Ville Lindsberg

VERKKOPALVELUN OHJELMOINTI SYMFONY2-SOVELLUSKEHYKSELLÄ

Case: Altima

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma

Marraskuu 2015

KUVAILULEHTI

MAMK University of Applied Sciences	**Opinnäytetyön päivämäärä** 27.11.2015
Tekijä(t) Ville Lindsberg	**Koulutusohjelma ja suuntautuminen** Tietojenkäsittely
Nimeke Verkkopalvelun ohjelmointi Symfony2-sovelluskehyksellä. Case: Altima	

Tiivistelmä

Opinnäytetyössä tehtävän ohjelmointityön tarkoituksena on olemassa olevan PHP-verkkosovelluksen ohjelmointi uudelleen käyttäen Symfony2-sovelluskehystä. Opinnäytetyössä tutustutaan lyhyesti sovelluskehysten historiaan yleisesti sekä tarkastellaan niiden tuomia hyötyihin ja haittoihin. Lisäksi tutustutaan MVC-arkkitehtuuriin ja sen merkitykseen etenkin verkkosovelluksissa.

Kehitettävä sovellus on opinnäyteyön toimeksiantajana toimivan Punavaara Oy:n käyttöön kehittämäni työajanseuranta- ja työnkirjausjärjestelmä nimeltään Altima. Olen kehittänyt sovellusta jo muutaman vuoden ajan omatoimisesti, mutta ajan myötä sovelluksen ylläpitämisestä on tullut erittäin haasteellista, johtuen huonosti suunnitelluista ja toteutetuista ohjelmointiratkaisuista. Ratkaisuna ongelmaan päädyttiin kirjoittamaan koko sovellus uudelleen hyödyntäen modernia PHP-sovelluskehystä, Symfony2:ta.

Opinnäytetyössä käydään vaihe vaiheelta läpi uuden Symfony2-projektin luominen sekä yhden suuremman sovelluksen osakokonaisuuden kehittäminen, käsittäen yleisimpiä verkkosovelluksessa tarvittavia toimintoja. Lisäksi tarkastellaan Symfony2:n tietoturva-ominaisuuksia sekä sovelluksen käyttöönottoa tuotannossa. Samalla verrataan sovelluskehyksen käytäntöjä Altiman vanhaan versioon.

Kehityksen tuloksena valmistui uusi versio Altimasta, joka tarjosi valtavia parannuksia kaikilla sovelluksen osa-alueilla. Lopputuloksena todetaan Symfony2:n olleen oikea valinta Altiman uuden version kehitykseen, huolimatta sen varsin jyrkästä oppimiskäyrästä ja projektin aloituksessa syntyneistä ongelmista. Symfony2 mahdollistaa helpomman jatkokehityksen ja tässä mielessä Altiman tulevaisuus näyttää nyt hyvältä.

Asiasanat (avainsanat)
Ohjelmointi, PHP, WWW-sivut, olio-ohjelmointi, Symfony2

Sivumäärä 34	**Kieli** Suomi
Huomautus (huomautukset liitteistä)	
Ohjaavan opettajan nimi Janne Turunen	**Opinnäytetyön toimeksiantaja** Punavaara Oy

DESCRIPTION

MAMK University of Applied Sciences	**Date of the bachelor's thesis** November 27, 2015
Author(s) Ville Lindsberg	**Degree programme and option** Business Information Technology
Name of the bachelor's thesis Building a web application with the Symfony2 framework. Case: Altima	

Abstract

The main goal of this thesis was to rewrite an existing PHP application by using the Symfony2 framework. The application in question was a web-browser-based time- and work-tracking application called Altima. I began the development of the application in 2013 for Punavaara Oy and I developed the original version continuously for two years. As the application grew, the development became increasingly difficult as it unveiled some bad planning and programming choices in the original code. As the best solution to these problems I decided to rewrite the whole program by using a modern software framework.

In the theoretical part of this thesis I took a brief look at the history of software frameworks in general and summarized some of the advantages and disadvantages of using them. I also described what MVC was and its role in software frameworks. The practical part described the process of creating and setting up a new Symfony2 project for Altima. I explained the main points of development of the new version and also compared how the Symfony2 application development differed from the earlier version of Altima.

The outcome of the thesis was a successfully rewritten version of Altima. The new version provided countless advantages over the old application regarding maintainability, security and future development. As a result, I was convinced that moving to Symfony2 was the correct solution for all of the problems the old version had.

Subject headings, (keywords) Software, PHP, frameworks, Symfony2	
Pages 34	**Language** Finnish
Remarks, notes on appendices	
Tutor Janne Turunen	**Bachelor's thesis assigned by** Punavaara Oy

SISÄLTÖ

1 JOHDANTO 1

2 SOVELLUSKEHYS 2

2.1 Hyötyjä ja haittoja 3

2.2 PHP-sovelluskehykset 4

2.3 MVC- arkkitehtuuri 8

3 SYMFONY2 10

3.1 Komponentit 11

3.2 Projektin rakenne ja bundlet 13

3.3 Reititys, mallit ja ympäristöt 15

4 ALTIMA 16

4.1 Kehitysympäristön ja uuden projektin luominen 18

4.2 Sovelluksen kehitys 21

4.3 Tietoturvasta 28

4.4 Käyttöönotto ja ylläpitäminen tuotantopalvelimella 29

5 PÄÄTÄNTÖ 31

LÄHTEET 33

LIITTEET

1 Twig-templaatti base.html.twig
2 Lomakkeen rakentaminen ja käsitteleminen
3 Kuvankaappauksia uudesta ulkoasusta

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on toteuttaa verkkopalvelu käyttäen Symfony2-sovelluskehystä. Palvelu tulee olemaan täydellinen uudelleenkirjoitus olemassa olevasta sovelluksesta, jonka kehitin alun perin täysin itsenäisesti ilman sovelluskehystä. Tästä johtuen opinnäytetyössä päästään hieman havainnollistamaan sovelluskehyksen tuomia hyötyjä ja haittoja verrattuna perinteiseen täysin omaan toteutukseen. Lisäksi opinnäytetyössä tutustutaan MVC-arkkitehtuuriin ja sen merkitykseen sovelluskehyksissä.

Kyseessä oleva sovellus on internetselaimella toimiva työajanseuranta- ja työnkirjausjärjestelmä nimeltään Altima. Altima on vuonna 2013 alkanut projekti, jota olen kehittänyt oman toiminimeni Smokevisionin välityksellä mäntyharjulaiselle Punavaara Oy:lle, joka toimii myös tämän opinnäytetyön toimeksiantajana. Punavaara Oy on kiinteistöalan huolto- ja ylläpitotehtäviin erikoistunut yritys, joka toimii Etelä-Savossa Mäntyharjun, Pertunmaan, Ristiinan ja Suomenniemen alueilla. Vuonna 1999 perustettu perheyritys on kasvanut nykyään parikymmentä henkeä työllistäväksi osakeyhtiöksi.

Altimaa alettiin kehittää, koska yrityksen työntekijöiden suuri liikkuvuus laajalla toimialueella aiheutti haasteita luotettavaan töiden kirjaukseen ja työaikojen seurantaan. Altimasta tehtiin alun perin mahdollisimman yksinkertaiseksi ja helppokäyttöiseksi, mutta kehityskaaren aikana ilmennyt tarve lisäominaisuuksille on tuonut ilmi lukuisia ongelmia huonosti suunnitelluissa ja toteutetuissa ohjelmointiratkaisuissa. Ratkaisuna päädyttiin toteuttamaan koko sovellus uudelleen hyödyntäen modernia Symfony2 PHP-sovelluskehystä.

Luvussa kaksi käydään läpi sovelluskehysten toimintaperiaatteita ja termistöä sekä hyötyjä ja haittoja. Tämän jälkeen tehdään lyhyt katsaus PHP-sovelluskehysten historiaan ja esitellään suosituimpia PHP-sovelluskehyksiä. Lisäksi luvussa tutustutaan MVC-arkkitehtuurin historiaan, toimintaperiaatteisiin sekä tarkastellaan sen merkitystä etenkin PHP-sovelluskehysten näkökulmasta. Luvussa kolme esitellään tarkemmin opinnäytetyön ohjelmointityöhön valitun sovelluskehyksen, eli Symfony2:n ominaisuuksia ja käytäntöjä. Luvussa neljä tutustutaan tarkemmin opinnäytetyön toimeksiantajaan ja käydään tarkemmin läpi sovelluksen kehitysprosessi. Luvussa käydään vaihe

vaiheelta läpi Symfony2-sovelluskehyksen kehitysympäristön asennus, yhden sovelluksen osakokonaisuuden kehittäminen sekä sovelluksen käyttöönotto ja ylläpitäminen tuotantopalvelimella. Lisäksi tarkastellaan sovelluksen tietoturvaa ja verrataan Altiman uuden version kehitystä vanhaan versioon.

2 SOVELLUSKEHYS

Sovelluskehys (engl. software framework) tarjoaa valmiin rungon, jonka päälle sovellus kehitetään hyödyntäen kehyksen tarjoamia työkaluja. Haikalan ja Mikkosen (2001, 189) mukaan sovelluskehys on kokoelma jonkin arkkitehtuurin mukaan toteutettuja luokkia, komponentteja ja rajapintakokoelmia, joihin sisältyy sovelluksen perustoiminnallisuus. Näitä osia voidaan käyttää uudelleen sovelluksen eri osissa sekä kokonaan erillisissä projekteissa. Sovelluskehyksen tarkoituksena on etenkin helpottaa ja nopeuttaa sovelluksen kehitystä hyödyntämällä valmiita komponentteja ja noudattamalla testattuja, hyväksi todettuja ratkaisuja (Wayner 2015). Sovelluskehykset pohjautuvat lähes poikkeuksetta olio-ohjelmointiin. Sovelluskehyksen voi käsittää aukkoja sisältävänä ohjelmiston runkona, joka itsessään ei yleensä ole vielä käyttökelpoinen sovellus. Ohjelmoijan tehtävänä on täyttää nämä aukot omalla koodillaan kehyksen arkkitehtuuria noudattaen saavuttaakseen halutun lopputuloksen. Esimerkkejä tunnetuimmista kehyksistä ovat graafisten käyttöliittymien rakentamiseen tarkoitetut kehykset, kuten Java Swing. Muita yleisiä käyttökohteita ovat esimerkiksi pelimoottorit, verkonhallintasovellukset sekä erilaiset graafiset editorit. (Haikala & Mikkonen 2011, 189.)

KUVA 1. Kirjaston ja sovelluskehyksen ero (Petricek 2015 mukaillen)

Sovelluskehys sekoitetaan terminä useasti kirjastoihin, sillä ero niiden välillä ei välttämättä ole kovin selkeä. Näiden kahden välistä eroa on havainnollistettu kuvassa 1. Kirjasto on kokoelma koodia, joka useimmiten hoitaa jonkin määrätyn tehtävän. Kirjaston tavoitteena on koodin uudelleenkäytettävyys, eli toistuvia toimintoja ei tarvitse ohjelmoida jokaisella kerralla uudelleen. Kirjastot toimivat liitännäisinä, eli ne tarjoavat rajapinnat, joita kutsumalla omasta koodista voidaan toteuttaa kirjaston tarjoama toiminto. (Petricek 2015.) Oleellinen ero kirjaston ja sovelluskehyksen välillä on siinä, kuka ja miten koodia kutsutaan. Perinteisessä ohjelmistosuunnittelussa sovellus toteutetaan käyttämällä apuna valmiita kirjastoja, joita sovellus kutsuu. Sovelluskehyksissä tämä suhde toteutuu päinvastoin; kehittäjä toteuttaa luokkia joita puolestaan sovelluskehys kutsuu. (Porebski ym. 2011, 2.)

2.1 Hyötyjä ja haittoja

Sovelluskehykset pyrkivät helpottamaan kehittäjien työtä monella tavalla. Sovelluskehyksen käyttäminen kehityksen työvälineenä tarkoittaa, että sovelluksen rakenteet ja toimintaperiaatteet ovat jäsennelty hyväksi havaittujen tapojen mukaan ja samalla koodin ylläpitämisestä ja jatkokehittämisestä tulee helpompaa (Symfony 2015). Tämä helpottaa myös tiimityöskentelyä projektissa, kun kaikilla osallistujilla on selkeät ohjeet joita noudattaa. Valmiiden koodikirjastojen käyttäminen säästää aikaa useasti toistuvien toimintojen tekemisessä ja osia voidaan viedä myös toisiin sovelluksiin. (Jacobs 2014.) Sovelluskehys tarjoaa yleisesti ottaen myös hyvän tietoturvan, sillä koodi on laadukasta ja laajasti testattua (Potencier 2013). Jacobsin (2014) mukaan sovelluskehykset ovat useimmiten kattavasti dokumentoituja, joten uusienkin kehittäjien on mahdollista tulla projektiin mukaan lyhyellä koulutuksella omaamatta aiempaa kokemusta tekniikasta. Lisäksi suosituilla sovelluskehyksillä on internetissä aktiivinen käyttäjäyhteisö joilta voi kysyä asiantuntevaa apua ongelmatilanteissa.

Sovelluskehysten käyttäminen vaatii kehittäjältä myös tiettyjen kompromissien ja haasteiden tiedostamista. Moonin (2011) mukaan mahdollisesti suurin ongelma sovelluskehyksissä on se, etteivät ne ole suorituskyvyltään koskaan yhtä tehokkaita kuin yksittäiseen käyttötarkoitukseen rakennettu optimoitu koodi. Petersonin (2008) blogikirjoituksen mukaan PHP:n luoja Rasmus Lerdorf on todennut suorituskykyongelman koskevan myös PHP-sovelluskehyksiä, vaikka PHP:n suorituskykyä on perinteisesti pidetty varsin hyvänä. Porebski ym. (2011, 3) mukaan huono suorituskyky johtuu so-

velluskehysten kehityshistoriasta. Alkujaan kehittäjät tekivät sovelluskehyksiä ainoastaan helpottaakseen omia tarpeitaan varten ajattelematta sen suuremmin sovelluskehyksen tulevaisuuden käyttökohteista. Seuraavat kehittäjäryhmät laajensivat sovelluskehyksen toimintoja koodilla, joka oli kuitenkin usein huonosti yhteensopivaa aiemman koodin kanssa. Jossain vaiheessa kehittäjät huomasivat sovelluskehyksen tarvitsevan täydellisen uudelleenkirjoituksen. Tämän kaltainen kehityskaari on hyvin havaittavissa esimerkiksi suosittujen PHP-sovelluskehysten kohdalla. Ajan myötä koodin laatu ja kehysten suorituskyky on parantunut merkittävästi, mutta väittämä suorituskyvystä on edelleen voimassa etenkin laajojen sovelluskehysten kohdalla.

