LAHDEN AMMATTIKORKEAKOULU
Lahti University of Applied Sciences

WWW-PALVELUN RAKENTAMINEN

Private Pilot -palvelun realisointi Symfony-kehysohjelmistolla

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2016
Antti Aspinen

Lahden ammattikorkeakoulu
Tietotekniikan koulutusohjelma

ASPINEN, ANTTI: WWW-palvelun rakentaminen
Private Pilot -palvelun realisointi Symfony-kehysohjelmistolla

Ohjelmistotekniikan opinnäytetyö, 81 sivua, 6 liitesivua

Kevät 2016

TIIVISTELMÄ

Opinnäytetyössä käydään läpi yleisellä tasolla, millaista on rakentaa WWW-palvelu käyttäen Symfony 2 -ohjelmistokehystä. Työ pääasiassa keskittyy yleisimpiin vaiheisiin ja kokonaiskuvaan. Työssä perehdytään joihinkin merkittävimpiin Symfony-komponentteihin, joita tarvittiin Private Pilot -palvelun demo-version rakentamiseksi.

Projekti oli osa Lahti Demo Room -pilottihanketta, jossa opiskelijat rakensivat demo-versioita yritysten käyttöön. Private Pilot -palvelun rakensivat Lahden Ammattikorkeakoulun opiskelijat mainostoimisto Ilmeelle.

Private Pilot WWW-palvelu tarjoaa yksityislentäjille mahdollisuuden ilmoittaa omia lentoreittejään verkossa. Asiakkaat voivat hakea näitä reittejä sivustolla ja maksua vastaan matkustaa lentäjien kyydissä pienlentokoneilla. Palvelu on tarkoitettu Yhdysvaltojen markkinoille.

Opinnäytetyön näkökulmasta haasteellista oli rakentaa nopea, tietoturvallinen ja luotettava palvelu, joka toimii useilla eri aikavyöhykkeillä. Myös käyttömukavuus tuli huomioida, ja tähän liittyen esimerkiksi lentojen tilaustoiminnon tuli toimia ilman todennusvaatimuksia sivuston käyttäjälle.

Palvelun käyttämässä WWW-sovelluksessa on käytetty Symfonyn lisäksi komponentteja, joita ei toimiteta Symfony-kehysohjelmiston mukana. Mainitsemisen arvoinen on Propel ORM -tietokantakomponentti. Se auttoi WWW-palvelun tietokantasuunnittelussa, nopeassa prototyypittämisessä ja toteutuksessa. Tämän takia Propel ORM -välitasonohjelmistosta on kokonaan oma lukunsa opinnäytetyössä.

Työssä oli useita haasteita, mutta lopputuloksena syntyi projektin alkuperäiset tavoitteet täyttävä toimiva demo-versio. Opinnäytetyön lukijalle on eduksi tietää PHP-ohjelmoinnin perusteita. Lukijalle on myös hyödyllistä tuntea yleisellä tasolla joitain ohjelmistokehityksessä käytettyjä menetelmiä.

Asiasanat: Symfony, Twig, Propel, PHP, ORM, ohjelmistoarkkitehtuuri, SQL, Web-sovelluskehys

Lahti University of Applied Sciences
Degree Programme in Information Technology

ASPINEN, ANTTI: Building of a WWW service
Building a Private Pilot Service with the Symfony Framework

Bachelor's Thesis in Software Engineering, 81 pages, 6 pages of appendices

Spring 2016

ABSTRACT

This thesis goes through designing and implementing a WWW-based service using the Symfony software framework. This work does not go deep into the topic. Instead it presents itself in a general way. The work explains some of the most common software components which are required to build a moderately complex web service.

The resulting Private Pilot service is meant to provide private pilots in the USA with a method to publicly announce their flight routes and let the general public search for those flights. People can use the service to book a flight from small airports and fly to other small airports where big flight companies do not operate.

Private Pilot also uses other software components along with the Symfony 2 framework. One such component is Propel ORM, which allowed quick prototyping and implementation of the database related functionality.

From the practical point of view, the most challenging part of the work was to provide a quick, reliable, scalable service that can operate simultaneously within several time zones. The service also had to work without users authenticating to the system.

Keywords: Symfony, Twig, Propel, PHP, ORM, Software Architecture, SQL, Web Application Framework

LYHENTEET JA KÄSITTEET

API Application Programming Interface eli ohjelmointirajapinta on joukko määriteltyjä sääntöjä ja työkaluja ohjelmistosovellusten väliseen tietojen vaihtoon.

Apache HTTPD

Apache Foundationin avoimen ja vapaan lähdekoodin WWW-palvelinohjelmisto. Se löytyy tavallisesti asennettuna GNU/Linux-pohjaisista käyttöjärjestelmistä. Apache on maailman eniten käytetyin WWW-palvelinohjelmisto (Netcraft 2013).

Big Data Isot tietovarannot on sateenvarjokäsite, jolla usein tarkoitetaan hyvin suurien, usein järjestelemättömien, tietovarantojen käsittelyä. Big Dataan kuuluu myös näiden tietovarantojen säilyttäminen, analysointi ja järjestäminen.

CGI Common Gateway Interface on tärkeä WWW-standardi, jolla selain välittää tietoa palvelimella sijaitsevalle ohjelmalle käsiteltäväksi. CGI on riippumaton käytetystä ohjelmointikielestä.

CMS Content Management System eli sisällönhallintajärjestelmä on yleistermi sovelluksille, joita käytetään minkä tahansa sisällön tuottamiseen, järjestelemiseen, julkaisemiseen ja ylläpitoon keskitetyn käyttöliittymän avulla. Yleensä kyseessä on jokin blogi yms. sovellus, joka hallitsee multimediasisältöä verkossa.

CRUD Termi, joka kuvaa neljää tiedonkäsittelyyn liittyvää toimenpidettä. Nämä toimenpiteet ovat Luo (engl. Create), Lue (engl. Read), Päivitä (engl. Update) ja Poista (engl. Delete). CRUD on yleinen tietokantoihin liittyvä käsite.

CSRF Cross-site Request Forgery on tietoturvaongelma, jossa hyökkääjä pakottaa loppukäyttäjän suorittamaan tahtomattaan komentoja WWW-palvelussa, johon käyttäjä on kirjautuneena sisään. Eli

hyökkääjä niin sanotusti ratsastaa tai surffaa käyttäjän istunnon päällä.

DAL Data Access Layer on sovelluskerros, joka tarjoaa sovelluksessa pääsyn persistoituun data-varantoon kuten esimerkiksi tietokantaan. Usein ORM-työkalu tarjoaa tämän välitason sovellukselle.

Database Abstraction Layer

Tietokanta-abstraktiokerros on ohjelmointirajapinta, joka vakinaistaa tavan, jolla tietokonesovellus ja tietokannat keskustelevat keskenään. Tämän rajapinnan toteuttavat sovellukset ovat tietokanta-agnostisia, eli riippumattomia käytetystä tietokantaohjelmistosta ja SQL-tietokantakielen versiosta.

Garbage Collector

Roskien kerääjä (tietojenkäsittelytieteen merkityksessä) ottaa talteen olio-pohjaisissa ohjelmointikielissä muistiin kertyneet viittaukset, joihin sovellus ei enää osoita. Tämän jälkeen se tyhjentää ne keskusmuistista, mikä vapauttaa tilaa keskusmuistista muille ohjelmille.

HTML Hypertext Markup Language on merkintäkieli, jolla WWW-sivuja kirjoitetaan. Se on johdettu XML-merkintäkielestä.

IIS Internet Information Services eli IIS on Microsoftin WWW-palvelin ohjelmisto. Se tavallisesti tulee Windows-käyttöjärjestelmien mukana, mutta ei oletusarvoisesti ole käytössä.

MVC Model-View-Controller eli malli-näkymä-ohjain on ohjelmistoarkkitehtuurimalli, jossa sovellus on ajettu kolmeen eri osaan. Nämä osat ovat tietoja hakeva ja tallentava Model-osa, tietoa käyttäjälle esittävä View-osa sekä tietoja käsittelevä Controller-osa.

Nimiavaruus

Nimiavaruus on yhden tyyppinen tapa kapseloida ja rajata tietoa tietojenkäsittelytieteessä. Sovelluksessa voi esimerkiksi olla useampia

saman nimen omaavia funktioita, mutta vain yksi saman nimiavaruuden (engl. namespace) sisällä.

ORM Object-relational mapping on menetelmä, jolla dataa pyritään sovittelemaan yhteen keskenään yhteensopimattomien olio- ja relaatio-mallien välillä. ORM:lla tavallisesti mallinnetaan ja abstraktoidaan tietoa olio-pohjaisella ohjelmointikielellä kirjoitetun sovelluksen ja relaatio-pohjaisen tietokannan välillä esimerkiksi WWW-palveluiden taustajärjestelmissä.

PDO PHP Data Objects eli PDO on Database Abstraction Layer -toteutus PHP-ohjelmointikielessä.

Persistenssi Tietojen persistenssi tarkoittaa tietojen säilyvyyttä vielä senkin jälkeen kun tiedot tuottanut prosessi on lopettanut toimintansa.

PHP PHP Hypertext Preprocessor on olio-pohjainen ohjelmointi- ja skriptauskieli, joka alun perin kirjoitettiin tietojen hakemiseen, käsittelyyn ja esittämiseen suoraan HTML-sivuilla. Nykyään PHP on täysimittainen ohjelmointikieli. Se on löyhästi tyypitetty ja siitä löytyy muun muassa Garbage Collector -toiminto.

PSR PHP Standard Recommendation on joukko Framework Interoperability Groupin kehittämiä suosituksia siitä miten PHP-sovelluksia tulisi kehittää ja millä tyylillä niitä tulisi ohjelmoida.

SOAP Simple Object Access Protocol on tietoliikenneprotokolla proseduurien etäkutsuja eli RPC:tä varten.

SQL Structured Query Language on tietokantojen käyttöön liittyvä kieli, jolla tietokannoista haetaan ja tallennetaan tietoa. Sen muoto ja säännöt usein vaihtelevat tietokantaohjelmistosta riippuen, mutta monet noudattavat SQL-standardia ainakin osittain.

YAML Yet Another Markup Language on yksinkertainen merkintäkieli, joka luotiin tarjoamaan vaihtoehto monimutkaisemman XML-merkintäkielen rinnalle.

SISÄLLYS

1 JOHDANTO

Private Pilot -projektin tarkoituksena oli luoda WWW-palvelu, joka tulee tarjoamaan Yhdysvalloissa yksityisille lentäjille mahdollisuuden ilmoittaa lentoreittejään verkossa. Yksityiset lentäjät voivat myydä paikkoja pienlentokoneistaan reiteillä, joita pitkin he lentävät. Tarkoituksena on yhdistää pienlentokoneen kyytiä haluavat ihmiset pienlentokoneita lentävien yksityislentäjien kanssa.

Palvelun keskeinen idea perustuu siihen, että pienlentokoneiden lentäjien on aktiivisesti lennettävä vuodessa tietty määrä lentoja, jotta heidän lentolupansa pysyisi voimassa. Yhdysvalloissa yksityisiä lentokenttiä on huomattava määrä. Niitä käyttäviä aktiivisesti lentäviä yksityisiä pienlentokoneiden omistajia on arviolta noin miljoona. Olettaen, että he lentävät vähintään kerran kuukaudessa, mahdolliset markkinat Private Pilot -palvelulle koostuisivat vuosittain noin 24 miljoonasta mahdollisesta lennosta yhteen suuntaan palvelun ideoijien mukaan. (Kariste 2013.)

Opinnäytetyössä käydään läpi Private Pilot WWW-palvelun rakentamiseen liittyviä vaiheita Symfony 2 -kehysohjelmiston ja Propel ORM -välitasonohjelmiston näkökulmasta. Työssä käsitellään esimerkiksi MVC-ohjelmistoarkkitehtuurin toimintaa käytännön hankkeessa, jossa tavoitteena oli rakentaa palvelun konseptin pohjalta toimiva demo-versio. Demo-version piti olla sellaisessa toiminnallisessa tasossa, että sitä voitaisiin käyttää esittelemään ja markkinoimaan palvelun liikeideaa mahdollisille rahoittajille.

Symfony on laajalti tunnettu ja paljon käytössä oleva PHP-kehysohjelmisto. Se tarjoaa joustavan ohjelmistorungon, jonka varaan voidaan kehittää monia erityyppisiä sovelluksia. Tätä opinnäytetyötä varten se oli mielenkiintoinen alusta yleisien WWW-sovellusten rakentamiseen liittyvien vaiheiden tutkimiseen.

Tämän opinnäytetyön lukijalta odotetaan vähintään perustason tuntemusta PHP-ohjelmointikielestä ja olio-ohjelmointiin liittyvistä ohjelmointiparadigmoista. Myös ohjelmistotuotannosta olisi hyvä tietää perusteita.

2 OHJELMISTOKEHYS

Ohjelmistokehys on sovellusrunko, joka koostuu ohjelmistokomponenteista, luokista ja usein useammasta kuin yhdestä rajapinnasta, johon ohjelmoija voi halutessaan kytkeä sovelluksensa. Se ei itsessään ole sovellus, vaan hyvin pelkistetty abstrakti rautalankamalli sovelluksille tai sovellusosille. Sitä täydentämällä ja kehittämällä voidaan rakentaa kokonaan uusia sovelluksia. (Koskimies & Mikkonen 2005a, 187–189.)

Tavallisesti ohjelmistokehyksillä pyritään sovelluksen ohjelmakoodin rakenteelliseen standardointiin sekä luettavuuden ja tietoturvan parantamiseen. Vakinaistaminen ei rajoitu ainoastaan ohjelmiston ohjelmakoodiin, vaan ulottautuu aina sovelluksen arkkitehtuuriin eli yleiseen rakenteeseen asti. Hyvä ohjelmistokehys ei ole riippuvainen mistään erityisestä ohjelmointiparadigmasta. Sellaisen tulisi tukea useita eri tapoja lähestyä sovelluksen kehittämistä. (Gamma, Helm, Johnson, Vlissides & Booch 1995a, 26–28; Koskimies & Mikkonen 2005a, 187–189.)

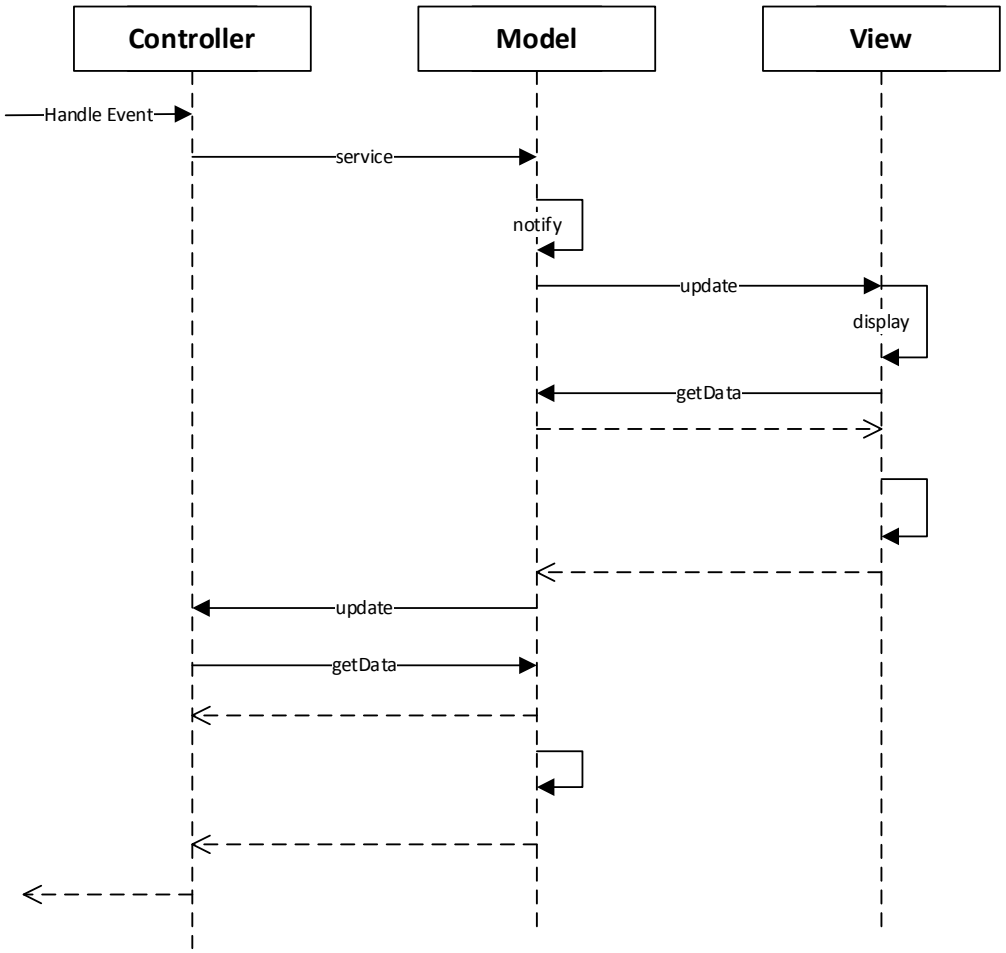
Kehittäjät, jotka työskentelevät ohjelmistokehysten parissa, pyrkivät tavallisesti tarjoamaan valmiita työkaluja ja puitteet sovelluksien rakentamiseen useilla tavoilla. Tämä koskee erityisesti sellaisia ominaisuuksia, jotka vaativat keskenään samantapaisten toimintojen toteuttamista usein.

On olemassa useita ohjelmistokehyksiä, ja ne voivat olla käyttötarkoituksesta riippuen hyvinkin erilaisia keskenään. Tunnettuja ja laajalti käytössä olevia kehyksiä ovat esimerkiksi Microsoftin tuottama pääasiassa Windows-ohjelmistokehitykseen tarkoitettu .NET-ohjelmistokehys sekä PHP-kehitykseen tehty Zend Framework.

Ensimmäinen laajalti levinnyt ohjelmistokehys oli SmallTalk-80-ympäristön mukana kulkenut Model-View-Controller-kehys. Sen mukaan on nimetty MVC-arkkitehtuurisuunnittelumalli. MVC-ohjelmistokehyksen tiettävästi kehitti Trygve Reenskaug 1970-luvulla työskennellessään Xerox PARC -tutkimuskeskuksessa Smalltalk-76-ympäristössä. (Reenskaug 2012; Reenskaug, Wold & Lehne 1996, 333.)

2.1 Malli-näkymä-ohjain

Malli-näkymä-ohjain (engl. MVC) on ohjelmistoarkkitehtuurimalli, jossa sovellus on jaettu kolmeen eri osaan. Kuvio 1 havainnollistaa näitä osia, jotka ovat tietoja hakeva ja tallentava Model-osa (suom. malli), tietoa käyttäjälle esittävä View-osa (suom. näkymä) sekä tietoja käsittelevä Controller-osa (suom. ohjain).

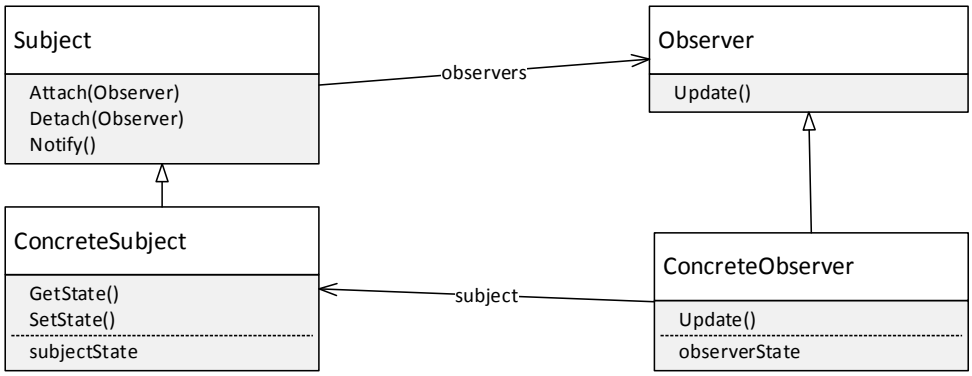


KUVIO 1. MVC:n toiminta (Koskimies & Mikkonen 2005c, 143)

MVC-malli on suosittu muun muassa dynaamisissa WWW- ja GUI-sovelluksissa. Se on eräs yleisimmistä arkkitehtuurimalleista ohjelmistoteollisuudessa. Siitä on olemassa myös muutamia eri variantteja, kuten Push- ja Pull-mallit (Rutenberg 2008).

MVC käyttää hyödykseen kuviossa 2 nähtävää Tarkkailija-suunnittelumallia (engl. Observer Design Pattern), jossa MVC:n View-osa (ja toisinaan myös

Controller-osa) rekisteröidään seuraamaan Model-osassa tapahtuvia muutoksia. Tarkkailija-suunnittelumallissa Observer-olio kuuntelee Subject-olion tilaa. Sitä tarvitaan kun halutaan useiden objektien vastaanottavan päivityksen, kun jonkin toisen objektin tila muuttuu. (Gamma, Helm, Johnson, Vlissides & Booch 1995b, 294–295; Koskimies & Mikkonen 2005d, 142–144.)



KUVIO 2. Tarkkailija-suunnittelumallin rakenne (Gamma, ym. 1995b, 294–295)

Sovelluksen hajauttaminen kolmeen eri osaan mahdollistaa niiden osien kehittämisen toisistaan riippumatta. Tämä johtaa usein nopeampaan kehitykseen, koska sovelluksen osien kehitys voidaan jakaa eri kehittäjien kesken. Se myös voi johtaa tietoturvallisempaan lopputulokseen ja helpottaa sovelluksen jakamista eteenpäin.

Ongelmiksi MVC-mallissa voidaan laskea sovelluksen kokonaisvaltaisen toiminnollisuuden hahmottamiseen liittyvät vaikeudet. Esimerkiksi sovelluksen sisältämien luokkien määrä voi myös kasvaa hyvin suureksi, mikä voi vaikeuttaa vian jäljittämistä sovelluksessa. Lisäksi Tarkkailija-mallin käyttäminen tarkoittaa sitä, että monet tarkkailijat saavat paljon päivityskutsuja, vaikka niiden ei tarvitsisi reagoida muuttuneeseen informaatioon mitenkään. Tämä osaltaan vaikuttaa sovelluksen suorituskykyyn. (Koskimies & Mikkonen 2005b, 144.)

2.2 PHP

PHP on yksi maailman suosituimmista ohjelmointikielistä (Stobart & Vassileiou 2004, 8–9). Siitä riippuvaisia verkkosivuja oli vuonna 2013 arviolta yli 244 miljoonaa, ja arviolta kolmasosa maailman palvelimista tukee PHP:tä (Ide 2013).

Ohjelmointikielenä PHP on helppokäyttöinen ja käytännönläheinen (Tatroe, MacIntyre, Lerdorf & Bourque 2013b, 1–2). Tästä syystä on tärkeää standardoida PHP-sovelluksien rakennetta ja käytäntöjä esimerkiksi käyttämällä johonkin tunnettuun arkkitehtuurisuunnittelumalliin perustuvaa ohjelmistokehystä (Sharp 2007). PHP on joustava olio-ohjelmointikieli, mutta siitä johtuen myös hyvin altis huonolle ohjelmakoodille, huonoille käytännöille ja tietoturvaongelmille (Marston 2004; Sharp 2007). Erityisesti PHP:n löyhä tyypitys johtaa helposti ongelmiin (Shell 2009).

Ongelmia voidaan ennaltaehkäistä esimerkiksi PSR-suosituksilla ja käyttämällä ohjelmistokehystä PHP-sovellusta kehitettäessä (Lockhart & Sturgeon 2016). PHP:n tyyppivihjauksen (engl. Type Hinting) käyttäminen on myös tärkeätä (Wojciech 2014). Myös muita niin sanottujen ankkatyypitettyjen kielien (engl. duck-typed language) hyviä käytäntöjä tulisi käyttää. Näihin käytäntöihin kuuluvat kattava dokumentaatio, selkeä ohjelmakoodi ja perusteellisesti suunniteltu testaussarja.

2.3 Symfony

On olemassa useita PHP-ohjelmointikielellä kirjoitettuja ohjelmistokehyksiä, ja niistä valtaosa pohjautuu MVC-arkkitehtuurimalliin. Alun perin Fabien Potencierin käynnistämä Symfony-projekti kehittää yhtä näistä monista PHP-ohjelmointikielellä kirjoitetuista ohjelmistokehyksistä.

Symfony on WAF (Web Application Framework), joka on luotu HTTP-pyyntöjen ja -vastausten käsittelyä varten (Wallsmith 2014). Se rakentuu Symfony-projektissa luotujen komponenttien päälle. Se on myös Push-tyyppinen MVC-kehys, jossa ohjain tulkitsee käyttäjän toimet, hakee dataa malleilta ja työntää informaation näkymälle (Rutenberg 2008).

Projektin kehitystyötä johtaa Potencierin perustama yritys SensioLabs, joka sijaitsee Ranskassa. Symfonyn kehitystyö käynnistyi virallisesti lokakuussa 2005, kun projektin verkkosivusto julkistettiin. Sen ensimmäinen versio julkaistiin 19. tammikuuta 2007. (Zaninotto 2007.)

Nykyään käytössä oleva toisen sukupolven versio Symfony 2.0 julkaistiin 28. heinäkuuta 2011. Se on kirjoitettu kokonaan uudelleen käyttäen PHP-standardeja, kuten esimerkiksi PHPUnit-yksikkötestausta, PHP-nimiavaruuksia ja PSR-0 Autoloader -mekanismia. Isoin rakenteellinen muutos oli Symfonyn käyttämien komponenttien irrottaminen toisistaan. (Potencier 2011a.)

Symfony on suoraan tarkoitettu internetissä toimivien palveluiden rungoksi. Se kilpailee suosiosta muun muassa Zend- ja ASP.NET -kehysten kanssa. Se on vapaan- ja avoimen lähdekoodin ohjelmistokehys, joka on lisensoitu MIT-lisenssillä (Wallsmith 2014).

Ohjelmistokehyksen vahvuuksiksi voidaan mieltää Symfonyn kohdalla sen sisältämien työkalujen määrä. Niillä kyetään nopeuttamaan WWW-sovelluskehitystä merkittävästi, kun kehittäjien ei tarvitse kirjoittaa sovelluskohtaisia työkaluja uudelleen. Se johtaa myös parantuneeseen tietoturvaan. Symfonyn mukana tulevilla työkaluilla voidaan WWW-sovellus suojata yleisimmiltä tietoturvaongelmilta, joihin esimerkiksi lukeutuvat CSRF-hyökkäys sekä SQL-injektointi. SensioLabs tarjoaa Symfonylle kattavan ja selkeän dokumentaation sivustollaan. Symfonyn ja sen komponenttien lähdekoodit ovat myös saatavilla esimerkiksi GitHub-palvelussa.

Symfonyn kattavuus voidaan nähdä sekä positiivisena että negatiivisena asiana: hyvänä siksi, että sillä voidaan rakentaa monia erityyppisiä WWW-sovelluksia, koska työkaluja ja ominaisuuksia on käytännössä jokaiseen tarpeeseen, ja huonona siksi, että siihen ensikertaa tutustuvalle on paljon opeteltavaa, vaikka kyseessä olisikin kokenut PHP-kehittäjä. Symfonyn useimpien käsitteiden ja paradigmojen ymmärtäminen vie aikaa.

On myös mainittava, että Symfonystä on olemassa vain vähän painettua kirjallisuutta. Tämä osittain johtuu varmaankin siitä, miten kattava ja hyvin kirjoitettu

SensioLabsin verkkosivuillaan tarjoama dokumentaatio on. Symfonya ja sen dokumentaatiota päivitetään usein, mistä johtuen Symfony-kirjat vanhenisivat noin puolen vuoden välein.

Monet WWW-palvelut, jotka saavat päivittäin merkittävän paljon käyttäjiä sivustoilleen, on rakennettu Symfonyn varaan. Sitä käyttävät esimerkiksi video-sivusto Dailymotion, TED-organisaatio, Askeet-hakukone, Delicious-linkkipalvelu sekä monet muut tunnetut ja vähemmän tunnetut sivustot. (Potencier 2007a; Potencier 2009a; Potencier 2012.)

Yahoon suosittu Yahoo Answers -palvelu toimii Symfonyn varassa. Sillä oli vuonna 2008 noin 135 miljoonaa käyttäjää (Whittle 2008). Myös Yahoon oma, nykyään jo lakkautettu, linkkipalvelu Yahoo Bookmarks toimi Symfonyn varassa. Yahoo Bookmarks -palvelulla oli parhaimmillaan yli 20 miljoonaa käyttäjää (Zaninotto 2006). Symfony on Yahoon lisäksi käytössä lukuisissa muissakin isoissa yrityksissä. Esimerkiksi yhdysvaltalainen radio- ja tv-yhtiö CBS ja Yhdistyneen kuningaskunnan yleisradioyhtiö BBC käyttävät Symfonya. (Potencier 2012.)

