Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Juho Jaakkola

Opinnäytetyö

Koulutusrekisteri

Moodlen koulutusten seurantaa helpottava sovellus Zend Framework -ohjelmistokirjastoa hyväksikäyttäen

Työn ohjaaja Petri Heliniemi
Työn tilaaja Mediamaisteri Group Oy
Tampere 4/2010

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä	Juho Jaakkola
Työn nimi	Koulutusrekisteri - Moodlen koulutusten seurantaa helpottava sovellus Zend Framework -ohjelmistokirjastoa hyväksikäyttäen
Sivumäärä	44
Valmistumisaika	toukokuu 2010
Työn ohjaaja	Petri Heliniemi
Työn tilaaja	Mediamaisteri Group Oy

TIIVISTELMÄ

Tämä opinnäytetyö käsittelee Mediamaisteri Group Oy:lle tehtyä sovellusta, jonka tarkoituksena oli helpottaa Moodle-oppimisympäristössä toteutettavien koulutusten seurantaa ja hallintaa. Sovelluksen kohderyhmänä olivat Mediamaisterin asiakkaana olevat yritykset, joissa Moodlea käytetään sisäisen oppimisen ja kouluttamisen apuvälineenä.

Sovellus suunniteltiin käytettäväksi WWW-ympäristössä Internet-selaimen kautta, ja se toteutettiin käyttäen hyväksi PHP-ohjelmointikielelle tehtyä Zend Framework -ohjelmistokirjastoa. Tämän ohella käytettiin XHTML-koodia sovelluksen rakenteen esittämiseen, CSS-tyylitiedostoja ulkoasun määrittämiseen sekä MySQL-tietokantaa sovelluksessa käytettävän tiedon säilyttämiseen.

Ohjelmistoprojektin tuloksena Mediamaisterin tuotevalikoimaan syntyi sovellus, joka oli vaivaton asentaa asiakkaiden käyttöön. Asiakkaille sovellus tarjosi helpon tavan pysyä perillä työntekijöiden käymistä koulutuksista keskittämällä kurssien edistymisen seurannan sekä käyttäjäkohtaisen koulutushistorian yhteen paikkaan. Lisäksi sovellukseen toteutettiin mahdollisuus määrittää koulutuskohtaisia ehtoja, joiden toteuttamisen jälkeen koulutus laskettiin suoritetuksi.

Sovellus täytti kaikki sille asetetut tavoitteet, mutta tarvittaessa sitä voisi kehittää esimerkiksi antamaan tarkempia tietoja kurssien etenemisestä. Myös suorituskriteerien määrittämiseen voisi antaa useampia mahdollisia vaihtoehtoja, tai mahdollistaa kriteerien entistä tarkemman määrittämisen.

Avainsanat PHP, Zend Framework, Moodle, E-learning, MVC-arkkitehtuuri

Tampere University of Applied Sciences
Business Information Systems

Writer	Juho Jaakkola
Thesis	Course Registry - Zend Framework -based application for monitoring courses in Moodle
Pages	44
Graduation time	June 2010
Thesis supervisor	Petri Heliniemi
Co-operating Company	Mediamaisteri Group Oy

ABSTRACT

This thesis is about a course registry application made for Mediamaisteri Group Oy. The purpose of the application was to make it easier to monitor and administer courses in the Moodle e-learning platform. The target group of the application was Mediamaisteri Group's client companies who use Moodle as a tool for internal learning and training.

The application was build to be used in the web environment and it was engineered using Zend Framework, a PHP-based MVC framework for web develepment. The representation and appearance of the application was created with XHTML-code and CSS-files. In addition, a MySQL-database was used to store the data needed for the application.

As a result of the project Mediamaisteri Group received a modular and easily maintainable new software for their product range. For the customers the application provided an easy way to be aware of the courses their employees had completed. Thanks to the application, all the course follow-up and user-specific course history were now centralized into one simple but informative application. In addition it was now also possible to define for each course the obligatory tasks that have to be completed before the course is marked as completed.

The application met all the required specifications defined for it, but it could be developed further, for example to give more detailed information about the progress of the courses. Also there could be more options for defining the obligatory tasks for courses so that managing the courses would be even more precise.

Keywords PHP, Zend Framework, Moodle, E-learning, MVC Framework

Sisällysluettelo

Käsiteluettelo

Eväste	Aputiedosto tai tunniste, jonka sisältämiä tietoja voidaan käyttää selaimen ja palvelimen välisen yhteydenpidon ohjaamiseen. (Sanastokeskus TSK ry 2001)
Funktio	Itsenäinen ohjelman osa, joka suorittaa tietyn toiminnon, ja jota voidaan kutsua eri puolilta pääohjelmaa tai muista funktioista. (Wikipedia 2010a)
PDF	Pääasiassa sähköiseen julkaisemiseen, tulostamiseen ja painamiseen tarkoitettu avoin tiedostostandardi. (Wikipedia 2010b)
RSS	Verkkosyötemuoto, jota käytetään usein päivittyvän digitaalisen sisällön, kuten blogien ja uutisten julkaisemiseen. (Wikipedia 2010c)
Selain	Ohjelma, joka on tarkoitettu WWW-sivujen esittämiseen käyttäjän laitteistolla (Sanastokeskus TSK ry 1999)
Skripti	Komentokielellä kirjoitettujen peräkkäin toteutettavien komentojen muodostama kokonaisuus.
SOAP	Internet-yhteyskäytäntö, joka mahdollistaa ulkoisen verkkosovelluksen käyttää hyväkseen verkkopalvelimen tarjoamia palveluja. (Sanastokeskus TSK ry 2009)
Tietokanta	Tietotekniikassa käytetty termi tietovarastolle. Se on kokoelma tietoja, joilla on yhteys toisiinsa. (Wikipedia 2010d)
XHTML	Erityisesti Internetissä esitettävien dokumenttien rakenteen ja ulkoasun kuvaukseen käytettävä merkintäkieli. (Sanastokeskus TSK ry 2007)
XML-RPC	Internet-yhteyskäytäntö, jonka avulla toisessa tietokoneessa toimivia palveluita voidaan käyttää Internet-verkon yli. (Wikipedia 2010e)

1 Johdanto

Moodle-oppimisympäristö on viime vuosien aikana saavuttanut suosiota oppilaitosten lisäksi myös yritysorganisaatioissa. Yritysten organisaatiorakenne ei ole kuitenkaan suuntautunut yhtä vahvasti koulutukseen kuin oppilaitoksissa, mikä varsinkin laajoissa yrityksissä saattaa hankaloittaa Moodlessa tapahtuvien koulutusten hallinnointia. Kouluissa jokaisella kurssilla on oma vastuuopettajansa, joka valvoo yksittäisen kurssin etenemistä ja oppilaiden menestymistä, mutta yritysorganisaatioissa tämä työ saattaa kaatua yksittäisen henkilöstöjohtajan harteille.

Tämän ongelman kanssa tekemisissä on ollut myös Mediamaisteri Group Oy, joka on organisaatioiden sisäisen osaamisen kehittämiseen erikoistunut asiantuntijayritys. Mediamaisteri tarjoaa Moodleen ja muihin avoimen lähdekoodin sovelluksiin liittyviä monipuolisia palveluita, ja kehittää myös omatoimisesti erilaisia WWW-sovelluksia.