Ongelmia aiheuttaa myös sovelluskehysten valtava määrä eri kielille ja niiden eroavaisuudet toisistaan. Uuteen sovelluskehyksen opettelu voi toisinaan vaatia kehittäjältä paljon aikaa ja tämä luo myös haasteita yrityksille, jotka tuottavat koodia asiakkaiden olemassa oleviin ohjelmistoihin (Haikala & Mikkonen 2011, 190). Mikäli sovelluskehys ei tarjoa valmista ratkaisua tai työkalua johonkin ongelmaan, voi sen ratkaiseminen olla monimutkaista, ja samalla voidaan pohtia sovelluskehyksen perimmäistä tarkoitusta, eli kehityksen helpottamista. Vaikka lähtökohtaisesti sovelluskehykset parantavat tietoturvaa, aiheuttavat ne sen suhteen myös oman riskinsä. Laajasti käytössä olevasta koodista löydetty haavoittuvuus on hyödynnettävissä kaikissa samaa sovelluskehystä käyttävissä ohjelmissa. Tietoturva-aukko voi olla haasteellista korjata muiden kuin alkuperäisten kehittäjien toimesta, ja sovelluksesta asiakkaalle syntyneet ongelmat jäävät lopulta aina sovelluskehyksen käyttäjän vastuulle. (Potencier 2013.)

2.2 PHP-sovelluskehykset

PHP on etupäässä verkkosivujen kehitykseen tarkoitettu palvelinpään ohjelmointikieli. Rasmus Lerdorfin kehitti PHP:n vuonna 1994 helpottamaan omien verkkosivujensa kehitystä. Lerdorf halusi nähdä, ketkä kävivät katsomassa hänen kotisivuillaan sijainnutta ansioluetteloaan ja päätyi kirjoittamaan C-kielellä joukon kirjastoja, joista muodostui PHP. Nimi PHP oli alun perin lyhenne sanoista Personal Home Page, mutta nykyään virallinen nimi kielelle on PHP: Hypertext Preprocessor. PHP:n suosio on kasvanut voimakkaasti alusta alkaen, sillä jo julkaisua seuranneena vuonna 1995 PHP:llä tehtyjä sivustoja oli noin 15 000 ja sitä seuranneena vuonna jo 50 000. (Holzner 2008.) Iden artikkelissa (2013) mainitun tutkimuksen mukaan vuonna 2013 vas-

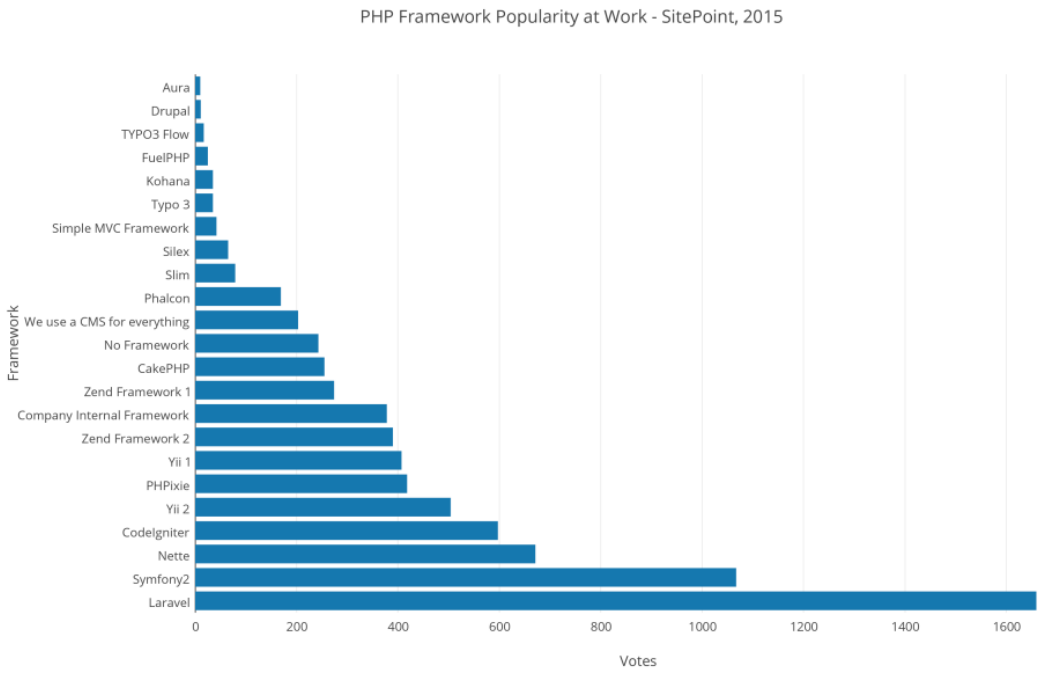
taava luku oli noin 244 miljoonaa ja määrä on edelleen kasvussa. PHP:stä onkin tullut selvästi suosituin internetpalvelinten ohjelmointikieli.

Ennen PHP:tä verkkosivut olivat lähinnä staattisia HTML-sivuja ja dynaamista sisältöä oli mahdollista luoda esimerkiksi C- tai Perl-kielillä (Moon 2011). Kehittäminen oli kuitenkin hyvin työlästä, sillä koko HTTP-prosessi piti käsitellä ja määritellä kokonaisuudessaan itse. PHP tarjosi ongelmaan ratkaisun sisäänrakennetulla HTTP-pyyntöjen käsittelijällä ja mahdollistamalla esimerkiksi PHP-koodin sisällyttäminen HTML-dokumenttien sekaan. Moon (2011) toteaakin PHP:n olevan itsessään tietyntyyppinen sovelluskehys. PHP on korkean tason komentosarjakieli, eli sitä ei käännetä perinteisten ohjelmointikielten, kuten Javan tapaan, vaan se tulkitaan suoritusvaiheessa erillisen PHP-tulkin avulla. PHP-tulkki on saatavilla kaikille suosituimmille käyttöjärjestelmille. Yksi PHP:n suosioon vaikuttavista tekijöistä onkin ollut sitä tukevien edullisten palvelinten valtava määrä (Porebski ym. 2011, 5). Lukuisat PHP-sovelluskehykset juontavat juurensa vuoteen 2005, jolloin Ruby on Rails -sovelluskehys nousi suureen suosioon verkkosivujen kehityksessä. Ruby on Rails -sovelluskehyksestä pidettiin erityisesti sen helppolukuisen koodin sekä olio-rakenteen vuoksi. PHP:n versio 5 mahdollisti olio-ohjelmoinnin myös PHP:ssä, jolloin myös PHP-sovelluskehysten määrä alkoi kasvamaan nopeasti. Porebskin ym. (2011, 5) mukaan Ruby on Rails -sovelluskehyksen suosion takia lähes kaikki PHP-sovelluskehykset olivat aluksi jonkin asteisia kopioita siitä.

Kuten sovelluskehyksissä yleisesti, Petersonin (2008) mukaan suurin ongelma myös PHP-sovelluskehysten kanssa on perinteisesti ollut huono suorituskyky verrattuna alemman tason PHP-koodiin. Tulevan PHP7-version pitäisi tuoda helpotusta ongelmaan parantamalla PHP:n alemman ohjelmointikerroksen suorituskykyä merkittävästi (Shafik 2015). Huono suorituskyky aiheuttaa ongelmia sovelluksen käyttäjien lisäksi myös esimerkiksi palvelinkeskuksissa, joissa optimoidulla koodilla voitaisiin alentaa palvelinten käyttökapasiteettiä ja samalla sähkönkulutusta

Kuvassa 2 on nähtävissä SitePoint-sivuston tekemän PHP-sovelluskehysten suosiota kartoittaneen kyselyn tulokset. Tuloksista voidaan havaita, että Laravel ja Symfony2 ovat tällä hetkellä selvästi suosituimmat PHP-sovelluskehykset. Viiden suosituimman joukkoon sisältyvät myös Nette, CodeIgniter sekä Yii2. Huomattavaa on myös ilman

sovelluskehystä toimivien vastanneiden määrä. Kaikki suosituimmat PHP-sovelluskehykset perustuvat MVC-arkkitehtuuriin, josta lisää seuraavassa luvussa.



KUVA 2. PHP-sovelluskehysten keskinäinen suosio (Skvorc 2015)

Laravel on osittain Symfony2-sovelluskehyksen sisältämien komponenttien päälle rakennettu verrattain tuore PHP-sovelluskehys. Ensimmäinen versio julkaistiin vuonna 2011 ja tuorein versio 5 vuoden 2015 helmikuussa. Sovelluskehys vaatii toimiakseen PHP:n version 5.5.9 ja sen käyttäminen on ilmaista MIT-lisenssin alaisena. Laravel perustuu komponentteihin, joiden asennus ja riippuvuudet hallitaan Composer:llä, eli PHP:lle kehitetyllä riippuvuuksienhallintatyökalulla. Laravelin omien sanojensa mukaan sen tärkeimpiä ominaisuuksia ovat käyttöönoton helppous sekä sovelluskoodin selkeä rakenne. Laravel-projekti sisältää oletuksena muun muassa Eloquent ORM olio- ja relaatiomallinnuskomponentin sekä Blade-templaattimoottorin. Versiosta 3 lähtien yksi sovelluskehyksen tärkeimmistä työkaluista on ollut Artisan-komentorivikomponentti, jonka avulla voidaan esimerkiksi luoda valmista koodia, hallita asennettuja kirjastoja tai tehdä toimenpiteitä ORM-komponentilla. Versiossa 4 sovelluskehys koki suuren uudelleenkirjoituksen, jonka jälkeen kehyksen suosio on kasvanut erittäin voimakkaasti ja Laravelista on tullut tämän hetken suosituin PHP-sovelluskehys (Skvorc 2015). Laravelin kotisivusto sisältää erittäin kattavan dokumentaation, laajan kokoelman video-oppaita sekä aktiivisen käyttäjäyhteisön. (Way 2012.)

Nette on tšekkiläisen Nette Foundationin kehittämä PHP-sovelluskehys, jonka kehitys aloitettiin vuonna 2008. Sen uusin versio 2.3 on julkaistu kesäkuussa 2015. Nette on asennettavissa ilmaisten New BSD- ja GNU GPL-lisenssien alaisuudessa. Laravelin tapaan myös Nette perustuu toisistaan riippumattomiin PHP-komponentteihin ja vaatii toimiakseen PHP version 5.3.1. Nettellä on erittäin aktiivinen käyttäjäyhteisö, mutta sovelluskehyksen heikkona puolena on tällä hetkellä puutteellinen dokumentaatio sekä eräiden ominaisuuksien keskeneräisyys verrattuna esimerkiksi Symfony2:een. (Ren 2015.)

CodeIgniter on alun perin EllisLab:n vuonna 2006 kehittämä PHP-sovelluskehys. Vuodesta 2014 lähtien sovelluskehystä on kehittänyt British Columbia Institute of Technology. Sovelluskehyksen uusin 3.0-versio julkaistiin maaliskuussa 2015 ja se vaatii toimiakseen PHP version 5.2.4. Myös CodeIgniter on ilmainen MIT-lisenssin alaisuudessa. Sovelluskehyksessä on panostettu etenkin konfiguraation helppouteen sekä suorituskykyyn. Tästä johtuen CodeIgniter on yksi nopeimmista suosituista PHP-sovelluskehyksistä. Viime vuosina CodeIgniter:n suosio on kuitenkin laskenut lähinnä modernimpien Laravel- ja Symfony2-sovelluskehysten suosion kustannuksella. (Stones 2015.)

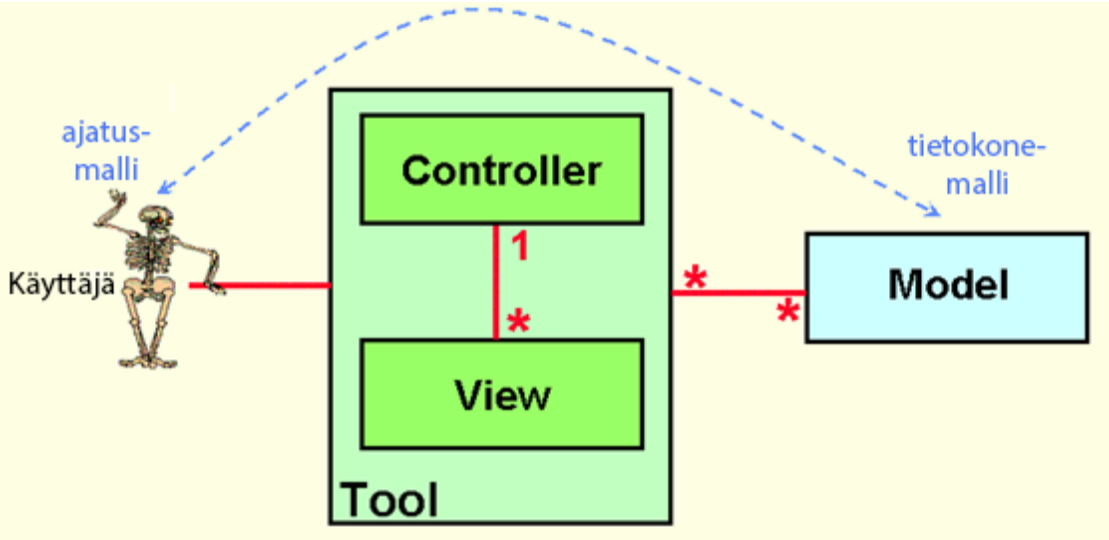
Qiang Xue julkaisi kehittämänsä Yii-sovelluskehyksen vuonna 2008 ja sen täysin uudelleenkirjoitettu versio 2.0 julkaistiin vuonna 2014. Yii2 on omien sanojensa mukaan komponentteihin perustuva, etenkin suorituskykyyn ja tietoturvaan panostava PHP-sovelluskehys. Yii2 on kehittäjille ilmainen BSD-lisenssin alaisuudessa. Sovelluskehys vaatii toimiakseen PHP version 5.4.0 tai uudemman. Komponenttien riippuvuudet ja asentaminen hoidetaan Laravelin tapaan Composer:llä. Yii2:ssa on myös integroituna tuki jQuery JavaScript-kirjastolle sekä Twitter Bootstrap HTML-sovelluskehykselle. (Beaumont 2014.)