Symfony on kohdennettu raskaiden WWW-sovellusten rakentamiseen, ja se on alusta lähtien kirjoitettu yritysten tarpeita ajatellen. Tämä ei silti estä Symfonya skaalautumasta myös pienempiin sovelluksiin. Symfony on rakenteeltaan täysin modulaarinen, ja kehittäjät voivat ottaa omaan sovellukseensa vain ne komponentit tai ohjelmistokehyksen ne osat, joita he kokevat tarvitsevansa. Kokenut sovelluskehittäjä voi halutessaan käyttäen Symfony Components -osia kirjoittaa esimerkiksi oman MVC-mallin mukaisen mikro-ohjelmistokehyksen.

2.3.1 Symfony Components

Symfony koostuu kahdesta laajemmasta kokonaisuudesta, jotka jakautuvat pienemmiksi osiksi. Näistä keskeisin on Symfony Components, joka on joukko toisistaan irrallaan olevia ja kierrätettäviä itsenäisiä osia. Useat PHP-sovellukset on rakennettu niitä käyttäen.

Käytännössä useimmat Symfony Components -osat ovat oliopohjaisia PHP-kirjastoja, joilla WWW-sovelluksissa usein tarvittuja toimintoja on helppo

ottaa käyttöön. Näihin toimintoihin lukeutuvat muun muassa lokimerkintöjen hallinta, tiedostojen käsittely, WWW-sivujen reititys ja HTML-lomakkeiden hallinta.

Tunnettuja Symfony Components -sovelluksia ovat esimerkiksi autonomiset selain-pohjaiset sisällönhallintajärjestelmät, kuten Drupal CMS ja eZ Publish, sekä lukuisten internetfoorumeiden käytössä oleva phpBB-sovellus. Symfony Components muodostaa keskeisen osan Symfony Framework -kehyksestä, joka on Components-osien ympärille rakennettu ohjelmistokehys. (SensioLabs 2015af.)

2.3.2 Symfony Framework

Verkkosivustoilla on usein esimerkiksi jokin yrityksen tai yhteisön kuulumisista kertova blogi- tai RSS-uutissyöte. Tällaisen ominaisuuden toiminta on pääsääntöisesti hyvin samanlaista sivustosta riippumatta. Tämän vuoksi ei ole järkevää kirjoittaa samaa RSS-uutissyötteen esittäjää aina uudelleen, kun uutta sivustoa ollaan pystyttämässä.

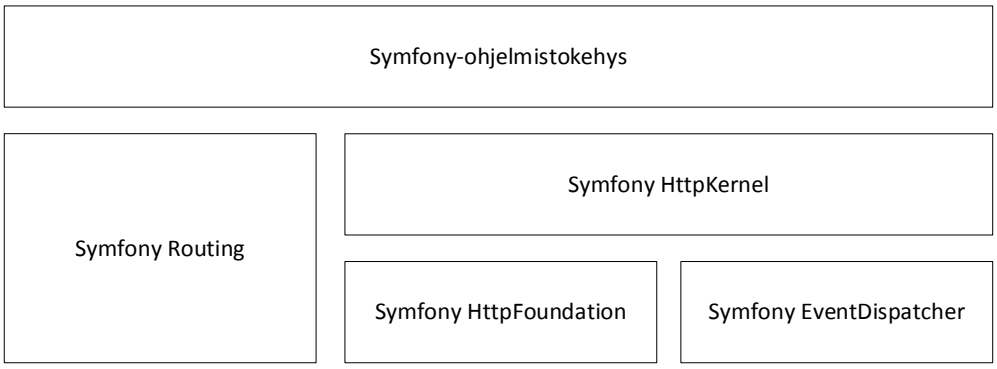
Symfony on kehitetty nopeuttamaan verkkosisällön kehitystä tekemällä usein käytetyistä ominaisuuksista mahdollisimman paljon uudelleen kierrätettäviä itsenäisiä osia (SensioLabs 2015c, 4). Symfony-sovelluksissa pyritään tähän muun muassa eristämällä sivustojen varsinainen sisältö muualle MVC-arkkitehtuurimallin periaatteiden mukaisesti, vaikka Symfony ei varsinaisesti pyri olemaan MVC-kehys (Potencier 2011b).

Kun näitä osia ei tarvitse aina kirjoittaa uudelleen, kehittäjät kykenevät keskittymään muun muassa tietoturvaan ja tällä tavoin tekemään osasta kykyjensä mukaan parhaan mahdollisen. Samaa osaa voidaan käyttää monta kertaa uudelleen. Näitä osia voidaan myös kehittää eteenpäin sisältämään hyvinkin edistyneitä toimintoja, joista kaikki saman kehittäjän sivustot hyötyvät. Näitä uudelleen kierrätettäviä osia kutsutaan Symfonyssä nimellä Software Bundle (suom. Ohjelmistopaketti). (SensioLabs 2015aj, 22.)

Symfony Framework on vain yksi bundle muiden joukossa (SensioLabs 2015aj, 22). Tämän FrameworkBundlen tehtävä on sitoa valittu joukko Symfony Components -osia ja kolmansien osapuolien kirjastoja yhdeksi toimivaksi ohjelmisto-

kehykseksi. Kuviossa 3 on nähtävissä tyypillinen Symfony-kehyksen rakenne ja siinä käytetyt komponentit.

Yksi bundle tavallisesti keskittyy sisältämään yhden toiminnallisen tarkoituksen, joka on jaettu tietoja hakevaan, tietoja käsittelevään ja tietoja esittävään osaan MVC-arkkitehtuurimallin mukaisesti. Tyypillisesti yksi bundle sisältää esimerkiksi blogin ja sen hallinnoimiseen liittyvät toiminnot. Bundle voi myös olla muun muassa isomman sivuston hallintapaneeli tai julkaisujärjestelmän osa.



KUVIO 3. Symfony rakentuu Symfony-komponenttien varaan

Bundleja on saatavilla vapaiden- ja avointen lähdekoodien lisenssien kanssa verkosta. Nämä vapaasti saatavilla olevat usein hyvin yleisluontoiset bundlet voidaan erikoistaa hyvinkin pitkälle, esimerkkinä mainittakoon Sonata-projekti. Se on joukko Symfony 2 -bundleja verkkokauppa ratkaisujen rakentamiseen. Esimerkiksi Sonata AdminBundle on tarkoitettu ylläpito- ja hallintapaneelien luomiseen WWW-sovelluksiin. (Rabaix 2016.)

Kehittäjät voivat erikoistaa bundlen tarkasti ennalta määriteltyihin toimintoihin ja jakaa näitä eteenpäin muille kehittäjille. Jos kokeneemman kehittäjän bundlen lisenssi sallii muokkaamisen voi aloitteleva kehittäjä halutessaan ottaa kokeneemman kehittäjän bundlen käyttöön, opiskella sen toimintaa ja erikoistaa sen omia tarpeitaan vastaavaksi.

Symfony on pyritty pitämään modulaarisena, jotta sen komponenttien muodostama kehys sallisi mahdollisimman monenlaisen sovelluksen kehittämisen. Modulaarisuuden ansiosta Symfonyllä on myös hyvin pieni suorituskykyrasite.

Suorituskykyyn liittyen Symfonystä löytyy myös erityinen välimuisti (engl. bytecode cache), johon se tallentaa jo kerran käsiteltyjen HTTP-pyyntöjen ja -vastausten osia (SensioLabs 2015al, 230). Esimerkiksi jokaista PHP-ohjelmakoodia sisältävää sivua ei välttämättä käsitellä useammin kuin yhden kerran.

Käyttäjän selaimen palvelimelta pyytämä data voidaan suoraan lähettää käyttäjälle, mikä merkittävästi nopeuttaa käyttäjän näkökulmasta WWW-palvelun toimintaa. Tämä HTTP-spesifikaatiossa mainittuihin standardeihin nojaava välimuisti myös keventää palvelimen suorittimen kuormaa, kun PHP-ohjelmakoodia ei tarvitse jokaista vierailijaa varten erikseen käsitellä. (SensioLabs 2015al, 230.)

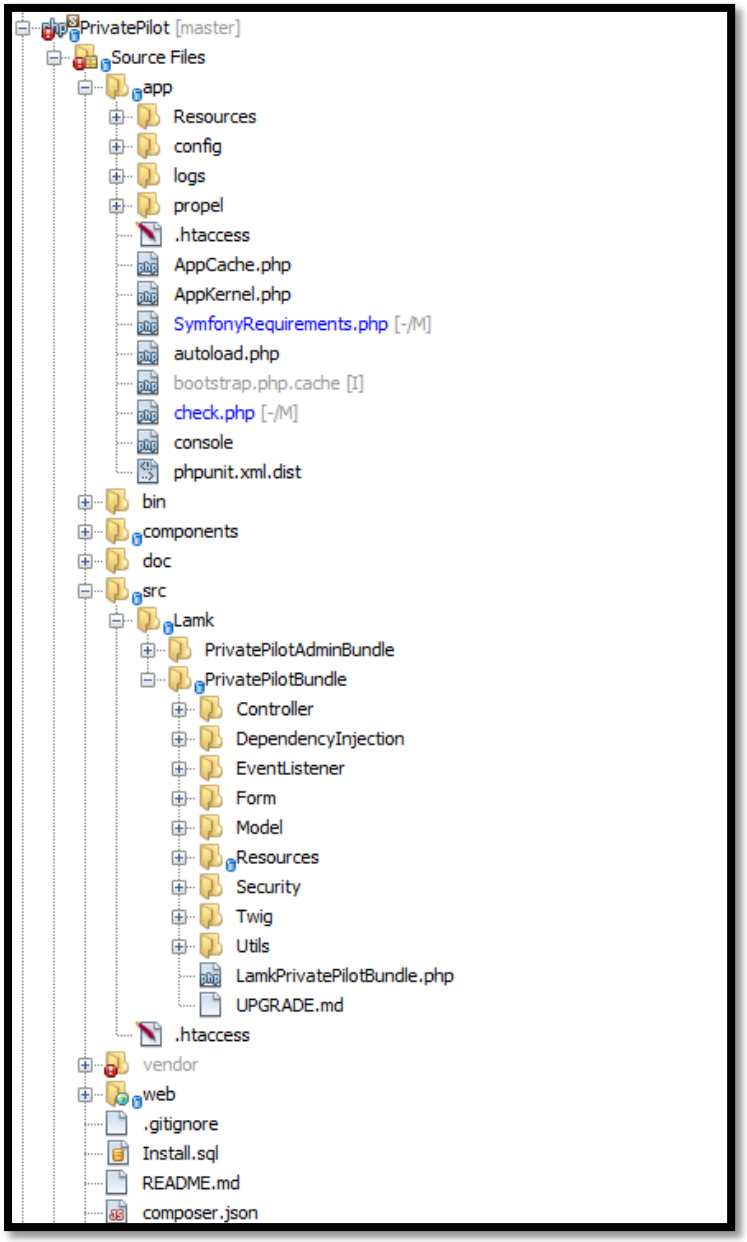
2.4 Arkkitehtuuri Symfony-sovelluksessa

Se, miten kehysohjelmistoja rakennetaan ja mihin tarkoitukseen, vaihtelee suuresti tavoitteiden ja teknologioiden mukaan. Tavallisesti WWW-kehysohjelmistoissa pyritään usein toistuvien tehtävien, kuten esimerkiksi sivujen reitityksen ja tietoturvan, rakentamisprosessin parantamiseen. Näin kehittäjät ja suunnittelijat pystyvät keskittymään WWW-sovelluksien sisältöön toistuvien pienien teknisten ongelmien ratkomisen sijaan.

Symfonyssä sisäisten osien keskeisenä tehtävä on välittää informaatiota eri järjestelmien välillä. Se on suunniteltu yhdistämään monet WWW-sovelluksien kehityksessä käytetyt työkalut yhdeksi järjestelmälliseksi kokonaisuudeksi, jolla on yhtenäiset asetukset ja yhtenäinen rakenne.

Symfony-terminologiassa sovellus tarkoittaa kansiota, joka sisältää asetustiedostot projektissa käytössä oleville bundleille (SensioLabs 2015ai). Symfonyssä on suhteellisen joustava hakemistorakenne, joka mahdollistaa useantyyppisten sovellusten rakentamisen. Kehittäjät kuitenkin suosittelevat Symfony-sovellukselle kuvion 4 mukaista organisointia. (SensioLabs 2015ak, 21–22.)

Oletusarvoisesti projektissa käytetty app-kansio sisältää Symfony-sovelluksen. Se sisältää pääasiassa Symfony-sovelluksen asetukset, mallineille (engl. templates), kuten Twig-päämallinteet ja käännökset muille kielille. App-kansiossa sijaitsee myös AppKernel-luokka, joka on keskeinen osa Symfony-sovelluksessa. Se muun muassa alustaa ajoympäristön ja määrittää, mitä bundle-paketteja sovelluksessa käytetään. (SensioLabs 2015ak, 21–22.)



KUVIO 4. Symfony-projektille tyypillinen kansiorakenne

AppKernel-aliluokka erikoistaa ja rajoittaa Symfony-kehyksen abstraktia Kernel-kirjastoluokkaa, joka hallitsee Symfonyn bundle-ympäristöjä. Se toteuttaa registerBundles()- ja registerContainerConfiguration()-metodit, kuten kuviossa 5 nähdään. Ensimmäinen metodi palauttaa taulukon niistä bundleista, joita sovelluksen ajamiseksi tarvitaan. Jälkimmäinen metodi lataa config-kansiossa sijaitsevat asetustiedostot, joita näiden ladattujen bundlejen kanssa käytetään. (SensioLabs 2015ak, 21–22.)

Kehittäjän omat Symfony-sovellusta varten kirjoitetut bundlet sijaitsevat src-kansiossa. Tämä tarkoittaa projektia varten tuotettua lähdekoodia ja projektissa tarjolle asetettuja resursseja, kuten esimerkiksi Twig-näkymät ja bundle-kohtaiset Symfony Event -laajennukset. (SensioLabs 2015ak, 21–22.)

Kolmansien osapuolien bundlet, komponentit ja Symfony-sovelluksen käyttöön asetetut kirjastot löytyvät vendor-kansion sisältä. Kyseistä kansiota käytetään Composer-paketinhallintasovelluksen oletusarvoisena asennussijaintina Symfony-sovelluksissa. Tästä johtuen sovelluskehittäjän ei tulisi muokata vendor-kansiossa sijaitsevia osia. Esimerkiksi Symfonyn käyttämien kolmansien osapuolien komponenttien, kuten SwiftMailer-kirjaston, Doctrine ORM:n ja Twig-järjestelmän keskeiset osat sijaitsevat oletusarvoisesti siellä. (SensioLabs 2015am, 25.)

Sovelluksen juurihakemisto on web-kansio (toisin sanottuna public_html-kansio, johon WWW-palvelin osoitetaan). Symfonyllä rakennetun WWW-sovelluksen suoritus käynnistyy tästä kansiosta. Se sisältää kaikki ne staattiset tiedostot, kuten kuvat, CSS-tyylitiedostot ja Javascript-tiedostot, jotka halutaan julkisesti saattaa tarjolle verkkoon. Kyseisessä kansiossa sijaitsee myös Symfonyn Front Controller -osa.

Projektin varsinaiset lähdekoodit Symfony eristää src-kansioon pois web-kansiosta, jotta WWW-palvelimella ei ole niihin pääsyä muun muassa tietoturvasyistä. Symfony Front Controller on PHP-skripti (app.php-tiedosto), joka tyypillisesti vastaanottaa ja valmistelee kaikki saapuvat pyynnöt käsittelyä varten.

Toisin sanoen Front Controller on alkulatausohjelma (engl. bootstrapper) Symfony-sovellukselle. Se osoitetaan WWW-palvelimelle sivuston index-tiedostoksi. Front Controller alustaa sovelluksen AppKernel-osan sekä luo Request-objektin PHP:n järjestelmänlaajuisista muuttujista ja välittää luodun objektin Symfonyn HttpKernel-komponentille. Front Controller -osan tehtävänä on myös välittää HttpKernelin tuottama sisältö takaisin käyttäjälle. (SensioLabs 2015ae, 9–12; SensioLabs 2015f, 75–77.)

```php
<?php

use Symfony\Component\HttpKernel\Kernel;
use Symfony\Component\Config\Loader\LoaderInterface;

class AppKernel extends Kernel
{

    public function registerBundles()
    {
        $bundles = array(
            new Symfony\Bundle\FrameworkBundle\FrameworkBundle(),
            new Symfony\Bundle\SecurityBundle\SecurityBundle(),
            new Symfony\Bundle\TwigBundle\TwigBundle(),
            new Symfony\Bundle\MonologBundle\MonologBundle(),
            new Symfony\Bundle\SwiftmailerBundle\SwiftmailerBundle(),
            new Symfony\Bundle\AsseticBundle\AsseticBundle(),
            new Braincrafted\Bundle\BootstrapBundle\BraincraftedBootstrapBundle(),
            new Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle(),
            new Propel\PropelBundle\PropelBundle(),
            new Lamk\PrivatePilotBundle\LamkPrivatePilotBundle(),
            new Lamk\PrivatePilotAdminBundle\LamkPrivatePilotAdminBundle(),
        );

        if (in_array($this->getEnvironment(), array('dev', 'test'))) {
            $bundles[] = new Symfony\Bundle\WebProfilerBundle\WebProfilerBundle();
            $bundles[] = new Sensio\Bundle\DistributionBundle\SensioDistributionBundle();
            $bundles[] = new Sensio\Bundle\GeneratorBundle\SensioGeneratorBundle();
        }

        return $bundles;
    }

    public function registerContainerConfiguration(LoaderInterface $loader)
    {
        $loader->load(__DIR__ . '/config/config_' . $this->getEnvironment() . '.yml');
    }

    public function init()
    {
        date_default_timezone_set('UTC');
        parent::init();
    }
}
```

KUVIO 5. Tyypillinen AppKernel-luokka Symfony-sovelluksessa

On huomionarvoista mainita, että Symfony ei automaattisesti huolehdi tuotantojärjestelmissä välimuistin päivittämisestä. Tämä johtuu siitä, että Symfony-sovelluksessa on tavallisesti useita lähdekooditiedostoja ja monia eri asetustiedostoja, jotka voivat olla samassa WWW-sovelluksessa useissa eri tiedostomuodoissa. Niiden jäsentämisessä menee paljon aikaa, ja se vaatii paljon resursseja. Sen sijaan, että Symfony jäsentäisi kaikki nuo tiedostot jokaista pyyntöä kohti aina uudelleen, ne tallennetaan välimuistiin uudelleenkäyttöä varten.

Symfonyssä on myös kattava lokijärjestelmä, joka perustuu Monolog-kirjastolle. Sillä on helppo jäljittää vikoja, kun niitä ilmenee kehitystyön aikana tai kun järjestelmä on tuotantokäytössä.

2.4.1 Symfony Bundle -järjestelmä

Symfonyssä WWW-sovellus jakaantuu uudelleenkierrätettäviin kokonaisuuksiin. Näistä kokonaisuuksista käytetään nimeä bundle. Ohjelmistokehyksen kaikki osat, mukaan lukien Symfonyn omat komponentit, on jaettu näiden ohjelmistopakettien sisään. (SensioLabs 2015ad, 93–94.)

Käytännössä yksi bundle on kansio, joka sisältää yhden loogisen toiminnallisen kokonaisuuden (kuten esimerkiksi blogin ja keskustelupalstan) asetukset ja logiikan muun muassa PHP-tiedostojen muodossa. Se voi myös sisältää esimerkiksi kuvia, CSS-tyylitiedostoja ja Javascript-tiedostoja. (SensioLabs 2015ai.)

Symfonyssä bundle-paketit voidaan jakaa pääasiassa kahteen kategoriaan. On olemassa sovelluskohtaisia paketteja, joita on käytetty vain rakentamaan kehittäjän omaa sovellusta. On myös olemassa uudelleen kierrätettäviä bundleja, jotka on tarkoitettu jaettavaksi ja käytettäväksi useissa eri sovelluksissa. Sovelluskohtainen bundle voidaan tarvittaessa muuntaa uudelleen kierrätettäväksi bundleksi.

Joitain keskeisimpiä bundleja, joita tavallisesti käytetään Symfony-pohjaisessa WWW-sovelluksessa, ovat muun muassa SecurityBundle, TwigBundle ja WebProfilerBundle. SecurityBundle tarjoaa kattavan turvajärjestelmän sovellukselle. TwigBundle lisää Twig kielen tuen. WebProfilerBundle antaa Symfonylle visuaalisen PHP-vianjäljittimen, suorituskyvyn tarkkailijan ja tietoturvan profilointi-ominaisuudet.

Bundlet tulee rekisteröidä AppKernelissä ennen kuin niitä voi käyttää (SensioLabs 2015ad, 93–94). Myös bundlejen asetukset tulee syöttää sovellukselle. Asetukset voidaan luoda käyttäen YAML-, XML- tai PHP-merkintätapaa.

Bundle voi sisältää myös lisätoiminnallisuutta muihin bundle-paketteihin. Ne voivat siten vaikuttaa sovelluksen kaikkiin tasoihin ja muovata niiden toimintaa.

Esimerkiksi SensioFrameworkExtraBundle sisältää monia laajennusosia FrameworkBundleen. Se sisältää muun muassa tuen määrittää Symfony-sovelluksen tietoturvaominaisuuksia ja reittejä käyttäen PHP-annotaatioita.

2.4.2 Symfony HttpFoundation

Symfonyn yksi kaikkein keskeisimpiä komponentteja on HttpFoundation. Se tarjoaa sovellukselle sen tarvitsemat työkalut HTTP-pyyntöjen ja -vastausten hallintaa varten. HttpFoundation on rakenteeltaan käytännössä joukko olio-keskeisiä abstraktioita PHP:n sisäänrakennetuille funktioille ja muuttujille. Se sisältää myös muun muassa tarvittavat metodit esimerkiksi Request- ja Response-objektien elinkaarien hallintaan, joita eritysesti HttpKernel-komponentti käyttää. (SensioLabs 2015n, 234.)

HttpFoundation-komponentin Request-luokka on abstraktio PHP:n globaaleille muuttujille, kuten esimerkiksi $_GET, $_COOKIE, $_SERVER ja $_POST. Yleisin tapa luoda Request-luokka on kutsua PHP:n senhetkiset globaalit muuttujat käyttämällä createFromGlobals()-funktiota. (SensioLabs 2015n, 234.)

HttpFoundationin Response-luokka taas abstraktoi PHP:n funktioita kuten echo(), setcookie() ja header(). HttpFoundationin Session-luokka (yhdessä SessionStorageInterfacen kanssa) toimii abstraktiona PHP:n sessionhallintatoiminnollisuudelle. (SensioLabs 2015o, 239; SensioLabs 2015p, 245–247.)

2.4.3 Symfony HttpKernel

HTTP on yksinkertainen tekstipohjainen tiedonsiirtoprotokolla, jossa jokainen vuorovaikutteinen HTTP-kysely (engl. Request) päättyy HTTP-vastaukseen (engl. Response). Näiden kyselyiden ja vastausten käsittely on Symfonyn tärkein tehtävä.

HttpKernel on komponentti Symfonyn sydämessä, jonka vastuulla on järjestelmällisesti käsitellä asiakassovelluksilta (kuten WWW-selaimilta) tulevat Request-objektit ja muuntaa ne aina Response-objekteiksi (SensioLabs 2015r, 264–266). Tähän Symfony käyttää apunaan Symfony EventDispatcher -komponenttia,

joka on yhden tyyppinen viestinvälitysarkkitehtuurin toteutus (SensioLabs 2015q, 161).

Symfonyssä HttpKernel ei varsinaisesti tee itse mitään. Se on tapahtumaohjautuva (engl. Event-driven), eli kaikki varsinainen työ tapahtuu tapahtumakuuntelijoiksi (engl. Event Listeners) rekisteröidyissä luokissa, jotka kuuntelevat HttpKernelistä tulevia ydintapahtumia (engl. Kernel Events) (SensioLabs 2015v, 266–267). Tämä on melko tavanomainen esimerkki neuvottelija-suunnittelumallista (engl. Mediator Software Design Pattern).

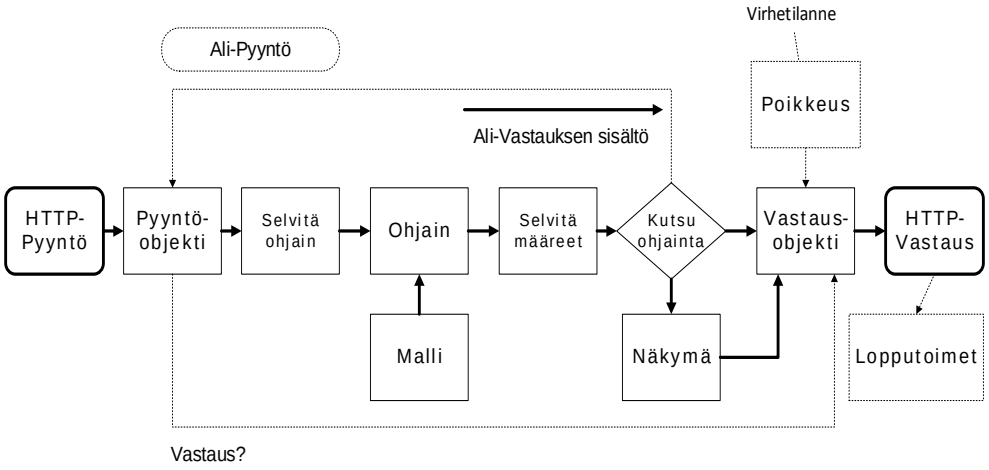
HttpKernelissä näistä ydintapahtumista ilmoitetaan Symfony EventDispatcher -komponenttia käyttäen, joka välittää tiedon niitä odottaville kuuntelijoille. Kuviossa 6 näkyvää HttpKernel-prosessia on tarkasteltu yleisellä tasolla erityisesti Symfony-sovelluksen näkökulmasta. On huomionarvoista, että HttpKernel-komponentin toiminta voi vaihdella sovelluksesta riippuen. Kehittäjät voivat halutessaan käyttää HttpKernel-komponenttia omissa sovelluksissaan tai kehysohjelmistoissaan. Esimerkiksi Drupal-sisällönhallintajärjestelmä käyttää Symfony HttpKernel-komponenttia HTTP-kyselyiden ja -pyyntöjen käsittelyyn (Drupal 2016).

Drupal-kehittäjät ovat implementoineet HttpKernelin tavalla, jonka he kokivat palvelevan heidän tarpeitaan parhaiten. Se eroaa tässä käsiteltävästä Symfony-kehysohjelmiston referenssi implementaatiosta. Näihin eroihin ei tässä opinnäytetyössä kuitenkaan syvennytä.

HTTP-pyynnön muunnos vastaukseksi työstetään HttpKernel-luokan handle()-metodissa. Se ajaa hallitusti saman luokan handleRaw()-jäsenfunktion, jossa tietojen välitys varsinaisesti tapahtuu (SensioLabs 2015v, 266–267). HandleRaw():n ensimmäisenä tehtävänä on ilmoittaa pyynnön saapumisesta kernel.request-ydintapahtumaa odottaville kuuntelijoille, kuten esimerkiksi SecurityListenerille ja RouteListenerille.

Kernel.request-tapahtumaan liitettyjen tapahtumakuuntelijoiden tehtävänä on tutkia HTTP-pyyntöjen luonne ja esikäsitellä ne. Tämä tapahtuu esimerkiksi lisäämällä informaatiota niihin ja muodostamalla Request-objekti HttpFoundati-

on-komponenttia hyväksi käyttäen. Tässä on keskeisenä tarkoituksena antaa sovellukselle mahdollisuus luoda vastaus ennen kuin mitään muuta ohjelmakoodia suoritetaan.



KUVIO 6. HttpKernelin välitysketju Symfony-sovelluksessa (SensioLabs 2015ac, 273)

Request-objekti toisinaan muunnetaan välittömästi Response-objektiksi, joka lähetetään asiakkaalle. Näin käy esimerkiksi tapauksissa, joissa SecurityListener evää reitin perusteella asiakkaalta pääsyn järjestelmän siihen osaan johon pyyntö kohdistui.

Näissä tietoturvaan liittyvissä tapauksissa HttpKernel ei välttämättä edes selvitä, mihin ohjaimeen pyyntö kohdistui, vaan siirtyy suoraan kernel.response-tapahtumaan ja lähettää HTTP-vastauksena esimerkiksi 403 Pääsy estetty HTTP-tilakoodin (SensioLabs 2015y, 267–268). Jos mikään tapahtumakuuntelija ei luonut Response-objektia, HttpKernel selvittää, mihin ohjaimeen HTTP-pyyntö kohdistui. Tämä päättää kernel.response-tapahtuman.

Luokka, joka implementoi ControllerResolverInterfacen, huolehtii Symfony-sovelluksessa ohjaimen selvittämisestä. Symfony Framework käyttää sisäänrakennettua ControllerResolver-luokkaa. HttpKernelin handle()-metodi kutsuu siltä getController()- ja getControllerArguments()-jäsenfunktioita, jotka kertovat ytimelle sen tarvitsemat tiedot ohjaimesta. Tämä tapahtuu RouteLis-

tener-kuuntelijan Request-objektiin lisäämän informaation perusteella. (SensioLabs 2015w, 268–270.)