Monet Mediamaisterin asiakkaat olivat esittäneet edellä mainittuun ongelmaan liittyviä kysymyksiä ja toiveita, minkä johdosta yrityksessä alettiin miettiä ratkaisuksi aivan omaa sovellustaan, joka helpottaisi Moodlen käyttöä yritysasiakkaiden kannalta. Työharjoitteluni Mediamaisterilla oli tällöin juuri päättynyt, joten sovellus päätettiin toteuttaa opinnäytetyöprojektinani.

Projektin tavoitteena oli kehittää sovellus, joka erilaisten raporttien ja yhteenvetojen avulla helpottaisi Moodlessa pidettävien kurssien seurantaa ja hallintaa yritysorganisaatioissa. Opinnäytetyön tavoitteena on kertoa, miten tämä sovellus toteutettiin, ja millaisia ominaisuuksia valmis sovellus sisälsi.

Opinnäytetyön taustatutkimusosiossa tutustutaan Moodlen toimintaan sekä Zend Framework -ohjelmistokirjastoon, jolla sovellus toteutettiin. Varsinaisessa tutkimusosiossa kerrotaan, miten sovellus rakentuu, mitä toiminnallisuuksia se sisältää, ja miten sovelluksen käyttäjät hyötyvät siitä. Lisäksi mennään hieman syvemmälle Zend Frameworkin käyttöön ja tutustutaan Moodlen tietokantaan sovelluksen vaatimilta osilta.

Zend Frameworkin käyttäminen vaati sen kirjaston tarjoamien käyttömahdollisuuksien selvittämistä, mutta lisäksi tutustumista sen yleisiin käyttösuosituksiin sekä MVC-arkkitehtuuriin. Taustatutkimukseen käytettiin kirjoja *Guide to Programming with Zend Framework* sekä *Zend Framework in Action*, jotka olivat sovelluksen toteutusvaiheessa tuoreimmat saatavilla olevat kirjat. Kaikki teosten kirjoittamisessa mukana olleet henkilöt ovat olleet kiinteästi mukana kehittämässä Zend Frameworkia, joten kirjojen sisältö oli siis todellisten asiantuntijoiden kirjoittamaa.

Kirjallisten teosten lisäksi oli erityisesti ohjelmointivaiheessa käytössä kirjaston online-dokumentaatio, josta löytyi kaikkein ajantasaisin dokumentaatio kirjaston tarjoamien toiminnallisuuksien käytöstä. Kirjalliset teokset kertoivat, miten kirjastoa tuli käyttää, ja mitä sillä oli mahdollista tehdä, kun taas online-dokumentaatio auttoi eteenpäin, kun vastassa oli jokin yksittäinen ohjelmointiin liittyvä ongelma.

Moodleen liittyen tutkimustyötä joutui tekemään lähinnä tietokannan osalta. Oli toki tärkeää tietää perusajatus siitä, miten Moodle toimii, mutta lopullisessa työssä riitti pelkkä tietokannan rakenteen ja siitä löytyvän tiedon tunteminen. Tietokannan rakenteestakin oleellista oli vain pieni kursseihin liittyvä osa. Yleiskuvaa Moodlen toiminnasta avasivat kirjat *Moodle* ja *Using Moodle*, jotka oli osoitettu lähinnä Moodlea opettajan roolissa käyttäville henkilöille. Kirjat olivat jo hieman ikääntyneitä, mutta Moodlen perusidea ei kuitenkaan ole vuosien saatossa juurikaan muuttunut.

Tietokannan rakenteen selvittäminen oli enimmäkseen empiiristä tutkimusta, mutta pariin otteeseen piti tukeutua Moodlen online-dokumentaatioon ymmärtääkseen tietokantataulujen suhteita. Dokumentaation ollessa melko puuttellinen, sai ratkaisun ongelmiin viimeistään, kun kysyi neuvoa kokeneemmilta Mediamaisterin IT-henkilöiltä, jotka olivat ehtineet työskennellä Moodlen kanssa jo pidemmän aikaa.

Lähteenä käytettiin myös Wikipediaa, mutta käyttö rajoittui lähinnä yksittäisten termien ja käsitteiden tarkentamiseen. Opinnäytetyön varsinaisen sisällön tukemiseen sitä ei käytetty, joten tässä käyttötarkoituksessa sen tarjoama tieto oli tarpeeksi luotettavaa.

2 Moodle

Moodle on ilmainen, avoimeen lähdekoodin perustuva virtuaalinen oppimisympäristö, jota käyttävät yritykset, organisaatiot sekä kaikentyyppiset oppilaitokset aina peruskoulusta korkeakouluihin. Se soveltuu erilaisiin käyttötarkoituksiin, joten eri käyttäjäryhmät voivat käyttää sitä haluamallaan tavalla. Monet instituutiot käyttävät sitä pitääkseen täysin virtuaalisia verkossa suoritettavia koulutuksia, mutta sitä voidaan käyttää myös perinteisen kasvokkain tapahtuvan opetuksen ohella. (Wikipedia 2010f)

Moodlen loi Australialainen tietojenkäsittelijä ja kouluttaja Martin Dougiamas, joka kyllästyi käytössä olleisiin oppimisympäristöihin, ja halusi luoda järjestelmän, jonka olisivat insinöörien sijasta kehittäneet opettajat. Hän uskoi, että näin järjestelmästä tulisi paremmin tarkoitukseensa sopiva. Sovelluksen ensimmäinen versio valmistui vuonna 2002. (Rice 2005, xiii)

Moodle tarjoaa työvälineitä mm. vuorovaikutukseen, sisällöntuottamiseen ja materiaalin jakamiseen. Sen käyttäjämäärät eri käyttäjäryhmissä voivat vaihdella kymmenistä käyttäjistä satoihin tuhansiin käyttäjiin. Moodlessa pidettävät kurssit voivat olla joko avoimia kaikille, tai ne voidaan rajata vain tietylle ryhmälle määrittämällä niille salasanan kaltainen avain, jonka avulla vain kyseiset henkilöt oppilaat pääsevät osallistumaan kurssille. (Wikipedia 2010f)

Moodle tarjoaa kouluttajille mahdollisuuden luoda pitämäänsä kurssia varten oman oppimisympäristön ja lisätä ympäristöön erilaisia materiaaleja, tehtäviä ja aktiviteetteja, jotka tehostavat kurssia oppimisen kannalta. (Cole 2005, 1-2) Tällaisia työkaluja ovat muun muassa keskustelualue, oppitunti, sanasto sekä erillisenä tiedostona palautettava tehtävä.

Moodlessa pidettävä kurssi voisi edetä esimerkiksi seuraavasti: kouluttaja luo kurssia varten oman oppimisympäristön ja lisää kurssille staattista materiaalia, johon oppilaiden pitää tutustua. Tämä voi olla joko kurssin oppimisympäristöön tuotettua materiaalia tai

erillinen tiedosto, joka on ladattu palvelimelle ja liitetty osaksi kurssia. Kun materiaali on käsitelty oppilaiden kanssa, voi kouluttaja käyttää joitakin interaktiivisia toiminnallisuuksia käsiteltyjen asioiden harjoittamiseen.

Oppilaat voivat luennon jälkeen esimerkiksi ratkoa siihen liittyviä tehtäviä yhdessä Moodlen keskustelualueella, ja tämän jälkeen palauttaa kirjoittamansa vastauksen erillisenä tiedostona tehtävien palautukseen tarkoitetulla toiminnolla. Kurssin lopussa kouluttaja voi luoda Moodleen kyselylomakkeen, jossa esimerkiksi monivalintakysymyksillä selvitetään, miten hyvin oppilaat ovat kurssilla opetetut asiat oppineet.