Yksi toistuvista kysymyksistä liittyen PHP-sovelluskehyksiin on, milloin ja millaisissa projekteissa niitä tulisi käyttää. Yleisesti ottaen sovelluskehyksen käyttöä ei suositella aloitteleville ohjelmoijille, sillä on tärkeää hallita PHP:n perusteet ja sisäistää sovelluskehyksen käytöstä syntyvät hyödyt ja haitat ennen käyttöön siirtymistä. Kuten sovelluskehykset yleisesti, PHP-sovelluskehykset nopeuttavat kehitystyötä sekä helpottavat luotettavan ja tietoturvallisen koodin luomista (Jacobs 2014). Symfony2:n kotis-

ivujen mukaan (Why should I use a framework? 2015) sovelluskehys on yksinkertaisuudessaan pelkkä työkalu, joka ei mahdollista mitään sellaista, jota ei olisi mahdollista toteuttaa omalla koodilla. Porebskin ym. (2011, 3–4) mukaan sovelluskehys ei tarjoa hyötyä pienissä projekteissa, joissa sivut ovat sisällöltään ainoastaan staattista informaatiota tai tietokantakyselyiden määrä on pieni. Sovelluskehystä ei voida myöskään käyttää silloin, kun kehittäjän täytyy pystyä hallitsemaan sovelluksen lähdekoodia kokonaisuudessaan. PHP-sovelluskehykset sopivat kuitenkin suurimpaan osaan dynaamisten web-sovellusten toteuttamista, joissa sovelluksen toiminta perustuu tietokantoihin, joiden sisältöä käyttäjät muokkaavat. Esimerkkejä tällaisista sovelluksista ovat esimerkiksi sosiaalisen median sovellukset, blogit sekä verkkokaupat.

2.3 MVC- arkkitehtuuri

MVC-arkkitehtuuri on etupäässä käyttöliittymäohjelmointia varten suunniteltu sovellusmalli, jossa sovelluksen käyttöliittymä, toiminnalliset osat määritellään toisistaan erillään (Kasurinen 2011, 149). MVC on lyhennys sanoista Model-View-Controller, eli malli-näkymä-käsittelijä. Norjalainen Trygve Reenskaug kehitti MVC-arkkitehtuurin Smalltalk-ohjelmointikieltä varten työskennellessään Xerox PARC-tutkimuskeskuksessa vuonna 1978 (Reenskaug 2010). MVC-arkkitehtuuri suunniteltiin alun perin työpöytäsovellusten käyttöliittymien ohjelmointiin, mutta siitä on tullut erittäin suosittu etenkin internetsovellusten ja sovelluskehysten keskuudessa (Porebski ym. 2011, 15). Reenskaugin (2010) mukaan mallin perimmäinen tarkoitus on yhdistää käyttäjän kuvitteellinen sekä tietokoneen digitaalinen malli yhdeksi toimivaksi kokonaisuudeksi (kuva 3), jonka toimintamalli noudattaa määrättyjä sääntöjä.



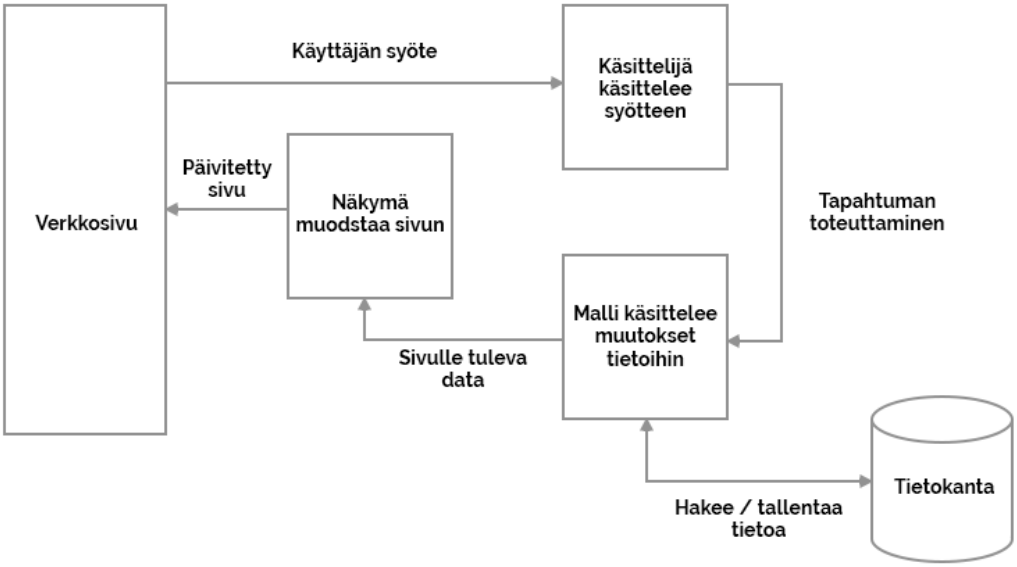
KUVA 3. MVC-arkkitehtuurin rakenne (Reenskaug 2010 mukaillen)

MVC-arkkitehtuurissa malli pitää sisällään sovelluksessa käytettävän tiedon sekä metodit tiedon käsittelemiseen. Useimmiten malli kuvaa tietokannan yksittäisen taulun yksittäistä riviä. Malli vastaa muun muassa pyyntöihin tietojen hakemisesta tai muuttamisesta. Se vastaanottaa käskyjä käsittelijältä ja välittää tarvittavat tiedot suoraan näkymälle. Malli pitää sisällään tiedon käsittelyyn tarvittavat toimenpiteet, mutta ei itse ole tietoinen mihin tai miten sen sisältämää tietoa käytetään. (Ullman 2009.) Lisäksi malli voi olla yhteydessä muihin malleihin (Kasurinen 2011, 151).

Näkymä kattaa sovelluksen käyttöliittymän, eli sen osan, jonka sovelluksen käyttäjä näkee. Näkymä hakee käsittelijän pyynnöstä tarvittavan tiedon mallilta ja yhdistää tiedot sivun ulkoasuun luoden käyttäjälle käyttöliittymän (Reenskaug 2010). Verkkosovelluksissa näkymä on usein sivupohja, eli templaatti, johon templaattimoottori (engl. template engine) tulostaa mallin sisältämät tiedot. Pelkkä HTML-dokumentti ei siis ole vielä MVC-arkkitehtuurin mukainen näkymä (Ullman 2009).

Käsittelijä sisältää sovelluksen varsinaisen bisneslogiikan, joka ohjaa tiedonkulkua sovelluksessa. Se on ainut osa sovelluksesta, jonka toimintoihin käyttäjä on suoraan yhteydessä. Käsittelijä ottaa vastaan käyttäjän toiminnon, suorittaa tarvittavat toimenpiteet mallille ja käskee näkymää päivittymään. Vaikka MVC-arkkitehtuuria kuvattaessa puhutaan usein yhdestä mallista, näkymästä ja käsittelijästä, voi yksittäinen käsittelijä olla myös yhteydessä useisiin malleihin ja näkymiin. (Hopkins 2013.)

Kasurisen (2011, 150) mukaan MVC-arkkitehtuurin mukaisesti suunnitellun sovelluksen toimintamallissa käyttäjä antaa jonkinlaisen syötteen näkymään, esimerkiksi kirjoittaa viestin keskustelupalstalle. Käsittelijä saa näkymään annetun syötteen ja luo sen pohjalta uuden viestiä kuvaavan mallin. Malli tallennetaan tietokantaan, jonka jälkeen käsittelijä käskee näkymään päivittämään itsensä. Näkymä pyytää mallilta uudet tiedot ja näyttää uuden kirjoitetun viestin käyttäjälle. Tämän jälkeen käyttöliittymä jää odottamaan uutta syötettä, jolloin prosessi alkaa alusta. Toimintamalli on havainnollistettuna kokonaisuudessaan kuvassa 4.



KUVA 4. MVC-arkkitehtuurin toimintamalli (Kasurinen 2011, 150)

Etenkin verkkosovelluksissa MVC-arkkitehtuuria käytetään usein hieman alkuperäistä toimintamallia soveltaen. MVC-arkkitehtuurin päälle onkin muodostunut niin kutsuttu MVP-arkkitehtuuri (Model-View-Presenter), jossa käsittelijä (Controller) on korvattu Presenter-komponentilla. Suurin ero näiden kahden arkkitehtuurin välillä on siinä, että Presenter toimii eräänlaisena välikomponenttina mallin ja näkymän välillä hoitaen niiden tiedonvälityksen, kun taas MVC-arkkitehtuurissa malli ja näkymä ovat suoraan yhteydessä toisiinsa. MVC-arkkitehtuuri on vakiintunut käsitteenä PHP-sovelluskehysten yhteydessä huolimatta siitä, että todellisuudessa suurin osa niistä toteuttaakin MVP-arkkitehtuuria. (Porebski ym. 2011, 15.)

3 SYMFONY2

Ranskalainen SensioLabs alkoi kehittää omaa Sensio Framework -sovelluskehystä vuonna 2005 helpottamaan yrityksen PHP-verkkosovellusten kehitystä. Sovelluskehys todettiin yrityksen sisällä onnistuneeksi työkaluksi, jonka myötä pääkehittäjä Fabien Potencier päätti samana vuonna julkaista sovelluskehyksestä avoimen lähdekoodin version, joka sai nimekseen Symfony. Symfony perustui alun perin vanhaan Mojavi PHP-sovelluskehykseen ja sai paljon vaikutteita muun muassa Ruby on Rails -sovelluskehyksestä. (Porebski ym. 2011, 8.) Alkuperäistä versiota kehitettiin kuuden vuoden ajan, kunnes vuonna 2011 julkaistiin Symfony2. Symfony2 on kokoelma uu-

delleenkäytettäviä itsenäisiä PHP-komponentteja, jotka yhdessä muodostavat sovelluskehyksen. Symfony-sovelluskehystä ei tule sekoittaa Symfony CMF-projektiin, joka on täysin erillinen, Symfonyn komponenttien pohjalle kehitetty sisällönhallintajärjestelmä. Symfony2:n oppimiskäyrää pidetään yleisesti ottaen varsin jyrkkänä verrattaessa muihin PHP-sovelluskehyksiin, mutta toisaalta sillä on erittäin kattava ja selkeä dokumentaatio sekä aktiivinen käyttäjäyhteisö. Oman dokumentaationsa lisäksi internetistä löytyy lukematon määrä oppaita tekstin ja videon muodoissa. (Porebski ym. 2011, 8.)

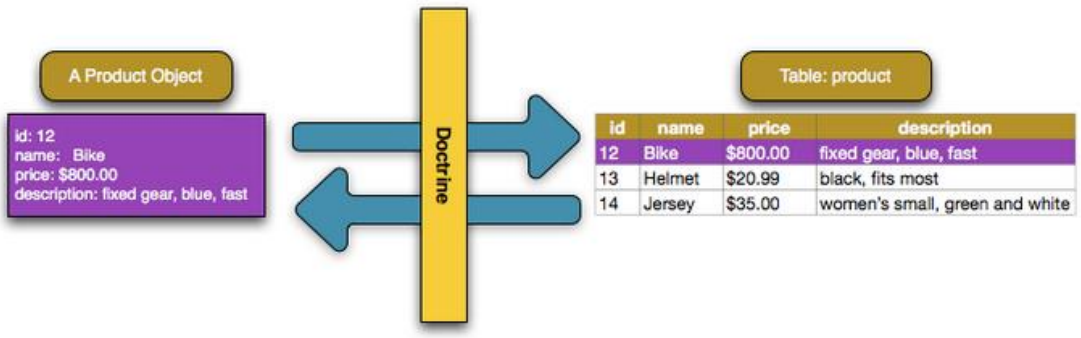
Symfony2 vaatii toimiakseen PHP-version 5.3.9 tai sitä uudemman. Opinnäytetyössä käsitellään kirjoittamishetkellä Symfonyn uusinta versiota 2.7. Versio 3.0 on tulossa ensi vuonna, mutta sen tuomat muutokset eivät ole yhtä suuria kuin aiemmassa versionumeron vaihdossa. Seuraavat luvut perustuvat pääosin Symfonyn omaan dokumentaatioon, sillä Symfony2 on verrattain tuore julkaisu, eikä muuta luotettavaa lähdemateriaalia ole saatavilla.

3.1 Komponentit

Symfony2:n peruspilareina toimivat komponentit. Ne ovat toisistaan riippumattomia uudelleenkäytettäviä PHP-kirjastoja, joita voidaan hyödyntää missä tahansa PHP-projektissa. Symfony Full Stack, eli Symfony2-sovelluskehyksen perusversio sisältää oletuksena 26 komponenttia, jotka hoitavat sovelluskehyksen ydintoiminnot. Symfony2-sovelluksen kehityksessä käytetään runsaasti konsoli-komponenttia, jolla voidaan tehdä laaja määrä erilaisia toimenpiteitä välimuistin tyhjentämisestä koodin luontiin. Kuten jo aiemmin todettiin, myös Laravel-sovelluskehys hyödyntää suuren joukon Symfonyn komponentteja. Muita esimerkkejä Symfonyn komponentteja käyttävistä projekteista ovat Drupal-sisällönhallintajärjestelmä ja phpBB-foorumisovellus.

Oletuksena Symfony2 käyttää Doctrine ORM-komponenttia, joka toteuttaa olio-relaatio-mallinnusta (engl. Object-Relational Mapping). Sen tarkoituksena on helpottaa tietokantojen hallintaa ja ylläpitää tiedon eheyttä. ORM toimii tietokannan ja mallien välisenä kerroksena (kuva 5), mahdollistaa kokonaisten mallien automaattisen tallentamisen ja noutamisen tietokannasta sen sijaan, että mallin sisältämät tiedot jouduttaisiin käsittelemään tietokantaa vasten yksitellen. Tämä nopeuttaa kehitystä merkittävästi, sillä ohjelmoijan ei tarvitse myöskään huolehtia tietokantayhteyksistä tai

monimutkaisista tietokantaoperaatioista. Doctrine mahdollistaa myös objektien ja tietokantojen helpon ylläpitämisen, esimerkiksi uuden muuttujan tarvitsee lisätä ainoastaan tietokantamalliin, jonka jälkeen Doctrine hoitaa tietokannan päivittämisen vastaamaan muuttunutta mallia. Kehitysvaiheessa Doctrinen lukuisia toimintoja suoritetaan konsolin kautta. (Databases and Doctrine 2015.)