Selvitettyään, mihin ohjaimeen pyyntö kohdistui, ja varmistuttuaan sen olemassaolosta HttpKernel lataa ohjaimen. Tässä vaiheessa HttpKernel myös välittää tiedon kernel.controller-tapahtumasta sitä tarkkaileville kuuntelijoille.

Tarkoituksena kernel.controller-tapahtumassa on Symfony-sovelluksen ja ohjaimen valmistelu suoritusta varten (SensioLabs 2015x, 270–271). Samassa yhteydessä ohjaimelle myös haetaan sen mahdollisesti tarvitsemat argumentit (SensioLabs 2015t, 271–272). Tämä tapahtuu sillä olettamuksella, että ydin onnistuu lataamaan ohjaimen sekä valmistelemaan sovelluksen ohjaimen suoritusta varten.

Seuraavaksi ohjain suoritetaan. Ohjaimen on aina tuotettava jotain, tai syntyy poikkeustilanne ja tieto kernel.exception-tapahtumasta lähetetään eteenpäin virheen hallittua käsittelyä varten (SensioLabs 2015s, 272–273; SensioLabs 2015u, 276–277). Onnistuneen suorituksen jälkeen ohjain useimmiten tuottaa Response-objektin, jolloin siirrytään kernel.response-ydintapahtumaan. (SensioLabs 2015s, 272–273.)

Ohjain ei kuitenkaan aina tuota valmista Response-objektia. Symfonyssä HttpKernel on jossain määrin vastuussa siitä, että vastaus pyydettyyn resurssiin tulee luoduksi. Jos ohjain tuottaa jotain muuta kuin Response-objektin, mutta ei myöskään tuota niin sanottua Null-vastausta, HttpKernel tekee ilmoituksen kernel.view-tapahtumasta. Tällöin kernel.view-tapahtumaan liitettyjen kuuntelijoiden tehtävänä on muuttaa ohjaimelta tuleva data-objekti Reponse-objektiksi käyttäen sen mukana kulkevia tietoja. Jos näin ei tapahdu, välitetään tieto kernel.exception-tapahtumasta. (SensioLabs 2015ab, 274.)

Olettaen, että joko kernel.controller tai kernel.view onnistuivat luomaan Response-objektin, päädytään aina kernel.response-tapahtumaan. Tämän jälkeen se lähetetään eteenpäin HTTP-tapahtuman käynnistäneelle asiakkaalle eli tavallisesti käyttäjän WWW-selaimelle. (SensioLabs 2015z, 275.)

Kun Request-Response-objektin käsittely on saatettu loppuun, välitetään kuuntelijoille vielä tieto kernel.complete_request- ja kernel.terminate-tapahtumasta. Se

kertoo Symfonylle ja kehittäjälle tapahtuman onnistumisesta tai epäonnistumisesta sekä jälkien siivoamisesta. (SensioLabs 2015aa, 275–276.)

HttpKernelin toiminnan ymmärtäminen on keskeistä kaikille Symfony-kehittäjille. Erityisesti ne kehittäjät, jotka haluavat kirjoittaa edistyneitä Symfony-sovelluksia ja muokata Symfonyn toimintaa haluavat ymmärtää tätä prosessia voidakseen muovata sen toimintaa tarkoituksiinsa sopivaksi.

2.5 YAML-merkintäkieli

YAML on vaihtoehto XML-merkintäkielelle. Se tulee sanoista YAML Ain’t Markup Language. YAML on tarkoitettu ihmisille helppolukuiseksi tavaksi serialisoida tietoja. YAML-formaatti toimii hyvin asetustiedostojen muotona. YAML-kielen ensimmäisen version loivat Clark Evans, Brian Ingerson ja Oren Ben-Kiki vuonna 2001. (Evans, Ben-Kiki & Ingerson 2009.)

Syntaksi YAML-kielessä on suunniteltu helposti kytkettäväksi useimpien tunnettujen ohjelmointikielien data-tyyppeihin, kuten listoihin ja taulukoihin. Kuviossa 7 on nähtävissä, millaista YAML-kielen tyypitys on.

```
imports:
    - { resource: parameters.yml }
    - { resource: security.yml }

framework:
    #esi:             ~
    #translator:      { fallback: "%locale%" }
    secret:          "%secret%"
    router:
        resource: "%kernel.root_dir%/config/routing.yml"
        strict_requirements: ~
    form:            ~
    csrf_protection: ~
    validation:
         enable_annotations: true
         enabled: true
    templating:
        engines: ['twig']
        #assets_version: SomeVersionScheme
    default_locale:  "%locale%"
    trusted_hosts:   ~
    trusted_proxies: ~
    session:
        # handler_id set to null will use default session handler from php.ini
        handler_id:  ~
    fragments:       ~
    http_method_override: true
    # Validation for Propel and other things.
    validation: { enabled: true }
```

KUVIO 7. Symfonyn käyttämät asetustiedostot tyypillisesti kirjoitetaan YAML-merkintäkielellä

YAML on yksi neljästä tavasta tallentaa ja muuttaa Symfony-sovelluksen asetuksia ja määreitä. Symfonyn asetukset voidaan kirjoittaa YAML-, PHP-, XML- tai INI-muotoa käyttäen. Oletusarvoisesti Symfonyn asetukset tehdään kuitenkin YAML-merkintäkielellä. (Zaninotto & Potencier 2007, 62.)

Kommenttimerkkinä YAML-kielessä toimii # (risuaita), paitsi jos se esiintyy merkkijonon (engl. string) sisällä, jolloin se edustaa numeron esityspaikkaa. Kommentti jatkuu rivin loppuun ja päättyy EOL-merkkiin.

YAML-kieli pyrkii hyvin hierarkkiseen tietojen esittämiseen. YAML:ssa perustyyppinä toimivat listat, jotka alkavat viivalla. Listoja voi myös kirjoittaa siten, että hakasulut sisältävät listan nimikkeet eroteltuina pilkuilla. YAML:n taulukot toimivat avain-arvo-muotoisina lohkoina.

Kielessä sarkaimet ja välilyönnit on määritelty kielenrakennetta kuvaaviksi tulostumattomiksi merkeiksi. Toisin kuin monessa muussa merkintäkielessä, YAML:ssa näillä tulostumattomille merkeille on annettu merkitys tietojen esittämisessä.

Esimerkiksi listoja tehtäessä lohkomerkin jälkeen rivit, jotka alkavat kahdella välilyönnillä, tulkitaan listan alkioiksi. Sarkaimia käytetään kertomaan näiden alkioiden sisältämistä jäsenistä tai niiden joukoista.

2.6 PHP- ja Twig-mallinekielet

Dynaamiset verkkosivut kirjoitettiin 90-luvun alussa tavallisesti C- tai Perl-kielellä. C-kielisen CGI-sovelluksen luominen käsittelemään HTML-sivujen dynaamisia osuuksia oli aikaa vievää ja hankalaa. Oli tarve saada jokin välitaso, jonka läpi informaatio kulkee backend-järjestelmältä HTML-mallineiden merkittyihin dynaamisiin lohkoihin sen sijaan, että CGI-sovellus luo kokonaisia HTML-sivuja.

PHP-ohjelmointikieli kehitettiin alun perin 90-luvun alussa mallinekieleksi (engl. template language) tähän kyseiseen tarkoitukseen. Sen luoja Rasmus Lerdorf suunnitteli PHP:n toimimaan ohuena kerroksena HTML-sivuston frontend-osuuden ja C-ohjelmointikielellä kirjoitetun backend-osuuden välillä. (Tatroe, MacIntyre, Lerdorf & Bourque 2013a, 2–6.)

PHP-kieltä ei koskaan otettu laajalti käyttöön tähän tarkoitukseen. PHP muuntui 90-luvun loppuun mennessä yleisluonteiseksi ohjelmointikieleksi. Se kehittyi nopeasti suuntaan, joka teki mahdolliseksi kirjoittaa kaiken tarvittavan bisneslogiikan ilman vaivalloista ja paljon resursseja vaativaa CGI:tä hyödyntävää C- tai Perl-ohjelmointia.

PHP:n yleistyminen aiheutti kuitenkin sen, että HTML-sivut ja HTML-mallineet usein täyttyivät PHP-ohjelmakoodista ja bisneslogiikasta. PHP-kielen käyttö HTML-merkintäkielen välissä siten hankaloittaa sivuston koodin luettavuutta ja ylläpidettävyyttä. Tästä syntyi tarve uudelle mallinekielelle, joka pelkästään muotoilisi ja esittäisi informaatiota, jota se vastaanottaa PHP:ltä.

Armin Ronacher kehitti Djangon mallinekieleen pohjautuvan Twig-mallinekielen vastaamaan tähän tarpeeseen vuosien 2007 ja 2008 välillä (Ronacher 2015). Twig on PHP-kielen versiolla 5 kirjoitettu mallinekieli, jonka olemassaolon oikeutus, eli *raison d'être*, on korvata PHP mallinekielenä HTML-mallinetiedostoissa. Twig-kielen pääasiallinen tehtävä on informaation sijoittaminen ja muotoileminen WWW-sivuille. Vuodesta 2009 lähtien Twig-kielen kehityksestä ovat vastanneet SensioLabs ja Fabien Potencier. Se on nykyään oletusarvoinen mallinekieli Symfony-ohjelmistokehyksessä. (Potencier 2009c.)

Twig käyttää kolmea rajamerkintätapaa. {{ … }}-merkintää käytetään merkitsemään kohtaa, johon tulostuu muuttujan sisältö tai suoritetun lausekkeen tulos (SensioLabs 2016g, 10-11). {% ... %}-merkintää käytetään kertomaan ohjausrakenteista, kuten for-silmukoista (SensioLabs 2016b, 13). {# … #}-merkintää käytetään kommentoimaan mallineiden sisältöä. Twig-kommentit eivät tulostu HTML-koodin sisään toisin kuin HTML:n <!-- … -->-merkinnät. (SensioLabs 2016a, 13.)

Twig-kielen perusominaisuus on lyhyet ja täsmälliset muotoilusäännöt. Muuttujan tulostamiseen PHP-kielessä käytetään yleisesti <?php echo $muuttuja ?> ja Twig-kielessä tähän riittää {{ muuttuja }}. (SensioLabs 2016g, 10–11.)

Toisin kuin PHP, Twig tukee automaattista output escaping -ominaisuutta tulostettaessa informaatiota. Tämä on tärkeä tietoturvaominaisuus, jolla voidaan pie-

nentää esimerkiksi XSS-hyökkäykseen liittyviä riskejä. Twig-kielen ympäristömuuttujista se voidaan myös kytkeä pois ja määrittää autoescape-tunnisteella tarkat lohkot, joissa sitä käytetään. (SensioLabs 2016d, 15–16.)

PHP-kielessä muuttujan syöte tulisi suodattaa esimerkiksi käyttäen htmlspecialchars()-funktiota, joka ottaa sisäänsä joukon parametreja. Twig-kielen kanssa riittää esimerkiksi escape-suodattimen käyttäminen. Kuviossa 8 on huomattavissa esimerkiksi {{ muuttuja | e }}, jossa e on alias escape-suodattimelle.

```
{# Lamk/Twig/Esimerkki.html.twig #}
{% extends "baseLayout.html" %}

{% block title %} Hopemage {% endblock %}

{% block head %}
    {# I never asked for this. #}
{% endblock %}

{# Navi-lista Bootsrapille. #}
{% block navigation %}
    {% spaceless %}
        <ul id="navigation">
            {% for item in navigation %}
                <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
            {% endfor %}
        </ul>
    {% endspaceless %}
{% endblock %}

{# Sisältö-lohko. #}
{% block content %}
    <h1>{{ blogHeader|striptags|title }}</h1>
    {% autoescape %}
        <p>{{ blogText }}</p>
    {% endautoescape %}
    <h2>{{blogReaders|striptags|upper}}</h2>
    {% if readers|length > 0 %}
        <ul>
            {% for reader in readers %}
                <li>{{ reader.nickname|e }}</li>
            {% endfor %}
        </ul>
    {% endif %}
{% endblock %}

{% block footer %}
    {{ parent() }}
    <!-- A cat is fine too. -->
{% endblock %}
```

KUVIO 8. Esimerkki Twig-kielen syntaksista

Informaation muotoiluun Twig-kielessä käytetään suodatinjärjestelmää. Oletus suodattimien käyttöön löytyy kattava dokumentaatio Twig-kielen verkkosivustolta.

Ohjelmoija voi halutessaan kirjoittaa Twig-kielelle myös omia suodattimiaan PHP-kielellä. Muotoiluun käytettävää suodatinta kutsutaan Twig-lohkoissa ja niitä voi ketjuttaa. Lähes kaikki muotoiluun liittyvä logiikka on irrotettu mallineesta ulkoisiin PHP-tiedostoihin. (SensioLabs 2016c, 11–12.)

Twig-mallineita voidaan myös periyttää. Tämä tapahtuu esimerkiksi laatimalla pohjamalli, josta kaikki muut mallinetiedostot johdetaan. Se tapahtuu käyttämällä extends-tunnistetta. (SensioLabs 2016f, 14–15.)

Merkintätavalla ei ole myöskään merkitystä Twig-kieltä käytettäessä. Ohjelmoijat voivat myös kirjoittaa esimerkiksi XML-mallineita. Vakinaistettu tapa merkitä Twig-tiedostot on tiedostonnimi.muoto.twig eli esimerkiksi esimerkki.html.twig HTML-mallineelle tai esimerkki.xml.twig XML-mallineelle. Twig sisältää välimuistin, ja tavallisesti Twig-tiedostot käännetään PHTML-mallineiksi ennen esittämistä. Tämä nopeuttaa sivujen esittämistä.

Twig-kieleen löytyy i18n-laajennus, joka lisää tuen GNU Gettextille (SensioLabs 2016j). Se mahdollistaa trans-tunnisteen käyttämisen eri kieliversioiden määrittämiseen mallineen sisällä. Monikielisten verkkosivujen kehittämisestä tulee helpompaa, kun samoja mallineita voidaan käyttää useiden eri kielien kanssa.

Makrot Twig-kielessä ovat periaatteessa vastine muiden ohjelmointikielien funktioille. Makroilla pyritään uudelleen kierrättämään vain pieniä muutoksia käyttökohteissaan kokevaa HTML-koodin lohkoa mahdollisimman paljon. Kun samaa koodia ei tarvitse kirjoittaa uudelleen montaa kertaa, se helpottaa mahdollisten ohjelmavikojen löytämistä ja korjaamista. (SensioLabs 2016e, 16.)

Twig on myös täysin laajennettavissa. Kehittäjät voivat halutessaan kirjoittaa omia Twig-käsittelijöitä, joilla voidaan muokata Twigin toimintaa (SensioLabs 2016h, 38–40). Lisäksi yksikkötestaus on osa Twig-kieltä. Jos ohjelmoija esimerkiksi kirjoittaa Twig-laajennuksia ja suodattimia, niitä varten voidaan laatia myös yksikkötestit (SensioLabs 2016i, 43). Näin voidaan yrittää varmistaa mallineiden toimivan suunnitellusti esimerkiksi sovelluksessa, joka käyttää kehittäjän omia laajennuksia.

2.7 Bootstrap

Bootstrap on CSS-kehys nopeaan WWW-käyttöliittymäkehitykseen (Otto & Thornton 2016). Se on joustava tapa luoda sivustoja, jotka skaalautuvat hyvin suuren tuumakoon ja kuvapistetarkkuuden omaavista 4K-näytöistä aivan pienimpiinkin muutaman tuuman kokoisiin matkapuhelimien näyttöihin.

Bootstrap on avointa ja vapaata lähdekoodia, ja sen kehitystyön on alun perin käynnistänyt vuoden 2010 puolessa välissä yhdysvaltalainen sosiaalisen median palveluita tuottava Twitter Incorporated (Otto & Thornton 2015). Yrityksen pääasiallinen tuote, mikroblogipalvelu Twitter, on rakennettu käyttäen Bootstrap-kehystä käyttöliittymässä.

Tästä kehyksestä on olemassa erityisesti Symfonya varten koottuja versioita, jotka integroituvat bundle-paketin muodossa sovellukseen. Näitä on olemassa muutama erilainen, mutta pääsääntöisesti niiden välillä ei ole merkittäviä toiminallisia eroja. Ei ole myöskään mitenkään outoa, jos sovellus käyttää Bootstrap CSS -rautalankamallia ilman Symfony-kohtaisia liitoksia.

3 OLIO-RELAATIOKUVAUSJÄRJESTELMÄ

Olio-relaatiokuvausjärjestelmällä (engl. Object Relational Mapping, ORM) abstraktoidaan ja välitetään tietoja kahden keskenään yhteensopimattoman tietojärjestelmän välillä. Tällaisella järjestelmällä voidaan kartoittaa esimerkiksi Java-luokkien tiedot vastaamaan yhteensopivalla tavalla PostgreSQL-tietokannan tauluja. ORM-kirjastoja on olemassa useita erilaisia eri tarkoituksiin, ohjelmointikieliin ja tietojärjestelmiin. (Hibernate.org 2016; Bauer & King 2007, 25–27; Bhatt 2014.)

On tavallista, että olio-relaatiokuvausjärjestelmää käytetään esimerkiksi isoissa dynaamisissa WWW-palveluissa yhdistämään palvelun käyttämän relaatio-tietokannan taulut ja olio-ohjelmointikielen olio-pohjaiset muuttujat keskenään. (Bowler & Bancer 2009, 8–9.)

Kun isoa sovellusta kehitetään, se tulee usein kiinteästi riippuvaiseksi useista eri komponenteista, kuten tietokannasta ja sen versiosta. Esimerkiksi MySQL-tietokantaan voi syntyä riippuvuus WWW-sovelluksissa, joissa käsitellään jatkuvasti ja paljon niin sanottuja suuria tietovarantoja (engl. Big Data). (Bowler & Bancer 2009, 8–9.)

Jos esimerkiksi Oracle päättää olla julkaisematta päivityksiä MySQL-tietokantaan, saattaa WWW-palvelua ylläpitävä taho ehkä vaihtaa tietokantaratkaisua. Olettaen, että WWW-palvelu ei käytä ORM-järjestelmää tai jotain muuta Database Abstration Layerin toteuttavaa DAL-kerrosta, ja se sisältää paljon MySQL-kohtaista ohjelmakoodia, sen tietokantaan kytkeytyvät osat pitää kirjoittaa käytännössä lähes tyhjästä lähtien uudelleen vastaamaan uutta tietokantaa. (Bowler & Bancer 2009, 8–9.)

Palvelun muiden osien kuten esimerkiksi frontendin kehittyminen saattaa hidastua tai jopa pysähtyä backend-järjestelmien uudelleen kirjoituksen ajaksi. Tästä seuraa myös, että palvelun MySQL-tietokanta joudutaan siirtämään uuteen tietokantaan kuten esimerkiksi Microsoftin MSSQL-tietokantaan. Tietokannan tiedot tulee vielä sovittaa ja optimoida uutta tietokantaa sekä sen rajapintaa varten. Tämä olisi kallis

ja aikaa vievä operaatio, joka pitää tehdä uudelleen myös seuraavalla kerralla, kun sovelluksen taustalla oleva tietokanta vaihdetaan.

Database Abstraction Layer ratkaisee ongelman luomalla väliin puskurin, joka toimii siltana sovelluksen ja tietokannan välillä. DAL:ää käyttäviä ORM:n tapaisia järjestelmiä on ollut olemassa jo vuosikymmeniä, mutta niiden käyttö alkoi yleistyä vasta olio-pohjaisten ohjelmointikielien yleistyttyä. Se johtui tarpeesta selvittää keskenään ristiinkytkettyjen luokkien aiheuttamia ongelmia.

Vaikka tietokanta taustalla vaihtuisi, ei koko sovelluksen tietokantatoimintoja tarvitse kirjoittaa uudelleen. Riittää kun ORM:n käyttämän DAL:n asetukset vaihdetaan vastaamaan uutta tietokantaa ja migraation avulla vanha tietokanta siirretään uuteen tietokantajärjestelmään.

Kehittyneemmillä ORM-järjestelmillä voidaan myös takaisinmallintaa (engl. reverse engineer) olemassa oleva tietokanta väliaikaiskuvaustiedostoksi, jota muokkaamalla voidaan tuottaa täysin uusi ja toimiva kantakuvaus. Kuvaustiedostolla voidaan myöhemmin tuoda uuteen tietokantaan vanhaa kantaa vastaavat taulut.

Tietojen siirtäminen onnistuu esimerkiksi käyttämällä Propellissa Fixtures-ominaisuutta, joka poimii vanhasta tietokannasta tiedot Propelin omaan JSON-tyyppiseen tiedostoon. Ne voidaan luovuttaa uuteen tietokantaan muutamalla komennolla.

Tapa, jolla ORM-työkaluilla tuotetaan kuvaus tietojen tyypeistä ja säilytyksestä, vaihtelee kirjastosta riippuen. Java Derbyn tapauksessa kuvauksen ovat Java-luokkia. Propel ORM kuvaa tietoja tavallisesti XML-muotoisena tekstitiedostona DTD-validoidun skeeman kanssa. Siitä rakennetaan mekaanisesti kaikki tarvittavat luokka-tiedostot. Doctrine ORM:ssä taas tiedot kuvataan puhtaina PHP-luokkina, jotka johdetaan Doctrinen omista luokista.

ORM on loppukäyttäjälle täysin näkymätön asia. Käyttäjä kuitenkin tulee käyttäneeksi ORM-järjestelmää sitä mukaan kun hän joko lukee tai kirjoittaa tietojaan tietojärjestelmään sovelluksen frontend-osassa, joka käyttää tietoja ORM:n läpi.

Pääsääntöisesti ORM-ratkaisun käyttäminen lisää sovellukseen vielä yhden riippuvuuden, mikä lisää sovelluksen monimutkaisuutta. Tietokantaa ei myöskään voi vapaasti kehittää, sillä kehityksen tulee tapahtua jatkuvasti päivittyvien kuvaus- tai luokka-tiedostojen läpi.

Joissain tapauksissa ORM saattaa lisätä sovelluksen järjestelmävaatimuksia. Erillisen välitason ohjelmiston läpi tietojenkäsittely voi vähemmän suorituskykyisillä tietokoneilla johtaa hidastuneeseen tietojen hakemiseen tai jopa tietojen katoamiseen. Etenkin jos käytössä oleva Data Access Layer -välitaso on suunniteltu tai toteutettu huonosti.

Kerran kehitettyjen luokkien katoaminen, jotka ovat riippuvaisia toisista abstraktoiduista luokista, aiheuttaa ORM-järjestelmää käytettäessä saman tapaisen ongelman kuin jos sitä ei käytettäisi. Tällöin sovelluksen tietokantaan, tai tässä tapauksessa ORM-järjestelmään kytkeytyvä toiminnollisuus, tulee suunnitella uudelleen tai tuottaa uudelleen siltä osin kuin muutoksia kuvaus- tai luokka-tiedostojen esittämiin tietoihin tehtiin.

ORM-järjestelmää käytettäessä voi myös joutua kohtaamaan joukon ongelmia, jotka tunnetaan nimellä Object-relational Impedance Mismatch (Ambler 2013). Yksi tähän liittyvistä ongelmista liittyy tietojen kätkentään. ORM-järjestelmä saattaa esimerkiksi paljastaa olion rajapinnoille, joilta saapuvaa tietovirtaa olion jäsenfunktiot eivät osaa käsitellä yhteensopivalla tavalla. Tämä heikentää kapseloinnista saatavia hyötyjä. Tämä ja monet muut tähän liittyvät ongelmat pääsääntöisesti johtuvat siitä, etteivät relaatiotietokannat tue olio-ohjelmoinnissa käytettyjä konsepteja.

3.1 Propel ORM

Propel on vapaan- ja avoimen lähdekoodin MIT-lisenssillä lisensoitu olio-relaatiokuvauskirjasto ja kokoelma työkaluja, jotka on kirjoitettu PHP-ohjelmointikielellä. Niiden kehitys käynnistyi vuonna 2000-luvun alussa ja versio 1.0 julkaistiin 2004. (Propelorm.org 2015b; Lellelid 2005.)

Näiden komponenttien pääasiallinen tehtävä on yhdistää PHP-luokat ja tuetun tietokannan taulut keskenään. Tuetut tietokannat ovat versiossa kaksi MySQL, PostgreSQL, SQLite, MSSQL ja Oracle DB.

Tietokannan taulujen kuvaamiseen Propel käyttää XML-skeemaa. Se on yhden tyyppinen toteutus Active Record -ohjelmistosuunnittelumallista, jossa yksi PHP-luokka vastaa yhtä tietokannan taulua. (Fowler 2002.)

Propel muodostuu kolmesta komponentista, joiden nimet ovat Generator, Runtime ja Library. Generator-komponentti rakentaa mekaanisesti PHP-luokkia XML-kuvaustiedostojen pohjalta, joissa tietokannan sisältö ja rakenne on määritelty. Runtime-komponentti hallitsee tietokantayhteyksiä PHP:n PDO-rajapintaa apunaan käyttäen, suorittaa transaktioita sekä kuvailee relaatiotietokannan hallintaan liittyviä sääntöjä ja rajoituksia.

PHP-kirjasto, jolla Propel voidaan integroida osaksi muita sovelluksia, kulkee nimellä Library. Tämän komponentin avulla voidaan kirjoittaa esimerkiksi käyttöliittymiä ja uusia työkaluja, jotka helpottavat tietokannan kehitystä ja hallintaa.

PHP:n versio viisi päivitti PHP:n olio-ohjelmointiominaisuuksia tehden mahdolliseksi Propelin kaltaisten projektien luomisen (Tatroe, MacIntyre, Lerdorf & Bourque 2013a, 2–6; The PHP Group 2016). Se myös loi laajalti kysyntää komponenteille, joita ei aikaisemmin ollut olemassa. Propelin esikuvana ja pohjana on ollut Apache Torque -projekti, jonka ORM-komponentti on kirjoitettu Java-ohjelmointikielellä (Lellelid 2003).

Propel oli hyvin kiinteä osa Symfony-kehysohjelmistoa. Se oli oletusarvoinen ORM-komponentti aina versioon 1.2 asti (Potencier 2007b). Symfony-ohjelmakehyksen versiosta 2.0 eteenpäin oletusarvoisena ORM-järjestelmänä sen korvasi Doctrine ORM (Potencier 2009b).

3.2 Propelin keskeisiä ominaisuuksia

Keskeisenä ominaisuutena Propelissa on sen kyky nopeaan prototyypitykseen. Tarvittavien luokkien ja muun ohjelmakoodin luominen tapahtuu automaattisesti,

mikä nopeuttaa kehitystyötä sitä käyttävissä sovelluksissa. Siinä on mukana myös joitain takaisinmallinnusominaisuuksia, jotka mahdollistavat olemassa olevien, Propelin tukemien tietokantojen, kääntämisen tietokantakuvaustiedostoiksi. (Propelorm.org 2015b.)

Propel on avointa-lähdekoodia ja siten muokattavissa kehittäjien tarpeita vastaavaksi. Projektin kehittäjät ovat dokumentoineet sen hyvin laajalti mikä tekee siitä kehittäjäystävällisen. Projekti myös noudattaa PHP-standardeja PSR-0, PSR-1, PSR-2 ja PSR-3. Propel on kehittäjiensä mukaan kattavasti yksikkötestattu, ja sen kirjastot ovat käyneet läpi kattavan validointiprosessin. (Propelorm.org 2015b.)

3.3 Tietokannan kuvaaminen merkintäkielellä Propel-järjestelmälle

Hyvän skeema-tiedoston kirjoittaminen edellyttää hyvää suunnittelua sekä kokemusta tietokantasuunnittelusta. Propel olio-relaatiokuvausjärjestelmä käyttää oletusarvoisesti schema.xml-tiedostoa, johon kuvataan tietokannan taulut XML-kielellä. Tietokannan voi kuvata myös YAML-kielellä. (Propelorm.org 2015a.)

```
<table name="received_order">
    <column name="id" type="INTEGER" required="true" primaryKey="true" autoIncrement="true" />
    <column name="id_user_personal_info" type="INTEGER" />
    <column name="id_flight_route" type="INTEGER" />
    <column name="id_order_payment_method" type="INTEGER" />
    <column name="number_of_reserved_seats" type="INTEGER" />
    <!--
    <column name="order_date_added" type="DATE" />
    <column name="order_time_added" type="TIME" />
    -->
    <behavior name="timestampable" />
    <foreign-key foreignTable="user_personal_info">
        <reference local="id_user_personal_info" foreign="id" />
    </foreign-key>
    <foreign-key foreignTable="flight_route">
        <reference local="id_flight_route" foreign="id" />
    </foreign-key>
    <foreign-key foreignTable="payment_method">
        <reference local="id_order_payment_method" foreign="id" />
    </foreign-key>
    <vendor type="mysql">
        <parameter name="Engine" value="InnoDB" />
        <parameter name="Charset" value="utf8" />
        <parameter name="Collate" value="utf8_swedish_ci" />
    </vendor>
</table>
```

KUVIO 9. Ote Private Pilot -sovelluksen Propel Schema -tiedostosta

Kuvion 9 tyylisiä skeema-tiedostoja voi olla yhdessä projektissa useampia. Kehittäjä voi myös halutessaan sekoittaa XML- ja YAML-tyyppejä keskenään. Tämä ei ole kuitenkaan suositeltavaa, sillä mahdollisen virhetilanteen sattuessa ongelmien

ratkominen voi olla melko haastavaa. Lisäksi tiedostojen jäsentäminen useista eri kielillä kirjoitetuista skeema-tiedostosta voi olla hidasta.