Tällä tavalla erilaisten materiaalien ja aktiviteettien avulla voidaan Moodlessa pitää kursseja, jotka sisältävät monipuolista oppimateriaaleja sekä niihin liittyviä interaktiivisia tehtäviä. Erilaisia aktiviteetteja löytyy Moodlesta useita, ja niitä voidaan tarvittaessa asentaa erillisinä moduuleinaan lisää Moodlesta oletuksena löytyvien moduuleiden rinnalle. Tämän opinnäytetyön kannalta oleellisia aktiviteetteja ovat kuitenkin vain aktiviteettimoduulit tentti, kirja sekä kyselylomake.

Tentti-moduuli on työkalu, jolla voidaan luoda monia erilaisia kysymystyyppejä sisältäviä kysymyssarjoja. Erilaisia tyyppejä ovat muun muassa monivalintakysymys, kyllä/ei-valinta sekä kysymys, johon vastataan lyhyellä kirjallisella vastauksella. Kysymykset ovat varastoituina kysymyspankissa, josta niitä voidaan käyttää uudelleen monissa eri tenteissä. Tentteihin voidaan sallia useita vastausyrityksiä, ja vastaamisen jälkeen opettaja voi joko antaa kirjallista palautetta kysymyksistä tai laittaa esille kysymysten oikeat vastaukset. (Moodle.org 2008)

Tentin yleisin käyttötarkoitus on selvittää kurssin eri vaiheissa, kuinka hyvin oppilaat ovat sisäistäneet siihen asti käsitellyt asiat. Tarvittaessa tentteihin voidaan sallia uudelleenyrittäminen joko kysymys- tai tenttikohtaisesti. Vaihtoehtoisesti voidaan kysymykset arpoa täysin sattumanvaraisesti, jolloin tentin sisältöön saadaan vaihtuvuutta. (Moodle.org 2008)

Kirja-moduuli on lohko, joka tuottaa monisivuisen kirjaa muistuttavan oppimateriaalikokonaisuuden, jossa olevia sivuja oppilas voi vapaasti selata. Kirjaan voi luoda kahdella eri tasolla olevia lukuja eli päälukuja ja näiden alilukuja. Kirjamoduuli ei sisällä interaktiivisia toimintoja, mutta kirjan tekijä voi kuitenkin lisätä siihen linkkejä muihin aktiviteetteihin. Lisäksi on mahdollista lisätä kirjan sisällöksi Flash-animaatioita. (Moodle.org 2008)

Kyselylomake-moduulin tarkoituksena taas on mahdollistaa palautteen kerääminen kurssin osallistujilta. Moduuli sallii tentti-moduulin tapaisesti omien kysymysten tekemisen sekä useita eri kysymystyyppejä. Muita vaihtoehtoja palautteen keräämiselle olisivat kyselymoduuli sekä palautemoduuli. Ensimmäinen sisältää kuitenkin vain valmiita kyselyitä, ja jälkimmäinen on ominaisuuksiltaan hyvin pelkistetty, eikä anna mahdollisuutta kovinkaan monipuolisen kyselyn luomiseen.

Kyselylomake-moduuli on siis oikea vaihtoehto, jos kouluttaja haluaa tehdä monipuolisen kyselyn, jonka sisältämien kysymysten täytyy olla juuri tiettyä kurssia varten kohdistettuja. Omien kysymysten tekemisen lisäksi moduuli sallii kouluttajan päättää, kerätäänkö palaute anonyymisti vai tallennetaanko vastausten lisäksi vastaajan nimi.

3 Zend Framework

Koulutusrekisterisovelluksen tekninen puoli toteutettiin käyttämällä hyväksi Zend Framework -ohjelmistokirjastoa, joka on otettu käyttöön Mediamaisterilla helpottamaan sovellusten tekoa ja ylläpitoa asiakkaiden tarpeiden ja vaatimusten jatkuvasti kasvaessa. Oikein käytettynä ohjelmistokirjastot tekevät sovelluksista loogisia kokonaisuuksia ja mahdollistavat niiden helpon ylläpidon ja muokkaamisen, mikä pidentää merkittävästi sovelluksen käyttöikää.

Tässä luvussa käsittelen ohjelmistokirjastoja lyhyesti yleisellä tasolla, ja keskityn sitten hieman tarkemmin juuri Zend Framework -kirjaston ominaisuuksiin ja mahdollisuuksiin. Opinnäytetyössä käsiteltävän Zend Frameworkin versio on 1.8.2.

3.1 Miksi käyttää ohjelmistokirjastoa

Aloittelevat koodaajat tekevät dynaamisia WWW-sivustoja perinteisesti yhdistelemällä suoraan HTML- ja PHP-koodia. PHP mahdollistaa erilaisten vuorovaikutuksellisten toimintojen toteuttamisen, ja HTML-koodin avulla nämä toiminnot voidaan esittää käyttäjälle. Tavallisesti jokaista WWW-sivuston sivua kohden on olemassa oma tiedostonsa, joka sisältää kaiken kyseiseen sivuun liittyvän toiminnallisuuden ja esitystavan.

Pienissä ja yksinkertaisissa sivustoissa tämä tekniikka toimii täysin moitteetta ja tuottaa nopeasti ja helposti tuloksia. Sivustojen laajuuden ja toiminnallisuuksien kasvaessa alkaa kuitenkin syntyä ongelmia. Koodia tutkiessaan huomaa, että kaikki ulospäin näkyvä sisältö sekä eri toiminnallisuudet ovat ripoteltuina pitkin useita eri tiedostoja. (Allen, Lo & Brown 2009, 3)

Jos tällä tavalla toteutettua sivustoa haluaa jollakin tapaa päivittää, täytyy haluttuja muutoksia lisäillä useisiin kohtiin useissa eri tiedostoissa. Esimerkiksi sivuston ulko-

asua muokatessaan voi joutua muokkaamaan saman muutoksen HTML-koodiin joka ikiseen tiedostoon. Tai jotakin PHP:n toiminnallisuutta, kuten lomakkeilta syötetyn tiedon käsittelyä, parannellessaan voi joutua lisäämään muutoksen jokaiselle lomakkeita sisältävälle sivulle.

Itsestään selvää on, että tämä tuottaa pidemmällä aikavälillä todella paljon ongelmia ylläpidettävyyden ja laajennettavuuden suhteen. Varsinkin kun PHP- ja HTML-koodi ovat sekoitettuina keskenään niin, että merkintäkoodia ja toiminnallisuuksista vastaavaa koodia on hankala erottaa toisistaan. Pahimmassa tapauksessa toisen muokkaaminen vaikuttaa myös toiseen, mikä voi aiheuttaa vakaviakin virheitä sivuston toiminnassa.