KUVA 5. Doctrine toimintamalli (Databases and Doctrine 2015)

Doctrine mahdollistaa myös niin sanotut code-first- ja database-first-lähestymisen uuden sovelluksen kehittämiseen. Code-first-ohjelmointi on paras ratkaisu, kun sovellusta ryhdytään kehittämään täysin puhtaalta pöydältä ilman mitään olemassa olevaa tietoa. Doctrine hoitaa tarvittavien tietokantarakenteiden luomisen ja hallinnan tietokantamallien mukaisesti. Vastaavasti database-first-ohjelmointi mahdollistaa sovelluksen kehittämisen käyttäen olemassa olevaa tietokantaa ja luomalla sitä vastaavat tietokantamallit. Tämä on hyödyllistä etenkin kun halutaan vaihtaa kokonaan ohjelmointikieltä, mutta uuden sovelluksen tulee olla yhteensopiva vanhan tietokannan ja sen sisältämän tiedon kanssa. Doctrine on suosituin Symfony2-projekteissa käytetyistä ORM-työkaluista, mutta kuten kaikki komponentit, se on täysin riippumaton muista komponenteista eikä sen käyttö ole pakollista. Symfony2:n kanssa voidaan käyttää myös esimerkiksi Propelia, joka on hyvin samankaltainen ORM-kirjasto kuin Doctrine. (Databases and Doctrine 2015.)

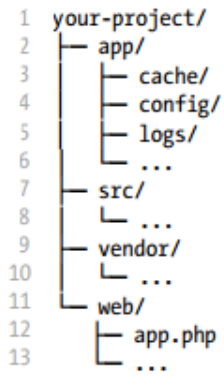
```
{% extends 'base.html.twig' %}

{% block title %}My cool blog posts{% endblock %}

{% block body %}
    {# Page content goes here #}
    {% for entry in blog_entries %}
        <h2>{{ entry.title }}</h2>
        <p>{{ entry.body }}</p>
    {% endfor %}
{% endblock %}
```

KUVA 6. Esimerkki Twig-templaatista (Creating and Using Templates 2015 mukaillen)

Symfony2 käyttää oletuksena templaattimoottori Twig:tä, joka on myös SensioLabs:n kehittämä komponentti. Sen tarkoituksena on helpottaa PHP:llä tuotetun sisällön sijoittamista HTML-koodin joukkoon. Se mahdollistaa myös useiden eriytettyjen sivupohjien ja koodi-lohkojen sisällyttämisen yhden dokumentin sisään. Käytännössä kaikki mitä Twig-templaattien avulla voidaan tehdä, voidaan tehdä myös perinteisesti sijoittamalla PHP-koodia HTML:n sekaan, mutta Twig selkeyttää syntaksia sekä helpottaa merkittävästi koodin ylläpitämistä. Twigissä käytetään kolmea erilaista tagia, jotka tekevät erilaisia asioita. *{{ ... }}* tulostaa jonkin muuttujan arvon, *{% ... %}* suorittaa toiminnon, kuten for-loopin ja *{# ... #}* mahdollistaa kommenttien kirjoittamisen. Esimerkki tagien käytöstä on nähtävissä kuvassa 6. (Databases and Doctrine 2015.)

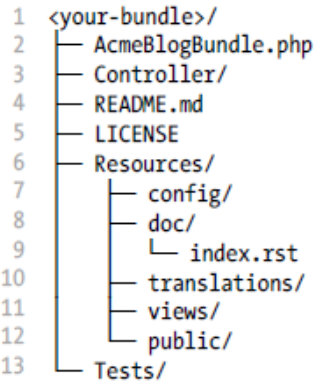
3.2 Projektin rakenne ja bundlet

Symfony2-projektin rakenne on erittäin joustava, mutta oletuksena sovelluksen kansiorakenne jakautuu neljään pääkansioon, *app*, *src*, *vendor* ja *web*. Projektin kansiopuu on kuvattu kuvassa 7. *App*-kansio sisältää sovelluksen asetustiedostot, käännökset sekä templaatit. Lisäksi sen alla sijaitsee kehittäjän kannalta erittäin tärkeät *cache*-kansio johon tallennetaan välimuistitiedostot, sekä *logs*-kansio johon sijoitetaan lokitiedostot. *Src*-kansion alle sijoitetaan kaikki sovelluksen bundlet. *Vendor*-kansiossa sijaitsee kaikki sovelluskehyksen ulkopuoliset riippuvaisuudet. *Web*-kansio puolestaan sisältää kaikki sovelluksen suoritusvaiheessa tarvittavat julkiset ja staattiset tiedostot, kuten kuvat, tiedostot, CSS-tyylitiedostot sekä JavaScript-tiedostot. *Web*-kansio sisältää myös niin sanotun Front Controller-luokan, jonka kautta kaikki pyynnöt kulkevat. (The Architecture 2015.)



KUVA 7. Symfony2-projektin rakenne (The Cookbook 2015, 69)

Projektin ei ole välttämätöntä noudattaa oletuksena määritettyä kansiorakennetta ja kansiorakenteen voi määrittää vapaasti. Välimuistitiedostot kannattaa kuitenkin eriyttää aina käytössä olevan ympäristön mukaan eri kansioihin, sillä eri ympäristöissä tuotettavat välimuistit voivat sisältää keskenään yhteen sopimatonta tietoa. Välimuisti- ja lokikansioiden sijainti voidaan määrittää *app*-kansion alla sijaitsevassa *AppKernel.php*-tiedostossa. (The Cookbook 2015, 69.)



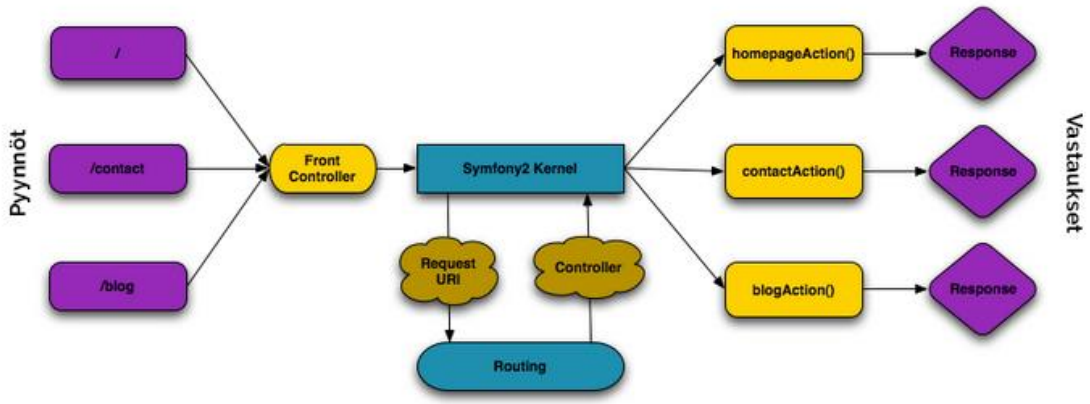
KUVA 8. Bundlen rakenne (The Cookbook 2015, 31)

Käytännössä kaikki ohjelmointityö Symfony2:ssa tehdään *src*-kansion alle sijoitettaviin bundleihin. Ne ovat eräänlaisia lisäosia, jotka toteuttavat jonkun suuremman kokonaisuuden, kuten blogin tai foorumin. Bundle on kansio, joka sisältää kaikki toteutettavaan toimintoon tarvittavat tiedostot, kuten PHP-tiedostot, CSS-tyylitiedostot, JavaScriptit ja kuvat. Bundlen täytyy sisältää bundle-luokka, joka muuntaa tavallisen kansion bundleksi. Määritellyn nimeämiskäytännön mukaan bundle-luokan nimen tulee olla kirjoitettuna yhteen isoin alkukirjaimin (camel case), päättyä sanaan bundle ja alkaa yrityksen nimellä. Esimerkkinä bundlen nimi voisi olla *CompanyCMSBundle*. Bundlen nimi saa sisältää ainoastaan aakkosnumeerisia merkkejä ja noudattaa PHP:n

nimiavaruuteen liittyviä sääntöjä. Kuvassa 8 bundle-luokka on *AcmeBlogBundle.php*. Symfony2:n konsoli mahdollistaa uuden bundlen ohjatun luomisen, jolloin kansiorakenne sekä tarvittavat PHP-luokat luodaan automaattisesti. (The Cookbook 2015, 30-31.)

3.3 Reititys, mallit ja ympäristöt

Tavallisesti pyynnöt kulkevat Symfony2:ssa kuvassa 9 näkyvän kaavan mukaan. Tapahtumaketju saa alkunsa, kun käyttäjä suorittaa pyynnön, eli esimerkiksi siirtyy selaimella määrättyyn url-osoitteeseen. Pyyntö vastaanotetaan Front Controller -luokassa, joka ohjaa sen Symfony Kernel -ydinkirjastolle. Kernel tulkitsee pyydetyn url-osoitteen ja etsii sitä vastaavan reititysasetuksen. Reittiasetus määrittää mihin käsittelijään ja metodiin pyyntö ohjataan. Tämän jälkeen käsittelijä vastaa pyynnön suorittamisesta, eli esimerkiksi tarvittavan tiedon käsittelemisestä sekä näkymän renderöinnistä. (Routing 2015.)



KUVA 9. Symfony toimintamalli (Routing 2015 mukaillen)

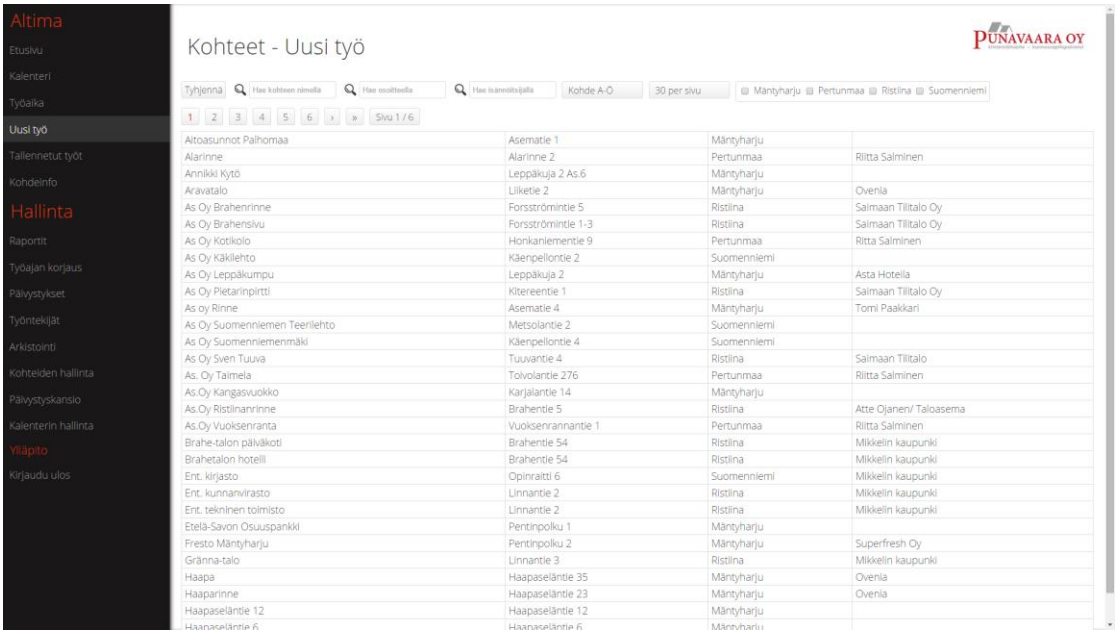
Symfony2:ssa MVC-arkkitehtuurin mukaista mallia kutsutaan entity:ksi. Mallit kuvaavat sovelluksen tarvitseman tiedon PHP-luokkina. Olio-relaatio-mallinnusta käytettäessä tietokanta luodaan mallin sisältämien muuttujien mukaiseksi. Muuttujiin voidaan liittää myös erilaisia annotaatioita, joihin on mahdollista määritellä esimerkiksi muuttujien tietotyypit. Näitä sääntöjä hyödynnetään muun muassa lomakkeiden muodostamisessa ja tiedon eheyden varmistamisessa. Lisäksi mallit sisältävät perinteiset niin sanotut getter- ja setter-metodit, joiden avulla tietoa viedään ja tuodaan luokalle. Entity-luokkien tueksi luodaan usein Repositorty-luokkia, jotka sisältävät kaikki tiedon käsittelemiseen tarvittavat monimutkaisemmat toimenpiteet. Repository-luokkien

tarkoituksena on keskittää kaikki usein käytetyt metodit yhden luokan alle, jota voidaan käyttää uudelleen missä tahansa sovelluksen osassa. (Routing 2015.)

Symfony2 mahdollistaa sovelluksen suorittamisen samalla tietokoneella erilaisissa ympäristöissä. Ympäristö on yksinkertaisuudessaan asetustiedosto, joka määrittelee, miten sovellus ajetaan. Symfony2 sisältää oletuksena kolme ympäristöä; tuotanto eli *prod*, kehitys eli *dev* sekä testaus eli *test*. Valmiiden ympäristöjen lisäksi omien ympäristöjen luominen halutuilla asetuksilla on myös mahdollista (The Cookbook 2015, 67). Oletusympäristöistä kehitysympäristö mahdollistaa muun muassa virheenkorjauksen, eli debuggauksen sekä hyödyllistä tilannetietoa sisältävän WebProfiler-työkalupalkin. Tuotantoympäristö puolestaan on optimoitu erityisesti tietoturvaa silmällä pitäen, sekä nopeudelle käyttämällä hyödyksi välimuistiin pakattuja tiedostoja. Ympäristö ladataan käyttämällä vastaavaa Front Controller -luokkaa, joka on esimerkiksi tuotantoympäristölle *app.php* ja *app_dev.php* kehitysympäristölle. Ympäristöjen erilaisia asetuksia hyödynnetään myös konsolikomentoja suoritettaessa. (The Cookbook 2015, 64.)