3.4 Propel Fixtures

Propel Fixtures on tarkoitettu säilömään tietokannassa olevat tiedot sekä talletta-maan ne myöhempää käyttöä varten. Se on eräs tapa säilöä dynaamista dataa tie-tokannan ulkopuolelle kehityksen aikana. Tätä ominaisuutta voidaan myös käyttää tallentamaan muuttumatonta staattista sisältöä, kuten tekstiä ja lukuja. Sitä voidaan käyttää tallentamaan lähestulkoon kaikkea dataa, jota tarvitaan tuotantokäyttöön tulossa olevassa järjestelmässä.

```
Lamk\PrivatePilotBundle\Model\UserPersonalInfo:
    UserPersonalInfo_1:
        first_name: 'Antti'
        last_name: 'Testinen'
        line1: 'Testikatu 10A'
        line2: Haapa
        city: Lahti
        state: Häme
        postal_code: '13423'
        country_code: FI
Lamk\PrivatePilotBundle\Model\File:
    File_1:
        name: Avatar
        mime_type: image/jpeg
        url: 545d3d30f2ff3.jpeg
Lamk\PrivatePilotBundle\Model\UserRole:
    UserRole_1:
        role: ROLE_ADMIN
    UserRole_2:
        role: ROLE_PILOT
    UserRole_3:
        role: ROLE_USER
```

KUVIO 10. Ote Propel Fixtures -tiedostosta

Symfony Console toimii käyttöliittymänä Fixtures-toiminnoille. Fix-tures-tiedostoja voidaan kirjoittaa XML- tai YAML-muotoon, kuten kuviossa 10 on nähtävissä. Tämä on hyödyllinen ominaisuus tietokantaa kehitettäessä, jota käyttämällä testi-dataa ei jatkuvasti tarvitse kirjoittaa ja syöttää uudelleen sovel-lukseen. On myös mahdollista syöttää WWW-sovellukseen alustavia tietoja, kuten alkuasetuksia, kun sovellusta ollaan valmistelemassa käyttöä varten tuotantoym-päristöön. (Propelorm.org 2016.)

Fixtures-toimintoa voidaan käyttää myös yhteistyössä muiden PHP-kirjastojen kanssa. Esimerkiksi PHP Faker -kirjastoa voidaan käyttää testidatan generoimiseksi Fixtures-tiedostoon, ja täten nopeuttaa tietokantakehitystä.

4 PRIVATE PILOT -PALVELU

Private Pilot on WWW-palvelu, jossa yksityiset pienlentokoneiden lentäjät voivat tarjota eräänlaista taksi-palvelua yksityishenkilöille. Lentäjät ilmoittavat palvelussa käyttämiään lentokoneita, lentokenttiä ja reittejä, joita he voivat lentää maksua vastaan.

Palvelun asiakkaat voivat varata lentoja ja maksaa lennosta lentäjälle palvelun läpi esimerkiksi luottokortilla tai verkkomaksupalvelulla, kuten PayPal. Käyttäjät voivat vertailla lentäjien tarjoamia reittejä ja hintoja. He voivat myös kommentoida lentoja ja tarjota palautetta lentäjille ja palvelulle lentäjistä.

Tähän liittyy teknisiä haasteita, kuten esimerkiksi palvelun on kyettävä esittämään nopeasti ja luotettavasti tietokannassa olevaa informaatiota saatavilla olevista lennoista. Palvelun on myös kyettävä toimimaan useilla eri aikavyöhykkeillä. Sen tulee myös pystyä skaalautumaan ylöspäin käyttäjien kasvun mukaan. Ja käyttäjien tuottamaa sisältöä on kyettävä hallitsemaan monen muun asian lisäksi.

Palvelun kehitys käynnistyi keväällä 2014 osana Lahti Demo Room -hanketta, jossa opiskelijat toteuttavat prototyyppejä yrityksille (Mattila 2014). Mukana oli kaksi ryhmää, jotka kumpikin keskittyivät omaan demo-projektiinsa. Private Pilot -projekti oli toinen niistä kahdesta demosta, joita hankkeessa rakennettiin. Projektin mentorina toiminut henkilö ohjeisti käyttämään Symfony 2 -sovelluskehystä. Se oli projektiin osallistujille entuudestaan tuntematon ja loi omat haasteensa.

Käytetystä Symfony-ohjelmistokehyksestä on saatavilla muutama erilainen jakelupaketti. Sovellusta luodessa olisi suotavaa käyttää pitkäaikaistuen eli niin sanottua LTS-jakelupakettia, joka on kirjoitushetkellä versio 2.3. SensioLabs on lupautunut tukemaan sitä vuoden 2016 toukokuuhun asti. (SensioLabs 2015ag.)

Private Pilot -hankkeen käynnistyessä uusin saatavilla ollut Symfony-jakelu oli versionumeroltaan 2.4. Se ladattiin SensioLabsin verkkosivustolta ja otettiin pohjaksi Private Pilot -sovellukselle. Päätöksen taustalla oli idea kehittää sovellusta seuraavaa myöhemmin julkaistavaa LTS-versiota vasten pohjalla olevien komponenttien maksimaalisen tuen varmistamiseksi.

4.1 Palvelun suunnittelu

Palvelun kehitystyö käynnistyi ohjelmointipuolella kehitysympäristön perustamisella ja Symfonyn dokumentaation lukemisella. Kehitysympäristön rakentaminen ei tuottanut projektissa ongelmia ja oli hyvin mekaaninen työvaihe.

Projektin käynnistyttyä sen toteutus ohjelmointipuolella eteni aluksi varsin hitaasti. Aikaa kului palavereissa ja suunnittelussa, joissa ei pystytty käymään kaikkia tarpeellisia asioita aina läpi, minkä takia projektin ohjelmointi toimivaksi tuotteeksi koki myöhemmin vaikeuksia. Private Pilot -ryhmän osallistujilla oli vain niukasti kokemuksia ryhmätyöskentelystä eri alojen osaajien kanssa. Sen takia työvirtojen suunnittelu oli haasteellista koko projektin ajan.

Haasteista huolimatta palvelun toimiva prototyyppi toteutui ja toteuttaa sille asetetut vaatimukset, vaikka projekti myöhästyi sovitusta aikataulusta. Tavoitteena oli rakentaa toimiva demo-versio, joka esittelee palvelun toimintaa käytännössä. Tähän tavoitteeseen päästiin.

Käytännössä isoin osa Private Pilot -sovelluksen kehittämiseen kuluneesta ajasta käytettiin tutkimustyöhön ja dokumentaation lukemiseen. Dokumentaatio on erittäin kattava ja käy järjestelmällisesti läpi kaikki olennaisesti Symfony-kehitystyöhön liittyvät asiat. SensioLabsin sivustolta löytyy melkein kaikki tarvittavat asiakirjat ja tieto sovellusten kehittämiseen Symfony-ohjelmistokehyksellä.

Symfonyn dokumentaatio on jaettu muutamaan isompaan kirjalliseen kokonaisuuteen. Niistä tärkeimpinä voidaan pitää Symfony Book-, Symfony Cookbook-, Symfony Components-, Symfony Reference- ja Symfony Best Practises -kirjoja. Ne ovat saatavilla SensioLabsin verkkosivustolla. (SensioLabs 2015ah.)

Dokumentaation lukemisen lisäksi oli tärkeää myös ymmärtää, miten Symfony käytännössä toimii. Tämä tapahtui rakentamalla muutamia yksinkertaisia sovelluksia, jotka keskittyivät joihinkin perusasioihin. Myös joidenkin Symfonyn toiminnan kannalta kriittisten komponenttien lähdekoodia luettiin projektia varten. Tämä vei aikaa projektissa varsinaisen työn ohjelmoimiselta, etenkin alkuvaiheessa.

4.2 Kehitysympäristö

Kehitysalustaksi valittiin Debian GNU/Linux Jessie -jakelu sen ollessa yhä kehityksessä. Näin saatiin uusimmat Debian-projektin paketoimat avoimen ja vapaan lähdekoodin työkalut käyttöön. Debian on avoimen- ja vapaan lähdekoodin GNU/Linux-pohjainen käyttöjärjestelmä (Debian Project 2016).

Kehitysympäristöksi projektin ajaksi valittiin NetBeans 8.0, joka oli sen uusin versio Private Pilot -palvelun kehitystyön alkaessa. NetBeansin etuna on muun muassa monialustaisuus, hyvä tuki PHP-ohjelmointikielelle, tuki PHP-ohjelmointikielen standardoidulle muotoilusäännölle (PHP-CS-Fixer -laajennusosa) ja muokattavuus (Oracle Corporation 2016b).

Symfony-sovelluksen osia hallitaan Console-komponentilla. Se toimii muun muassa bundle-pakettien edistyneempien toimintojen merkkipohjaisena käyttöliittymänä helpottaen WWW-sovelluksen kehitystä ja hallintaa palvelimen puolella.

NetBeans-kehitysympäristö kytkeytyy saumattomasti Symfonyn ja sen eri komponentteihin, kuten muun muassa Console-komponenttiin. Tämä mahdollisti Symfonyn osien helpon hallinnan visuaalisen käyttöliittymän avulla.

Projektin kehityksessä käytössä oli Apache Foundationin WWW-palvelin HTTPD, joka yleisesti tunnetaan myös nimellä Apache. Debian-projektin työkalut ja PHP-liitännäiset kytkeytyvät siinä oletusarvoisesti Apache-palvelimeen. Kehitystyössä käytettiin Debian-jakelun oletusarvoisen PHP-ympäristön osia.

Tietokannaksi valittiin avoimen- ja vapaanohjelmakoodin tietokanta MySQL. Aiemmin Sun Microsystemsin tukema, mutta nykyisin Oraclen omistama, MySQL on luotettava ja pienistä keskisuuriin käyttötapauksiin hyvin skaalautuva tietokantaohjelmisto (Oracle Corporation 2016a).

4.2.1 Composer

Composer on paketinhallintajärjestelmä, jota käytetään hallitsemaan WWW-kehitysprojektien tarvitsemia ulkoisia riippuvuuksia, kuten esimerkiksi CSS-käyttöliittymäkirjastoja sekä Javascript- ja PHP-kirjastoja. Sen avulla on

vaivatonta päivittää sovelluskehityksessä käytetyt komponentit tarvittaessa uudempaan saatavilla olevaan julkaistuun versioon. (Adermann & Boggiano 2016.)

Symfony-sovelluksessa sitä käytetään hallitsemaan Symfony Components -osien eri versioita. Composer myös helpottaa projektille keskeisten komponenttien riippuvuussuhteiden hallintaa.

Composerin toiminnan edellytyksenä on composer.json-tiedoston olemassaolo sovelluksen kansion juuressa. Se sisältää JSON-kielellä kirjoitetun kuvauksen projektin vaatimista ulkoisista komponenteista ja niiden versioista, joita tarvitaan sovelluksessa.

Symfony-jakelun mukana tulee esimerkki composer.json-tiedosto, johon on valmiiksi kirjoitettu tärkeimmät Symfonyn liittyvät paketit. Tiedoston voi tarvittaessa luoda automaattisesti tai kirjoittaa itse alusta lähtien. Tämä on tarpeellista, kun halutaan käyttää vain joitain yksittäisiä komponentteja tai tarkasti määrittää, mitä paketteja ja niiden versioita sovelluksessa halutaan käyttää.

4.2.2 Symfony-projektin perustaminen

Symfony WWW-sovellusprojektin kehitystyön käynnistämiseksi on olemassa muutamia tapoja. Ohjelmistokehyksen kehittäjät suosittelevat käyttämään yksinomaan Composer-paketinhallintajärjestelmää. Se tapahtuu lataamalla Symfony-jakelu SensioLabsin verkkosivustolta ja purkamalla se itse valittuun projekti-kansioon. Halutessaan kehittäjä voi myös aloittaa esimerkiksi luomalla uuden projektin IDE:n ohjatulla toiminnolla. Sen jälkeen Composerilla tulisi päivittää Symfonyn tarvitsemat komponentit, sikäli kuin se on tarpeellista.

Private Pilot -projektissa käytettiin NetBeans IDE:n ohjattua toimintoa Symfony-projektin luomiseksi. Ohjattu toiminto ei käytä Composeria projektin perustamiseen. Ohjattu toiminto luo NetBeans-projektin, minkä jälkeen se purkaa ja rekisteröi luotuun projektiin ennalta määritellyn Symfony-jakelun tiedostot.

Projektin luomisen jälkeen on suositeltavaa hakea ja tarvittaessa päivittää Symfony-jakelu Composerilla. Jakelun mukana tulee esitehty composer.json-tiedosto,

joka sisältää tiedot Symfonyn sovellusprojektissa käyttämistä komponenteista. NetBeans-sisältää kontekstivalikon Composerin käyttämiseen. Sillä voi suorittaa kaikki keskeiset toiminnot, joita tarvitaan projektin hallitsemiseen.

Private Pilot -sovellusprojektissa composer.json-tiedostoon lisättiin muun muassa Propelin sen hetkinen vakaa Symfony bundle -jakelu. Näin se integroitui osaksi luotua Symfony-projektia lähes ongelmitta. Propelin asetuksia piti kirjoittaa asentamisen jälkeen. Näiden asetusten tekemiseen on Propelin sivulla olemassa yksityiskohtaiset ohjeet.

WWW-palveluissa tietoturva on hyvin keskeinen asia erityisesti ylläpidon näkökulmasta. On suositeltavaa tarkistaa säännöllisin väliajoin Symfony-sovellusprojektissa käytetyt komponentit tunnettujen tietoturvaongelmien varalta käyttämällä Symfony Consolen security:check-komentoa

Komento security:check selvittää, mitä komponentteja projektissa on käytössä ja mitä riippuvuuksia niillä on. Tämän jälkeen se hakee Symfonyn palvelimelta tiedot Components-osissa olevista tunnetuista haavoittuvuuksista ja varoittaa sovelluskehittäjää, jos tunnettuja haavoittuvuuksia on havaittu sovelluksen käyttämissä paketeissa.

4.2.3 Versionhallintajärjestelmän käyttöönotto

Kun Symfony-sovellusprojekti on luotu, on suositeltavaa lisätä se versionhallintajärjestelmään. Private Pilot -projektissa käytetään Git-versionhallintaa ja Atlassian Bitbucket-lähdekoodivarastopalvelua.

Git on hajautettu sosiaalinen versionhallintajärjestelmä (Git Project 2016). Sen kehityksen käynnisti Linux-ytimen kehityksestä tunnettu Linus Torvalds 2000-luvun puolessa välissä (Cloer 2015). Projekti pitää valmistella Gitin käyttöönottoa varten. Sitä varten tulee luoda muun muassa gitignore-tiedosto ja lisätä siihen sovellusprojektin eteenpäin jakamisen kannalta tarpeettomat kansiot ja tiedostot.

Näitä tiedostoja ovat esimerkiksi Composerilla haetut ulkoiset Symfony-komponentit. Esimerkki gitignore-tiedosto tulee usein valmiiksi Symfony-ohjelmistokehyksen standard-jakelun mukana. Tämän jälkeen voidaan Git-puu alustaa projektin kansiossa komennolla git init ja lisätä kaikki sovellusprojektin tiedostot puuhun komennolla git add. Lopuksi projektin tiedostot kommitoidaan git commit -komennolla.

Bitbucket on Atlassianin ylläpitämä sosiaalinenpilvipalvelu lähdekoodien varastointiin (Atlassian 2016). Se kilpailee muun muassa tunnetumman GitHubin kanssa Git-tyyppisten tietovarantojen säilytyksestä.

Projektin eteenpäin viemisessä käytettiin Bitbucketin tarjoamia toimintoja. Esimerkiksi tavoitteita listattiin virheenjäljitinjärjestelmään ominaisuuksina, joita vielä tarvitaan. Näitä lisättiin ja merkittiin selvitetyiksi sitä mukaan kun Private Pilot -projekti eteni.

Bitbucket-sivustoa käytettiin hahmottamaan mitä ominaisuuksia WWW-palvelulta sen käyttäjät haluavat. Private Pilot prototyypitettiin Kanban-tyylisesti pieniksi ominaisuuksiksi ja toiminnoiksi, joille annettiin tärkeysjärjestys ja aika virheenjäljitinjärjestelmässä. Näitä tehtäviä tehtiin sitä mukaan kun niitä voitiin alkaa toteuttamaan. Priorisointia jouduttiin tekemään, kun ilmeni ohjelmavikoja, joita piti korjata päivän aikana tai myöhemmin samalla viikolla.

4.2.4 WWW-palvelimen asetusten määrittäminen

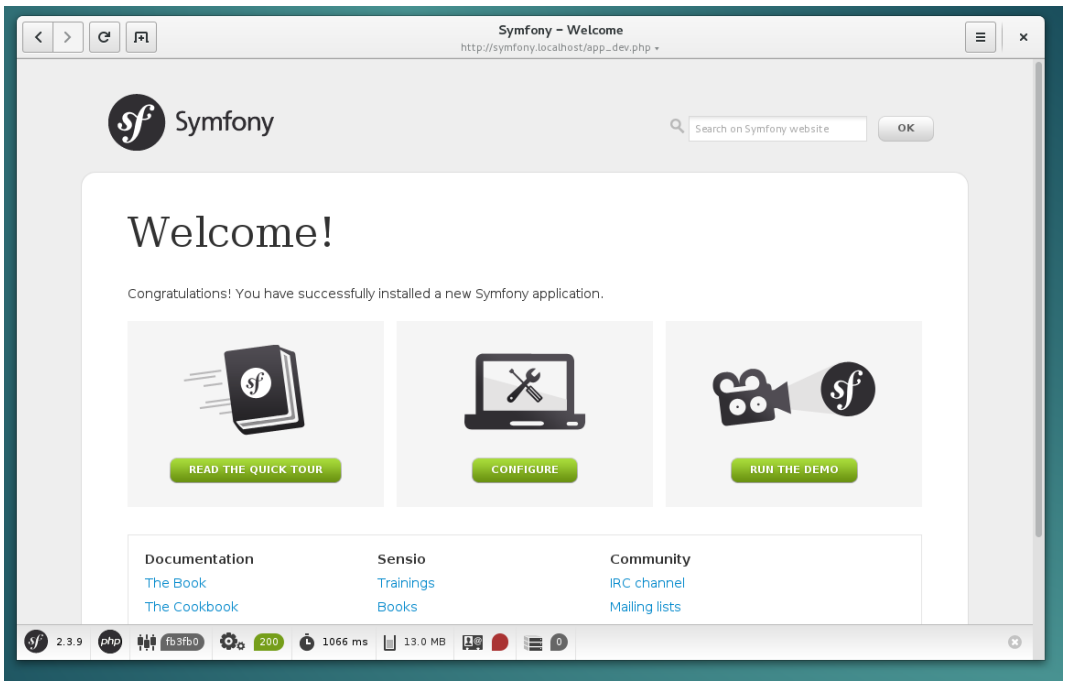
Symfonylla on jotain vaatimuksia sen suhteen miten WWW-palvelin on asetettu toimimaan. Apachen Virtual Host -asetustiedosto luodaan Debianissa kansioon /etc/apache2/sites-available. Private Pilot -projektissa kopioitiin esimerkiksi tarjottu host config -mallitiedosto default.conf, joka uudelleennimettiin privatepilot.conf-tiedostoksi. Tämä tiedosto muokattiin lisäämällä siihen ServerName-muuttuja privatepilot.localhost.

Seuraavaksi voidaan tehdä Apachelle Virtual Host -asetukset. Tämä on riippuvainen siitä millä käyttöjärjestelmällä projektia ajetaan. Tavallisesti se koostuu host-tiedostoon lisättävästä osoitteesta ja Apachen vhost.conf-tiedoston kirjoitta-

misesta. On tärkeätä myös kytkeä PHP Apachen moduuliksi ja tehdä tarvittavat määritykset niin, että se lataa PHP-tyyppisten tiedostot MIME-tyyppiin pohjautuen.

Symfonyn toiminta edellyttää, että WWW-palvelimella ja PHP-komentorivitulkilla on oikeus lisätä ja poistaa tiedostoja Symfonyn välimuistissa. Mikäli sama käyttäjäryhmä ei hallinnoi molempia, tulee ACL-määritys tehdä (SensioLabs 2015g, 30–31). Se tapahtuu Darwin-käyttöjärjestelmässä (toisin sanoen Mac OS X:ssä) ja muissa BSD-jakeluissa chmod +a -komentorivimääreellä. Linux-jakeluissa käytetään esimerkiksi setfacl-komentoa kertomaan tiedostojärjestelmälle useita käyttäjäoikeuksista yhdessä tiedostossa.

Kun Apachen asetukset on tehty ja palvelimen kirjoitusoikeudet välimuistiin tarkistettu, on hyvä yrittää testiajaa Symfonyn demo-bundle WWW-selaimessa. Jos kaikki menee hyvin, on tuloksena jotain kuvion 11 tapaista.



KUVIO 11. Demo-bundlen etusivu esillä WWW-selaimessa

Symfonyn demo-bundle sisältää esimerkkejä siitä, mitä kaikkea Symfonyllä voi tehdä, linkkejä dokumentaatioon ja harjoituksiin. Kuviossa 11 näkyy myös Symfonyn debug-paneeli, joka on tavallisesti aktiivinen kehitysympäristössä. Jos tes-

tiajo onnistuu, Symfony-projektien mukana tuleva demo-bundle poistetaan ja aloitetaan kehittämään sovelluksen bundleja.

4.3 Private Pilot -palvelun kehitys

Sovelluksen ohjelmoiminen voidaan jakaa muutamiin vaiheisiin. Näitä vaiheita voidaan eritellä ja ryhmitellä niissä tuotetun toiminnollisuuden perusteella. Niitä voidaan ajatella myös rakenteellisina kerroksina sovelluksessa.

Symfonyn dokumentaatio antaa ohjeita ja suosituksia Symfonyn tutustuvalle ohjelmoijalle ja kehottaa rakentamaan näitä kerroksia määritellyssä järjestyksessä. Näistä vaiheista oli apua Private Pilot -sovelluksen rakentamisessa toimivaksi WWW-palveluksi, jota voi oikeasti käyttää.

Yksinkertaisemmissa perussivustoissa, joissa on vähän tai ei lainkaan dynaamista sisältöä, ei ole kuin muutama vaihe. Paljon dynaamista sisältöä sisältävissä monimutkaisemmissa WWW-palveluissa on kehitysvaiheita enemmän ja niiden toteutus edellyttää jo kerroksilta tarkempaa suunnitelmallisuutta. Private Pilot sovelluksessa näitä vaiheita olivat yleisluontoisesti ilmaistuna muun muassa tarvittavien toimintojen ja ominaisuuksien selvittäminen, niiden jakaminen bundleiksi, bundlejen ohjelmointi ja testaus, sekä käyttöönottaminen.

Private Pilot -sovellus keskittyy lentojen tilausjärjestelmään ja käyttäjien hallintaan. Mutta tavallisesti tämän tapaisista palveluista löytyy myös jonkin tyyppinen sisällönhallintajärjestelmä varsinaisen sivuston hallintaan. Private Pilot -palvelussa ei ole CMS-järjestelmää, mutta sellaisen myöhempää integrointia varten on jätetty mahdollisuus.

Yhdessä vaiheessa työstettiin paljon perusasioita kuten perussivustoa ja liittyviä asioita, jotka tuovat informaatiota käyttäjän näytölle. Tarkoituksena oli saada Symfony esittämään Private Pilot -sivustolle liittyvää informaatiota ja luoda pohjaa kehittyneemmille ominaisuuksille. Sovellukselle määriteltiin reittejä sivuston osiin kuten etusivulle, profiileihin, sekä funktioihin kuten lisää lentoreitti, poista lentoreitti, muokkaa lentoreittiä, ja niin edespäin. Eräs vaihe oli eri kirjastojen ja työ-

kalujen integrointia osaksi sivustoa. Esimerkiksi Propel oli ensimmäinen iso ulkoinen komponentti, joka integroitiin osaksi Private Pilot -sovellusta.

Yksi merkittävimmistä kehitysvaiheista koostui ongelmien ratkomisesta eräiden kehittyneempien toimintojen toteuttamiseksi ja testaamiseksi. Myös eräänlainen bundlejen uudelleen organisointi ja ohjelmakoodin refaktorointi olivat osa tätä vaihetta Private Pilot -sovelluksen kehityksen aikana.

Esimerkiksi yritys integroida Bootstrap CSS-kehys sellaisenaan osaksi Symfonya epäonnistui. Sen kanssa koettujen ongelmien vuoksi se myöhemmin korvattiin Florian Eckerstorferin kehittämällä BraincraftedBootstrapBundle-paketilla (Eckerstorfer 2015). Se ratkaisi monia integrointiongelmia ja helpotti merkittävästi muun muassa Bootstrapin sisällyttämistä Symfony Forms -komponenttia käyttäviin Twigillä kirjoitettuihin lomakkeisiin.

Bundlejen uudelleen järjestelystä on kerrottava hieman taustatietoa. Symfonyssä on alusta lähtien ollut bundle-käsitteenä. Symfonyn ensimmäisissä versioissa yksi bundle vastasi yhtä loogista toimintojen kokonaisuutta, joka oli siirrettävissä mihin tahansa muuhun Symfony-sovellukseen sellaisenaan. Tämä kuitenkin usein johti siihen, että Symfony bundlet muodostivat keskenään riippuvuussuhteita toisiinsa, eli ne eivät oikeasti olleet niin siirrettävissä Symfony-asennusten välillä kuten oli tarkoitus.

Nykyään SensioLab ohjeistaa, että yhden bundle-paketin tulisi periaatteessa sisältää vain yksi kokonainen sivusto ilman riippuvuuksia muihin bundle-paketteihin (SensioLabs 2015a). Eli esimerkiksi blogi ja kaikki sen toiminnot ovat yhdessä paketissa. Private Pilot -projektissa kyseistä nyrkkisääntöä ei noudatettu.

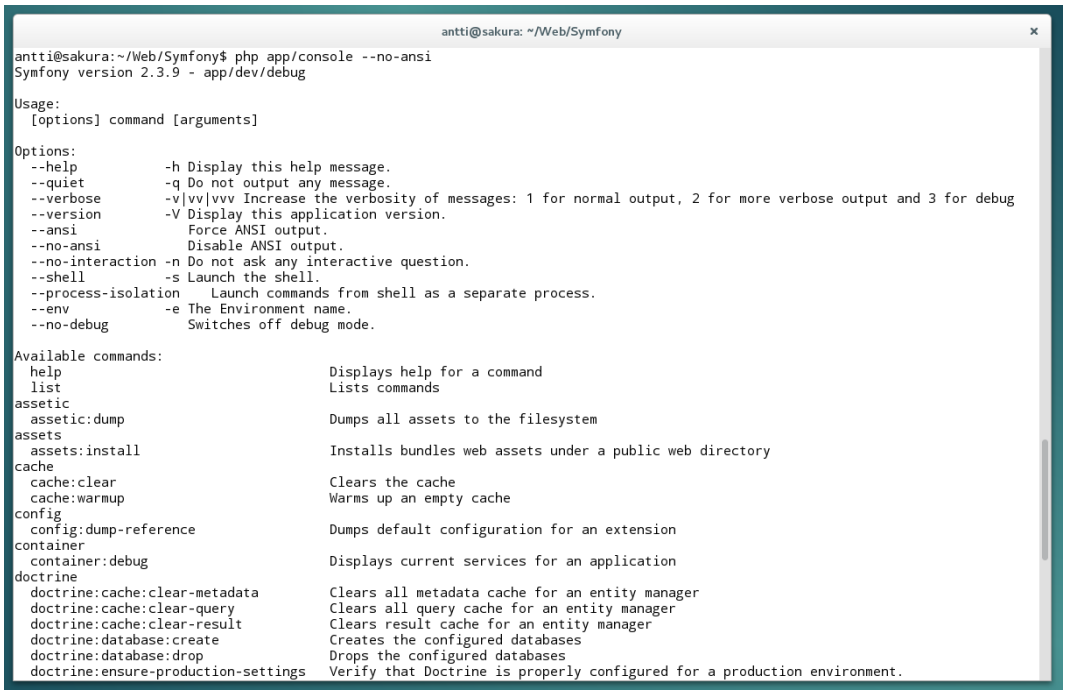
Private Pilot -sovellus on rakennettu osittain Symfonyn ensimmäisen version ohjeistukseen nojaten, jossa laajempien toimintojen kokonaisuudet on jaettu useampiin bundle-paketteihin tehden niistä keskenään riippuvaisia toisistaan. Toisaalta ne ovat myös helposti irrotettavissa toisistaan, erikoistettavissa ja kierrätettävissä muiden sivustojen bundleiksi.

