: { _controller: OntBankBundle:Bank:login, bankId: 987 }


// src/Ont/BankBundle/Controller/BankController.php

// 1)
public function indexAction($bankId) {
  ...
}
```

Koodiesimerkki 24. Aktiivisen pankin tunnistamisen toteutus.

Kirjautumisen toteutus

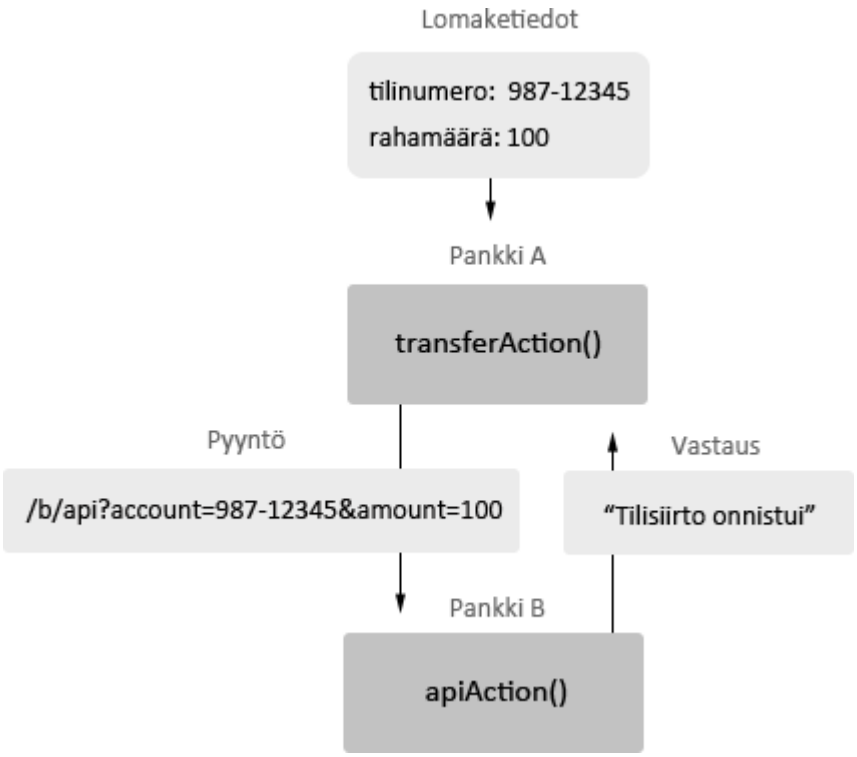
Koodiesimerkissä 25 on esitetty kirjautumistietojen käsittelyn toteutus. Käyttäjän lähetettyä kirjautumistiedot lomakkeen kautta, tarkastaa ohjain ensin tilin olemassaolon, minkä jälkeen tarkastetaan täsmäävätkö salasanat (2). Salasanojen täsmätessä tilin tunnus tallennetaan istuntoon (3) ja selain ohjataan etusivulle (4). Muussa tapauksessa käyttäjälle näytetään virheviesti (5).

```
public function logInAction($bankId) {
  ...
  $request = $this->getRequest();
  if ($request->getMethod() === "POST") {
    $username = $request->request->get("username");
    $password = $request->request->get("password");
    // 1)
    $account  = $this->getBank()->findAccountByUsername($username);

    // 1)                    2)
    if ($account !== false && $account->authenticate($password)) {
      $session = $this->get("session");
      // 3)
      $session->set(
        "account_id_{$this->bankId}",
        $account->getId()
      );