Ratkaisu tämänkaltaisen sotkun välttämiseen on strukturointi. Yksinkertaisimmillaan tämä tarkoittaa sitä, että WWW-sovelluksen esittämiseen ja erilaisiin toiminnallisuuksiin, kuten tietokannan kanssa keskusteluun, käytetty koodi sijoitetaan eri tiedostoihin. Tällöin erityyppisille koodeille on oma paikkansa, ja halutut koodin osat löytyvät helpommin. (Allen ym. 2009, 4)

Kun koodi on saatu hyvään järjestykseen, nousee seuraavaksi kysymykseksi koodin uudelleenkäyttö. Laajan WWW-sovelluksen sisällä saattaa toistua useita kertoja aivan samanlaiset toiminnallisuudet, kuten tietojen noutaminen tietokannasta tai lomakkeelta lähetettyjen tietojen käsitteleminen. Tällöin on loogista, että toiminnallisuudet kirjoitetaan vain kertaalleen omaan tiedostoonsa. Kun tarvitaan tiettyä toimintoa, ei sitä enää tarvitse kirjoittaa muun koodin sekaan, vaan valmista koodia voidaan käyttää jo olemassa olevasta tiedostosta.

Seuraavaksi tulee ajankohtaiseksi funktioiden käyttö. Useat suoraan koodin sekaan sisällytetyt toiminnallisuudet saattavat sisältää samannimisiä muuttujia, jolloin ne voivat sekoittua keskenään. Tämän voi välttää tekemällä toiminnallisuuksista itsenäisiä funktioita. Funktion sisässä oleva koodi on täysin eristetty muusta koodista, jolloin sen sisällä voi suorittaa toimintoja varmana siitä, että ne eivät vaikuta muuhun koodiin.

Sovelluksessa olevien funktioiden määrän kasvaessa myös niiden hallinta hankaloituu. Tätä voi helpottaa jakamalla funktiot erilaisiin luokkiin sen perusteella, minkälaisia toimintoja ne suorittavat. Tämän jälkeen esimerkiksi kaikki tietokantaan liittyvät funktiot löytyvät omasta luokastaan, taulukoita generoivat funktiot omastaan ja niin edelleen. Ja näin on syntynyt ohjelmistokirjasto.

Nyt kaikki koodi on omassa määrätyssä paikassaan, jokaiselle toiminnallisuudelle on toteutettu oma funktionsa, ja nämä on vielä jaettu selkeisiin luokkiin. Tämän ansiosta sovellusta on paljon helpompi ylläpitää, ja samoja valmiita toimintoja voidaan helposti käyttää uudelleen pitkin sovellusta.

Monille koodaajille tämä on täysin riittävä tapa toimia, mutta vielä kaiken tämän jälkeenkin on mahdollista törmätä aivan uusiin ongelmiin. On lähes väistämätöntä, että ennemmin tai myöhemmin syntyy uusia tarpeita, joita kirjasto ei vielä täytä. Tai kokemuksesta viisastuneena tajuaa, että jotkin kirjaston toiminnallisuudet olisi voinut toteuttaa paljon paremmalla tavalla.

Uusia toimintoja lisätessään ja vanhoja korjaillessaan huomaa nopeasti pelkästään kirjaston ylläpidon vievän kaiken ajan sovellusten kehittämisestä. Voi olla, että luokkia ja funktioita on niin paljon, ettei kirjaston käyttäjä enää edes kunnolla muista, miten kaikki toimii. Kirjaston kokonaiskuva muuttuu pikkuhiljaa epäselväksi, ja sen käytön tehokkuus alkaa laskea.

Tällöin tulee ajankohtaiseksi ryhtyä käyttämään julkisia, avoimeen lähdekoodiin perustuvia kirjastoja, joita kehittämässä ja ylläpitämässä ovat yhteisöt, joihin kaikkein suosituimpien kirjastojen tapauksessa saattaa kuulua useita satoja jäseniä ympäri maailmaa (Zend Technologies Ltd. 2010). Yksi tällainen kirjasto on Zend Framework.

3.2 Kirjasto

Zend Frameworkin kirjasto on täysimääräinen sovelluskehys, ja sen tarkoituksena on tarjota kaikki tarvittavat ominaisuudet yritysmaailman sovellusten tarpeisiin. Se on kuitenkin myös hyvin joustava ja suunniteltu niin, että käyttäjä voi halutessaan ottaa käyttöön vain osat, jotka hän tarvitsee. Erilaisia osia on useita, mutta ne voi jakaa pääpiirteissään kuuteen eri kategoriaan. Nämä ovat MVC, käyttäjien autentikointi ja pääsyn sääteleminen, kansainvälistäminen, ohjelmien välinen tiedonvälitys, WWW-palvelut sekä kirjaston ydin.

MVC-komponentit tarjoavat täysipainoisen *model-view-controller*-mallin, joka yhdessä kirjaston ytimen komponenttien kanssa mahdollistaa sovelluksen ulkoasun, toimintalogiikan sekä ohjaustiedostojen erottamisen toisistaan. Tämän ansiosta sovelluskehitys on joustavaa ja sovellukseen on helppo tehdä jälkeenpäin laajojakin muutoksia. MVC-arkkitehtuuria käsitellään myöhemmin hieman tarkemmin.

Monissa sovelluksissa on tarvetta käyttäjien autentikoinnille ja näiden käyttäjien pääsyn rajoittamiselle. Autentikointi perustuu yleensä tunnus-salasana-pariin, jonka avulla käyttäjä tunnistetaan, mutta tunnistaminen voi perustua muihinkin menetelmiin. Pääsyn rajoittamisessa taas selvitetään, mihin sovelluksen resursseihin tunnistetulla käyttäjällä on oikeus päästä käsiksi, ja miten käyttäjä voi vaikuttaa näihin resursseihin. Zendin autentikointi ja pääsylista -kategoria tarjoaa toimintoja näiden ominaisuuksien toteuttamiseen.

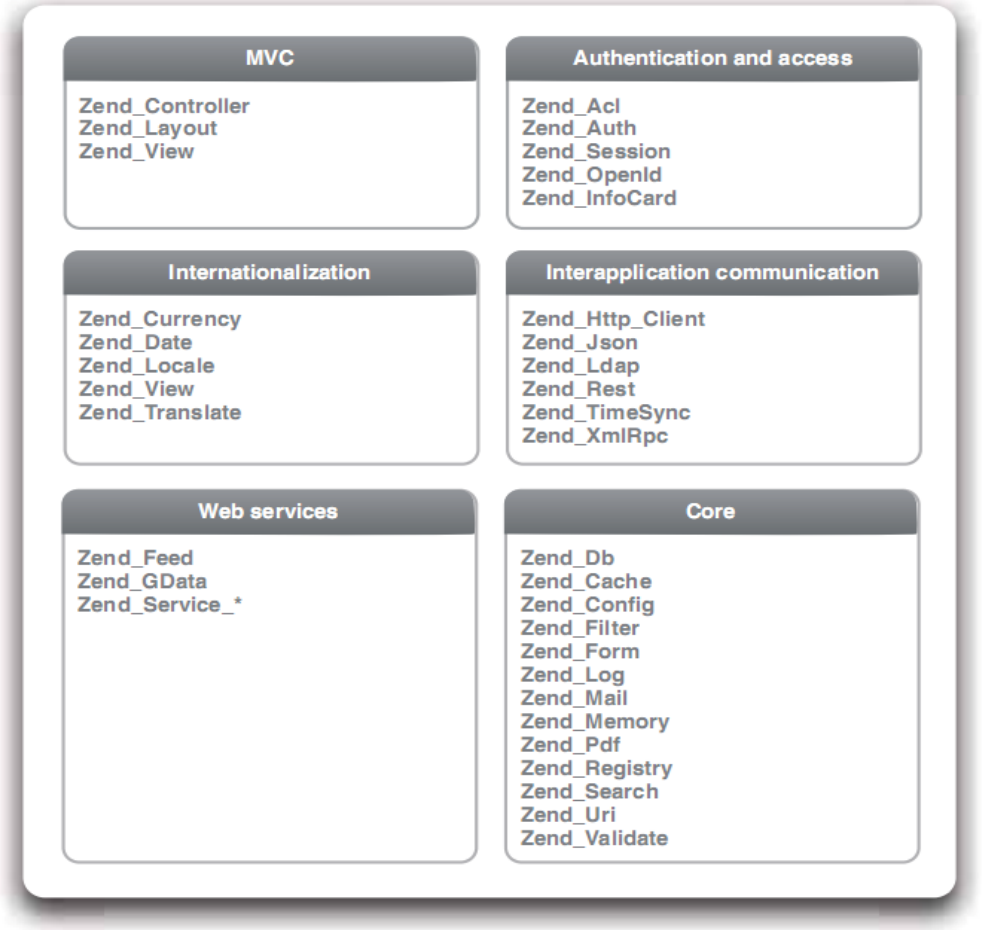
Kansainvälistämiseen tarkoitettu kategoria sisältää ominaisuuksia, joilla voidaan helposti muuttaa sovellusten sisältö vastaamaan eri maiden ja alueiden yksilöllisiä tarpeita. Ominaisuuksiin kuuluu hyvin pieniä toimintoja, kuten eri valuuttasymboleiden näyttäminen maakohtaisesti, mutta tämän lisäksi voidaan kääntää kaikki sovelluksen tekstimuotoinen sisältö sanoittain tai lauseittain kielestä toiselle. Myös päivämäärien ja kellonaikojen muotoa voidaan tähän tarkoitettujen toiminnallisuuksien avulla vaihtaa maakohtaisesti.