Näin myös tehtiin kun huomattiin, että niin sanottu CommonBundle oli kasvanut isommaksi kuin mitä sille määriteltyjen tehtävien ja vastuiden olisi pitänyt olla.

Niitä päädyttiin rajaamaan pilkkomalla bundle kahteen toisistaan riippuvaiseen osaan.

4.3.1 Symfony Console

Symfony Console on Symfony Components osa, jolla voidaan vaivattomasti lisätä komentorivikomentoja PHP-sovellukseen keskitetysti. Tavallisesti tällä konsolilla automatisoidaan Symfony-ohjelmistokehyksen eri toimintoja, joista osa on nähtävissä kuviossa 12. Sillä esimerkiksi luodaan ja siivotaan välimuisti, tutkitaan reittien toimivuutta, käsitellään tietokantaa, jne.



KUVIO 12. Console -komponentin tarjoamia toimintoja

Private Pilot -sovellukseen integroituja komponentteja on helppo käskyttää konsolilla. Esimerkiksi Propel-komponentin voi ohjeistaa luomaan tietokanta kuvaustiedostosta, generoimaan model-luokat, luomaan taulut tietokantaan ja täyttämään ne Propel Fixturesia käyttäen esitäytettyjä tietoja.

4.3.2 Private Pilot -palvelun bundle-paketit

Kehitysympäristön rakentamisen ja projektin perustamisen jälkeen Symfony-sovelluksen ohjelmoiminen alkaa tavallisesti bundlen luomisella. Tämä tapahtuu Symfony Consolea käyttäen, mikä on varsin yksinkertainen tehtävä. Bundlen voi luoda myös käsin, mutta Console helpottaa prosessia merkittävästi.

Bundlen voi luoda käsin luomalla kansio-rakenne ja lisäämällä sinne bundlen toiminnan kannalta tärkeät tiedostot. Käsin luotu bundle pitää manuaalisesti rekisteröidä Symfonyn AppKernelin kanssa. Muuten Symfony ei tiedä katsoa bundlen sisältävän kansion sisälle sovelluksen suorituksen aikana.

Luominen Symfony Consolella tapahtuu yhdellä yksinkertaisella komennolla. Se käynnistää ohjatun toiminnon bundlen luomiseksi, jonka aikana annetaan vielä lisämääreitä. Niihin kuuluu muun muassa nimiavaruuden ja asetustiedostojen tiedostomuodon määrittäminen.

Symfonyssä asetustiedostot voidaan tallentaa muun muassa YAML-, XML- ja PHP-muotoisina. Niissä on muitakin kuin pelkästään semanttisia eroja. Esimerkiksi PHP-tyyppisissä asetustiedostoissa on se etu, että niiden väliin voidaan laittaa muutakin PHP-koodia, ja siten kytkeä muiden PHP-sovellusten tuottamaa informaatiota niihin.

Private Pilotin kanssa käytettiin suositeltua YAML-merkintätapaa. Tavallisesti se nopeuttaa kehitystyötä muun muassa siksi, että sitä on helppo lukea ja se sallii rennomman muotoilun verrattuna muihin muotoihin.

Bundlet ovat käytännössä ryhmä PHP-luokkia ja metodeja, CSS-tyylejä, TDD-testejä sekä muita tiedostoja, jotka hoitavat yhtä laajempaa toiminnallista kokonaisuutta. Kansiorakenne bundlen sisällä on joustava, mutta on olemassa hyviä käytäntöjä ja suosituksia siitä, millainen sen tulisi olla.

Monimutkaisemmissa Symfony-sovelluksissa sivuston loogiset kokonaisuudet tavallisesti jaetaan useammaksi bundleksi. Bundle-jako voisi tapahtua esimerkiksi seuraavalla tavalla yrityksen sisäisessä blogi-palvelussa.

Blogin sisällön näyttämisestä yrityksen työntekijöille huolehtisi Company-ViewBlogBundle-paketti. Sen näyttämien merkintöjen ylläpidosta ja tietokantayhteyksistä vastaisi CompanyBlogBundle. Blogi-palvelun hallinnasta vastaisi CompanyAdminBlogBundle ja kirjautumistoiminnoista huolehtisi CompanyLoginBundle.

Esimerkissä Company on nimiavaruus, jota seuraa bundlen nimi, ja lopuksi vielä sana bundle, sillä kaikkien Symfony Bundlejen on päätyttävä sanaan bundle. Muitakin sääntöjä on, kuten kamelinselkätyypitys nimissä, koska bundlen nimen tulee noudattaa PHP-yhteentoimivuusstandardeja. (SensioLabs 2015a.)

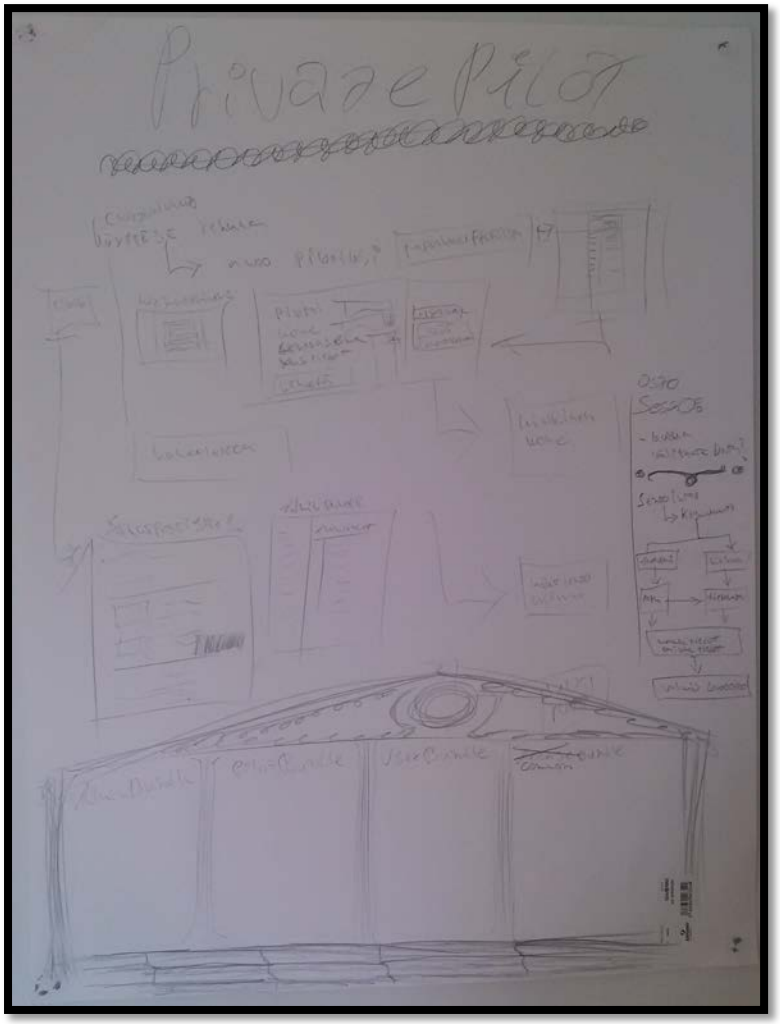
Private Pilot -sovelluksen suunnittelu alkoi listaamalla sovelluksen toiminnan kannalta tärkeät toiminnot ja järjestelemällä ne toisiaan tukeviksi loogisiksi ryhmiksi. Näistä suunnitelluista ryhmistä rakennettiin sovelluksessa käytetyt kaksi Symfony Bundlea. Jokainen bundle tarvitsee oman PHP-nimiavaruuden. Nimiavaruutena toimii esimerkiksi yrityksen tai yhteisön akronyymi. Private Pilot -sovelluksessa nimiavaruudeksi on valittu Lamk.

Private Pilot -projektissa suunniteltiin alun perin neljän bundlen luomista, jotka olivat AdminBundle, UserBundle, PilotBundle, ja CommonBundle. Nämä ovat nähtävissä kuviossa 13. Vain kaksi näistä suunnitelluista bundleista toteutui. Toiminnot, jotka UserBundle olisi sisältänyt, yhdistyivät hyvin varhaisessa vaiheessa kehitystyötä CommonBundle-pakettiin. Tämä iso yhdistynyt bundle kulkee nimellä LamkPrivatePilotBundle. Se ei ole kierrätettävissä uudelleen ilman sen pilkkomista kahdeksi tai kolmeksi pienemmäksi bundle-paketiksi.

LamkPrivatePilotBundle, eli toiselta nimeltään CommonBundle, sisältää pääasiassa kaikki sivuston toiminnot ja enemmänkin kuin mitä sen pitäisi, koska siihen on integroituneena yhtenä möhkäleenä kaikki se, mitä piti tulla UserBundle-, PilotBundle- ja CommonBundle-paketteihin.

Ylläpitopaneelin toiminnot on sijoitettu LamkPrivatePilotAdminBundlen sisälle. Se luotiin, kun CommonBundlessa oli liian paljon sisältöä ja se jakaa muun muassa tietoturvaan liittyvistä pääsyoikeussyistä toiseen bundle-pakettiin. Tämä tehtiin järjestelmän ohjelmakoodin selkeyttämiseksi.

Näin myös varmistettiin, että tavallisilla käyttäjillä ei olisi edes vahingossakaan mahdollisuutta päästä ylläpidollisiin toimintoihin käsiksi. Kaikki AdminBundlen ohjaimet vaativat sisäänkirjautumista ja pääkäyttäjä-tason käyttäjäoikeuksia.



KUVIO 13. Alkuperäinen Private Pilot -suunnitelma

PrivatePilotBundlen ja AdminBundlen välillä on merkittäviä eroja siinä, miten ne toimivat ja minkä tyyppistä toiminallisuutta ne sisältävät. Ne eroavat niin ohjelmointityyleiltään kuin sisäiseltä toiminnaltaan.

Esimerkiksi AdminBundlen kanssa on onnistuttu paremmin noudattamaan ”laihat ohjaimet, lihavat mallit”-nyrkkisääntöä. AdminBundle käyttää myös SensioExtraFrameworkBundlen tarjoamia ominaisuuksia, kuten PHP-annotaatioita ohjaimissa reittien määrittämiseen. PrivatePilotBundle käyttää routes.yml-tiedostoa osoitteiden määrittämiseen.

4.3.3 Reititys Symfony-sovelluksessa

Symfony Routing -komponentti pyrkii tarjoamaan kehittäjille työkalut monimutkaisten reittien kuvaamiseen vaivattomasti. Symfony-sivuston reitit voi määritellä useammalla eri tavalla, mutta oletusarvoisesti reitit kirjoitetaan routing-asetustiedostoon. (SensioLabs 2015k, 55–56.)

Jokaisella bundlella on tavallisesti config-kansio, josta löytyy routing.yml-tiedosto tai sen vastine. Siihen on määriteltynä bundlen tarjoamat reitit sekä reittien määreet. (SensioLabs 2015l, 57; SensioLabs 2015m, 66.)

Reittien määreisiin tavallisesti kuuluvat muun muassa reitin nimi, virtuaalinen polku resurssiin ja viittaus nimiavaruuteen, josta löytyy reitistä huolehtiva ohjain. Kuten kuviosta 14 nähdään, voidaan myös tarkempia määreitä antaa. Esimerkiksi on mahdollista sallia vain GET-pyynnöt tai numeroita on löydyttävä polun jossain kohdassa.

Päätason reitit määritellään app/config/routing.yml-tiedostoon (SensioLabs 2015l, 57). Ne jakaantuvat alaspäin siirryttäessä bundle-kohtaisiksi routing-tiedostoiksi tai annotaatio-tyyppiä edustavaksi resurssiksi ohjainten otsakkeissa.

Reitteihin kohdistuvien pyyntöjen sisäänpääsy- ja ulospääsyoikeuksista vastaa Symfony Firewall. Siihen kuuluu muun muassa pääsyoikeuksien määrittäminen erillisessä security.conf-tiedostossa Symfonyn eri osiin. Sillä ei ole tekemistä tietoliikenneverkojen kanssa. (SensioLabs 2015d, 174.)

Private Pilot -sovellus rakennettiin siten, että se sisältää neljä käyttäjätasoa ja kolme firewall-lohkoa luotuihin route-määrityksiin. Yksi bundle muodostaa oman firewall-lohkonsa, jolla on omat alioikeutensa. Käyttäjätasot ovat anonyymi, rekisteröitynyt käyttäjä, pilotti ja admin.

Symfony Routing -komponentti käyttää hyväkseen Apachen mod-rewriteä. Tästä syystä muun muassa IIS:n käyttäminen Symfonyn kanssa on haastavampaa kuin Apachen kanssa.

```
46 lamk_private_pilot_flight_show:
47     pattern: /flight/{id}/show
48     defaults: { _controller: LamkPrivatePilotBundle:Flight:show }
49     requirements:
50         id: \d+
51         page: \d+
52
53 lamk_private_pilot_flight_update:
54     pattern: /flight/{id}/update
55     defaults: { _controller: LamkPrivatePilotBundle:Flight:update }
56     requirements:
57         id: \d+
58
59 lamk_private_pilot_flight_delete:
60     pattern: /flight/{id}/delete
61     defaults: { _controller: LamkPrivatePilotBundle:Flight:delete }
62     requirements:
63         id: \d+
64
65 # User
66 lamk_private_pilot_user_show:
67     pattern: /user/{id}/show
68     defaults: { _controller: LamkPrivatePilotBundle:User:show }
69     methods: [GET]
70     requirements:
71         id: \d+
72
73 lamk_private_pilot_user_avatar_upload:
74     pattern: /user/settings/avatar
75     defaults: { _controller: LamkPrivatePilotBundle:User:upload }
```

KUVIO 14. Ote PrivatePilotBundlen routing.yml-tiedostosta.

4.3.4 Symfony-annotaatiot

Englannin kielessä sana annotation tarkoittaa mitä tahansa reunahuomautusta, joka luetaan muun tekstinlomassa. Tietojen käsittelytieteessä annotaatiot ovat tapa liittää metadataa, jolla on vaikutusta ohjelman suorituksen aikaiseen toimintaan. Tämän tapaista metadataa voi esimerkiksi olla online-ohje, kommentti tai ohjeita ohjelmakoodin jäsentäjälle. Annotaatiot eivät suoraan vaikuta ohjelman semanttiseen toimintaan, mistä syystä niitä pidetään maagisina kekseinä, joilla piilotetaan niiden alla olevia monimutkaisempia mekanismeja.

Symfony tarjoaa joukon PHP-annotaatioita, joilla tähdätään ohjaimien toiminnan ja sovelluskoodin luettavuuden tehostamiseen. Näistä keskeisimpinä ovat @security-, @template- ja @route-merkinnät, joita käytettiin apuna, kun AdminBundle-pakettia ohjelmoitiin. Nämä merkinnät ovat osa SensioBundleFrameworkExtraBundle-pakettia, joka tulee olla käytössä annotaatioiden toimimiseksi.

Kuten Symfonyn reitityksestä kerrottiin aikaisemmin, Symfonyn polut jäsennetään tyypillisesti erillisiin tiedostoihin, josta Symfony käyttää selvittämään HttpKer-

nelin tarvitsemat tiedot muun muassa reitityksestä ja ohjaimista. Symfonyssä on kuitenkin mahdollista määrittää ohjainten polut suoraan ohjainten annotaatioissa, ja tällä tavalla suoraviivaistaa web-sovelluksen toimintaa ohjelmoijan perspektiivistä käsin. Esimerkiksi @route annotaatiota on käytetty AdminBundle-paketissa siten, että se on lisätty ohjaimen kommentti-lohkoon tarvittavien määreiden kanssa, jolloin erillistä routing-tiedostoa ei ole tarvittu.

Symfonyn käyttämien annotaatioiden tyyli on mallinnettu Java-ohjelmointikielessä olevien annotaatioiden mukaan. PHP-ohjelmointikielessä ei ole sisäänrakennettua tukea Java- ja C#-tyyppisille merkinnöille. Siihen voidaan liittää vain ulkoisilla kirjastoilla hyvin rajallinen tuki annotaatioille. Tähän tarkoitukseen Symfony käyttää Doctrinen annotaatiokirjastoa.

PHP-ohjelmointikielessä annotaatiot merkitään PHPDocBlock-kommenttien sisään. Sellainen edustaa huonoa ohjelmistosuunnittelua, sillä minkään kommentoidun tekstin ei tulisi vaikuttaa ohjelmakoodin loogiseen toimintaan.

Annotaatioita käyttävän PHP-sovelluksen vikojen jäljittäminen on monissa tapauksissa vaikeampaa kommenttien sisältämästä ohjelmakoodin vuoksi. Lisäksi riippuvuus ulkoisesta kolmannen osapuolen kirjastosta tarkoittaa sitä, että merkityt annotaatiot eivät ole universaalisti yhteensopivia keskenään. (Bell 2012.)

Yrityksistä huolimatta virallisesta vakinaistetusta tuesta PHP-annotaatioille ei ole päästy sopuun. Ennen kuin PHP-kehittäjät saavat virallisen annotaatiotuen PHP-kieleen sekasorto PHP-annotaatioiden kanssa jatkuu, mikä osaltaan vaikeuttaa ratkaisun löytymistä. Tämä on merkittävää muun muassa siksi, että suurin osa merkittävimmistä PHP-projekteista käyttää jonkin tyyppistä annotaatiokirjastoa hyväkseen. Näihin lukeutuvat muun muassa phpUnit-, Doctrine-, Zend-, TYPO3-, Symfony2- ja monet muut PHP-kielellä kirjoitetut ohjelmistot. (Blanco & Charron 2010.)

Private Pilot -sovelluksessa PrivatePilotBundle on kirjoitettu käyttämällä Symfonyn varsinaista routes-tiedostoa, johon on määriteltynä hyvinkin tarkasti kaikki polut ohjaimineen. PrivatePilotAdminBundle taas käyttää paljon kuviossa 15 nähtäviä Symfonyn PHP-annotaatioita parantamaan koodin ylläpidettävyyttä. Näin

on toimittu muun muassa siksi, että muodostuisi parempi kuva siitä, mitä etuja ja haittoja annotaatioiden käyttämisellä on.

Hyvänä seikkana annotaatiot, kuten @route, helpottavat ohjelmoijan työtä, kun kaikki ohjaimen toimintaan liittyvä koodi löytyy yhdestä paikasta. AdminBundle käyttää annotaatioita myös turvamääreisiin, mikä niin ikään myös helpotti bundlen kehittämistä. Sen ansiosta monimutkaisempia firewall-ratkaisuja ei tarvittu erikseen määritellä security-asetustiedostoon.

4.3.5 Symfony-ohjaimet

Symfonyssä ohjain (engl. Controller) on tavallisesti PHP-luokka, joka erikoistaa FrameworkBundlen Controller-luokkaa ja jonka ohjelmoija luo käsittelemään HTTP-objekteja. Tavallisesti ohjaimen tehtävä on ottaa sisäänsä Pyyntö-objekti ja käsitellä se Vastaus-objektiksi ohjelmoijan haluamalla tavalla action-funktiossa ja siirtää se eteenpäin Request-Response-mallin mukaisesti. Vastaus-objekti voi sisältää esimerkiksi HTML-sivun, XML-asiakirjan, serialisoidun JSON-taulun tai 404-virheen. (SensioLabs 2015h, 43; SensioLabs 2015i, 44.)

Ohjaimen voi muodostaa Symfonyssä myös ilman FrameworkBundlea. Tallöin käytetään HttpFoundation-komponentin Reponse-luokkaa muodostamaan PHP-luokka (niin sanottu raakile-ohjain), joka sisältää vain absoluuttisen minimimäärän informaatiota Vastaus-objektin muodostamiseksi.

Tavallisesti Symfonyn ohjainten action-funktioissa suoritetaan sivuston toimintaan liittyviä tehtäviä, kuten kuvion 15 kahdessa action-funktiossa on nähtävissä. Näihin lukeutuu esimerkiksi tiedon hakeminen malleilta ja puskeminen näkymälle (nähtävissä kuvion 15 riviltä 29 olevassa indexAction-funktiossa) sekä näkymältä vastaanotetun lomake-objektin käsitteleminen ja vieminen malleille (kuvion 15 rivin 53 addAction-funktio).

Ohjainten tulisi Symfonyn kehittäjien ohjesääntöjen mukaisesti sisältää vain koodia, joka sitoo yhteen sovelluksen eri osat ja ohjaa niiden toimintaa. Nyrkkisääntö on, että ohjaimissa olisi vain viisi muuttujaa, kymmenen tai vähemmän toimintoa, ja jokainen toiminto sisältää vain kaksikymmentä riviä koodia. Liiketoimintalo-

giikka ja muut isommat kokonaisuudet, joilla ei ole ohjaimen tai näkymän toiminnan kannalta merkitystä tulisi siirtää malleihin. Tämä voidaan tiivistää idiomiksi: laihat ohjaimet, lihavat mallit. (SensioLabs 2015j, 19–22.)

```
21 class AirplaneController extends Controller
22 {
23
24     /**
25      * @Route("/admin/airplane/{page}", requirements={"page" = "\d+"}, defaults=
   {"page" = 1}, name="lamk_private_pilot_admin_airplane")
26      * @Security("has_role('ROLE_ADMIN')")
27      * @Template()
28      */
29     public function indexAction($page)
30     {
31         $planes = AirplaneQuery::create()
32             ->orderById()
33             ->paginate($page, $rowsPerPage = 10);
34         $nextPage = $planes->getNextPage();
35         $previousPage = $planes->getPreviousPage();
36         $links = $planes->getLinks(10);
37
38         return array(
39           'planes'       => $planes,
40           'page'         => $page,
41           'links'        => $links,
42           'rowsPerPage'  => $rowsPerPage,
43           'nextpage'     => $nextPage,
44           'previouspage' => $previousPage
45         );
46     }
47
48     /**
49      * @Route("/admin/airplane/add", name="lamk_private_pilot_admin_airplane_add")
50      * @Security("has_role('ROLE_ADMIN')")
51      * @Template()
52      */
53     public function addAction(Request $request)
54     {
55         $airplane = new Airplane();
56         $form = $this->createForm(new AirplaneType(), $airplane);
57
58         if ('POST' === $request->getMethod()) {
59             $form->handleRequest($request);
60
61             if ($form->isValid()) {
62                 $airplane->save();
63
64                 return $this->redirect($this->generateUrl
   ('lamk_private_pilot_admin_airplane', array(
65                         'id' => $airplane->getId()
66                 )));
67             }
68         }
69         return array('form' => $form->createView());
70     }
```

KUVIO 15. Ote PrivatePilotAdminBundlen Symfony-ohjaimesta

Isompia toiminnallisia osia varten ohjain voidaan myös refaktoroida Symfony Service Container -tapahtumiksi. Service-tapahtumat ovat Symfonyssä erityisiä olioita, jotka suorittavat sovelluksen laajuisia pitkälle erikoistuneita yksittäisiä rutiineja tarvittaessa. Niitä käytetään sovelluksessa esimerkiksi lähetettäessä käyttäjälle vahvistusviesti sähköpostilla käyttäjätilin luomisen yhteydessä ja muissa ”kun tämä tapahtuu, tuon pitää tapahtua”-tapauksissa. (Wallsmith 2014; SensioLabs 2015e, 217–218.)

4.3.6 Symfony Form -komponentti

Lomakkeiden käsittelyä varten Symfonyssä on monipuolinen valikoima työkaluja. Näillä työkaluilla on pyritty muun muassa yksinkertaistamaan Web-kehittäjille tavallisesti haastavaa HTML-lomakkeiden käsittelyä ja pyrkimään ehkäisemään tietoturva-ongelmia.

Useimmat näistä työkaluista on koottu Symfony Form -komponentin alaisuuteen, joka on lomakkeiden luomiseen tehty komponentti. Sen avulla voidaan kirjoittaa PHP-luokkia, joista mekaanisesti muodostetaan HTML-lomakkeita. Lomakkeet luodaan Symfony FormBuilder-objektia apuna käyttäen. Sitä käyttäen informaatio myös sidotaan halutun mallin kuten esimerkiksi Propel Model tai Doctrine Entity -luokan kanssa.

Private Pilot -sovelluksessa käytetty Propel sisältää omat liitoksensa Symfony Form -komponentille. Tämä mahdollisti sen, että sovelluksessa käytettyjen lomakkeiden, kuten kuviossa 16 nähtävä OrderDetails-luokka, luomista voitiin automatisoida käyttäen Propelin Symfony Console -komentoja. Näitä mekaanisesti luotuja pohjia ei tule käyttää sellaisenaan, sillä niistä puuttuvat esimerkiksi kaikki rajoitukset ja muut kenttien määreet. Lisäksi niissä saattaa tulla ylimääräisiä lomake-osia, joita ei haluta mukaan varsinaiseen lomakkeeseen. Pohjia on siivottava ja muovata sisältämään muun muassa raja-arvoja ja tiukemmin määriteltyjä data-tyyppejä ennen käyttöönottoa.

Lisäksi lomakkeet purettiin useimmissa tapauksissa pienemmiksi osiksi Twig-mallinteiden väliin. Tämä johtui tarpeesta muovata yksityiskohtaisemmin lomakkeiden CSS-tyylimäärityksiä, joihin oli aluksi vaikeata päästä käsiksi johtuen muun muassa Bootstrapin kanssa kohdatuista ongelmista. Myöskään sovelluksen visuaalisen puolen tarpeista ei ollut riittävästi tietoa. Tämän vuoksi liikkumavaraa jätettiin visuaalisen ilmeen myöhempää rakentamista varten jo kehityksen aikana.

Lomakkeen voi purkaa hyvinkin pieniksi osiksi, joiden tyyliä ja käytöstä voi muokata täysin vapaasti. Tästä huolimatta lomakkeita ei tavallisesti kuulu hajottaa osiin ja kasata niitä uudelleen esimerkiksi Twig-mallineessa. CSS-mallinteiden määrittäminen Twigille riittää kertomaan Symfonylle miten ja millaista teemaa ja

tyyliä lomakkeiden tulisi käyttää. Tähän voidaan käyttää apuluokkia, jotka poimivat tarvittavat tiedot esimerkiksi CSS-kehyksestä.

Usein lomakkeet turvataan suojauksella, jolla pyritään ehkäisemään CSRF-hyökkäyksiä. Lomakkeisiin generoidaan käyttäjältä piilossa oleva kenttä, joka sisältää yksilöllisen avaimen. Lomakkeen tietojen väärentämistä voidaan ehkäistä tällä tavoin.

```
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

/**
 * Description of DetailType
 *
 * @author Antti Aspinen <antti.aspinen@student.lamk.fi>
 */
class OrderDetailType extends BaseAbstractType
{

    /**
     *  {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add('numberofreservedseats', 'choice', array(
          'label'       => 'How many seats do you want to book?',
          'empty_value' => 'Choose amount of seats',
          'required'    => true,
          'choices'     => array(
            1 => 'One',
            2 => 'Two',
            3 => 'Three',
            4 => 'Four',
            5 => 'Five',
          ),
          'multiple'    => false,));
        $builder->add('idorderpaymentmethod', 'model', array(
          'label'       => 'What payment method do you wish to use?',
          'empty_value' => 'Choose a payment method',
          'required'    => true,
          'class'       => 'Lamk\PrivatePilotBundle\Model\PaymentMethod',
          'property'    => 'name',
          'multiple'    => false,
        ));
        $builder->add('Submit order', 'submit');
    }

    public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
        $resolver->setDefaults(array(
          'data_class' => 'Lamk\PrivatePilotBundle\Model\ReceivedOrder',
        ));
    }

    public function getName()
    {
        return 'orderdetails';
    }
}
```

KUVIO 16. Symfony Form -lomakeluokka

On huomion arvoista, että lomakkeet voidaan muodostaa ohjaimen sisällä. Tosin tavallisesti ne vain esitellään ohjaimessa. Kun lomake on esitelty, siitä muodoste-

taan erityinen näkymä-objekti createView-metodilla. Se välitetään ohjaimen läpi Twig-mallinteelle sijoitusta varten.

Twigillä merkitään lomakkeen paikka ja tyyli tiedostossa käyttäen Twig-muuttujaa {{ form() }}. Tämä muuttuja voidaan pilkkoa esimerkiksi {{ form_start() }}, {{ form_error() }} ja {{ form_end() }} osiksi, jotka puolestaan voidaan pilkkoa vielä pienemmiksi paloiksi. Kaikille näille osille voi myös antaa hyvinkin tarkasti omat määritteensä ja muodostaa niistä esimerkiksi lomake-ryhmiä, kuten Private Pilot -sovelluksen lentojen luonti lomakkeessa tehtiin.