      // 4)
      return $this->redirect($this->generateRouteUrl());
    }
    else {
      // 5)
      $this->get("session")->setFlash(
        "error",
        "Virheellinen käyttäjätunnus tai salasana."
      );
    }
 }
 ...
}
```

Koodiesimerkki 25. Kirjautumistietojen käsittely.

Tilisiirtotoiminnon toteutus

Kaaviossa 3 on kuvattu tilisiirtotoiminnon toiminta. Käyttäjä syöttää lomakkeelle tilinumeron, jolle haluaa siirtää rahaa ja rahamäärän. Käyttäjän lähettämät lomaketiedot tarkastetaan ensin käyttäjän oman pankin ohjaimessa: tilinumeron tulee kuulua johonkin olemassa olevaan pankkiin ja käyttäjän tilillä tulee olla tarpeeksi rahaa tilisiirron suorittamiseksi. Mikäli vastaanotetut lomaketiedot läpäisevät tarkastuksen, ohjain lähettää HTTP-pyynnön kohdepankkitilin omistavan pankin sovellusrajapintaan. Pyyntö sisältää lomaketiedot parametreinaan. Sovellusrajapinnassa tarkastetaan, että kohdepankkitili on olemassa, jotta raha voidaan tallettaa tilille. Sovellusrajapinta tulostaa vastauskoodin, joka kertoo tilisiirron onnistumisesta tai vaihtoehtoisesti syyn, miksi siirto ei onnistunut. Tilisiirron onnistuessa käyttäjän oman pankin ohjain veloittaa käyttäjän pankkitililtä tilisiirron mukaisen rahamäärän ja näyttää ilmoituksen onnistuneesta tilisiirrosta (kuva 2). Myös mahdollisista virhetilanteista ilmoitetaan käyttäjälle virheviestein.



Kaavio 3. Tilisiirtotoiminnon toiminta.

Kuva 2. Pankkisovellus ilmoittaa tilisiirron onnistuneen.

7 POHDINTA

Sovelluskehityksen nopeuttamiseksi on saatavilla useita avoimen lähdekoodin ohjelmistokehyksiä. Enää kehittäjän ei tarvitse keskittyä työkalujensa tekoon, vaan ohjelmistokehykset tarjoavat monipuoliset työkalut, jolloin kehittäjä pystyy keskittymään pelkästään oman sovelluksensa kehitystyöhön. Symfony on omasta mielestäni yksi parhaista ohjelmistokehyksistä PHP-ohjelmointikielelle sen ammattimaisuuden ja monipuolisten ominaisuuksien johdosta.

Opinnäytetyöprosessin aikana perehdyin monipuolisesti Symfonyn tarjoamiin mahdollisuuksiin. Opin paljon sen käyttämistä ohjelmistosuunnittelumalleista, joita hyödyntämällä myös omista sovelluksista saadaan uudelleenkäytettäviä. Vaikka Symfony alkuun vaatiikin kehittäjältä melko paljon aikaa ja vaivaa, johdonmukaisesti edettäessä käytön oppiminen ja sisäistäminen on nopeaa. Ongelmatilanteissa apu löytyy yleensä Symfonyn laajasta ja monipuolisesta dokumentaatioista.

Symfony-sovelluksen testaus pystytään toteuttamaan suurilta osin PHPUnitilla ja Symfonyn omilla web-testaukseen tarkoitetuilla työkaluilla. Usein nykyaikaiset web-sovellukset käyttävät käyttöliittymissään vasta loppukäyttäjän selaimessa tulkittavaa JavaScriptia ja ns. AJAX-menetelmiä, jolloin testausta ei pystytä toteuttamaan kaikilta osin pelkillä opinnäytetetyössä esitellyillä testaustyökaluilla. Laajasti JavaScriptia käyttävien sovelluksien funktionaaliseen testaukseen on saatavilla työkaluja, jotka suorittavat testit oikean verkkoselaimen avulla, jolloin myös sovelluksen JavaScript-osat on mahdollista testata. Jatkotoimenpiteenä aion perehtyä sovelluksen JavaScript-osien testaukseen.

Opinnäytetyöni kirjoitettin opasmuotoon, jotta sitä voidaan voidaan käyttää uuden kehittäjän perehdytyksessä Symfonyn perusteisiin. Pankkisovelluksen teon aikana opin paljon ohjelmistotestauksesta, josta olen ollut kiinnostunut jo pidemmän aikaa.

Kaiken kaikkiaan Symfony on erinomainen vaihtoehto ammattimaiseen sovelluskehitykseen PHP-ohjelmointikielellä: se hyödyntää toteutuksessaan useita hyväksi havaittuja suunnittelumalleja, sen toiminnallisuutta on mahdollista laajentaa lähes rajattomasti ja se tarjoaa jo oletuspaketissaan kaikki nykyaikaiseen web-kehitykseeen tarvittavat työkalut. Aion jatkossa toteuttaa suuremmat PHP-projektit aina Symfonylla.

LÄHTEET

Bergmann, S. 2012a. PHPUnit Manual. Tulostettu 9.5.2012.
http://www.phpunit.de/manual/3.6/en/phpunit-book.pdf

Bergmann, S. 2012b. PHPunit not expect generic exception? Tulostettu 28.5.2012.
https://github.com/sebastianbergmann/phpunit/issues/454

Evans, C. C. YAML 1.2. Tulostettu 19.5.2012.
http://yaml.org/

Jansch, I. 2008. php|architect's Guide to Enterprise PHP Development. Kanada: Marco Tabini & Associates.

McArthur, K. 2008. Pro PHP: Patterns, Frameworks, Testing and More. USA: Apress.

O'Phinney, M. W. 2010. PSR-0. Tulostettu 23.5.2012.
https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-0.md

PHP Manual. 2012. Defining namespaces. Tulostettu 9.12.2012.
http://fi2.php.net/manual/en/language.namespaces.definition.php

PHP Manual. 2012. Namespaces overview. Tulostettu 23.5.2012.
http://www.php.net/manual/en/language.namespaces.rationale.php

Potencier, F. 2009. Dailymotion, powered by symfony. Tulostettu 19.5.2012.
http://symfony.com/blog/dailymotion-powered-by-symfony

Potencier, F. 2011. Symfony 2.0. Tulostettu 23.5.2012.
http://symfony.com/blog/symfony-2-0

Potencier, F. 2012. Symfony2 meets Drupal 8. Tulostettu 23.5.2012
http://symfony.com/blog/symfony2-meets-drupal-8

Symfony. About. Tulostettu 3.6.2012.
http://symfony.com/about

Symfony. Community. Tulostettu 9.5.2012.
http://symfony.com/community

Symfony. Contributors. Tulostettu 9.5.2012.
http://symfony.com/contributors

Symfony. Requirements for running Symfony2. Tulostettu 19.5.
http://symfony.com/doc/current/reference/requirements.html

Symfony. SensioFrameworkExtraBundle. Tulostettu 28.5.2012.
http://symfony.com/doc/current/bundles/SensioFrameworkExtraBundle/index.html

Symfony. 2012. The Book. Tulostettu 9.5.2012.
http://symfony.com/pdf/Symfony_quick_tour_2.0.pdf?v=2

Symfony. The YAML Component. Tulostettu 9.5.2012.
http://symfony.com/doc/current/components/yaml.html
Swift Mailer. Free Feature-rich PHP Mailer. Luettu 19.5.2012.
http://swiftmailer.org/

Whittle, D. 2008. Yahoo! Answers powered by symfony. Tulostettu 19.5.2012.
http://symfony.com/blog/yahoo-answers-powered-by-symfony

Zandstra, M. 2010. PHP Objects, Patterns, and Practice. Third Edition. USA: Apress.

Zaninotto, F. 2007. Symfony 1.0 released. Tulostettu 3.6.2012.
http://symfony.com/blog/symfony-1-0-released