Ohjelmien väliseen keskusteluun tarkoitettu kategoria mahdollistaa tiedon lukemisen muista sovelluksista Internetin välityksellä. Sen ominaisuuksien avulla voi helposti kerätä dataa muista sovelluksista ja esittää ne omallaan. Yleisimmin tähän käytetään joko XML-RPC- tai SOAP-protokollaa, joihin molempiin löytyy kategoriasta omat toiminnallisuutensa. Myös nousevassa suosiossa oleva JSON-protokolla on tuettu (Allen ym. 2009, 13).

WWW-palvelu-komponenttien avulla voi päästä käsiksi muiden WWW-sivustojen tarjoamiin palveluihin. Nämä Zendin komponentit käsittelevät yksinkertaisimmillaan tavallista RSS-syötettä, mutta niistä löytyy myös valmiita toiminnallisuuksia muun muassa Googlen, Yahoo!:n ja Amazonin ohjelmistorajapintojen kanssa keskusteluun. Esimerkiksi Googlen rajapinnan kanssa keskustelevat toiminnot voivat hakea tietoa Googlen kalenterista, Base-tietokannasta sekä YouTube-videopalvelusta.

Edellä mainittujen komponenttien lisäksi Zendin kirjasto sisältää osia, joita on hankala sijoittaa mihinkään tiettyyn kategoriaan. Tämän vuoksi ne on kerätty yhteen kirjaston ydinkomponenteiksi. Tämä kategoria sisältää muun muassa luokkia, joiden avulla voi keskustella tietokantojen kanssa, luoda lomakkeita, lähettää sähköpostia ja lukea sovelluksen konfiguraatiotiedostoja. Näiden perustoimintojen lisäksi se sisältää myös erikoistuneempia toiminnallisuuksia, kuten tiedon välimuistiin tallentamisen, sivuston sisäisten hakutoimintojen tekemisen sekä PDF-tiedostojen luomisen.

Tarkemmat listaukset kirjaston eri kategorioista ja niiden sisältämistä luokista on nähtävissä kuviossa 1.



Kuvio 1: Zend Frameworkin kirjastokomponentit (Allen ym. 2009, 10)

3.3 MVC-arkkitehtuuri

Sen lisäksi, että Zend Framework on luokkakirjasto, on se myös ohjelmistokehys. Tämä tarkoittaa, että erilaisten valmiiden luokkien lisäksi sitä varten on suunniteltu myös runko, jonka päälle voidaan kehittää uusia sovelluksia. Ohjelmistokehykset itsessään eivät yleensä sisällä mitään suoritettavia toimintoja, vaan vain erilaisia valmiiksi rakennettuja sovelluksen osia, joiden ansiosta uutta sovellusta ei tarvitse alkaa kehittää aivan puhtaalta pöydältä (Wikipedia 2009). Oleellinen piirre Zend Frameworkin ohjelmistokehystä on MVC-arkkitehtuuri.

MVC-arkkitehtuurin periaatetta on käytetty hyväksi ohjelmoinnissa jo vuodesta 1979, jolloin se kehitettiin Xeroxin PARC-tutkimuskeskuksessa. Tänä päivänä se on hyvin yleisesti käytössä eri luokkakirjastojen yhteydessä (Evans 2008). Arkkitehtuuri perustuu koodin jakamiseen kolmeen loogiseen ryhmään: *malli* (model), *näkymä* (view) ja *ohjain* (controller).

Malli vastaa sovelluksessa datan käsittelystä. Monesti puhutaan joko kevyestä tai raskaasta mallista. Kevyt malli hoitaa tiedon lisäämisen, päivittämisen ja poistamisen eli perustoiminnot, joita tarvitaan laajaa tietomäärää käyttävässä sovelluksessa. Tästä käytetään myös lyhennettä CRUD (*Create*, *Update*, *Delete*). Raskaasta mallista puhutaan, kun näiden toimintojen lisäksi malli hoitaa myös monimutkaisempia sovelluskohtaisia toimintoja eli niin sanottua bisneslogiikkaa (Evans 2008, 42). Tällaisia toimintoja ovat esimerkiksi tilausten ja maksujen käsittely verkkokaupassa tai matemaattisten laskutoimitusten ja yhteenvetojen tekeminen suurista tietomääristä.

Näkymän tehtävä on muuttaa mallin tuottama data muotoon, jota sovelluksen loppukäyttäjä pystyy hyödyntämään. Yleisimmin tämä tarkoittaa HTML-koodia, jonka selain muuttaa ihmisen ymmärrettävään muotoon, mutta tämän sijasta näkymä voi yhtä hyvin muuttaa datan esimerkiksi PDF-, XML- tai JSON-muotoon (Evans 2008, 60). Koska malli ja näkymä toimivat itsenäisesti ja ovat selkeästi erotettuina toisistaan, pystyy saman datan esittämään usealla eri tavalla. Datan esittämisen lisäksi näkymän kautta tapahtuu myös tiedon kerääminen käyttäjältä erilaisten lomakekenttien kautta.

Ohjain vastaa nimensä mukaisesti sovelluksen toiminnan ohjaamisesta. WWW-sovellusten tapauksessa ohjaimen tehtävä on ottaa vastaan pyyntöjä käyttäjältä, ja päättää niiden perusteella, mikä WWW-sivu tulisi näyttää, ja mitä toimintoja toteuttaa ennen sivun lähettämistä käyttäjälle. Ohjain tekee tarvittavat pyynnöt malleille ja ohjaa niiden tuottaman datan eteenpäin näkymälle, jonka jälkeen valmis sisältö lähetetään takaisin sovelluksen käyttäjälle.