4 ALTIMA

Altima on Punavaara Oy:n käyttöön kehitetty internetselaimella käytettävä työajanseuranta- ja työnkirjausjärjestelmä. Se sisältää helppokäyttöisen työajanseurannan, josta työntekijä voi nappia painamalla aloittaa tai lopettaa työajan, kirjoittaa vapaamuotoisen kuvauksen työpäivästä sekä valita pitikö ruokataukoa työpäivän aikana. Kuukausittain työajoista muodostetaan työntekijäkohtaiset paperiset raportit, joiden perusteella palkat lasketaan. Sovellus erittelee raporttiin automaattisesti työntekijän tekemät tunnit sekä laskee mahdolliset tuntilisät kellonaikoihin määritettyjen rajojen mukaan. Toinen sovelluksen pääominaisuuksista on työnkirjaus, johon työntekijät tallentavat kaikki tehdyt laskutettavat työt. Työntekijä valitsee uutta työtä tallennettaessa kohteen missä työ on tehty ja syöttää sen jälkeen tarvittavat tiedot lomakkeeseen. Myös töistä muodostetaan kuukausittain paperiset raportit, jotka puolestaan syötetään edelleen yrityksen käyttämään laskutusohjelmaan. Kuvassa 10 on nähtävissä Altiman vanhan version ulkoasu.



KUVA 10. Altiman vanha ulkoasu

Altiman kehitys alkoi kesällä 2013 ja sovellus otettiin käyttöön jo muutaman kuukauden kehityksen jälkeen saman vuoden syksyllä. Aluksi koko järjestelmä tehtiin niin ominaisuuksien kuin koodinkin puolesta mahdollisimman yksinkertaiseksi. Sovelluksen PHP-koodi sijoitettiin suoraan HTML-koodin joukkoon ja myöhemmin toimintoja eriytettiin omiin luokkiin. Sovellusta kehitettiin jatkuvasti käytöstä saadun kokemuksen perusteella ja jokaisen muutoksen jälkeen koodin hallinnasta tuli haastavampaa. Ainoana järkevänä ratkaisuna ongelmien ratkaisuun pidin koko sovelluksen uudelleenkirjoittamista käyttäen tunnettua sovelluskehystä. Symfony2 valikoitui Altiman uuden version pohjaksi useasta syystä. Koska kehitän sovellusta ainakin toistaiseksi yksin, on sovelluskehyksen käyttö ainut mahdollisuus laadukkaan koodin tuottamiseen näin laajaksi kasvaneessa sovelluksessa. Symfony2:n valintaa puolsivat lukemani keskustelut asiantuntevilla keskustelufoorumeilla sekä dokumentaation kattavuus että selkeys ensisilmäyksellä. Laravel ei vaikuttanut mielestäni kiinnostavalta, vaikka tutkimusten mukaan se onkin suositumpi kuin Symfony2. Laravel vaikuttaa turhan monimutkaiselta, sillä se koostuu monien eri sovelluskehysten osista, joiden päälle on puolestaan kehitetty Laravelin omia kirjastoja. Ajatuksena tämä järjestely ainoastaan lisäsi mielikuvaa sovelluskehysten monimutkaisuudesta ja hitaudesta. Lisäksi Symfony2 tarjoaa pidempää tukea sen eri versioille kuin Laravel. Uusi sovellus tullaan kehittämään puhtaalta pöydältä, koska myös tietokannan suhteen vanhassa versiossa oli monia ongelmia. Uudet tietokantamallit suunnitellaan kuitenkin vanhaa tietokantaa silmällä pitäen, jotta olemassa oleva tieto voidaan siirtää kohtuullisella vaivalla uuteen

järjestelmään. Kuvia uuden version ulkoasusta on nähtävissä liitteessä 3. Sovellusta varten hankittiin lisenssi INSPINIA-nimiseen ulkoasuun, koska uuden ulkoasun kehittäminen olisi vienyt liikaa aikaa. Tästä johtuen sovelluksen käyttöliittymään ei käsitellä sen tarkemmin tässä opinnäytetyössä.

4.1 Kehitysympäristön ja uuden projektin luominen

Kuten kaikkien PHP-sovellusten, myös Symfony2-sovelluksen kehittämiseen tarvitaan PHP:tä tukeva palvelin. Kehitysvaiheessa asioiden helpottamiseksi käytetään paikallista palvelinta, kun taas tuotannossa käytössä on dedikoitu verkkopalvelin Debian Linuxilla. Paikalliseksi palvelinsovellukseksi valitsin XAMPP-paketin, koska sovelluksesta on olemassa versiot sekä Windowsille että UNIX-käyttöjärjestelmille, mukaan lukien OS X:n, jolla osa Altiman kehityksestä tehdään. XAMPP sisältää PHP:n, Apachen sekä MySQL-tietokantapalvelimen, jota voidaan hallita mukana tulevalla phpMyAdmin-hallintatyökalulla. Kaikki palvelimella käytettävät tiedostot sijoitetaan XAMPP-paketin asennuskansion alla sijaitsevaan *htdocs*-kansioon. Sovelluksen kehityksessä käytetään PHP:tä komentorivin kautta, jota varten XAMPP-paketin asennuskansion alla sijaitsevan *php*-kansion sijainti tulee lisätä Windowsin PATH-ympäristömuuttujaan. Lisäksi Symfony2 vaatii toimiakseen *php-intl*-lisäosan, jota käytetään sovelluskehyksen lokalisoinnissa. Lisäosa otetaan käyttöön etsimällä Apache-serverin *php.ini*-konfiguraatiotiedostosta rivi *;extension=php_intl.dll*. Rivin alusta poistetaan puolipiste ja muutokset tallennetaan. Kun Apache käynnistetään seuraavan kerran, lisäosa on käytettävissä.

Sovelluskehyksen käyttöä varten asennetaan myös PHP:n riippuvuuksienhallintasovellus Composer. Lisäksi Symfony suosittelee asentamaan *PHP accelerator*:n, jonka avulla sovelluksen PHP-suorituskykyä voidaan parantaa. Altimassa tähän tehtävään käytetään *Alternative PHP Cache*-lisäosaa (APC). Näiden asennusvaiheiden jälkeen voidaan luoda uusi tietokanta sovellusta varten käyttäen phpMyAdmin-hallintatyökalua. Suositeltu merkistö tietokantaa varten on *utf8_general_ci*. Tietokantapalvelimelle luodaan myös uusi MySQL-käyttäjä, jolle annetaan kaikki oikeudet kyseisen tietokannan käyttöön.

Editoriksi sovelluksen kehitykseen valitsin NetBeans IDE:n. NetBeans on helppo asentaa ja konfiguroida, jonka lisäksi se sisältää sisäänrakennetun tuen useille PHP-

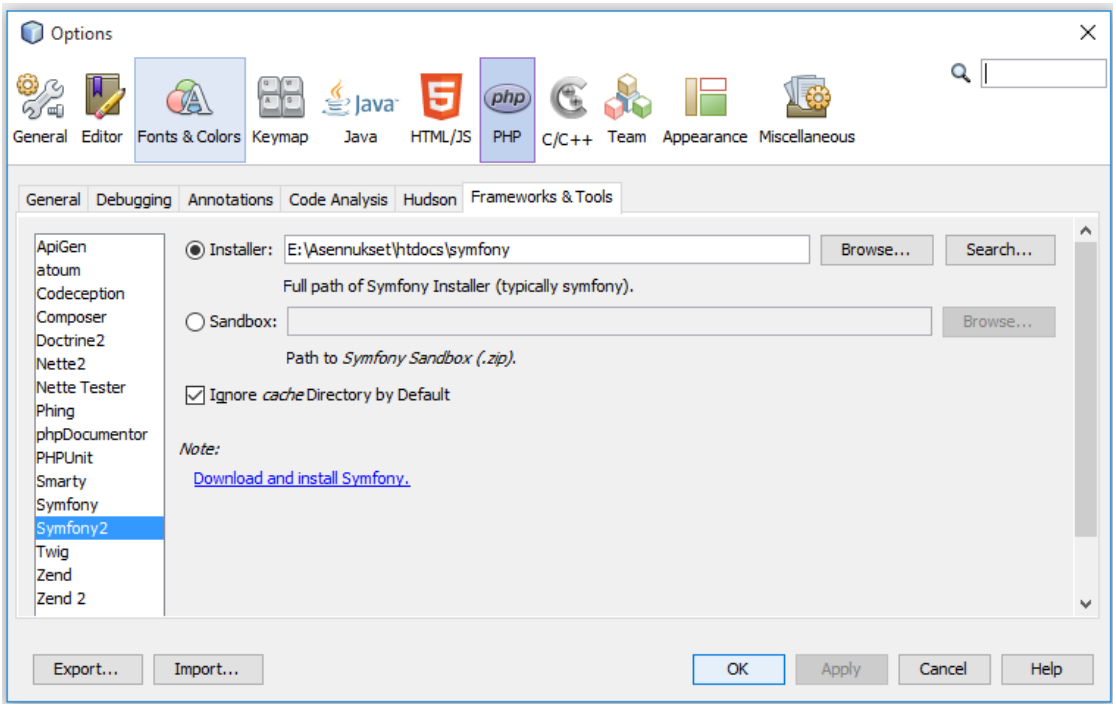
sovelluskehyksille, mukaan lukien Symfony2:n. Johtuen Symfony2:n äskettäin muuttuneesta jakelutavasta, jouduin kuitenkin käyttämään kehityksen aikana NetBeans:n kehittäjäversiota, sillä Symfony2-integraatio ei toiminut oikein uusimman virallisen version kanssa. Aiemmin Symfony jakoi projektin rungon ladattavana zip-pakettina suoraan kotisivuilta, mutta versiosta 2.7 lähtien asennus tapahtuu yhden PHAR-tiedoston (PHP Archive) avulla. NetBeans hakee tiedostosta muun muassa kaikki Symfony2:ssa käytettävät komennot, eikä niiden suorittaminen ole muuten mahdollista. Tulevaisuudessa julkaistavat NetBeans-versiot sisältävät tuen uudelle asennustavalle. Vaadittava PHAR-tiedosto suositellaan ladattavaksi kaikilla käyttöjärjestelmillä käyttäen komentoriviä (kuva 11).

```
C:\Users\Ville>E:

E:\>cd E:\Asennukset\xampp\htdocs

E:\Asennukset\xampp\htdocs>php -r "file_put_contents('symfony',
file_get_contents('http://symfony.com/installer'));"

E:\Asennukset\xampp\htdocs>php symfony
```

KUVA 11. Symfony Installerin lataus ja suorittaminen

Asennustiedosto ladataan haluttuun hakemistoon, joka tässä tapauksessa on XAMPP-asennuksen alla sijaitseva htdocs-kansio. Tiedosto sisältää Symfony Installer-asennusohjelman, jota käytetään myös uuden Symfony2-projektin luomiseen. Tiedosto suoritetaan komentorivillä käyttäen PHP:tä, jonka jälkeen käynnistyvä asennusohjelma ohjaa uuden projektin luonnissa. Altimaa varten uusi projekti luodaan komennolla *php symfony new altima*. Oletuksena asennusohjelma luo uuden projektin uusimman Symfony2:n uusimman version pohjalta, mutta halutessaan projektin voi luoda myös vanhemmille Symfony2-versioille lisäämällä haluttu versionumero komennon perään. Asennusohjelma luo projektille tarvittavat kansiot ja tiedostot.

Edellä ladatun *symfony.phar* tiedoston polku täytyy vielä asettaa NetBeans:n Symfony2-asetusten *Installer*-kohtaan (kuva 12). Symfony2-asetukset löytyvät editorin PHP-asetusten *Frameworks & Tools*-välilehden alta. Tämän jälkeen NetBeans tunnistaa edellä luodun projektikansion automaattisesti Symfony2-projektiksi ja editori tunnistaa nyt kaikki projektissa käytettävissä olevat komennot.



KUVA 12. NetBeansin Symfony-asetukset

Kun projektin pohja on luotu, voidaan aloittaa perusasetusten määrittäminen. Ensimmäisenä asetetaan tietokannan asetukset *app/config/parameters.ym*-tiedostoon (kuva 13). Tiedosto toimii sovelluksen keskitettynä konfiguraatiotiedostona, josta arvoja haetaan muihin sovelluksen asetustiedostoihin avainten avulla. Koska tässä kehitysympäristössä tietokanta sijaitsee samalla palvelimella kuin sovellus, isännäksi määritetään localhost tai 127.0.0.1. Mikäli porttia ei ole muutettu, se voidaan jättää tyhjäksi tai arvoksi voidaan asettaa oletusportti 3306. Tietokannan nimi, käyttäjätunnus ja salasana määritetään vastaamaan edellä luotujen tietokannan ja käyttäjän tietoja. Kuvassa 13 osa tiedoista on piilotettu tietoturvan vuoksi.

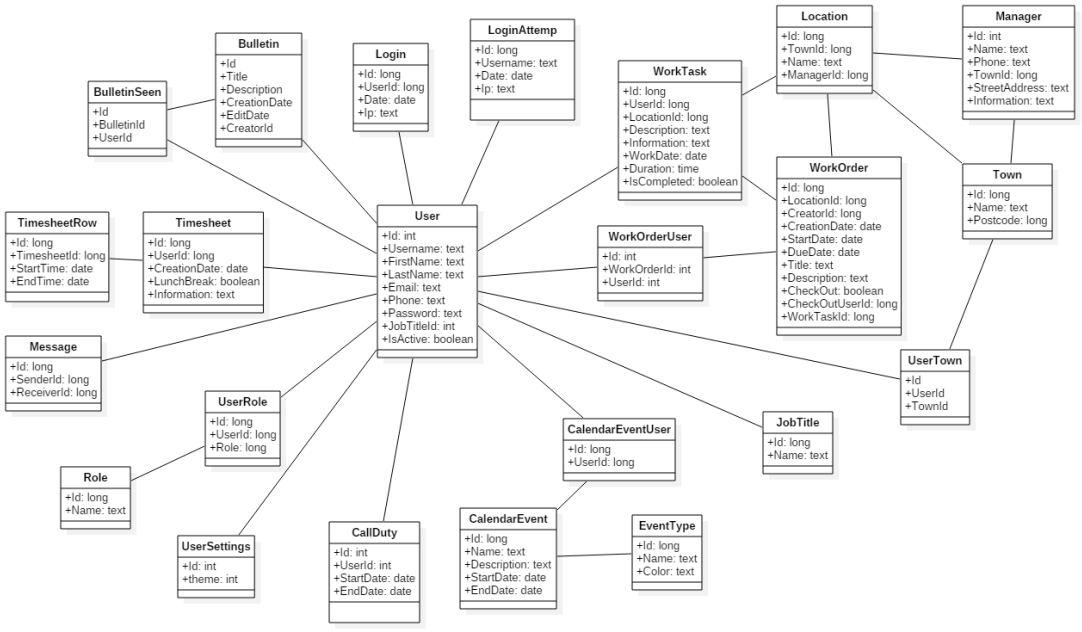
```
parameters:
    database_host: localhost
    database_port: null
    database_name:
    database_user:
    database_password:
    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: null
    mailer_password: null
    secret:
    database_driver: pdo_mysql
    database_path: null
```

KUVA 13. Parameters.yml

Kun projektin pohja on luotu, voidaan suunnata selaimella osoitteeseen *http://localhost/altima/web/config.php*. Avautuva sivu ilmoittaa, mikäli asennus tai ympäristö sisältää virheitä, jotka tulee korjata ennen jatkamista. Useimmiten ongelmat liittyvät etenkin UNIX-käyttöjärjestelmillä kansioiden käyttöoikeuksiin tai puuttuvaan *php-intl*-lisäosaan. Sekä käytettävällä palvelimella että komentorivin käyttäjällä tulee olla kirjoitusoikeudet *app/cache*- ja *app/logs*-kansioihin (Installing and Configuring Symfony 2015). Kun sivu kertoo kaiken olevan kunnossa, voidaan siirtyä varsinaiseen ohjelmointiin.