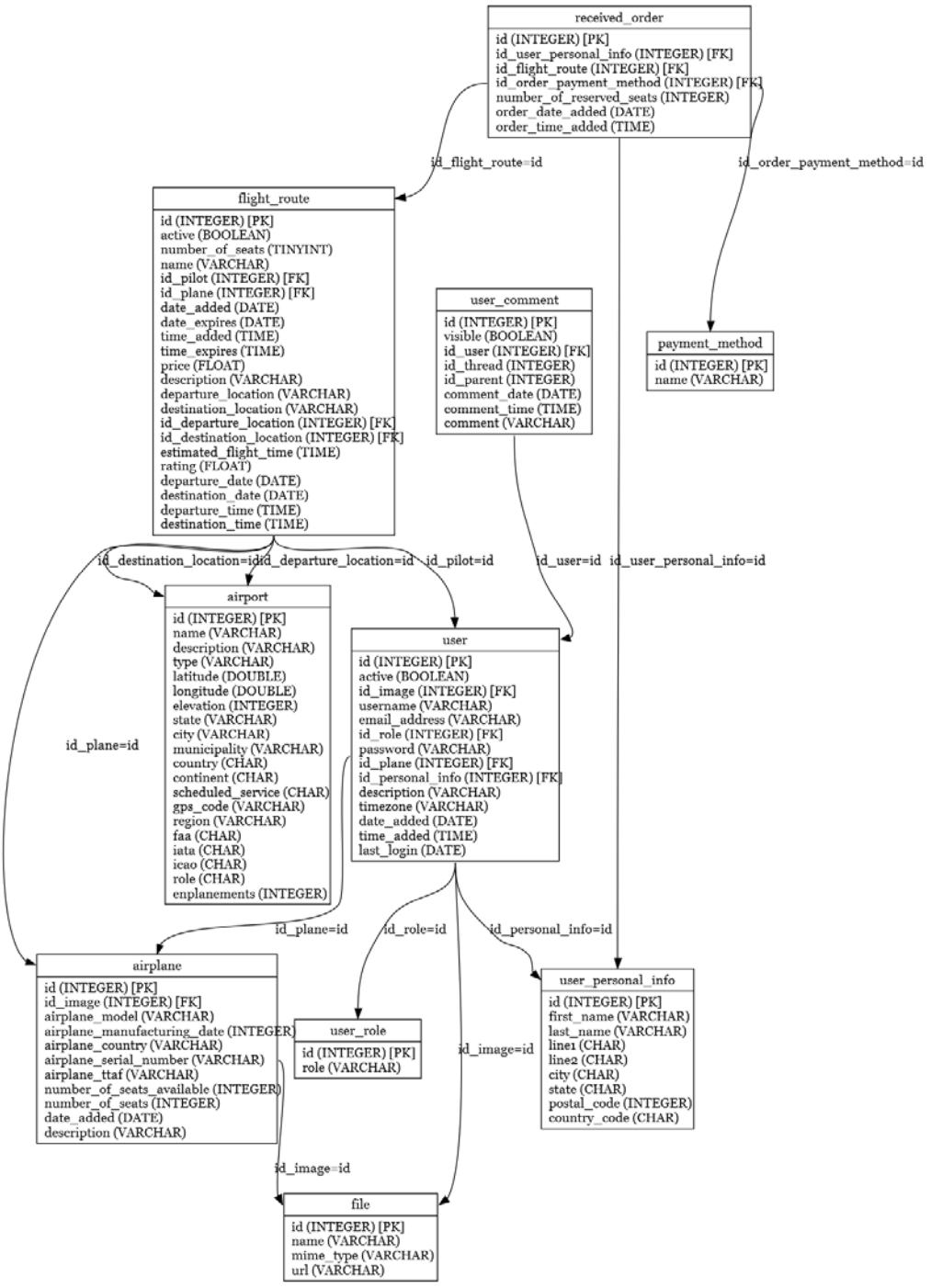
4.4 Propel-kehitys

Private Pilot -sovelluksen tietokannan suunnittelu alkoi siitä, millaista tietoa tavallisesti tämän tapaisissa palveluissa tarvitaan. Sitä varten laadittiin kevyt UML-tyyppinen sketsi tietokannan rakenteesta paperille. Sitä siistittiin hieman Debian-jakelun mukana tulevalla Umbrello-sovelluksella.

Näin saatiin suunnitelmallisuutta tietokannan rakentamiseen. Tämän UML-mallin pohjalta kirjoitettiin schema.xml-tietokantatiedosto. Propelin schema.xml-tiedostoa muokattiin ja parannettiin sitä mukaan kuin projekti edistyi. Kuviosta 17 on nähtävissä, minkä kaltainen tietokannasta lopulta kehkeytyi 1.0 versioon mennessä.

Tietokannan kehityksessä ei heti alkuun huomioitu esimerkiksi sitä, kuinka paljon kentät vaativat tilaa. Tästä seurasi se, että esimerkiksi etu- ja sukunimitauluissa saattoi olla rajattoman merkkimäärän kokoisia tauluja. Näitä optimoitiin myöhemmin järkevimmiksi ja pienemmiksi. Se tehosti tietokannasta tietojen hakemista ja käyttöä.

Päivämääriä varten voidaan käyttää kenttiä, joissa päivämäärät tallennetaan leimoina, joita haetaan tietokannasta. Vaihtoehtoinen tapa on tehdä soluja, joihin tallennetaan aika esimerkiksi sekunteina vuodesta 1970 eteenpäin. Kummassakin mallissa on hyviä ja huonoja puolia.



KUVIO 17. Propelin luoma GraphViz-diagrammi tietokannan lopullisesta versiosta

Leimoja varten joudutaan rakentamaan käsittelijät, jotka muotoilevat tietokannassa olevat aikaleimat oikein. Private Pilot -sovelluksessa aluksi käytettiin sekunteina vuodesta 1970 eteenpäin -mallia, mutta sovelluskehityksen edetessä tehtiin vaihto

aikaleimoihin. Tämä tehtiin, koska se yksinkertaisti haastavaksi koettua ajan käsittelyä sovelluksessa.

Sovelluksen tulee toimia useilla aikavyöhykkeillä ja vastaamaan asiakkaiden tarpeisiin. Järjestelmän ylläpitäjien tulee myös voida hallita järjestelmää, jonka käyttäjät sijaitsevat useilla ei aikavyöhykkeillä.

4.4.1 Propel-luokat

Vaikka Propel-luokan voi käsin kirjoittaa, ne tavallisesti luodaan automaattisesti Symfony-sovelluksessa konsolia käyttäen. Symfony luo luokille Model-kansion sen bundlen sisälle, jonka luokista on kyse. Tämän jälkeen niitä voi erikoistaa tarvittaessa.

Symfony Security -komponentin arkkitehtuurissa on määritelty vakionimet joillekin tunnetuimmille määreille. Esimerkiksi käyttäjänimi-kentälle on olemassa User, salasana-kentälle on Password ja niin edespäin. Propel yrittää automaattisesti kytkeytyä näihin sillä oletuksella, että sovelluskehittäjä on käyttänyt näitä nimiä Propelin skeema-tiedostossa ja sallii Propelin automaattisesti kytkeytyä niihin.

Jos jokin tarpeellinen kenttä kulkee tietokannassa nimellä, johon Propel ei osaa kiinnittäytyä, voidaan muokata Propelin automaattisesti luomaa mallia ja kertoa siinä, mihin sarakkeeseen informaation kuuluu mennä. Private Pilot -projektissa tätä käytettiin implementoimalla Securityä varten eraseCredentials- ja getRoles-funktiot User-mallissa.

Jos tietokantaan on tallennettuna esimerkiksi sähköposti-kenttä, joka toimii käyttäjien nimenä, niin silloin pitää ohjelmoijan käsin käydä määrittämässä Propelin Symfonylle generoimissa luokissa, miten näitä määreitä käsitellään ja haetaan. Tämä tapahtuu implementoimalla Propelin mekaanisesti luoma malli-luokka, kuten Private Pilot -sovelluksessa tehtiin User-luokan kanssa.

4.4.2 Tietojen hakeminen Propelia käyttäen

Private Pilot -sovelluksessa jokaisella sivulla käsitelty dynaaminen informaatio haetaan ja tallennetaan ohjaimessa Propelin rajapintaa käyttäen. Se ohjataan suoraan objektina Symfonyssä näkymästä vastaavalle sivulle Twigin käsiteltäväksi.

Tietojen hakeminen tapahtuu Propelin mekaanisesti luomia Query-objekteja käyttäen. Esimerkiksi tietokannan user-taulusta Propel luo userQuery-objektin, jonka create()-tehdasmetodilla muodostetaan haku. Tämän jälkeen kutsumalla esimerkiksi findPK()-metodia haussa voidaan hakea primääriavaimeen perustuen rivi Kirjat-taulusta. FindPK()-metodi on erikoistapaus, sillä se ei tarvitse niin sanottua terminaattori-metodia hakulausekkeen päätteeksi.

Haettaessa muuta informaatiota tai sen osia käytetään joukkoa mekaanisesti luotuja get-metodeita. Tallennettaessa informaatiota käytetään joukkoa set-metodeita. Nämä Propel-lausekkeet päättyvät tavallisesti esimerkiksi find()-, save()- tai update()-terminaattorimetodiin, riippuen siitä millaista transaktiota ollaan suorittamassa.

Luotuun hakuun voidaan lisätä reunaehtoja määrittämällä niitä suoraan hakuun, mutta joskus tämä ei riitä. Silloin voidaan luoda Propelin version 1.4 tyyppinen vanhahtava Criteria-objekti, jolla hausta saadaan haluttu informaatio käyttäen edistyneempiä ehtomäärityksiä.

Propel ei varsinaisesti aseta rajoituksia siihen, miten ja millaista tietoa tietokannasta voidaan hakea. Se kykenee ottamaan vastaan suoraan vapaasti muovattuja PDO-lauseopilla kirjoitettuja SQL-komentoja ohjaimessa. Tämä ei kuitenkaan ole suositeltavaa kuin eritystapauksissa. Lisäksi SQL-koodin käyttöön liittyy myös jotain sääntöjä, joita pitää noudattaa sitä käytettäessä.

Mahdollisuus tähän on olemassa tilanteisiin, joissa Propelin automaattisesti luomat metodit eivät riitä kuvaamaan monimutkaisempaa hakua tai hakua, joka on yksinkertaisempaa kirjoittaa SQL:nä. Myös tilanteissa, joissa haussa käytettyjen Propelin metodien määrän vuoksi suorituskyky kärsii tai jos Propelin metodeissa ilmenee jotain vikaa, voidaan SQL:ää käyttää. Esimerkiksi Private Pilot -sovelluksessa oli muutaman kuukauden ajan vika, jossa SQL-koodilla paikattiin

Propelin ensimmäisen version puutetta eräässä tietojen hakemiseen liittyvässä skenaariossa.

Propel pystyy suoraan ajamaan POST-metodilla vastaanotetun isvalid()-metodilla tarkistetun lomakkeen päälle usein suoraan save()- tai update()-metodin ilman objektin purkamista erillisiksi set-lausekkeiksi. Purkaminen on myös mahdollista, jos siihen on tarvetta.

4.5 Käyttöliittymän kehitys

Käyttöliittymän suunnittelu lähti liikkeelle siitä, että Private Pilot -palvelu on tulossa Yhdysvaltojen markkinoille. Länsimaissa kuten Yhdysvalloissa ihmiset lukevat ylhäältä alas ja vasemmalta oikealle. Tämä vaikutti siihen, miten käyttöliittymän elementit on sijoitettu.

Private Pilot -sovelluksen käyttöliittymä on laadittu siten, että käyttäjän kannalta mielenkiintoinen informaatio on pyritty esittämään keskellä näyttöä tekstin kulkiessa ylhäältä alas ja vasemmalta oikealle. Jokainen valinta on sijoitettu mahdollisimman paljon ylätasoista alkaen matalammille tasoille siirtymään vinosti vasemmasta yläreunasta oikealle alas.

Käyttöliittymän suunnitteluvalinnoilla on pyritty vähentämään hiiren kulkemaa liikettä ja siten tehostamaan sivuston käyttöä Fitts'-lakiin perustuen. Fitts'-laista voidaan johtaa, että mitä vähemmän hiiri liikkuu ruudulla matkaa, sitä nopeammin käyttäjä voi tehdä töitä. (Haixia 2002; Hale 2007.)

Käyttöliittymän kulkiessa oikealle merkitykseltään vähiten tärkeät aktiiviset elementit on sijoitettu aivan keskelle. Näin lukeutuvat toiminnot, joita tavallisesti käyttäjät eivät tarvitse. Aivan oikeassa laidassa ovat erotin välin jälkeen sijoitettuna elementit, jotka vaativat käyttäjältä keskeisen osan sisällön lukemisen ymmärtämistä, kuten esimerkiksi lennon tilauspainike.

Värikoodausta olisi tarvittu, mutta käyttöliittymän visuaalisen designin jäädessä keskeneräiseksi siihen ei paneuduttu asian edellyttämällä tavalla. Tästä johtuen

esimerkiksi käyttäjäkokemuksen suunnittelu on jäänyt keskeneräiseksi, vaikka siihen liittyvä pohjatyö on tehtynä ja valmis käyttöä varten.

Yleisesti käyttöliittymänkehityksessä pyrittiin hyödyntämään yleisiä ohjesääntöjä, kuten esimerkiksi Jakob Nielsenin kymmentä periaatetta. Näihin periaatteisiin kuuluvat muun muassa käyttäjien ymmärtämän termistön hyödyntäminen ja käyttäjien muistikuorman vähentäminen (Nielsen 1993, 20). Palvelun käyttöliittymää yritettiin rakentaa siten, että se olisi helppo ymmärtää, nopea käyttää ja tarjoaisi käyttäjille esteettömästi informaatiota.

4.6 Bootstrap Symfony-sovelluksessa

Symfony-sovelluksessa käyttäjän selaimeen lähetettäviä resursseja hallitaan Assetic-sisällönhallintakomponentilla, jonka keskeisenä tehtävänä on erotella ne sovelluksen muista resursseista. Käyttäjän selaimeen lähetettäviä resursseja ovat esimerkiksi CSS- ja Javascript-koodi sekä kuva-, ääni- ja video-tiedostot. (SensioLabs 2015b, 7–8.)

Private Pilot -sovelluksessa käytetään Bootstrapistä erityistä Braincrafted-BoostrapBundle-versiota. Verrattuna Bootstrap-projektin varsinaiseen jakeluun on Braincrafted-jakelun erityisominaisuutena mahdollisuus kytkeytyä Symfonyn rajapintojen, Asseticin ja Twigin kanssa yhteen. Se voidaan myös määrittää toimimaan laajennusosana Twigin käyttöliittymän tyylimäärityksille ja siten vaikuttaa yhdestä paikasta esimerkiksi kaikkien sovelluksen lomakkeiden ja malline-sivujen ulkoasuun ja toimintaan.

Se mahdollisti muun muassa CSS-koodin automaattisen kääntämisen muun muassa SASS- ja LESS-menetelmillä Assetic-komponenttia käyttämällä. Tämä tekee tyylikoodista ylläpidettävämpää, laajennettavampaa ja monipuolisempaa. Private Pilot -projektissa käytettiin node.js-pohjaista LESS-menetelmää esikäsittelemään CSS-tyylikoodia.

4.7 Symfony Service -tapahtumat

Symfonyssä voi luoda Service Containerilla tapahtumia, jotka suoritetaan aina kun määritetyt ehdot täyttyvät. Tämän avulla voi esimerkiksi Twigiä pyytää tekemään jokin toiminto, joka laukeaa asetettujen ehtojen perusteella. Esimerkiksi Private Pilot -palvelussa, kun käyttäjä kirjautuu järjestelmään, haetaan tietokannasta käyttäjän aikavyöhyke ja se asetetaan Twigin asetuksissa käyttäjän session aikavyöhykkeeksi. Ottaen huomioon, miten Private Pilot on tulossa Yhdysvaltoihin palveluksi, tämä ominaisuus oli hyvin tärkeä saada osaksi Private Pilotin toimivaa demoa. Se mahdollistaa käyttäjien nähdä kiinnostavien lentojen lähtöajat ja lentäjien ilmoittamat saapumisaikojen arviot oman aikavyöhykkeensä ajassa.

Yksinkertaistettuna Service on Symfonyssä PHP-objekti, jolla on jokin yksi tarkasti rajattu tehtävä. Service-tapahtumat ovat kuitenkin laajakokonaisuus Symfonyssä, joiden parissa voi kuluttaa helposti paljon aikaa. Niillä on muun muassa mahdollista laajentaa Symfonyn toimintaa ja muovata sitä ohjelmoijan haluamalla tavalla. Tavallinen käyttötapaus on esimerkiksi muodostaa Service, joka lähettää jokaisen uuden käyttäjän rekisteröitymisen yhteydessä heille sähköpostiosoitteen vahvistamisesta viestin.

4.8 Private Pilot versio 1.0

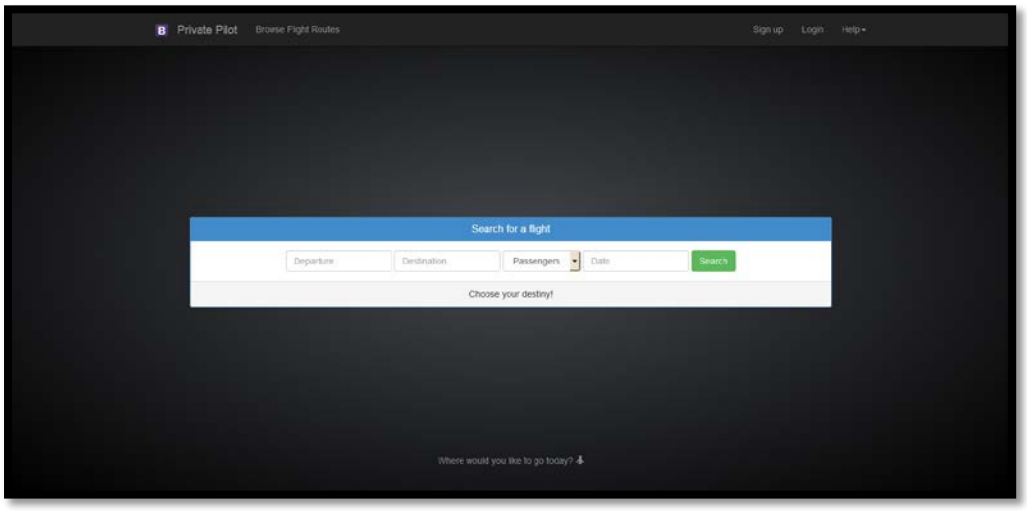
Kun sovellus saavutti toiminnallisen tason, jossa kaikki keskeiset tavoitteet oli saavutettu, se julistettiin versioksi 1.0 ja se ladattiin verkkoon WWW-palveluksi. Siihen tehtiin alkuvalmistelut ja muutamia testilentoja lisättiin loppukäyttäjän roolia esittäviä mahdollisia testikäyttäjiä varten.

Kun loppukäyttäjä ensimmäistä kertaa avaa Private Pilot -sovelluksen hänen eteensä avautuu kuvion 18 näköinen haku ja tietosivu. Sivu on suunniteltu ohjaamaan käyttäjän silmät kiinnittämään huomio ensin haku ruutuun. Tähän käytetään värejä ja sisällönkeskittämistä.

Muita elementtejä etusivulla ovat linkit palveluun rekisteröitymiseksi, kirjautumiseksi, lentojen selaamiseksi listana, yms. Näiden elementtien sijoittelussa on pyritty huomioimaan olemassa olevat käytännöt, jotta uusien käyttäjien olisi

mahdollista käyttää aikaisempia kokemuksiaan muista vastaavista sivustoista hyödyksi.

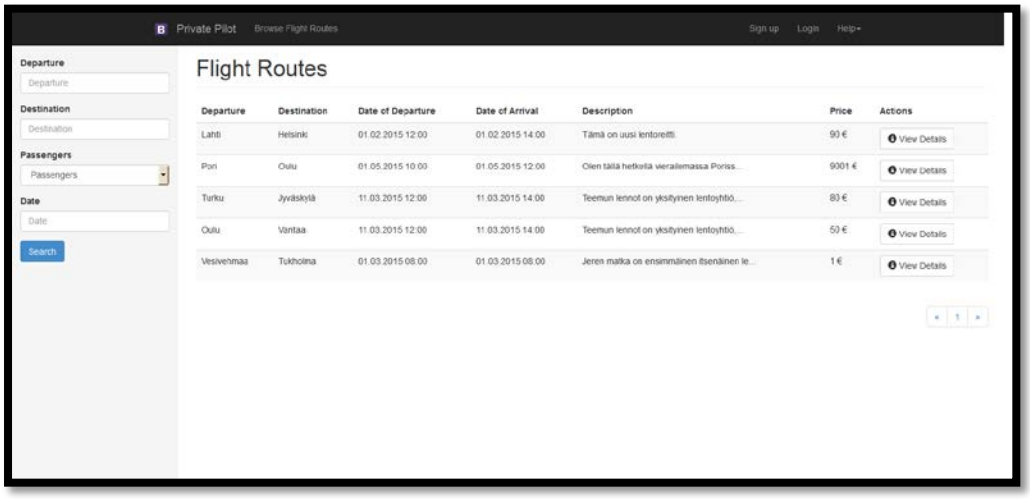
Hakulomakkeeseen syötetään haluttu lähtöpaikka, määränpää, tarvittavien paikkojen määrä sekä päivämäärä. Syötettäessä lähtöpaikkaa ja määränpäätä järjestelmä kyselee taustalla Private Pilot -palvelun JSON-rajapinnalta aktiivisena olevia paikkoja, joista lähtee lentoja ja automaattisesti täydentää käyttäjän syöttämiä tietoja tarvittaessa. Automaattinen kenttien täydennys JSON-tietolähteestä ja päivämäärän valitsin ovat JQueryUI-kirjaston standardi komponentteja.



KUVIO 18. Private Pilot -palvelun etusivu

Käyttäjä voi joko hakea tai selata järjestelmään kirjattuja lentoja. Muiden käyttäjien tietojen näkemiseen tarvitaan kirjautuminen ja käyttäjä voi halutessaan luoda käyttäjätunnuksen ja kirjautua sisään.

Käyttäjät voivat tilata lentoja kirjautumatta järjestelmään, mutta sitten heitä pyydetään täyttämään ylimääräinen lomake omista tiedoistaan. Nämä tiedot tallennetaan tietokantaan pysyvästi, mutta käyttäjät eivät voi ilman tunnusta tarkistella sitten omia tietojaan. Sähköpostiosoite ja varaustunnus liitetään mukaan, jos käyttäjä haluaa myöhemmin luoda tunnuksen ja liittää aiemmat lentonsa omaan tunnukseensa.



KUVIO 19. Saatavilla olevien ja toistuvien lentojen näkymä

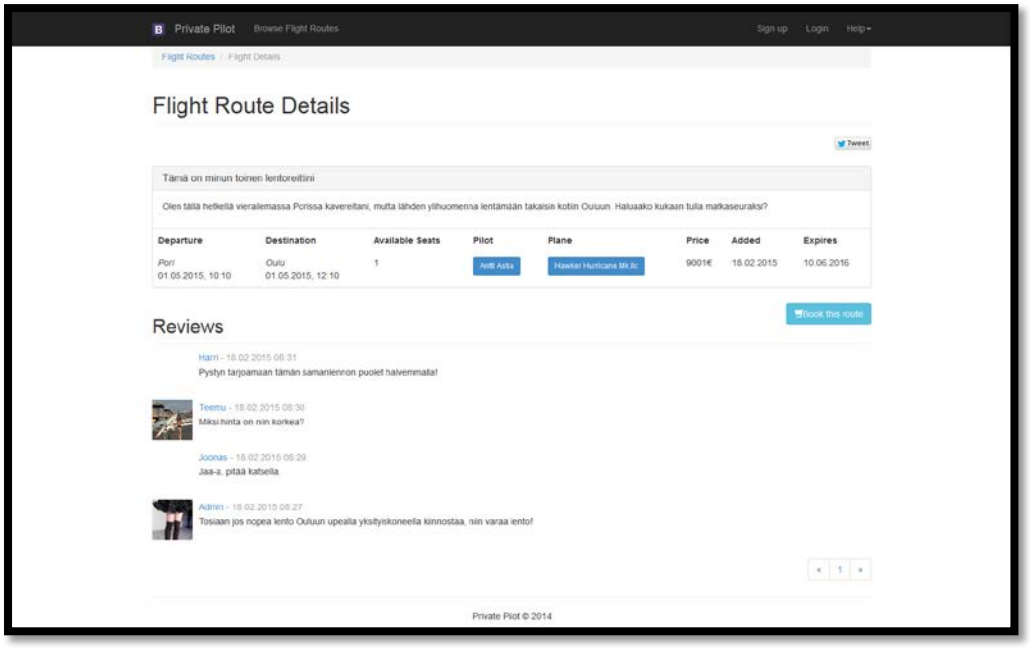
Tyypillisesti käyttäjät katsovat verkkosivuja F-kirjaimen muotoisena alueena (Kadavy 2011a, 135–136). Tätä tietoa käytettiin hyväksi muun muassa lentoreittien selausnäkymää ja lentoreitin tietonäkymää suunniteltaessa.

Lentojen selausnäkymässä näkyy hakuruutu sivupaneelina vasemmassa laidassa. Symfony mahdollistaa lomakkeiden kierrättämisen, joten kyseessä on sama lomake kuin etusivulla, mutta uudelleen tyyliteltynä. Informaatio levittäytyy oikealle kuljettaessa listaukseksi. Visuaalista hierarkiaa on tuettu muun muassa käyttämällä eriväritettyjä rivejä, jotka helpottavat käyttäjää lukemaan taulukosta tarvitsemaansa informaatiota (Kadavy 2011b, 170–171). Listassa on jokaisen lennon kohdalla painike, joka vie sivulle, jossa kerrotaan lisää informaatiota lennosta ja sen tilaus mahdollisuuksista.

Visuaalisesti Private Pilot -palvelun käyttöliittymä ei valmistunut. Tästä keskeneräisyydestä johtuen jotkin kuvioissa 18–25 näkyvät asiat tulevat muuttumaan. Esimerkiksi kuvion 19 lentojen selausnäkymän taulukko-tyyppinen lista ehkä palvelisi asiakkaita paremmin, jos se olisi tyyliteltyjen info-laatikoiden lista. Sellaisia käytetään esimerkiksi internethuutokauppa eBayssä esineiden tarkempien tietojen näyttämiseen suoraan hakutuloksissa. Käyttöliittymän elementtien järjestys ei kuitenkaan tulisi oleellisesti muuttumaan, jos Private Pilotin käyttöliittymä kehitetään loppuun asti.

Käyttäjät voivat katsoa toistensa julkisia profiileja. Tämä oli sikäli tärkeää, että palvelua käyttävien asiakkaiden on voitava nähdä haluamansa lentoa lentävän

pilotin saamia arvosanoja. Tämä auttaa asiakkaita etsimään parhaan pilotin kyseiselle reitille ja myös kannustaa palvelua käyttäviä yksityislentäjiä tarjoamaan parasta mahdollista palvelua asiakkailleen.

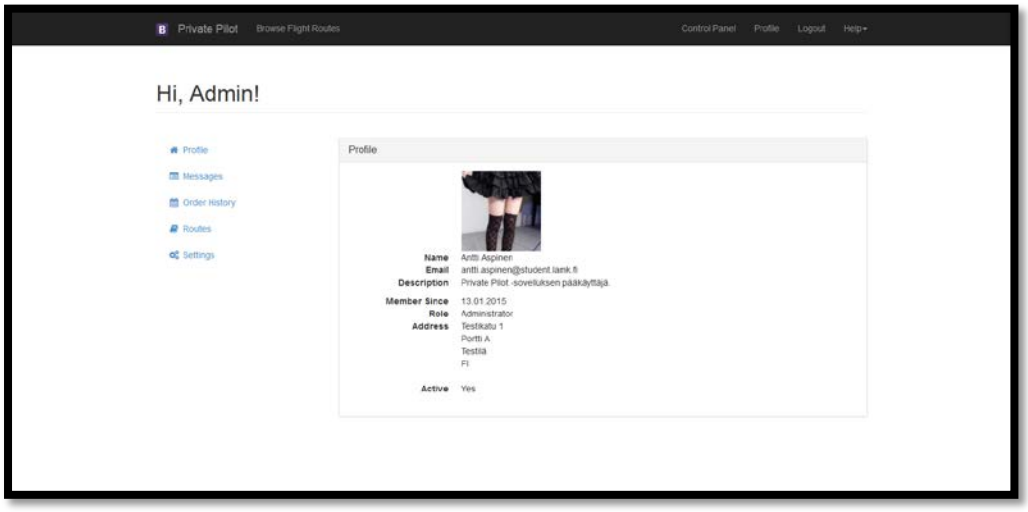


KUVIO 20. Lennon tietonäkymä

Julkinen profiili ja käyttäjien oma henkilökohtainen profiili näyttävät eri määrän ja erilaisia tietoja käyttäjästä. Tällä on tarkoitus muun muassa suojella käyttäjien yksityisyyttä. Julkinen profiili ei suoraan kerro tavallisen käyttäjän nimeä ja tarjoaa nimimerkin. Kun asiakas tilaa lennon, hän saa varsinaisesti sitten pilotin palveluun rekisteröimän nimen, osoitteen, lentokentän tiedot, yms.