3.4 Muita piirteitä

Yksi Zend Frameworkin ominaispiirre on ”puhtaiden” URL-osoitteiden käyttö. Tämä tarkoittaa, että osoite, joka perinteisesti olisi esimerkiksi muotoa *www.osoite.com?sivu=uutiset&osio=urheilu*, näkyy muodossa *www.osoite.com/uutiset/urheilu*. Näin osoitteet ovat selkeämpiä, ja helpommin muistettavissa. Puhtaat URL-osoitteet ovat selkeytensä ja helpon muistettavuutensa ansiosta nopeasti yleistymässä Internetissä. (Berners-Lee 1998).

MVC-mallin yhteydessä Zend Framework hyödyntää myös *Front Controller* -toimintamallia, jossa kaikki sovellukseen tehtävät pyynnöt kohdistetaan vain yhteen tiedostoon, joka yleensä on nimetty *index.php*. Toinen, nykyään jo hieman vanhentunut, tapa tehdä WWW-sovelluksia olisi *Page Controller* -malli, jossa jokaista toimintoa kohden on oma tiedostonsa (*sivu1.php*, *sivu2.php* jne.). Tässä mallissa kuitenkin ilmenee turhaa toistoa, sillä usein joka sivulla on käytössä aivan samoja toimintoja, kuten sivun rakenteen määrittely. (Wikipedia 2010g)

Pyyntöjen keskittäminen yhteen tiedostoon, ja eri toimintojen suorittaminen ohjaimen kautta vähentää tätä toistoa ja helpottaa sovelluksen ylläpitoa. Prosesseista, joilla ohjain selvittää käyttäjän haluaman sivun ja ajaa kaikki kyseisen sivun näyttämiseen tarvittavat toiminnot, käytetään termejä *routing* ja *dispatching* (reitittäminen ja suorittaminen). Tämän toiminnan hoitaa ohjain, joka URL-osoitteen avulla päättelee, minkä sivun käyttäjä haluaa nähdä.

Jos käyttäjä pyytää sivuston etusivua, esimerkiksi *www.osoite.com*, pyrkii ohjain tällöin suorittamaan *IndexController*-nimisen ohjaimen *IndexAction*-nimisen toiminnon, joka on sivuston oletussivu. Jos pyyntö on esimerkiksi muotoa *www.osoite.com/tapahtumat/pikkujoulut*, suorittaa Zend Framework *pikkujoulutAction*-nimisen toiminnon *tapahtumatController*-nimisestä ohjaimesta. Tällä tavoin ohjaintiedostot generoivat ja esittävät käyttäjälle tämän pyytämän WWW-sivun.

4 Sovelluksen määrittelyt

Tässä luvussa käydään läpi, mitä vaatimuksia sovellukselle määritettiin. Määrittelyssä piti ottaa huomioon asiakkaan tarpeet, mutta myös sovelluksen käytön tekniset vaatimukset.

4.1 Vaaditut toiminnallisuudet

Sovellukselle asetettu tärkein vaatimus oli keskittää Moodlen kurssien seuranta sekä kurssihistoria yhteen paikkaan niin, että niitä olisi helpompi hallinnoida. Koska sovelluksen kohderyhmänä olivat yritysorganisaatiot, piti kiinnittää huomiota erityisesti esimiesten ja johtotehtävissä olevien henkilöiden tarpeisiin. Esimiesten piti päästä seuraamaan kaikkien alaistensa kurssien etenemistä, ja henkilöstöpäälliköiden piti päästä käsiksi kaikkien yrityksen työntekijöiden kurssitietoihin. Tavalliselle työntekijälle riitti pääsy listaukseen omista kursseistaan.

Kurssihistorian tuli sisältää listaus kaikista kursseista, joille käyttäjä on jossain vaiheessa ollut ilmoittautuneena. Listaus sisälsi siis sekä suoritetut kurssit että kurssit, joiden suorittaminen oli vielä kesken. Tällaiseen listaukseen piti siis olla pääsy kaikilla työntekijöillä. Esimiesten piti lisäksi päästä näkemään vastaavat listaukset omien alaistensa osalta.

Kurssihistorialistauksen lisäksi sovelluksessa piti olla kurssien yleisestä tilanteesta yhteenvetoja antava sivu. Mahdollisimman havainnollisen tilannekuvan saavuttamiseksi yhteenvetoon päätettiin tehdä niin lukumäärällinen, prosentuaalinen kuin graafinenkin kuvaus. Tietoja piti päästä selaamaan kursseittain sekä kurssien sisällä esimiehittäin. Tämän lisäksi piti päästä näkemään jokaisen esimiehen alaisten tilanteesta lista, joka kertoi, moniko osallistuneista oli ehtinyt suorittaa kurssin.

Kurssien seurannan ja kurssihistorian lisäksi yritysten henkilöstöjohtajilla piti olla mahdollisuus päästä määrittämään pakolliset Moodlen kurssimoduulit, joiden tulee olla hyväksytysti suoritettuja ennen kuin kurssi merkitään suoritetuksi. Moodlessa oli käytettävissä useampiakin kurssimoduuleita, mutta sovelluksessa riitti, että määritettävissä olivat moduulit kirja, kysely sekä tentti, sillä nämä kolme moduulia ovat Mediamaisterin asiakkailla eniten käytössä.

4.2 Käyttöympäristö

Sovellus toteutettiin käytettäväksi Internetin kautta, joten sen käyttämiseen riittää asiakkaan osalta pelkkä WWW-selain. Koska asiakkailla saattaa olla käytössä hyvinkin laaja kirjo erilaisia selaimia, piti sovellusta tehdessä varmistaa, että se noudattaa yleisiä WWW-standardeja. Tämän lisäksi siihen piti tehdä myös joitakin tarkasti kohdistettuja selainkohtaisia määrityksiä, jotta se toimisi oikein myös vanhemmilla WWW-selaimilla, jotka kaikki eivät noudattaneet voimassa olevia standardeja.

Asiakkaat käyttävät sovellusta ottamalla yhteyden selaimellaan palvelimeen, joka Mediamaisterin tapauksessa on Debian Linux -palvelin. Palvelimella on käytössä Apachen WWW-palvelin Internet-palvelujen tarjoamiseen sekä MySql-tietokantapalvelin tiedon säilyttämiseen. Tämä on hyvin yleinen WWW-palvelujen tarjoamiseen käytettävä kokoonpano, joten sovellus olisi tarvittaessa hyvin helppo siirtää palvelimelta toiselle esimerkiksi korvattaessa vanha palvelin uudella ja tehokkaammalla palvelimella.

Sovelluksessa käytettäviä puhtaita URL-osoitteita varten piti Apachessa olla käytössä *mod_rewrite*-moduuli, joka uudelleenkirjoittaa osoitteet perinteisestä muodosta puhtaaseen muotoon ja päinvastoin. Toimintoa varten määritettiin osoitteiden kirjoitussäännöt hakemistokohtaisia asetuksia sisältävään *.htacces*-tiedostoon. Säännöissä määritettiin, että kaikki muut osoitteet paitsi ne, jotka viittaavan suoraan johonkin palvelimella sijaitsevaan tiedostoon, ohjataan *index.php*-tiedostoon, josta *Front Controller* voi selvittää, mille sivulle käyttäjä haluaa päästä käsiksi. Näin esimerkiksi palvelimella sijaitseviin

kuviin pääsee suoraan käsiksi puhtaista osoitteista huolimatta.