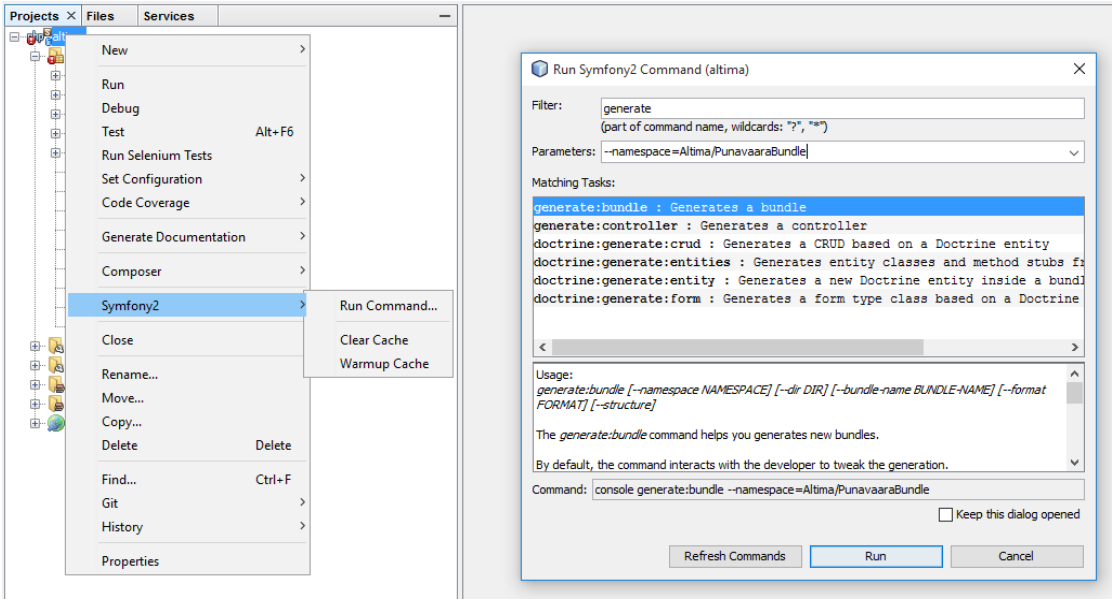
4.2 Sovelluksen kehitys

Kehityksen tueksi luotiin sovelluksen suunnitteluvaiheessa yksinkertainen luokkakaavio, johon määritettiin sovelluksessa tarvittavat mallit, eli entity-luokat. Luokkakaavio on erittäin hyödyllinen kehityksen työkalu, josta on paljon apua esimerkiksi tarvittavien reittien ja käsittelijöiden luonnissa. Luokkakaavio suunniteltiin toimeksiantajan kanssa keskusteltujen uusien ominaisuuksien pohjalta sekä mukailemalla vanhaa tietokantarakennetta. Eräät tietorakenteet, kuten työajat, suunniteltiin kuitenkin täysin uudelleen. Kuten kuvassa 14 esitetystä luokkakaaviosta voidaan todeta, käytännössä kaikki sovelluksen mallit ovat jollain tapaa yhteydessä käyttäjä-malliin. Tässä opinnäytetyössä käydään tarkemmin läpi yhden luokkakaavion osan, *WorkTask*:n kehitys.



KUVA 14. Altima-luokkakaavio

Kuten aiemmin todettiin, käytännössä kaikki ohjelmointi tehdään bundleihin. Toisin sanoen ohjelmointi aloitetaan luomalla uusi bundle. Tämä onnistuu Symfony2:n konsolikomennolla, joita voidaan suorittaa NetBeans:ssä klikkaamalla projektipuun juurta hiiren oikealla painikkeella ja valitsemalla Symfony2-valikon alta *Run Command…* (kuva 15). Aukeavasta ikkunasta valitaan komento *console generate:bundle*, jonka parametriksi syötetään *--namespace=Altima/PunavaaraBundle*. Komennon suoritus näkyy editorissa Output-ikkunassa, jossa ohjelma kysyy bundlen viitenimen ja tallennussijainnin, jotka voidaan jättää oletusasetuksille. Lisäksi komentoa suoritettaessa määritetään asetustiedostojen formaatti, joka voi olla yaml, xml, php tai annotaatiot. Altiman kehityksessä kaikissa asetuksissa käytetään yaml-formaattia, joka on ihmisen luettavissa oleva tiedon serialisointi-stantardi, jota voidaan käyttää millä tahansa kielellä. Komennossa on myös mahdollista luoda bundlelle suositeltava kansiorakenne sekä muutama välttämätön tiedosto.



KUVA 15. Uuden bundlen luominen

Bundlen luonnin jälkeen voidaan aloittaa mallien, reittien, käsittelijöiden ja näkymien ohjelmointi. Rakenteita aletaan määrittää luokkakaavion mukaisesti alkaen Doctrinen entity-luokista, eli MVC-arkkitehtuurin mukaisista malleista. Kuten luvussa kolme kerrottiin, entity on PHP-luokka, joka sijoitetaan tavallisesti bundlen alle *Entity*-kansioon. Uusi entity-luokka voidaan luoda konsolikomennolla *console doctrine:generate:entity*. Komennolle annetaan luotavan entity-luokan nimi sekä formaatti, jota käytetään Doctrinen määrityksissä. Altiman entity-luokissa käytetään formaattina ainoastaan annotaatioita. Luotu entity-luokka sisältää valmiina vaadittavan *id*-

muuttujan, jota käytetään tietokannassa tietueen avaimena sekä getter-metodin kyseiselle muuttujalle. Kuvassa 16 on nähtävissä Doctrinessa käytettäviä annotaatioita, jotka alkavat *@ORM*-merkinnällä. Näillä voidaan ohjeistaa Doctrinea muun muassa tietokannassa käytettävistä nimistä ja tietotyypeistä. Lisäksi annotaatioiden avulla määritetään säännöt, joiden mukaan muuttujien tietoja validoidaan sekä määritetään riippuvuudet toisiin entity-luokkiin.

```
/**
 * @ORM\Table(name="work_task")
 * @ORM\Entity(repositoryClass="Altima\PunavaaraBundle\Repository\WorkTaskRepository")
 */
class WorkTask
{
    /**
     * @var integer
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * Get id
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }
```

KUVA 16. Esimerkki WorkTask entity-luokasta

Entity-luokat ja objekti-relaatiomallinnus ovat erittäin suuri muutos vanhaan järjestelmään, jossa objekti-relaatiomallinnusta ei hyödynnetty laisinkaan. Kaikki tiedot ladattiin ja käsiteltiin aina tarpeiden mukaan joko suoraan HTML-koodin seassa tai erillisissä metodeissa, joista palautettiin toimintoa varten tehtyjä PHP-objekteja. ORM mahdollistaa samojen luokkien käyttämisen missä tahansa sovelluksen osassa, jonka lisäksi se hoitaa automaattisesti tiedon välityksen sovelluksen ja tietokannan välillä. Käytännössä tämä tarkoittaa, ettei käyttäjä joudu kirjoittamaan SQL-lauseita ollenkaan. Sen sijaan monimutkaisemmat tietokantahaut tehdään Repository-luokkiin käyttäen Doctrinen omaa Doctrine Query Languagea (DQL).

Entity-luokan luomisen jälkeen luokkaan lisätään loput tarvittavat muuttujat sekä määritellään niille tarvittavat annotaatiot. Tämän jälkeen suoritetaan konsolikomento *console doctrine:generate:entities AltimaPunavaaraBundle*. Komento käy läpi kaikki bundlen sisältämät entity-luokat ja luo kaikille muuttujille tarvittavat getter- ja setter-

metodit sekä luokalle Doctrinen EntityManagerin tarvitseman rakennusmetodin (constructor). Lopuksi suoritetaan komento *console doctrine:schema:update --dump-sql,* joka tarkistaa tietokannan rakenteet entity-luokkia vasten ja näyttää muutoksiin vaadittavat SQL-lauseet Output-ikkunassa. Mikäli komennon parametrin *--dump-sql* parametriin *--force,* komento suorittaa kyseiset tietokantatoimenpiteet automaattisesti. Näitä komentoja voidaan suorittaa uudestaan aina kun entity-luokkiin tehdään muutoksia, esimerkiksi lisätään uusi muuttuja. Komennot eivät vaikuta käyttäjän luomiin omiin metodeihin, eivätkä muutenkaan tavallisesti aiheuta muita toimenpiteitä. Näiden toimintojen avulla uuden tiedon lisääminen entity-luokalle voidaan tehdä erittäin helposti. Entisessä Altiman versiossa vastaava muutos olisi vaatinut tuntien työn, kun kaikki kyseiseen tietokantatauluun liittyvät tietokantafunktiot ja HTML-lomakkeet käsittelijöineen olisi täytynyt jokainen vuorollaan päivittää vastaamaan muutoksia.

Käsittelijä, eli *controller*, on MVC-arkkitehtuurin mukainen PHP-luokka, joka ohjaa mallia tiedon käsittelyssä sekä hallitsee näkymien näyttämisestä. Käsittelijöitä luodaan siis kaikille sovelluksen osille luokkakaavion mukaisesti, esimerkiksi Altiman tapauksessa *WorkTaskController*, *TimesheetController*, *BulletinController* ja niin edelleen. Käsittelijän nimi täytyy päättyä *Controller*-sana. Kuvassa 17 on esitetty esimerkkinä *WorkTaskController*:n hyvin yksinkertainen *indexAction*-metodi. Metodi saa parametrinaan *id*:n, joka vastaa pyydetyn *WorkTask* entityn *id*-muuttujaa. Metodi pyytää Doctrinen EntityManageria etsimään kyseistä *id*:tä vastaavaa entity-objektia tietokannasta. Löydetty entity-objekti annetaan eteenpäin renderöitävän näkymän, eli Twig-templaatin saataville. Jos entity-objektia ei löydy tietokannasta, tulostetaan virhenäkymä.

```
public function indexAction($id) {
    $repository = $this->getDoctrine()
            ->getRepository('AltimaPunavaaraBundle:WorkTask');
    $workTask = $repository->find($id);
    if (!$workTask) {
        throw $this->createNotFoundException('Työtä ei löytynyt.');
    }

    return $this->render('AltimaPunavaaraBundle:WorkTask:workTask.html.twig',
        array(
            'worktask' => $workTask
        ));
}
```

KUVA 17. WorkTaskController:n indexAction-metodi

Jotta käsittelijöitä voidaan kutsua, on määritettävä reititysasetukset. Bundlen käyttämät reitit määritetään oletuksena bundle-kansion alla sijaitsevaan *Resources/config/routing.yml*-tiedostoon yaml-formaatin mukaisesti. Lisäksi tämä reititystiedosto tulee määrittää käytettäväksi sovelluksen ylemmän tason reititysasetuksiin *app/config/routing.yml*-tiedostoon. Yksittäinen reititysasetus sisältää kolme olennaista osaa; nimi, path, eli reittiä vastaava url-osoite sekä käsittelijä ja toiminto, johon reitti ohjataan. Nimeä käytetään muun muassa linkkien luontiin näkymissä sekä ohjelmallisissa uudelleenohjauksissa. Path vastaa url-osoitetta, jota reititysasetuksen halutaan koskettavan ja sen osia voidaan poimia parametreina aaltosulkujen (*{ }*) avulla. Nämä parametrit välitetään automaattisesti samalla nimellä reitissä asetetulle käsittelijälle. Haluttu käsittelijä määritellään *defaults*-avaimella, johon määritetään kaksoispisteellä (*:*) eroteltuna järjestyksessä bundle, käsittelijä sekä metodin nimi. (Routing 2015.) Kuvassa 18 on nähtävissä muutama esimerkki Altiman reitityksistä. Esimerkkinä osoite *http://www.domain.com/worktask/1* ohjautuu edellä luotuun *WorkTaskController*:n *indexAction*-metodiin, johon välitetään *id*-parametriksi numero 1. Huomattavaa on, että metodin nimen lopusta poistetaan sana *Action*.

```
#src/Altima/PunavaaraBundle/Resources/config/routing.yml
worktask:
    path:     /worktask/{id}
    defaults: { _controller: AltimaPunavaaraBundle:WorkTask:index }

worktasks:
    path:     /worktasks
    defaults: { _controller: AltimaPunavaaraBundle:WorkTask:list }

worktask_location:
    path:     /worktask/add/
    defaults: { _controller: AltimaPunavaaraBundle:WorkTask:locationList }

#app/config/routing.yml
altima_punavaara:
    resource: "@AltimaPunavaaraBundle/Resources/config/routing.yml"
    prefix:   /
```