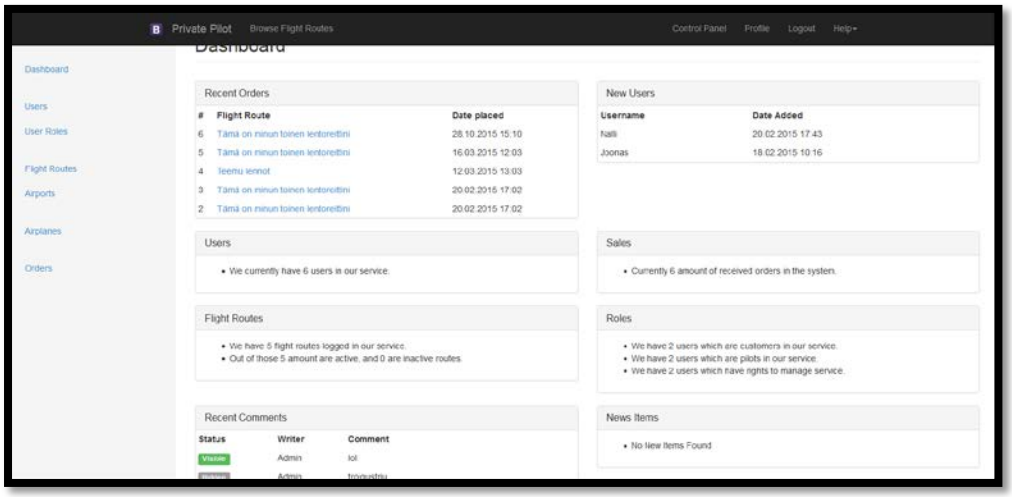
Käyttäjien omassa profiilissa voidaan hallita perustietoja kuten nimeä ja yhteystietoja. Tavalliset profiili toiminnot kuten avatarin asettaminen ja vaihtaminen, henkilötiedot, jne. ovat myös mahdollisia.

Käyttäjä voi voi halutessaan täyttää lomakkeen pilotiksi hakemista varten järjestelmään. Ylläpitäjät tarkistavat tarvittavat tiedot ja luvat pilotista ja myöntävät pilotille oikeuden toimia pilottina järjestelmässä.



KUVIO 21. Käyttäjäprofiili

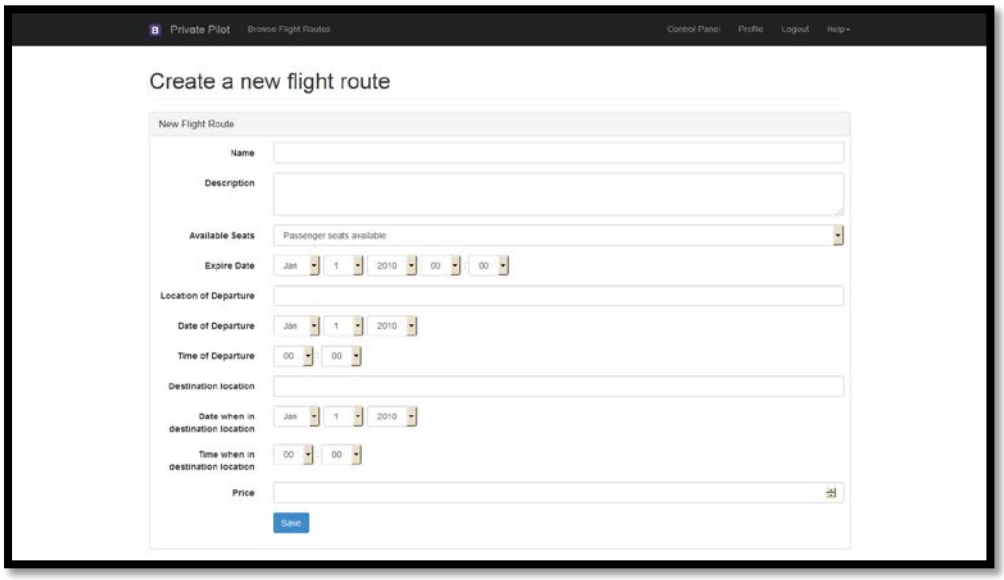
Profiilissa näkyy tilatut lennot, ja jos käyttäjä on pilotti, niin myös lennot, joita hän tarjoaa ja lentää säännöllisin väliajoin. Lentoja voi luoda siten, että se alkaa ja kestää vain kerran, tai siten, että lento kertautuu kyseisenä ajankohtana useamman kerran. Esimerkiksi on mahdollista asettaa järjestelmään päivämäärä, jolloin lento tapahtuu kerran viikossa tai useamman kerran muutaman kuukauden välein. Versiossa 1.0 on kuitenkin tekninen ongelma, jonka vuoksi vain lennon kertautumisen aika on päällä.



KUVIO 22. Hallintapaneeli

Hallintapaneelista, joka on kirjoitettu käyttämättä Sonata-projektin työkaluja, löytyy vähimmäisvaatimukset täyttävät admin-toiminnot. Tähän kuuluu esimerkiksi käyttäjien hallinta ja lentokoneiden hallinta.

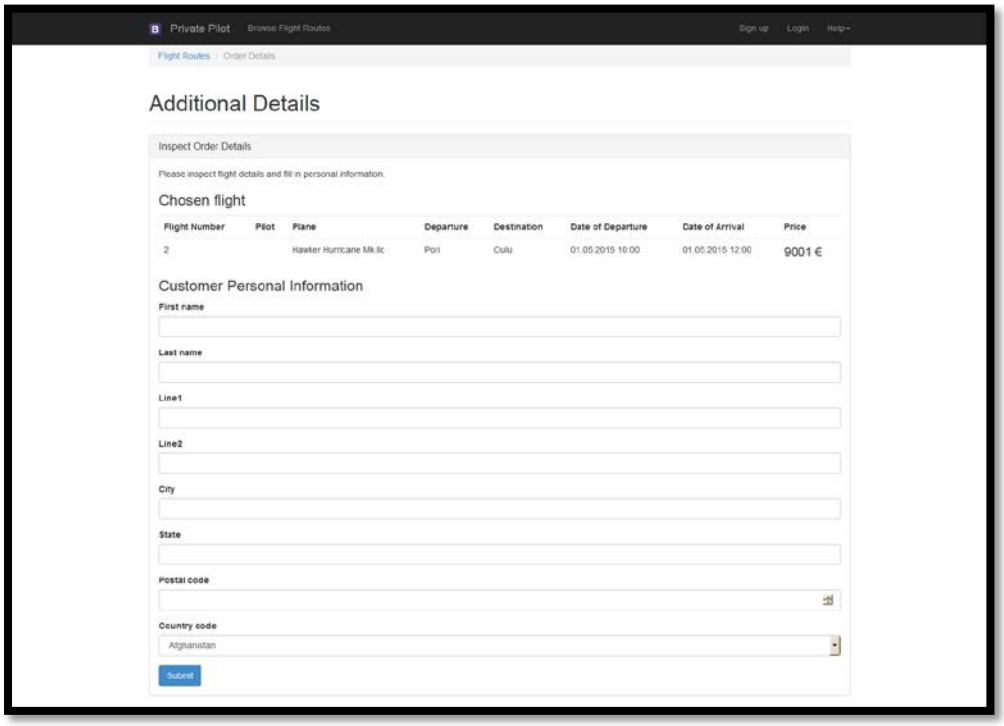
Lentoreittien hallintaa ei ole kytketty erillisenä osana hallintapaneeliin, koska lentokenttien kytkeminen koneisiin ja lennettäviin reitteihin on kesken vielä versiossa 1.0. Lentoreittejä hallitaan versiossa 1.0 CommonBundlen puolella yksityislentäjien käyttöön tarkoitetuilla toiminnoilla. Admin-käyttäjät eivät voi muokata niitä käyttämällä muiden käyttäjien lentoja.



KUVIO 23. Uuden lentoreitin luominen

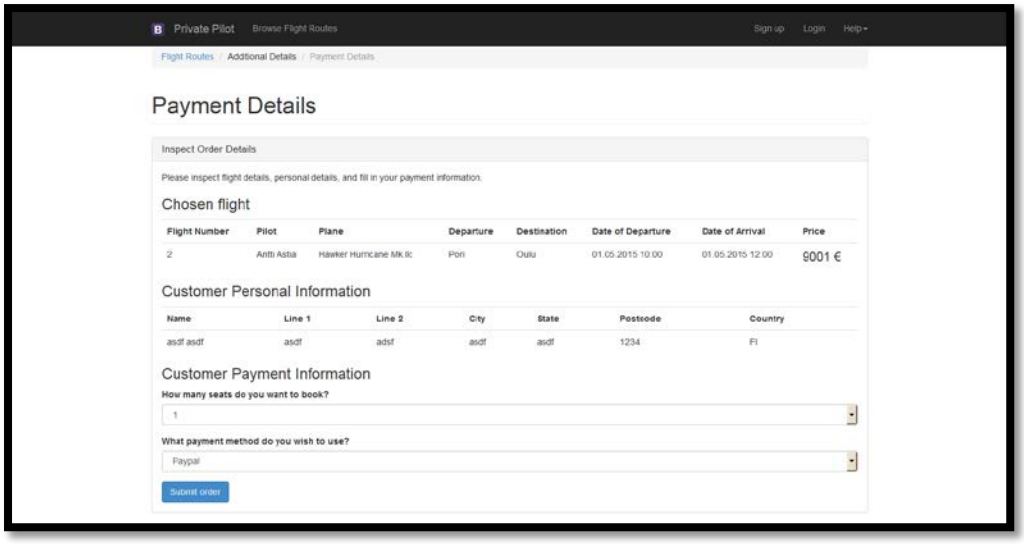
Lentoreittien luominen on suhteellisen suoraviivainen tapahtuma. Se on yksi lomake, johon täytetään kaikki oleelliset tiedot lennosta ja järjestelmä huolehtii lopusta.

Private Pilot versio 1.0 olettaa järjestelmänä, että pilotti on täyttänyt oman lentokoneensa tiedot. Jos näin ei ole, palvelu ei anna pilotin luoda lentoja ennen kuin yksityislentäjä on täyttänyt tiedot koneestaan, kotilentokentästään ja palvelun ylläpitäjät tarkistaneet ja hyväksyneet nämä tiedot.



KUVIO 24. Tunnuksettomien käyttäjien tietolomake

Tunnuksettomat asiakkaat voivat myös tilata lentoja. Mutta he joutuvat täyttämään ylimääräisen lomakkeen, jossa he kertovat transaktion kannalta henkilötietojaan. Nämä säilötään järjestelmään.



KUVIO 25. Lentojen tilauslomake

Jos käyttäjä on kirjautunut järjestelmään, hän pääsee suoraan lennon tilauslomakkeeseen. Private Pilot -palvelu olettaa, että käyttäjä on syöttänyt henkilötietonsa järjestelmään jo kerran. Jos näin ei ole, järjestelmä huomauttaa käyttäjää ja tämän jälkeen ohjaa käyttäjän kirjoittamaan henkilötiedot profiilin hallintapaneeliin.

5 YHTEENVETO

Private Pilot -projekti oli osa Lahti Demo Room -hanketta. Projektissa tavoitteena oli kehittää toimiva demo-versio web-sovelluksesta. Se toimisi verkossa WWW-palveluna, jossa yksityislentäjät ja yksityislentoja haluavien asiakkaiden tarpeet kohtaavat.

Palvelu jakaa monia samankaltaisuuksia ja toimintatapoja vastaavien matkailupalveluiden kanssa. Demo-version suunnittelussa tosin pyrittiin keskittymään vain niihin oleellisiin ominaisuuksiin, joita yksityislentäjien palveluksia tarjoava sivusto tarvitsee vähintään toimiakseen.

Näihin toimintoihin lukeutuivat muun muassa asiakastietojen hallinta, ostotapahtumat sekä kyky lisätä ja poistaa lentoreittejä järjestelmästä. Palvelu on tarkoitettu Yhdysvaltojen markkinoille, minkä vuoksi on huomioitava myös muun muassa useiden aikavyöhykkeiden olemassaolo ja paikallinen lainsäädäntö.

Private Pilot -sovellus toteutui toiminnallisen puolen osalta pääosin. Liiketalouden opiskelijat suorittivat oman tutkimuksensa. Tekniikan opiskelija suunnitteli ja toteutti käyttöliittymän sekä rakensi toimivan demo-version. Muotoiluinstituutin visuaalinen design ei valmistunut muun muassa teknisistä ongelmista johtuen ja jäi siten keskeneräiseksi, joten sitä ei implementoitu sovellukseen.

Sovelluksen ja käyttöliittymän suunnittelutyö käynnistyi 2014 alkuvuodesta. Tavoitteena oli saada projekti valmiiksi kesäkuun 2014 loppuun mennessä. Ohjelmointi alkoi varsinaisesti toukokuussa 2014. Demo valmistui alkuvuodesta 2015 noin 4–6 kk suunniteltua myöhemmin.

Demo-version ensimmäisessä iteraatiossa on puutteita, jotka pääasiassa johtuivat aikatauluongelmista. Sovelluksen ohjelmointiin kului pidempi aika kuin mitä alun perin suunniteltiin.

Odotukset projektin lopputuloksista olivat ehkä hieman liian korkeat siihen nähden, mitä yksi ohjelmoija voi tuottaa ja millaisella aikataululla se on mahdollista. Tämä on nähtävissä muun muassa sovelluksen puutteellisesta dokumentaatiosta. Myöskään merkittäviä optimointeja sovellukseen ei ole tehty tai ne ovat puutteellisia.

Kokonaisuutena työn voidaan arvioida onnistuneen. Tutkimusongelmanahan oli WWW-palvelun rakentaminen ja siihen liittyvien vaiheiden läpikäyminen. Niitä käsiteltiin opinnäytetyössä Symfony-alustan näkökulmasta.

Kaikkia ajateltuja toimintoja ja ominaisuuksia ei tosin saatu mukaan sovellukseen. Esimerkkinä tällaisesta pois jätetystä toiminnosta on eräänlainen hälytysjärjestelmä. Siinä asiakkailla on mahdollisuus jättää halutusta lentoreitistä järjestelmään tieto, jos kyseistä reittiä ei ole sillä hetkellä tarjolla. Jos myöhemmin joku palveluun kirjautuneista piloteista ilmoittaa lentävänsä kyseistä reittiä, saa asiakas siitä tiedon. Ne toiminnot ja ominaisuudet, jotka on saatu lisättyä sovellukseen, voivat olla myös joiltain osin puutteellisia.

Tämän hetkisessä tilassaan Private Pilot -palvelu 1.0 on toimiva demo-versio. Alkuperäisten suunnitelmien mukaan sitä voidaan käyttää esittelemään liikeideaa mahdollisille sijoittajille, joten se täyttää projektin pääasiallisen tavoitteen. Sovelluksessa olevat puutteet eivät oleellisesti haittaa palvelun toiminnan esittelemistä.

Käyttökokemuksia ei ole dokumentoitu ja käytettävyystutkimusta ei ole tehty. Lisäksi varsinaista palautetta ei kyetty keräämään palvelun ideoijilta. Mahdollisia testikäyttäjiä varten on olemassa verkkosivusto, jossa he voivat vierailla ja samalla testata palvelun toimintaa käytännössä.

5.1 Projektista opittua

Private Pilot -projekti opetti kaupallisten ohjelmistoprojektien matkasta suunnittelupöydältä valmistumiseen asti. Jotkin asiat sujuivat hyvin, mutta sitten oli asioita, joita olisi pitänyt tehdä toisin. Oikein mitoitetuilla teknologisilla ratkaisuilla ja hyvillä käytännöillä olisi voitu päästä parempaan lopputulokseen ja pitäytyä aikataulussa.

Ketterien ohjelmistokehitys menetelmien käyttäminen olisi ollut hyödyllistä. Projekti olisi pitänyt jaksottaa tasaisesti noin yhden tai kahden kuukauden pituisiksi jaksoiksi. Näihin jaksoihin olisi sovittu tavoitteita, jotka ovat valmiina jaksojen loppuun mennessä. Jaksojen lopussa olisi pitänyt olla katsaukset, joissa olisi kat-

sottu, mitkä jakson sovituista tavoitteista ovat täyttyneet. Lisäksi katsauksissa olisi arvioitu uudelleen tulevien jaksojen tavoitteiden aikataulujen realistisuutta.

Palvelun ideoijien luonnokset ja muu palvelua koskettava materiaali olisi pitänyt projektiin osallistuneiden yhdessä organisoida tehtävän tavoitteiksi heti ensimmäisessä projektin jaksossa. Lisäksi olisi pitänyt esittää enemmän kysymyksiä ideoijille, miten he haluavat palvelun toimivan ja miltä sen pitäisi näyttää.

Projektinhallintaan ei panostettu riittävästi. Sovelluksen kehityksessä olisi pitänyt käyttää projektinhallintatyökaluja, kuten Atlassian JIRA -ohjelmistoa. Olisi myös pitänyt huomioida projektin jäsenien kyky osata käyttää näitä työkaluja ja ehkä käyttää aikaa niihin tutustumiseen yhdessä.

Sovelluksenkehittäjien kannattaa käyttää työkaluja kuten esimerkiksi Project Hamsteria, Togglia ja Timelyä, jotka jäljittävät henkilön ajankäyttöä. Projektin tehtäviin kuluvan ajankäytön seuraaminen auttaa kehittäjiä seuraamaan omaa edistymistään. Tällaista ohjelmistoa olisi pitänyt käyttää myös jäljittämään suunniteltuihin jaksokohtaisiin tavoitteisiin käytettyä ajan määrää.

Jaksojen tavoitteet olisi pitänyt jakaa pienempiin hallittavimpiin tehtäviin, joita olisi voinut olla useita käynnissä samanaikaisesti jaksoa kohden. Jos jakson tavoitteeseen liittyvä tehtävä olisi vaikuttanut liian isolta, se olisi pitänyt jakaa metodisesti aina pienempiin osiin sikäli kun oli tarpeellista. Näitä tehtäviä olisi voitu kirjoittaa ylös esimerkiksi projektihallintaohjelmistoon, Google Docs -asiakirjaan tai tarramuistilapuille.

Projekti olisi pitänyt dokumentoida paremmin. Pienetkin viat olisi pitänyt kirjoittaa ylös jonnekin, kuten esimerkiksi projektinhallintaohjelmistoon, josta niitä olisi voitu myöhemmin käydä tarkastelemassa. Dokumentointi kehitystyön aikana on tärkeätä, koska tällöin voidaan myöhemmin esimerkiksi jäljittää, missä versioissa oli mitäkin ongelmia ja minkäkin liitännäisen kanssa. Ohjelmavikojen dokumentointi on yhtä tärkeää, ellei tärkeämpääkin kuin eri ominaisuuksien dokumentointi.

Kehitystyön etenemisen yleinen dokumentointi voidaan tehdä esimerkiksi versionhallintajärjestelmän lokeihin siitä, mitä ominaisuuksia on lisätty, korjattu, muutettu ja poistettu. Niistä voidaan myöhemmin kirjoittaa palvelun asiakkaan, tässä

tapauksessa esimerkiksi palvelun tilanneen ja ylläpitävän yrityksen, työntekijöille muutoslokeja. He voivat siten tarvittaessa tiedottaa asiakkailleen palveluun tapahtuvista muutoksista.

Vikoja olisi pitänyt kirjoittaa ylös Private Pilot -sovelluksen projektinhallintajärjestelmään. Tästä olisi myöhemmin ollut hyötyä muun muassa version 2.0 kehityksen aikana. Se olisi voinut tarjota myös tarkempaa lokia kohdatuista ongelmista, joiden ratkomisessa kului aikaa kehityskaaren loppusuoralla. Vähäpätöisiltäkin vaikuttavat tekniset ongelmat olisi pitänyt kirjoittaa ylös myöhempää tarkastelua varten. Näihin lukeutui esimerkiksi ongelma, jossa Propelin kuvake ei asentunut oikeaan paikkaan Symfonyn WebProfiler-osaan Composeria käytettäessä.

Olisi ollut hyvä huomioida kehityksen aikana, että aina ei ole tärkeätä se, miten hieno ratkaisu on, kunhan se toimii. Ei tulisi jäädä kiinni pieniin ongelmiin pitkäksi aikaa. Niihin voidaan myöhemmin palata, kun kokonaisuus on saatu rakennettua.

Tietokannan koon kasvu olisi Private Pilot -sovelluksessa ollut hyvä huomioida heti kehitystyön alussa. Myös tietokannan kenttien kokoa olisi pitänyt heti rajoittaa tarpeen mukaan. Jälkikäteen on usein vaikeampaa rajata kokoa kuin löysätä rajoituksia.

Kehitystyön aikana ryhmän työvirran kanssa oli ongelmia, jotka kumpusivat eri alojen osaajien yhteistyökokemuksien puutteesta. Tämä näkyi esimerkiksi katkonaisina viestintäketjuina, joissa ideat eivät välittyneet eri tasojen välillä. Jäsenillä oli myös viestintäongelmia, ja ryhmän keskenään jakama visio ei myöskään ollut kovin vahva. Myös ohjauksen puute ja vastuiden siirto alaspäin olivat ongelma, minkä takia suunnitteluvaiheessa tehty työ ei suoraan muuttunut varsinaiseksi konkreettiseksi lopputulokseksi.

Näitä ongelmia olisi voitu korjata esimerkiksi laittamalla projektin ryhmä kokoontumaan viikossa aina muutamana kertana samaan huoneeseen työstämään projektia yhdessä tunniksi tai pariksi projektin vetäjien kanssa. Nopea toimiva palauteketju ja yhteinen visio olisivat myös voineet parantaa viestintään liittyviä ongelmia. Projektissa tulee olla jokin yhteinen visio ja kannustin, joka ohjaa projektissa mukana olevia työskentelemään samaan tavoitteeseen pääsemiseksi.

Projektista vastuussa oleville tahoille olisi pitänyt myös viestiä selkeästi siitä mitä ollaan tekemässä. Palauteketju ei olisi saanut katketa, kuten Private Pilot -projektissa usein kävi. Siten olisi voitu varmistua projektin olevan kulkemassa asiakkaan haluamaan suuntaan. Palauteketjun tulee tuottaa käyttökelpoista dataa projektin kehittäjille, mutta ennen kaikkea myös projektin muille osapuolille.

Private Pilot -projektissa huomattiin miten tärkeätä on pyrkiä käyttämään työkaluja, joiden kanssa on aikaisemminkin työskennellyt. Voi olla vaikeaa omaksua uusia työkaluja lyhyessä ajassa, jos on tottunut joidenkin muiden vastaavien työkalujen paradigmoihin.

Hyvien käytäntöjen noudattaminen on tärkeätä. Jossain on jo huomattu ja luotu parhaat käytännöt sekä julkaistu ne muiden käyttöön. Projektissa olisi pitänyt sopia joistain käytännöistä jäsenten kanssa ja pyrkiä noudattamaan niitä. Olisi ollut esimerkiksi järkevää, että tietokannan ja funktioiden nimet käyttävät samaa nimi-standardia heti alusta lähtien.

Kehitystyössä on tärkeätä julkaista sovelluksesta versioita mahdollisimman aikaisin ja usein, jotta nähdään mihin suuntaan ollaan kulkemassa. Jos suunnan oikaisutarvetta on, se pitää tehdä mahdollisimman varhaisessa vaiheessa. Private Pilotin kanssa ei ollut todellista edistymisen seurantaa, vaikka projekti selkeästi olisi hyötynyt siitä.

Testikäyttäjien olisi pitänyt päästä katsomaan ja käyttämään olemassa olevaa järjestelmää sitä mukaan kun se sai uusia osia. Lisäksi mitä isompi testaajien ryhmä on, sitä enemmän ohjelmistovikoja he löytävät. Kun sovellus pääsi testattavaan vaiheeseen, ei testikäyttäjiä ollut.

5.2 Jatkokehitys ja versio 2.0

Jos Private Pilotin seuraavaa versiota aloitettaisiin kehittämään, olisi ohjelmistotekniikan näkökulmasta asialistalla ensimmäisenä PrivatePilotBundlen siivoaminen. Se koostuu joukosta toimintoja ja tyylejä, joita tulisi huoltaa parempaan kuntoon refaktoroimalla.

Työtehtävänä refaktorointi koostuisi muutamasta järjestelmällisestä toimenpiteestä. Näihin kuuluisi ensimmäisenä muun muassa lentojentilausjärjestelmän irrottaminen PrivatePilotBundle-paketista kokonaan omaksi paketikseen. Nämä irrotettavat osat tulisi järjestää PrivatePilotAdminBundle-paketin esimerkin mukaan, joka monesta näkökulmasta katsottuna edustaa sitä tapaa, miten koko sovellus olisi pitänyt kirjoittaa ja järjestellä.

Muitakin tunnistettavia kokonaisten toimintojen ryhmiä voi olla kuin lentojentilausjärjestelmä, jotka voidaan refaktoroida omiksi bundle-paketeikseen. Mahdolliset tulevat uudet osat, kuten Private Pilot -palvelun CMS-ominaisuudet esimerkiksi yrityksen uutisvirran ja blogin hallintaan, tulisi alusta lähtien kirjoittaa omiksi bundle-paketeiksi, joilla on oma vastuualueensa palvelussa.

Erityisesti PrivatePilotBundlen käyttämät ohjaimet tulisi siivota ja kirjoittaa hyvien käytäntöjen mukaisiksi. Private Pilot -palvelun käyttämissä ohjaimissa saattaa olla hyvinkin paljon tavaraa, jota siellä ei oikeasti tarvitsisi olla. Tämä on merkittävästi palvelun tietoturvaan ja suorituskykyyn vaikuttava asia.

Ohjaimet tulisi kirjoittaa käyttämään PHP-annotaatioita. Erityisesti route-määrittelyt tulisi kirjoittaa PHP-annotaatioina ohjaimeen sikäli kuin se on mahdollista, jotta kehittyneempiä route-määrittelyitä ei tarvita. Tästä seuraisi routing.yml-tiedoston keveneminen, joka selkeyttää reititystä ja ehkä nopeuttaa reitin selvittämistä suorituksen aikana.

Private Pilotin bundle-paketteja varten tulisi kirjoittaa TDD-testit ainakin tärkeimpien ja keskeisimpien osien osalta. Testien kattavuuteen tulisi kiinnittää huomiota tärkeimpien Private Pilot -komponenttien kohdalla.

Monet toiminnot jäivät pois versiosta 1.0 aikataulu- tai muista syistä johtuen. Demo-versiossa ei tarvinnut olla kaikkia niitä ominaisuuksia ja toimintoja, joita palvelu oikeasti tarvitsee. Versiossa 2.0 nämä poisjääneet asiat tulisivat mukaan.

Maksu- ja tilausjärjestelmää tulisi kehittää niin, että PayPal-tuki tulisi laittaa kuntoon ja saada maksuliikenne oikeasti toimimaan. Private Pilot -palveluun pitää lisätä oikea sessiotuki kirjautumattomille asiakkaille. Se yksinkertaistaisi ja tehostaisi maksujärjestelmää ohjelmakoodin puolella.

Yleisesti versiossa 2.0 tulisi erityisesti keskittyä visuaalisen yleisilmeen kohentamiseen ja muun muassa käyttöliittymän käytettävyyden testaamiseen. Private Pilotin dokumentaatio on puutteellinen, joten dokumentaatiota pitäisi ehdottomasti kehittää lisää.

Web-sovellukseen piti tulla Symfony Console -osuus, jolla Private Pilot -palvelun voi helposti asentaa palvelimen tietokantaan. Tämä tapahtuisi ajamalla komento, joka lisää palvelimeen sovelluksen toiminnan kannalta välttämättömät tiedot. Kyseisellä toiminnolla voitaisiin myös hallita sovelluksen kytkentöjä muualle järjestelmään.

Lentojen hakulomakkeessa oleva paikkojen määrä tulisi ehkä määräytyä eniten paikkoja sisältävän lentokoneen mallin mukaan. Järjestelmä voidaan rakentaa automaattisesti huolehtimaan siitä, että eniten vapaita paikkoja sisältävät lennot olisivat aina näkyvissä.

Asiakkaan näkökulmasta tarkasteltuna voi olla tärkeää kertoa näkyvämmin esimerkiksi tietoja yksityislentäjästä, hänen lentokoneensa mallista sekä luotettavuudesta. Hakutulosten lajitteleminen lentojen listausnäkymässä esimerkiksi hinnan, lentäjän saamien arvosanojen tai lähtö- ja saapumisaikojen perusteella olisi tärkeä asia seuraavassa versiossa. Se helpottaisi ihmisiä löytämään juuri ne lennot, joita he ovat hakemassa, ja juuri siihen hintaan kuin he ovat valmiita niistä maksamaan.