Koulutusrekisterin piti olla taaksepäin yhteensopiva vähintään Moodlen 1.7-version kanssa. Oleellista versioiden eroissa oli sovelluksen kannalta pelkästään tietokannan rakenne, sillä sovelluksen tarkoituksena on juuri tietokannasta löytyvän datan avulla generoida erilaisia raportteja ja yhteenvetoja. Tietokannan rakenteen ollessa vääränlainen, ei sovellus pysty suorittamaan siihen oikeanlaisia hakuja, vaan yritykset päättyvät virhetilanteisiin.

Moodlen versiossa 1.7 tuli tietokannan rakenteeseen muutoksia, jotka vaikuttivat merkittävästi käyttäjien rooleihin Moodlen kursseilla, joten aikaisempien versioiden kanssa sovellus ei ole suoraan yhteensopiva. Versiosta 1.7 eteenpäin ei tietokannan rakenteeseen ole kuitenkaan tullut sovelluksen toimintaan vaikuttavia muutoksia. (Naakka 2010)

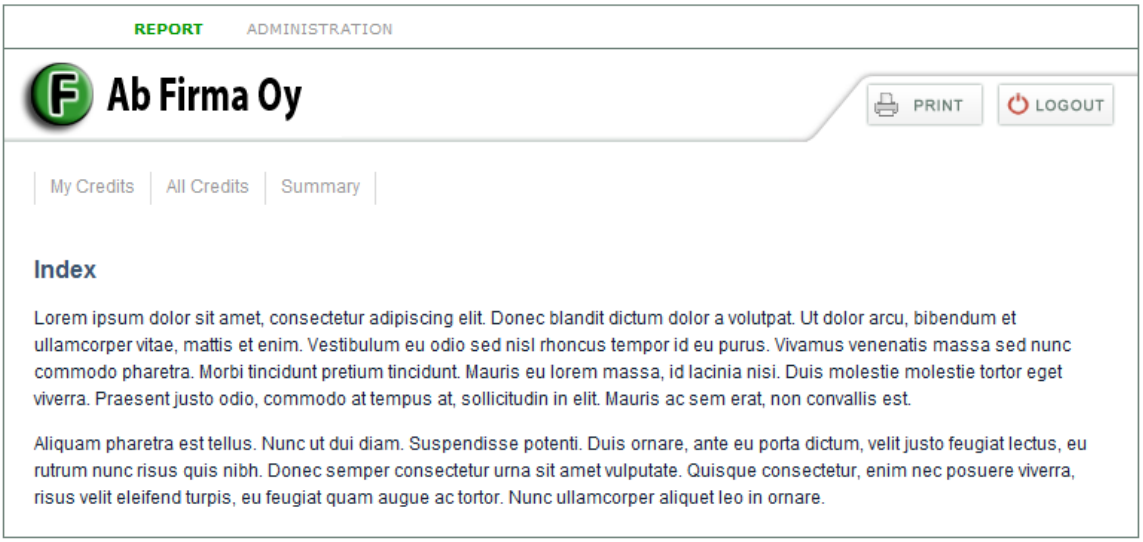
5 Sovelluksen toteuttaminen

Tässä luvussa käydään läpi, miten sovellus rakennettiin, ja millä tavalla se toimii sisäisesti. Lopussa käsitellään, miten sovellus toimi loppukäyttäjän näkökulmasta, ja mitä ominaisuuksia se tarjosi käyttäjilleen.

5.1 Sivupohja

Sovelluksen layout eli sivupohja tehtiin Mediamaisterin valmiista yleisestä sivupohjasta, johon on CSS-tyylitiedostojen avulla helppo vaihtaa teema ja grafiikat vastaamaan kunkin asiakkaan yksilöllistä visuaalista ilmettä. Varsinainen pohja oli suhteellisen lyhyt pääasiassa HTML-koodista koostuva tiedosto, johon lisättiin PHP-skripteillä kutsuja erilaisiin sisältöihin, jotka ohjelmistokirjastolla toteutetut toiminnallisuudet sitä varten generoivat.

Varsinaista bisneslogiikkaa itse sivupohjan koodin sekaan ei siis tullut ollenkaan, ja tällä tavalla selkeästi sisällöstä erotettuna sivupohjaa on helppo muokata ilman, että tarvitsee kiinnittää huomiota sovelluksen toimintaan. Kuviossa 2 on nähtävillä sivupohjan ulkoasu kuvitteellisen yrityksen logolla sekä täytetekstillä varustettuna.

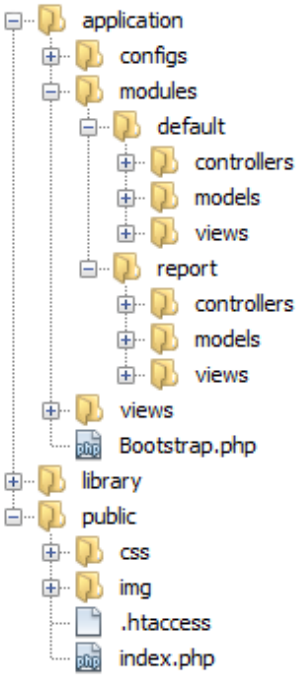


Kuvio 2: Sovelluksen sivupohja

5.2 Hakemistorakenne

Kuten Zend Frameworkin kirjaston elementtien käyttö, myös sovelluksen hakemistorakenne on täysin sovelluksen kehittäjän itse päätettävissä. Siihen, kuten MVC-mallin käyttöönkin, löytyy kuitenkin valmiita suosituksia, joiden mukaisesti sovellusten tiedostot kannattaa sijoittaa. Oleellista tässäkin on erottaa erityyppiset tiedot ja tiedostot omiin selkeisiin ryhmiinsä.

Kuten kuviosta 3 voidaan nähdä, hakemistorakenteen perustana on kolme hakemistoa: *application*, *library* ja *public*. *Application* sisältää kaiken sovelluksen vaatiman koodin, eli konfiguraatiotiedostot sekä MVC-mallin käyttämät tiedostot. Hakemisto *library* sisältää mitäpä muutakaan kuin Zend Frameworkin kirjaston. *Public*-hakemisto taas sisältää kaikki sovelluksen loppukäyttäjälle näkyvät tiedostot, eli kuvat sekä index-tiedoston, jonka kautta kaikki käyttäjälle näytettävä tieto ohjataan.



Kuvio 3: Sovelluksen hakemistorakenne

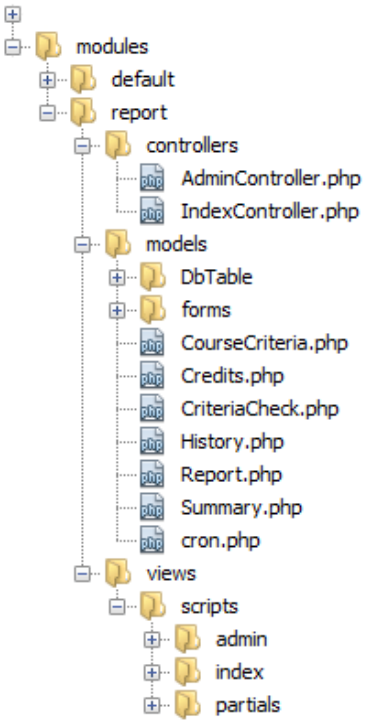
Public-kansio on erotettu omaksi hakemistokseen, sillä tarkoituksena on, että WWW-palvelimelle annetaan pääsy vain tähän hakemistoon. Koska Zend Framework käyttää *Front Controller* -toimintamallia, eli se ohjaa kaikki WWW-pyynnöt vain yhden tiedoston kautta, riittää, että vain tiedosto *index.php* on loppukäyttäjän saatavilla. Tämä lisää tietoturvaa, sillä näin loppukäyttäjä ei voi vahingossakaan päästä näkemään tärkeitä *application*-hakemistossa olevia tiedostoja, kuten salasanoja sisältävää asetustiedostoa.