KUVA 18. Esimerkki reitityksistä

Edellä määritettiin siis *WorkTaskController*:n *indexAction* palauttamaan mallin tiedoilla renderöidyn Twig-templaatin käyttäjälle. Twig-templaatti on HTML-tiedosto, jonka sisälle voidaan tulostaa sisältöä Twig-tagien avulla. Templaatit suositellaan sijoitettavaksi bundlen alle *Resources/views*-kansioon entity-luokkien mukaan nimettyihin kansiorakenteisiin. Altima sisältää kaikille sivuille yhteisen sivupohjan, joka on

määritetty *app/Resources/views/base.html.twig*-tiedostoon (liite 1). Kyseiseen templaattiin sijoitetaan ulkoasun yhteiset osat kuten sivu- ja yläpalkit, sekä linkitetään yhteiset CSS- ja JavaScript-tiedostot. Lisäksi pohja jaetaan useisiin osiin käyttäen *block*-elementtejä, joiden sisältö määritellään *{% block nimi %}...{% endblock %}*-tageilla. Yhteinen sivupohja periytetään kaikille sovelluksen sivuille *{% extends '::base.html.twig' %}*-tagilla. Templaatissa määritetyt block-elementit ovat nyt yliajettavissa määrittelemällä samanniminen elementti sivupohjaa laajentavalla sivulla. Käsittelijästä templaatille tuotujen PHP-objektien tietoja voidaan tulostaa *{{ ... }}*-tagilla. Tagin sisään kirjoitetaan objektin nimi sekä tulostettavan muuttujan nimi muodossa *objekti.muuttuja*. Ketjuttamalla voidaan myös hakea entity-luokan relaatioissa määritettyjen entity-objektien tietoja. Tämä on erittäin kätevää, sillä kaikki liittyvät tiedot ovat aina saatavilla ilman erillisiä tietokantahakuja käsittelijässä.

```
{% extends '::base.html.twig' %}

{% block body %}
    <p>{{ worktask.description }}</p>
    <p>{{ workTask.location.name }}</p>
    <p>{{ workTask.location.manager.name }}</p>
{% endblock %}
```

KUVA 19. worktask.html.twig-templaatti

Monet verkkosovelluksen perustavat toiminnollisuutensa HTML-lomakkeisiin, ja sama pätee myös Altimaan. Useat sovelluksen sivut pitävät sisällään yhden tai useamman HTML-lomakkeen, joiden avulla käyttäjä pystyy suorittamaan sovelluksen toimintoja. Lomakkeet ovat Symfony2:ssa erillinen komponentti, jonka toimintaperiaatteet voivat aluksi vaikuttaa varsin monimutkaisilta, mutta komponentti on erittäin tehokas työkalu oikein käytettynä. Lomake muodostetaan ensin käsittelijässä objektiksi, joka sisältää kaikki lomakkeen tarvitsemat kentät ja tiedot, ja tämän jälkeen se viedään Twig-templaatille renderöitäväksi. Lomake voidaan luoda automaattisesti perustuen entity-luokan tietoihin tai määrittelemällä itse kaikki lomakkeen sisältämät kentät. Lomake-komponentti sisältää myös automaattisen lomakkeen validoinnin perustuen entity-luokassa tai lomakkeen luomisessa määrättyihin ehtoihin.

Liitteessä 2 on nähtävissä *WorkTaskController*:n *addAction*, joka luo lomake-objektin *WorkTask* entity-objektin pohjalta. Lomake-objekti välitetään Twig-templaatille renderöitäväksi, jossa sitä käsitellään määrätyillä *form_*-tageilla (kuva 20). Symfony sisältää valmiina kolme erilaista HTML-rakennetta lomakkeille, ja näitä pohjia voi tarpeen mukaan yliajaa omilla tyyleillä. Mikäli lomake on lähetetty, eli käsittelijän sisältämä *Request*-objekti ei ole tyhjä, lomakkeen sisältämät tiedot tarkistetaan *isValid()*-metodilla. Jos lomakkeen validointi epäonnistui, virheelliset kentät saavat automaattisesti virheviestin Twig-templaatissa. Mikäli validointi onnistui, lomakkeen sisältämän entityn tiedot tallennetaan tietokantaan Doctrinen EntityManagerilla käyttäen *persist()*- ja *flush()*-metodeja, joista edellä mainittu valmistelee uuden tiedon tietokantaan tallentamista varten, kun taas *flush()* suorittaa varsinaisen tietokantakomennon. Tietojen muokkaaminen onnistuu täsmälleen samalla menetelmällä sillä erotuksella, että *persist()*-metodia ei tarvitse kutsua, koska entity-objekti on jo olemassa tietokannassa.

```
<div class="ibox-content">
    {{ form_start(form) }}
        {{ form_row(form.description) }}
        <div class="hr-line-dashed"></div>
        {{ form_row(form.information) }}
        <div class="hr-line-dashed"></div>
        {{ form_row(form.user) }}
        <div class="hr-line-dashed"></div>
        {{ form_label(form.workDate) }}
        <div class="hr-line-dashed"></div>
        {{ form_row(form.duration) }}
        <div class="hr-line-dashed"></div>
        {{ form_widget(form.isCompleted) }}
        <div class="hr-line-dashed"></div>
        {{ form_widget(form.save) }}
    {{ form_end(form) }}
</div>
```

KUVA 20. Lomakkeen renderöiminen Twig-templaatissa

Toistaen edellä käytyjä vaiheita voidaan toteuttaa käytännössä kaikki loput sovelluksen osat. Näin ollen muiden osien kehitys nopeutuu merkittävästi, kun yhden osan sovelluksesta saa valmiiksi. Verrattuna Altiman vanhaan versioon, uusi sovellus koostuu samoista elementeistä, mutta huomattavasti laadukkaammin toteutettuna. Sovelluksen näkymät, eli sivut oli vanhassa versiossa jaettu omiin PHP-tiedostoihin ja kaikille sivuille yhteinen sivupohja oli sijoitettu *index.php*-tiedostoon. Sama tiedosto si-

sälsi hieman reitityksiä muistuttaneen komponentin, joka latasi url-parametreja vastaavan sivu-tiedoston. Mahdollisesti suurin ero syntyy kuitenkin tiedon käsittelyssä, sillä vanhassa versiossa ei ollut ORM-komponenttia käytössä. Yhteydenpito tietokantaan tehtiin aina tapauskohtaisesti suoraan sivu-tiedostoissa kirjoittamalla kaikki SQL-lauseita myöten itse.

4.3 Tietoturvasta

Tässä vaiheessa sovelluksessa ei ole vielä kiinnitetty huomiota käyttöoikeuksiin tai käyttäjien hallintaan, sillä kaikille sovelluksen sivuille on tällä hetkellä vapaa pääsy. Symfony2:ssa on sisäänrakennettuna erittäin monipuolinen turvallisuuskomponentti, joka käsittelee käyttäjän kirjautumisen, käyttöoikeuksien hallinnan sekä käyttäjäobjektin hallinnan. Turvallisuuskomponentin asetukset määritetään *app/config/security.yml* tiedostoon, johon määritetään käyttäjäroolit, palomuurit sekä sovelluksen osien käyttämiseen tarvittavat roolit. Lisäksi tiedostoon voidaan määritellä esimerkiksi salasanojen koodaamiseen käytettävä kryptografia, joka on Altiman tapauksessa bcrypt. Palomuurin asetuksiin määritetään kirjautumislomakkeen ja uloskirjauksen polut. Pääsyoikeuksien avulla voidaan rajoittaa käyttäjien pääsyä määrättyihin sovelluksiin osiin käyttäjäroolien mukaisesti. Altimassa kaikki sovelluksen sivut on sallittu ainoastaan sisäänkirjautuneille käyttäjille, pois lukien kirjautumis-sivu, johon täytyy olla mahdollisuus päästä ilman sisäänkirjautumista. Jos kirjautumis-sivun määrittää vaatimaan kirjautumista, syntyy päättymätön uudelleenohjauskierre. Muutamia Altimassa käytettäviä määrityksiä on nähtävissä kuvassa 21.

```
role_hierarchy:
    ROLE_ADMIN:       ROLE_USER
    ROLE_SUPER_ADMIN: [ROLE_USER, ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH]

firewalls:
    default:
        anonymous: ~
        http_basic: ~
        form_login:
            login_path: /login
            check_path: /login_check
            csrf_provider: security.csrf.token_manager
        logout:
            path:   /logout
            target: /
        remember_me:
            key:      "%secret%"
            lifetime: 604800
            path:     /

access_control:
    - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/, roles: ROLE_USER }
```

KUVA 21. Security.yml

Asetusten jälkeen turvallisuuskomponentti hoitaa automaattisesti käyttöoikeuksien tarkistamisen. Tarvittaessa käyttäjä uudelleenohjataan automaattisesti kirjautumissivulle tai vaihtoehtoisesti käyttäjälle näytetään virheilmoitus. Roolitarkastuksia voidaan suorittaa myös esimerkiksi koodissa käsittelijöissä tai Twig-sivupohjissa. Symfony tarjoaa myös monia muita turvallisuutta parantavia ominaisuuksia, kuten eston cross site scripting-hyökkäyksille (XSS) sekä salatun yhteyden käyttämisen. Symfony2-projektissa on mahdollista tarkistaa käytössä olevat Composer-riippuvaisuudet tiedettyjen tietoturva-aukkojen osalta komennolla *php app/console security:check*. Tätä komentoa suositellaan käytettävän säännöllisesti, jotta mahdolliset riskit voidaan tunnistaa ajoissa.

Vanhaan versioon verrattuna parannus tietoturvan suhteen on todella merkittävä. Vanhassa versiossa kirjautuminen tarkistettiin ainoastaan sivuille yhteisessä *index.php*-tiedostossa. Kirjautuminen ja käyttöoikeuksien tarkastus oli toteutettu hyvin yksinkertaisesti pohjautuen PHP-sessioihin, jotka olisi ollut erittäin helppo kaapata ja ohittaa. Symfony2:ssa nämä asiat hoidetaan pääosin evästeisiin perustuen. Tietokannassa salasanat suojattiin md5-salauksella, joka on tyhjää parempi, mutta kuitenkin erittäin helppo murtaa, toisin kuin uudessa versiossa käytettävä bcrypt.

4.4 Käyttöönotto ja ylläpitäminen tuotantopalvelimella

Sovelluksen asentaminen ja käyttöönottaminen tuotantopalvelimella on erittäin tärkeä vaihe, joka kannattaa tehdä huolella. Altiman käyttämä palvelin on varustettu Debian Linuxilla, mutta tehtävät toimenpiteet ovat samoja kaikilla käyttöjärjestelmillä. Palvelimella tulee olla asennettuna samat Symfonyn vaatimat sovellukset ja kirjastot kuin paikallisessa kehitysympäristössä, eli muun muassa PHP vaadittavilla lisäosilla, Composer, MySQL-tietokanta ja niin edelleen. Ensimmäisenä vaiheena luodaan uusi tietokanta sekä MySQL-käyttäjä sovellusta varten samoilla asetuksilla kuin paikallisessa käytössä. Sovelluksen käyttöönotto aloitetaan lataamalla projektin tiedostot, pois lukien *vendor*-kansio palvelimen *www*-kansion alle haluttuun alikansioon käyttäen FTP:tä. Tiedostojen lataamisen jälkeen on suositeltavaa tarkistaa *app/cache*- ja *app/log*-kansioiden käyttöoikeudet. On suositeltavaa käyttää samaa UNIX-käyttäjää

verkkopalvelimelle ja komentoriville, jolloin näiden kahden välillä ei synny ristiriitoja.

Loput asennuksen vaiheet suoritetaan käyttäen komentoriviä esimerkiksi SSH-yhteyden avulla. Kun kaikki tiedostot on ladattu palvelimelle, siirrytään komentorivillä projektin juureen ja suoritetaan komento *php app/check.php*. Komento tarkistaa täyttääkö palvelin sovelluksen vaatimukset ja opastaa korjaamaan mahdolliset ongelmat. Seuraavaksi tehdään määritykset *app/config/parameters.yml*-tiedostoon samaan tapaan kuin paikallisessa kehitysympäristössä. Tämän jälkeen asennetaan sovelluksen käyttämät tuotantoympäristön riippuvaisuudet käyttäen Composer:a. Tämä tapahtuu suorittamalla komento *composer install --no-dev --optimize-autoloader* projektin juuressa. Komento asentaa myös bundlen assetit web-kansion alle käyttäen symbolisia linkkejä. Seuraava vaihe on tyhjentää sovelluksen välimuisti tuotantoympäristöä varten komennolla *php app/console cache:clear --env=prod --no-debug* sekä asennetaan Assetic Assetit komennolla *php app/console assetic:dump --env=prod --no-debug*. Nyt sovellus on käyttövalmis, mutta tietokanta on edelleen tyhjä. Mikäli käyttö halutaan aloittaa tyhjästä tietokannasta, ajetaan vielä komento *php app/console doctrine:schema:update --force*, joka luo tarvittavat taulut tietokantaan. Vaihtoehtoisesti voidaan esimerkiksi tuoda kehitysympäristön tietokannan sisältö tuotantopalvelimelle käyttäen phpMyAdmin-hallintatyökalua.

Näiden asennusvaiheiden jälkeen sovelluksen toiminta voidaan testata siirtymällä selaimella *web/app.php*-tiedostoon. Mikäli sivu antaa HTTP-virheilmoituksen 500, johtuu tämä luultavimmin *cache*- ja *logs*-kansioiden virheellisistä käyttöoikeuksista. Kun sovellus toimii ongelmitta, on jäljellä enää Apachen VirtualHost:n määrittäminen, jolla ohjataan haluttu domain-osoite sovelluksen *web*-kansioon. Altiman käyttämä VirtualHost-tiedosto on kuvan 22 mukainen. Näiden toimenpiteiden jälkeen on vielä suositeltavaa tarkistaa palvelimelta kansioiden käyttöoikeudet tietoturvallisuuden kannalta sekä siivota tarpeettomat tiedostot esimerkiksi *web*-kansion alta.

```
<VirtualHost *:80>
    ServerName domain.tld
    ServerAlias www.domain.tld

    DocumentRoot /var/www/project/web
    <Directory /var/www/project/web>
        AllowOverride All
        Order Allow,Deny
        Allow from All
    </Directory>

    <Directory /var/www/project>
        Options FollowSymlinks
    </Directory>

    ErrorLog /var/log/apache2/project_error.log
    CustomLog /var/log/apache2/project_access.log combined
</VirtualHost>
```

KUVA 22. VirtualHost määritykset

Sovelluksen päivittäminen tuotantopalvelimella on verrattain yksinkertainen prosessi. Tarvittavat koodimuutokset tehdään ja testataan kehitysympäristössä, jonka jälkeen muokatut tiedostot siirretään FTP:llä palvelimelle korvaten vanhat tiedostot. Tämän jälkeen ajetaan komentoriviltä aiemmin mainittu välimuistin tyhjennyskomento, sekä tarvittaessa luodaan uudet symboliset linkit bundlen asseteille ja päivitetään tietokanta Doctrinella. Riippuvaisuuksien päivittäminen hoidetaan Composer:n omilla komennoilla. Tässä mielessä Symfony2 tuo valtavan parannuksen verrattuna Altiman vanhaan versioon. Sovelluksen tiedostot ovat selkeästi jaoteltu erillisiin osiin. Symfony2:ssa päivitysprosessi on hyvin hallittu, ja kestää parhaimmassa tapauksessa ainoastaan muutamia sekunteja.