Kaiken kaikkiaan Private Pilot oli haastava web-sovellus rakennettavaksi. Se oli projekti, jonka parissa työskennellessä kului yksi vuosi. Se oli merkittävä opin lähde ja mielenkiintoinen opinnäytetyö. Jos siitä joskus tulee tunnettu WWW-palvelu, on mukavaa tietää olleensa käynnistämässä sen toimintaa.

LÄHTEET

Adermann, N. & Boggiano, J. 2016. Composer: Dependency Manager for PHP [viitattu 11.5.2016]. Saatavissa: https://getcomposer.org/

Ambler, S. 2013. Agile Data: The Object-Relational Impedance Mismatch. Ambysoft Inc. [viitattu 11.5.2016]. Saatavissa: http://www.agiledata.org/essays/impedanceMismatch.html

Atlassian. 2016. Bitbucket: the Git solution for professional teams [viitattu 12.5.2016]. Saatavissa: https://www.atlassian.com/software/bitbucket

Bauer, C. & King, G. 2007. What is ORM? Teoksessa: Java Persistence with Hibernate. In Action. Greenwich, CT 06830, United States of America: Manning Publications Co., 25-27.

Bell, J. 2012. PHP Annotations Are a Horrible Idea [viitattu 28.10.2015]. Saatavissa: http://theunraveler.com/blog/2012/php-annotations-are-a-horrible-idea/

Bhatt, L. 2014. Object Relationship Mapping (ORM). lalitbhatt.net [viitattu 25.4.2016]. Saatavissa: http://tech.lalitbhatt.net/2014/07/object-relationship-mapping-orm.html

Blanco, G. & Charron, P. 2010. Request for Comments: Class Metadata. The PHP Group [viitattu 28.10.2015]. Saatavissa: https://wiki.php.net/rfc/annotations

Bowler, T. & Bancer, W. 2009. The Model-View-Controller pattern. Teoksessa: Symfony 1.3 Web Application Development. Starting Series. Birmingham, B27 6PA, United Kingdom: Packt Publishing Ltd., 8–9.

Cloer, J. 2015. 10 Years of Git: An Interview with Git Creator Linus Torvalds. linux.com [viitattu 12.5.2016]. Saatavissa: https://www.linux.com/blog/10-years-git-interview-git-creator-linus-torvalds

Debian Project. 2016. Debian: The Universal Operating System [viitattu 11.5.2016]. Saatavissa: https://www.debian.org/

Drupal. 2016. Drupal API: class HttpKernel. drupal.org [viitattu 3.5.2016]. Saatavissa: https://api.drupal.org/api/drupal/vendor!symfony!http-kernel!HttpKernel.php/class/HttpKernel/8.2.x

Eckerstorfer, F. 2015. BraincraftedBootstrapBundle - Boostrap for Symfony 2 [viitattu 18.7.2014]. Saatavissa: http://bootstrap.braincrafted.com/

Evans, C., Ben-Kiki, O. & Ingerson, B. 2009. YAML Ain’t Markup Language (YAML™) Version 1.2 [viitattu 11.8.2015]. Saatavissa: http://yaml.org/spec/1.2/spec.html

Fowler, M. 2002. Patterns of Enterprise Application Architecture. The Addison-Wesley Signature Series. Boston, MA, USA: Addison Wesley.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. & Booch, G. 1995a. Frameworks. Teoksessa: Design Patterns: Elements of Reusable Object-Oriented Software. Boston, USA: Addison-Wesley, 26-28.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. & Booch, G. 1995b. Observer Design Pattern. Teoksessa: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series. Boston, USA: Addison-Wesley, 294–295.

Git Project. 2016. Git [viitattu 12.5.2016]. Saatavissa: https://git-scm.com/

Haixia, Z. 2002. Fitts' Law: Modeling Movement Time in HCI [viitattu 25.8.2015]. Saatavissa: http://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/printer/fitts_4print.html

Hale, K. 2007. Visualizing Fitts's Law [viitattu 25.8.2015]. Saatavissa: http://www.particletree.com/features/visualizing-fittss-law/

Hibernate.org. 2016. What is Object/Relational Mapping?. Inc Red Hat [viitattu 22.4.2016]. Saatavissa: http://hibernate.org/orm/what-is-an-orm/

Ide, A. 2013. PHP just grows & grows. Netcraft [viitattu 7.9.2015]. Saatavissa: http://news.netcraft.com/archives/2013/01/31/php-just-grows-grows.html

Kadavy, D. 2011a. Chapter 6: Holding the Eye: Composition and Design Principles - Reading direction. Teoksessa: Design for Hackers. Chichester, West Sussex, United Kingdom: Wiley, 135–136.

Kadavy, D. 2011b. Chapter 7: Enlivening Information: Establishing a Visual Hierarchy - Many Visual Factors can affect hierarchy. Teoksessa: Design for Hackers. Chichester, West Sussex, United Kingdom: Wiley, 170–171.

Kariste, O. 2013. Demo Room - Private Pilot konseptin kuvaus ja toimeksiantoehdotus. Lahti: Mainostoimisto Ilme.

Koskimies, K. & Mikkonen, T. 2005a. Johdatus ohjelmistokehyksiin. Teoksessa: Ohjelmistoarkkitehtuurit. Valikko-sarja. Helsinki: Talentum, 187–189.

Koskimies, K. & Mikkonen, T. 2005b. Malli-näkymä-ohjain-arkkitehtuurin soveltaminen. Teoksessa: Ohjelmistoarkkitehtuurit. Valikko-sarja. Helsinki: Talentum, 144.

Koskimies, K. & Mikkonen, T. 2005c. Malli-näkymä-ohjain-arkkitehtuurin toiminta. Teoksessa: Ohjelmistoarkkitehtuurit. Valikko-sarja. Helsinki: Talentum, 143.

Koskimies, K. & Mikkonen, T. 2005d. Malli-näkymä-ohjain-arkkitehtuurit. Teoksessa: Ohjelmistoarkkitehtuurit. Valikko-sarja. Helsinki: Talentum, 142–144.

Lellelid, H. 2003. [ANNOUNCE] PHP5 Torque port: Propel. Apache Foundation [viitattu 7.10.2015]. Saatavissa: http://mail-archives.apache.org/mod_mbox/db-torque-dev/200309.mbox/%3C20030917192803.A248F53858@hoggle.dreamhost.com%3E

Lellelid, H. 2005. Propel Changelog. Github.com [viitattu 21.8.2015]. Saatavissa: https://github.com/propelorm/Propel/blob/2030848844380dea8ed4d11d7cbcbf0d3ffd83f5/generator/CHANGELOG#L41

Lockhart, J. & Sturgeon, P. 2016. PHP: The Right Way [viitattu 18.4.2016]. Saatavissa: http://www.phptherightway.com/

Marston, T. 2004. What is/is not considered to be good OO programming. tonymarston.net [viitattu 27.4.2016]. Saatavissa: http://www.tonymarston.net/php-mysql/good-bad-oop.html

Mattila, A. 2014. Opiskelija, hae mukaan Demo Room Lahteen. Lahti: Lahden ammattikorkeakoulu [viitattu 24.8.2015]. Saatavissa: http://www.lamk.fi/ajankohtaista/Sivut/Demo-Room-Lahti-etsii-opiskelijoita.aspx

Netcraft. 2013. Netcraft: July 2013 Web Server Survey [viitattu 27.10.2015]. Saatavissa: http://news.netcraft.com/archives/2013/07/02/july-2013-web-server-survey.html

Nielsen, J. 1993. Table 2: The Usability Princibles. Teoksessa: Usability Engineering. San Diego, California, USA: Academic Press, 20.

Oracle Corporation. 2016a. MySQL: The world's most popular open source database [viitattu 11.5.2016]. Saatavissa: https://www.mysql.com/

Oracle Corporation. 2016b. NetBeans IDE: Fits the Pieces Together [viitattu 11.5.2016]. Saatavissa: https://netbeans.org/

Otto, M. & Thornton, J. 2015. About Bootstrap. Mark Otto [viitattu 21.8.2015]. Saatavissa: http://getbootstrap.com/about/

Otto, M. & Thornton, J. 2016. Bootstrap: The world's most popular mobile-first and responsive front-end framework [viitattu 10.5.2016]. Saatavissa: http://getbootstrap.com/

Potencier, F. 2007a. Delicious Preview built with Symfony. SensioLabs [viitattu 12.5.2015]. Saatavissa: http://symfony.com/blog/delicious-preview-built-with-symfony

Potencier, F. 2007b. Symfony, Propel, and Doctrine. SensioLabs [viitattu 7.10.2015]. Saatavissa: http://symfony.com/blog/symfony-propel-and-doctrine

Potencier, F. 2009a. Dailymotion, powered by Symfony. SensioLabs [viitattu 27.5.2015]. Saatavissa: http://symfony.com/blog/dailymotion-powered-by-symfony

Potencier, F. 2009b. Doctrine vs Propel. SensioLabs [viitattu 7.10.2015]. Saatavissa: http://symfony.com/blog/doctrine-vs-propel

Potencier, F. 2009c. Templating Engines in PHP. Potencier.org [viitattu 7.8.2015]. Saatavissa: http://fabien.potencier.org/templating-engines-in-php.html

Potencier, F. 2011a. Symfony 2.0. SensioLabs [viitattu 27.5.2015]. Saatavissa: http://symfony.com/blog/symfony-2-0

Potencier, F. 2011b. What is Symfony2?. potencier.org [viitattu 26.4.2016]. Saatavissa: http://fabien.potencier.org/what-is-symfony2.html

Potencier, F. 2012. Why Symfony?. Potencier.org [viitattu 05.12.2015]. Saatavissa: http://fabien.potencier.org/article/65/why-symfony

Propelorm.org. 2015a. Configuring Propel: Supported Formats [viitattu 25.10.2015]. Saatavissa: http://propelorm.org/documentation/10-configuration.html#supported-formats

Propelorm.org. 2015b. Propel, The Blazing Fast Open-Source PHP 5.4 ORM [viitattu 18.8.2015]. Saatavissa: http://propelorm.org/

Propelorm.org. 2016. Working With Symfony2: The Fixtures [viitattu 11.5.2016]. Saatavissa: http://propelorm.org/documentation/cookbook/symfony2/working-with-symfony2.html#the-fixtures

Rabaix, T. 2016. About Sonata. sonata-project.org [viitattu 27.4.2016]. Saatavissa: https://sonata-project.org/about

Reenskaug, T. 2012. MVC: Xerox PARC 1978-79 [viitattu 27.5.2015]. Saatavissa: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

Reenskaug, T., Wold, P. & Lehne, O.A. 1996. Model-View-Controller. Teoksessa: Working With Objects:The Ooram Software Engineering Method. Harlow, United Kingdom: Prentice Hall, 333.

Ronacher, A. 2015. Open Source Libraries. Pocoo.org [viitattu 28.5.2015]. Saatavissa: http://lucumr.pocoo.org/projects/

Rutenberg, G. 2008. Pull vs. Push MVC Architecture [viitattu 8.11.2015]. Saatavissa: http://www.guyrutenberg.com/2008/04/26/pull-vs-push-mvc-architecture/

SensioLabs. 2015a. Best Practices for Reusable Bundles [viitattu 9.10.2015]. Saatavissa: http://symfony.com/doc/current/cookbook/bundles/best_practices.html

SensioLabs 2015b. Chapter 1: How to Use Assetic for Asset Management. Teoksessa: Symfony: The Cookbook. http://symfony.com: SensioLabs, 7–8.

SensioLabs 2015c. Chapter 1: Symfony and HTTP Fundamentals. Teoksessa: Symfony: The Book. http://symfony.com: SensioLabs, 4.

SensioLabs 2015d. Chapter 15: Security. Teoksessa: Symfony: The Book. http://symfony.com: SensioLabs, 174.

SensioLabs 2015e. Chapter 18: Service Container - What is a Service? Teoksessa: Symfony: The Book. http://symfony.com: SensioLabs, 217–218.

SensioLabs 2015f. Chapter 21: Understanding how the Front Controller, Kernel and Environments Work together. Teoksessa: Symfony: The Cookbook. http://symfony.com: SensioLabs, 75–77.

SensioLabs 2015g. Chapter 3: Installing and Configuring Symfony - Checking Symfony Application Configuration and Setup. Teoksessa: Symfony: The Book. http://symfony.com: SensioLabs, 30–31.

SensioLabs 2015h. Chapter 5: Controller. Teoksessa: Symfony: The Book. http://symfony.com: SensioLabs, 43.

SensioLabs 2015i. Chapter 5: Controller - Requests, Controller, Response Lifecycle. Teoksessa: Symfony: The Book. http://symfony.com: SensioLabs, 44.

SensioLabs 2015j. Chapter 5: Controllers. Teoksessa: Symfony: The Best Practices Book. http://symfony.com: SensioLabs, 19–22.

SensioLabs 2015k. Chapter 6: Routing. Teoksessa: Symfony: The Book. http://symfony.com: SensioLabs, 55–56.

SensioLabs 2015l. Chapter 6: Routing - Creating Routes. Teoksessa: Symfony: The Book. http://symfony.com: SensioLabs, 57.

SensioLabs 2015m. Chapter 6: Routing - Including External Routing Resources. Teoksessa: Symfony: The Book. http://symfony.com: SensioLabs, 66.

SensioLabs 2015n. Chapter 61: The HttpFoundation Component. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 234.

SensioLabs 2015o. Chapter 61: The HttpFoundation Component: Response. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 239.

SensioLabs 2015p. Chapter 62: Session Management. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 245–247.

SensioLabs 2015q. Chapter 67: The EventDispatcher Component. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 161.

SensioLabs 2015r. Chapter 67: The HttpKernel Component. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 264–266.

SensioLabs 2015s. Chapter 67: The HttpKernel Component: Calling the Controller. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 272–273.

SensioLabs 2015t. Chapter 67: The HttpKernel Component: Getting the Controller Arguments. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 271–272.

SensioLabs 2015u. Chapter 67: The HttpKernel Component: Handling Exceptions: The kernel.exception Event. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 276–277.

SensioLabs 2015v. Chapter 67: The HttpKernel Component: HttpKernel: Driven by Events. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 266–267.

SensioLabs 2015w. Chapter 67: The HttpKernel Component: Resolve the Controller. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 268–270.

SensioLabs 2015x. Chapter 67: The HttpKernel Component: The kernel.controller Event. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 270–271.

SensioLabs 2015y. Chapter 67: The HttpKernel Component: The kernel.request Event. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 267–268.

SensioLabs 2015z. Chapter 67: The HttpKernel Component: The kernel.response Event. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 275.

SensioLabs 2015aa. Chapter 67: The HttpKernel Component: The kernel.terminate Event. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 275–276.

SensioLabs 2015ab. Chapter 67: The HttpKernel Component: The kernel.view Event. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 274.

SensioLabs 2015ac. Chapter 67: The HttpKernel Component: The Workflow. Teoksessa: Symfony: The Components Book. http://symfony.com: SensioLabs, 273.

SensioLabs 2015ad. Chapter 9: The Bundle System. Teoksessa: Symfony: The Book. http://symfony.com: SensioLabs, 93–94.

SensioLabs 2015ae. The Journey from the Request to the Response. Teoksessa: Symfony: The Book. http://symfony.com: SensioLabs, 9–12.

SensioLabs. 2015af. Projects using Symfony [viitattu 27.5.2015]. Saatavissa: http://symfony.com/projects

SensioLabs. 2015ag. The Release Process [viitattu 24.8.2015]. Saatavissa: http://symfony.com/doc/current/contributing/community/releases.html

SensioLabs. 2015ah. Symfony Documentation [viitattu 8.10.2015]. Saatavissa: http://symfony.com/doc/current/index.html

SensioLabs. 2015ai. Symfony Glossary [viitattu 25.5.2015]. Saatavissa: http://symfony.com/doc/current/glossary.html

SensioLabs 2015aj. Understanding the Bundle System. Teoksessa: Symfony: The Quick Tour. http://symfony.com: SensioLabs, 22.

SensioLabs 2015ak. Understanding the Directory Structure. Teoksessa: Symfony: The Quick Tour. http://symfony.com: SensioLabs, 21-22.

SensioLabs 2015al. Use a Byte Code Cache (e.g. APC). Teoksessa: Symfony: The Book. http://symfony.com: SensioLabs, 230.

SensioLabs 2015am. Using Vendors. Teoksessa: Symfony: The Quick Tour. http://symfony.com: SensioLabs, 25.

SensioLabs 2016a. Chapter 3: Twig for Template Designers: Comments. Teoksessa: The Twig Book. http://twig.sensiolabs.org/: SensioLabs, 13.

SensioLabs 2016b. Chapter 3: Twig for Template Designers: Control Structure. Teoksessa: The Twig Book. http://twig.sensiolabs.org/: SensioLabs, 13.

SensioLabs 2016c. Chapter 3: Twig for Template Designers: Filters. Teoksessa: The Twig Book. http://twig.sensiolabs.org/: SensioLabs, 11–12.

SensioLabs 2016d. Chapter 3: Twig for Template Designers: HTML Escaping. Teoksessa: The Twig Book. http://twig.sensiolabs.org/: SensioLabs, 15–16.

SensioLabs 2016e. Chapter 3: Twig for Template Designers: Macros. Teoksessa: The Twig Book. http://twig.sensiolabs.org/: SensioLabs, 16.

SensioLabs 2016f. Chapter 3: Twig for Template Designers: Template Inheritance. Teoksessa: The Twig Book. http://twig.sensiolabs.org/: SensioLabs, 14–15.

SensioLabs 2016g. Chapter 3: Twig for Template Designers: Variables. Teoksessa: The Twig Book. http://twig.sensiolabs.org/: SensioLabs, 10–11.

SensioLabs 2016h. Chapter 5: Extending Twig: Creating an Extension. Teoksessa: The Twig Book. http://twig.sensiolabs.org/: SensioLabs, 38–40.

SensioLabs 2016i. Chapter 5: Extending Twig: Testing an Extension. Teoksessa: The Twig Book. http://twig.sensiolabs.org/: SensioLabs, 43.

SensioLabs. 2016j. Twig Extensions: The i18n Extension. sensiolabs.org [viitattu 10.5.2016]. Saatavissa: http://twig.sensiolabs.org/doc/extensions/i18n.html

Sharp, J. 2007. Why you should be using a framework. joshsharp.com.au [viitattu 27.4.2016]. Saatavissa: http://www.joshsharp.com.au/blog/view/why_you_should_be_using_a_framework

Shell, W. 2009. Quicky 1: PHP is Loosely Typed - What does that Mean?. Wordpress.com [viitattu 27.5.2015]. Saatavissa: https://wshell.wordpress.com/2009/10/08/quicky-1-php-is-loosely-typed-what-does-that-meand/

Stobart, S. & Vassileiou, M. 2004. But How Popular is PHP Really? Teoksessa: PHP and MySQL Manual: Simple, yet Powerful Web Programming. Springer Professional Computing. London, United Kingdom: Springer-Verlag London, 8-9.

Tatroe, K., MacIntyre, P., Lerdorf, R. & Bourque, M. 2013a. Introduction to PHP: A Brief History of PHP. Teoksessa: Programming PHP. In a Nutshell Series. Sebastopol, California, USA: O'Reilly, 2–6.

Tatroe, K., MacIntyre, P., Lerdorf, R. & Bourque, M. 2013b. Introduction to PHP: What Does PHP Do? Teoksessa: Programming PHP. In a Nutshell Series. Sebastopol, California, USA: O'Reilly, 1–2.

The PHP Group. 2016. History of PHP. php.net [viitattu 10.5.2016]. Saatavissa: http://fr2.php.net/manual/en/history.php.php

Wallsmith, K. 2014. SymfonyLive London 2014: Kris Wallsmith - How Kris Builds Symfony Apps. SensioLabs [viitattu 29.10.2015]. Saatavissa: https://www.youtube.com/watch?v=W8MOIOyPbmM

Whittle, D. 2008. Yahoo! Answers powered by symfony. SensioLabs [viitattu 27.5.2015]. Saatavissa: http://symfony.com/blog/yahoo-answers-powered-by-symfony

Wojciech, S. 2014. Type Hinting is important. sznapka.pl [viitattu 18.4.2016]. Saatavissa: http://blog.sznapka.pl/type-hinting-is-important/

Zaninotto, F. 2006. Yahoo! bookmarks uses symfony. SensioLabs [viitattu 21.5.2015]. Saatavissa: http://symfony.com/blog/yahoo-bookmarks-uses-symfony

Zaninotto, F. 2007. Symfony 1.0 released. SensioLabs [viitattu 27.5.2015]. Saatavissa: http://symfony.com/blog/symfony-1-0-released

Zaninotto, F. & Potencier, F. 2007. YAML as Symfony's default configuration language. Teoksessa: The Definitive Guide to Symfony. Expert's Voice in Open Source. New York, USA: Apress, 62.

LIITTEET

LIITE 1. Private Pilot -esittelylehti (Kariste 2013)

ilme

MAINOSTOIMISTO

Demo Room – Private Pilot
Presentation of concept and assignment

Mainostoimisto Ilme Oy
Lahti: Svinhufvudinkatu 23, 15110 Lahti
Tampere: Polttimonkatu 4, 33210 Tampere
p. +358 (0) 20 7429 850
ilme@ilme.fi

ilme

Private Pilot – concept

Concept:

Private Pilot – concept will be a service working on US markets, which will help single pilots to offer their flights for customers. Basically we will help suppliers and customers to communicate with each others by launching this new way of thinking.

In US also private pilots needs to fly x number of flights annually to be able to keep their 'flying-licence'. Also quite many pilot just flies for fun. Theres alot of private airports and private plane owners around one million. If one million private pilots fly one in month, the total amount is 24 million trips annually

ilme

Private Pilot

We will take benefit of already invented ways to offer bookings, and use already known ways for customers.

ilme

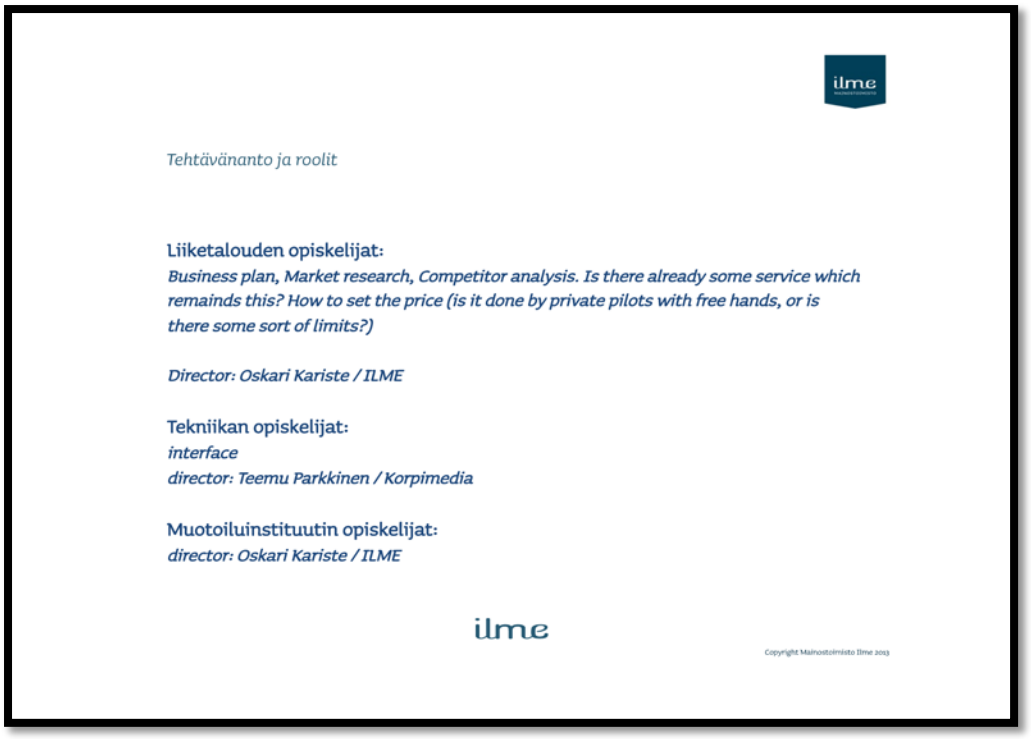
Private Pilot

Private Pilot – internetpage will contact flyers and customers. Theres few scenarios how to do that;

1. Fly with private plane
2. Fly differently
3. Be able to use better connections what airlines can offer
4. Be able to use smaller airports what big companies use

ilme


ilme
Private Pilot
Interface
Most important point is to help customers find private pilots.
The flights are offered by private pilots, who will update their own routes to the sytem. Also they have to update price and time when flight is scheduled.
Customer will search flights like in Ebookers. Most relevant thing is location of starting point. If theres no flights at the day you pick, then you will see when there would be next possibility.
ilme
Copyright Mainostoimisto Ilme 2013


ilme
Tehtävänanto ja roolit
Liiketalouden opiskelijat:
Business plan, Market research, Competitor analysis. Is there already some service which remainds this? How to set the price (is it done by private pilots with free hands, or is there some sort of limits?)
Director: Oskari Kariste / ILME
Tekniikan opiskelijat:
interface
director: Teemu Parkkinen / Korpimedia
Muotoiluinstituutin opiskelijat:
director: Oskari Kariste / ILME
ilme
Copyright Mainostoimisto Ilme 2013


Our goal
Our goal is to find funding, which will help us with marketing and launching of this new service/product.
ilme
Copyright Mainostoimisto Ilme 2013


Possible profits
If we are able to get 10 % of private pilots in US, and each of them will get ten customers annually with price of 250 USD/flight, then the turnover of this service could be 25 M.USD/ year
Our profit 3 % = 7,5 M.USD/year
Plus if we get 100 USD/year service-provider fee, thats gonna make +1 M.USD
Whole annually profit would be 8,5 M.USD
Then we gonna celebrate!
ilme
Copyright Mainostoimisto Ilme 2013

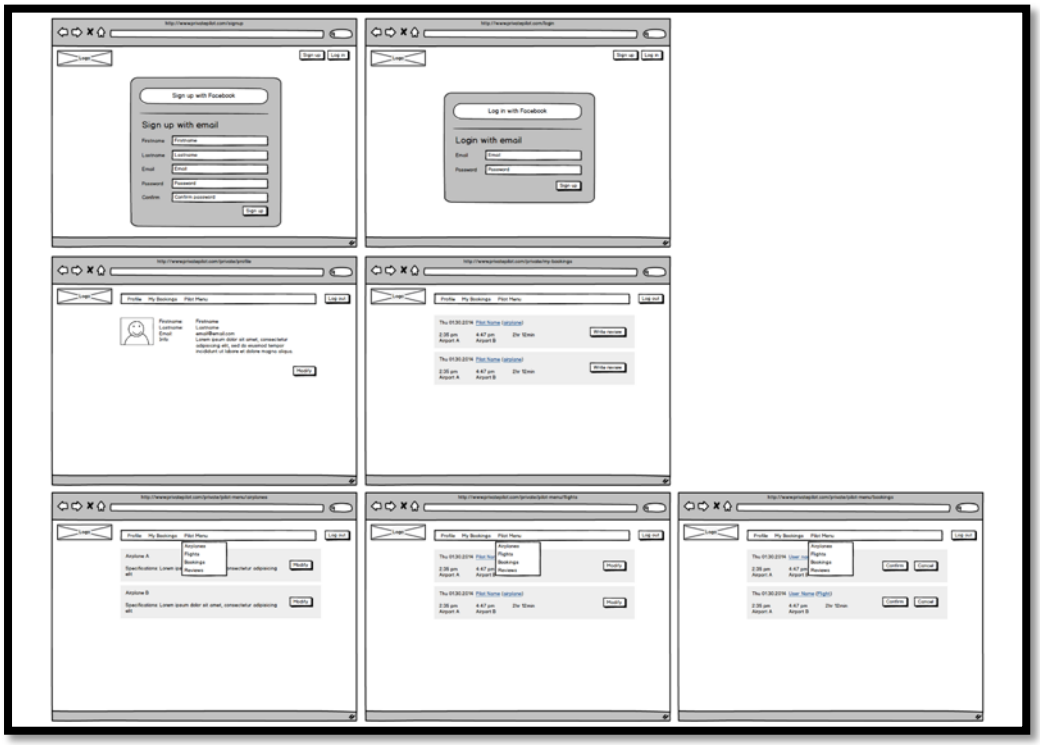
Backround *ideas*

Private airports in Us
http://airportguide.com/private_search_results.php

If pilots would fill up information about their planes
http://www.cessna.com/citation/mustang

And inform how many available seats there is

This might also be fun feature
http://www.seatguru.com/airlines/Air_France/
Air_France_Airbus_A330-200_International_Long_Haul.php


Sign up with Facebook
Sign up with email
Log in with Facebook
Login with email