5 PÄÄTÄNTÖ

Altiman uuden version kehityksessä perimmäisenä tarkoituksena oli päivitettävyyden ja jatkokehityksen helpottaminen. Symfony2-sovelluskehys osoittautui mielestäni oikeaksi valinnaksi näiden ongelmien ratkaisemiseen. Sovelluskehys sisältää erittäin tehokkaat ja helppokäyttöiset työkalut sovelluksen kehittämiseen. MVC-arkkitehtuurin myötä sovelluksen eri osat ovat selkeästi jaoteltu helposti hallittaviin osiin, ja projektin päivittäminen on jatkossa huomattavasti helpompaa kuin vanhassa versiossa. Symfony2 tarjoaa myös useita tietoturvaa parantavia ominaisuuksia ja vanhaan versioon verrattuna parannus asian suhteen onkin valtaisa. Oikeastaan ainut ne-

gatiivinen seikka Symfony2:n käytössä verrattuna vanhaan versioon on aavistuksen kasvanut sivun latausaika. Kyse on kuitenkin millisekunneista, eikä Altiman käyttäjämäärillä asialla ole käytännössä mitään merkitystä.

Altiman kehityksestä kertyneen kokemuksen perusteella voin todeta, että Symfony2:n oppimiskäyrä oli ainakin omasta mielestäni varsin jyrkkä. Ensimmäinen viikko kehityksestä kului pelkästään projektin ja kehitysympäristön asennuksessa syntyneiden lukemattomien ongelmien ratkaisemiseen. Asiaa ei myöskään helpottanut se, että tein kehitystyötä kahdessa eri ympäristössä Windows- ja OS X-käyttöjärjestelmillä, jotka molemmat vaativat hieman erilaisia toimenpiteitä. Yhden suuremman sovelluksen osan toteuttamisen jälkeen kehitystyö alkoi kuitenkin helpottua ja nopeutua. Onneksi Symfony2:n dokumentaatio on erittäin kattava, joten ongelmien ratkaisu kysyy lähinnä tahdonvoimaa kehittäjältä. Apuna toimii myös erittäin aktiivinen käyttäjäkunta esimerkiksi Stack Overflow-sivustolla. En kuitenkaan suosittele sovelluskehyksen käyttämistä aloitteleville kehittäjille, sillä mielestäni on välttämätöntä ymmärtää PHP-sovelluskehityksen periaatteet ennen sovelluskehykseen siirtymistä. Symfony2 on erittäin laaja sovelluskehys, jonka kaikkien ominaisuuksien ja yleisten käytäntöjen opiskelemiseen voi kuluttaa aikaa lähes loputtomasti.

Altiman kehitys tulee jatkumaan aktiivisena myös tästä eteenpäin. Sovellukseen on suunnitteilla useita uusia ominaisuuksia, kuten kieliversioiden lisääminen. Altimaa on mahdollisesti tulevaisuudessa tarkoitus myydä uusille asiakkaille, jolloin bundle-järjestelmä helpottaa eri asiakkaille räätälöityjen sovellusversioiden hallintaa. Vanhalla versiolla tämä olisi ollut lähestulkoon mahdotonta, joten sovelluksen uudelleenkirjoitus oli ainut ja oikea ratkaisu tulevaisuutta silmällä pitäen.

LÄHTEET

Beaumont, Matthew 2014. 7 Reasons to Choose the Yii 2 Framework. WWW-dokumentti. http://www.sitepoint.com/7-reasons-choose-yii-2-framework/. Päivitetty 13.10.2014. Luettu 30.8.2015.

Creating and Using Templates 2015. Symfony. WWW-dokumentti. http://symfony.com/doc/current/book/templating.html. Ei päivitystietoa. Luettu 25.9.2015.

Databases and Doctrine 2015. Symfony. WWW-dokumentti. http://symfony.com/doc/current/book/doctrine.html. Ei päivitystietoa. Luettu 26.9.2015.

Haikala, Ilkka & Mikkonen, Tommi 2011. Ohjelmistotuotannon käytännöt. Helsinki: Talentum.

Holzner, Steven 2008. PHP: The Complete Reference. New York: McGraw Hill Professional.

Hopkins, Callum 2013. The MVC Pattern and PHP. WWW-dokumentti. http://www.sitepoint.com/the-mvc-pattern-and-php-1/. Päivitetty 4.3.2013. Luettu 23.8.2015.

Ide, Andy 2013. PHP just grows & grows. WWW-dokumentti. http://news.netcraft.com/archives/2013/01/31/php-just-grows-grows.html. Päivitetty 31.1.2015. Luettu 15.9.2015.

Installing and Configuring Symfony 2015. Symfony. WWW-dokumentti. http://symfony.com/doc/current/book/installation.html. Ei päivitystietoa. Luettu 6.10.2015.

Jacobs, Sem 2014. When & Why to Use PHP Framework? Blogi. http://blog.templatemonster.com/2014/04/28/php-frameworks-when-and-why-to-use-them. Päivitetty 28.4.2014. Luettu 29.9.2015.

Kasurinen, Jussi Pekka 2011. Ruby on Rails ohjelmointi. Jyväskylä: Docendo.

Moon, Brian 2011. PHP Frameworks. Blogi. https://brian.moonspot.net/php-frameworks. Päivitetty 25.4.2011. Luettu 28.8.2015.

Peterson, David 2008. Rasmus Lerdorf: PHP Frameworks? Think Again. Blogi. http://www.sitepoint.com/rasmus-lerdorf-php-frameworks-think-again/. Päivitetty 29.8.2008. Luettu 30.8.2015.

Petricek, Tomas 2015. Library patterns: Why frameworks are evil. Blogi. http://tomasp.net/blog/2015/library-frameworks/. Päivitetty 3.3.2015. Luettu 15.9.2015.

Porebski, Bartosz & Przystalski, Karol & Nowak, Leszek 2011. Building PHP Applications with Symfony, CakePHP, and Zend Framework. Birmingham: Wrox.

Potencier, Fabien 2013. Don't use PHP libraries with known security issues. WWW-dokumentti. http://fabien.potencier.org/don-t-use-php-libraries-with-known-security-issues.html. Päivitetty 19.2.2013. Luettu 19.9.2015.

Reenskaug, Trygve 2010. MVC XEROX PARC 1978–79. WWW-dokumentti. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html. Päivitetty 12.6.2010. Luettu 24.8.2015.

Ren, Taylor 2015. Nette Framework: First Impressions. Blogi. http://www.sitepoint.com/nette-framework-first-impressions/. Päivitetty 22.6.2015. Luettu 30.8.2015.

Routing 2015. Symfony. WWW-dokumentti. https://symfony.com/doc/current/book/routing.html. Ei päivitystietoa. Luettu 7.10.2015.

Shafik, Davey 2015. What to Expect When You're Expecting: PHP 7. Blogi. https://blog.engineyard.com/2015/what-to-expect-php-7. Päivitetty 2.5.2015. Luettu 30.8.2015.

Skvorc, Bruno 2015. Best PHP Framework for 2015 – SitePoint Survey Results. Blogi. http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/. Päivitetty 28.3.2015. Luettu 24.8.2015.

Stones, Ben 2015. What is CodeIgniter, and how does it work? Blogi. http://www.eukhost.com/blog/webhosting/what-is-codeigniter-and-how-does-it-work/. Päivitetty 2.2.2015. Luettu 30.8.2015.

The Architecture 2015. Symfony. WWW-dokumentti. http://symfony.com/doc/current/quick_tour/the_architecture.html. Ei päivitystietoa. Luettu 15.9.2015.

The Cookbook. 2015. Symfony. PDF-dokumentti. https://symfony.com/pdf/Symfony_cookbook_2.7.pdf?v=4. Ei päivitystietoa. Luettu 15.9.2015.

Ullman, Larry 2009. Understanding MVC: The Basics. Blogi. http://www.larryullman.com/2009/10/08/understanding-mvc/. Päivitetty 8.10.2009. Luettu 30.8.2015.

Way, Jeffrey 2012. Why Laravel is Taking the PHP Community by Storm. WWW-dokumentti. http://code.tutsplus.com/tutorials/why-laravel-is-taking-the-php-community-by-storm--pre-52639. Päivitetty 28.11.2012. Luettu 30.8.2015.

Wayner, Peter 2015. 7 reasons why frameworks are the new programming languages. WWW-dokumentti. http://www.infoworld.com/article/2902242/application-development/7-reasons-why-frameworks-are-the-new-programming-languages.html. Päivitetty 30.3.2015. Luettu 26.8.2015.

Why should I use a framework? 2015. Symfony. WWW-dokumentti. https://symfony.com/why-use-a-framework. Ei päivitystietoa. Luettu 10.9.2015.

LIITE 1.

Twig-templaatti base.html.twig

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
        <title>Altima | Punavaara Oy | {% block title %}{% endblock %}</title>
        {% block stylesheets %}{% endblock %}
        <link rel="icon" type="image/x-icon" href="{{ asset('favicon.ico') }}" />
        <!-- Bootstrap 3.3.4 -->
        <link href="{{ asset('css/bootstrap.min.css') }}" rel="stylesheet" type="text/css" />
        <!-- Font Awesome Icons -->
        <link href="{{ asset('css/font-awesome.css') }}" rel="stylesheet" type="text/css" />
        {% block pagestyles %}{% endblock %}
        {% stylesheets 'bundles/altimapunavaara/sass/style.sass' output='css/*.css' filter='sass, cssrewrite' %}
            <link href="{{ asset_url }}" rel="stylesheet" type="text/css" />
        {% endstylesheets %}
    </head>
    <body>
        <div id="wrapper">
            <!-- Main Navigation -->
            {{ render(controller('AltimaPunavaaraBundle:Layout:navigation',
                    { 'route': app.request.attributes.get('_route') }
            )) }}
            <div id="page-wrapper" class="gray-bg">
                <div class="row border-bottom">
                    <nav class="navbar navbar-static-top white-bg" role="navigation" style="margin-bottom: 0">
                        <div class="navbar-header">
                            <a class="navbar-minimalize minimalize-styl-2 btn btn-primary" href="#">
                                <i class="fa fa-bars"></i>
                            </a>
                        </div>
                        <ul class="nav navbar-top-links navbar-right">
                            {{ render(controller('AltimaPunavaaraBundle:Layout:messages',
                                { 'max': 3 }
                            )) }}
                            <li>
                                <a href="{{ path('logout') }}">
                                    Kirjaudu ulos <i class="fa fa-sign-out"></i>
                                </a>
                            </li>
                        </ul>
                    </nav>
                </div>
                <div class="wrapper wrapper-content animated fadeInRight">
                    {% block body %}{% endblock %}
                </div>
            </div><!-- #/page-wrapper -->
        </div><!-- #/wrapper -->
        <!-- jQuery 2.1.4 -->
        <script src="{{ asset('js/jquery-2.1.4.min.js') }}"></script>
        <!-- jQuery UI custom -->
        <script src="{{ asset('js/jquery-ui.custom.min.js') }}"></script>
        <!-- Bootstrap 3.3.4 JS -->
        <script src="{{ asset('js/bootstrap.min.js') }}"></script>
        <script src="{{ asset('js/plugins/metisMenu/jquery.metisMenu.js') }}"></script>
        <script src="{{ asset('js/plugins/slimscroll/jquery.slimscroll.min.js') }}"></script>
        <!-- Custom and plugin javascript -->
        <script src="{{ asset('bundles/altimapunavaara/js/inspinia.js') }}"></script>
        <script src="{{ asset('js/plugins/pace/pace.min.js') }}"></script>
        <!-- Toastr script -->
        <script src="{{ asset('js/plugins/toastr/toastr.min.js') }}"></script>
        <!-- SweetAlert -->
        <script src="{{ asset('js/plugins/sweetalert/sweetalert.min.js') }}"></script>
        <!-- Main script -->
        <script src="{{ asset('bundles/altimapunavaara/js/main.js') }}"></script>
        {% block javascripts %}{% endblock %}
    </body>
</html>
```

Lomakkeen rakentaminen ja käsitteleminen

```
public function addAction(Request $request, $locationId) {
    $workTask = new WorkTask();
    $repository = $this->getDoctrine()
            ->getRepository('AltimaPunavaaraBundle:Location');
    $location = $repository->find($locationId);
    $userRepo = $this->getDoctrine()
            ->getRepository('AltimaPunavaaraBundle:User');

    $form = $this->createFormBuilder($workTask)
        ->add('description', 'textarea', array(
            'label'  => 'Kuvaus',
            'attr' => array('class' => 'textarea-bulletin')
        ))
        ->add('information', 'textarea', array(
            'label'  => 'Lisätiedot',
            'attr' => array('class' => 'textarea-bulletin'),
            'required' => false
        ))
        ->add('workDate', 'date', array(
            'label'  => 'Päivämäärä',
            'widget' => 'single_text',
            'html5' => false,
            'required' => false,
            'format' => 'dd.MM.yyyy',
            'data' => new \DateTime('now')
        ))
        ->add('user', 'entity', array(
            'label'  => 'Työntekijä',
            'class' => 'AltimaPunavaaraBundle:User',
            'choice_label' => 'fullName',
            'choices' => $userRepo->getActiveUsers(),
            'data' => $this->getUser()
        ))
        ->add('duration', 'number', array(
            'label' => 'Kesto'
        ))
        ->add('isCompleted', 'checkbox', array(
            'label'    => 'Valmis',
            'required' => false,
            'data' => true
        ))
        ->add('save', 'submit', array(
            'attr' => array('class' => 'btn btn-primary btn-save'),
            'label' => 'Tallenna'
        ))
        ->getForm();

    $workTask->setLocation($location);
    $workTask->setCreatedAt(new \DateTime('now'));

    $form->handleRequest($request);
    if ($form->isValid()) {
        $em = $this->getDoctrine()->getManager();
        $em->persist($workTask);
        $em->flush();
        return $this->redirectToRoute('worktask_location');
    }

    return $this->render('AltimaPunavaaraBundle:WorkTask:add.html.twig',
            array(
                'form' => $form->createView(),
                'location' => $location
            ));
}
```

LIITE 3

Kuvankaappauksia uudesta ulkoasusta