Hakemistorakenteesta huomaa, että MVC-mallin tiedostoja löytyy kahdesta hakemistosta, eli moduuleista *default* ja *report*. Sovellus päätettiin jakaa kahteen osaan, sillä tulevaisuudessa on mahdollista, että koulutusrekisterin ohelle toteutetaan myös joitakin muita toiminnallisuuksia. Kun sovellukset ovat erillisinä itsenäisinä osinaan, voi niitä tarvittaessa lisätä ja poistaa täysin vapaasti asiakkaan tarpeiden mukaan.

Ainoa pakollinen moduuli on *default*-moduuli, jonne toteutettiin toiminnallisuudet, joita käytetään aina huolimatta siitä, mitä muita moduuleita on käytössä. Toiminnallisuuksia ovat autentikointi sekä virheiden ja käyttäjätietojen hallinta. Käytännössä tämän moduu-

lin kautta käyttäjät pääsevät käsiksi lomakkeisiin, joilla he kirjautuvat sisään sovellukseen ja tekevät muutoksia käyttäjätietoihinsa. Tämän lisäksi myös mahdollisista virhetilanteista johtuvat ilmoitukset esitetään käyttäjälle sen kautta.

Moduulihakemistojen alta löytyvät MVC-mallin mukaisesti omiin ryhmiinsä jaetut tiedostot (Kuvio 4). *Controllers*-hakemiston alta löytyvät ohjain-tiedostot, joihin eri toiminnallisuuksien hallinta jaetaan tavallisesti sen perusteella, mikä niiden käyttötarkoitus on. Tässä tapauksessa hyvältä ratkaisulta tuntui tehdä kaksi erillistä ohjainta. *IndexController*-ohjaimen kautta hoidetaan kaikki kurssien seurantaan liittyvät toiminnot, ja *AdminController*-ohjaimella taas hallinnoidaan kurssikriteerien määrittämiseen tarvittavia toiminnallisuuksia.



Kuvio 4: MVC-tiedostot

Models-hakemistosta löytyvät tiedostot sisältävät koodin, joka CRUD-toimintojen lisäksi suorittaa kaiken sovelluskohtaisen toimintalogiikan. Koulutusrekisterin tapauksessa tämä tarkoittaa lähinnä Moodlen kurssien tilanteen tutkimista. *CriteriaCheck*-luokka esimerkiksi sisältää toimintoja, jotka tarkistavat, ovatko kurssin erilaiset suorituskritee-

rit täyttyneet. *History*-luokka käyttää hyväkseen näitä toimintoja päivittäessään eri käyttäjien kurssihistoriaa. *Summary*-luokka taas kerää yhteen kaiken kursseista tallennetun historiatiedon ja muuttaa sen muotoon, josta *Report*-luokka voi koostaa erilaisia suoritustilannetta havainnollistavia raportteja.

DbTable-hakemisto sisältää yhden tiedoston jokaista sovelluksen tarvitsemaa tietokantataulua kohden. Eri mallitiedostot kutsuvat tarvittaessa näitä tiedostoja, ja saavat niistä selville taulujen nimet ja pääavaimet. Näiden tietojen avulla ne voivat ottaa yhteyden tauluihin ja lisätä, muokata ja poistaa tauluissa olevia tietoja.

Forms-hakemisto sisältää sovelluksessa käytettävät lomakkeet. Hakemistossa olevat tiedot eivät sisällä ollenkaan lomakkeessa tarvittavaa varsinaista koodia, vaan pelkät määrittelyt siitä, millainen lomakkeen tulisi olla. Tämä mahdollistaa lomakkeiden joustavan käytön ja muokkaamisen, sillä eri määrittelyjä voidaan tarvittaessa ylikirjoittaa uusilla määrittelyillä, kun lomake otetaan käyttöön. Tällöin samaa lomakepohjaa voidaan käyttää eri tilanteissa eri tavalla. Lomakemäärittelyihin voidaan lisätä myös määrittelyt lomakkeelta syötetyn tiedon validoinnille sekä virheilmoituksia, jotka tarvittaessa ilmaisevat käyttäjän syöttäneen vääränlaista sisältöä.

Views-hakemisto sisältää sivupohjat, joissa mallien tuottama data muutetaan loppukäyttäjää hyödyttävään muotoon eli tavallisimmin HTML-koodiksi. Jokaista ohjaimissa olevaa toimintoa varten on olemassa oma sivupohjansa, joka generoi datasta halutunlaisen esitysmuodon, kuten taulukon, luettelon tai lomakenäkymän. Tämä generoitu koodi liitetään sitten sovelluksen layout-tiedostoon, jolloin syntyy kokonainen WWW-sivu.

5.3 Tietokanta

Suurin osa sovelluksen toiminnoista käyttää hyväkseen tietoja, jotka löytyvät jo valmiiksi Moodlen tietokannassa olevista tietokantatauluista. Sovellusta varten piti kuiten-

kin luoda neljä uutta taulua Moodlen taulujen ohelle samaan tietokantaan. Nämä taulut ovat *report_role*, *user*, *report_course_criteria* sekä *report_course_history*. Näistä kaksi ensimmäistä piti luoda, jotta saatiin lisättyä Moodlen käyttäjille joitakin uusia tietoja, sekä annettua heille sovellukseen käyttäjärooli. Kaksi jälkimmäistä liittyy kurssien suoritusten seurantaan.

Tässä kappaleessa käsitellään hieman tarkemmin näitä neljää taulua, sekä näiden taulujen suhdetta Moodlen tietokantatauluihin. Itse lisättyjen taulujen rakennetta on kuvattu taulukoilla, joista näkee taulujen sarakkeiden nimet, tietotyypit sekä joitakin tarkempia ominaisuuksia. Tietotyyppi *int* tarkoittaa positiivista kokonaislukua ja *varchar(n)* merkkijonoa, jonka lopussa on kerrottuna merkkien maksimimäärä. *Date* taas on muodossa *yyyy-mm-dd* oleva päivämäärä (esimerkiksi 2010-02-26).

Osassa taulujen sarakkeista on mainittuna ominaisuudet *not null*, *primary key* sekä *auto_increment*. *Not null* tarkoittaa, että kyseistä arvoa ei voi jättää tyhjäksi lisätessä uutta riviä tauluun, ja *primary key* kertoo taulun pääavaimen, jota käytetään rivien tunnistamiseen. Tämän sarakkeen arvo on yksilöllinen, eikä se toistu millään muulla rivillä. *Auto increment* -valinta taas saa sarakkeen arvon kasvamaan automaattisesti yhdellä numerolla aina kun tauluun lisätään uusi rivi.

User-taulu, jonka rakenne näkyy taulukossa 1, piti luoda Moodlen käyttäjätietoja sisältävän taulun ohelle, sillä useimmilla Mediamaisterin asiakkailla on tarve tallentaa työntekijöistään tietoja, joille ei Moodlen käyttäjätaulussa ole valmiiksi omaa sarakettaan. Taulussa oleva id on sama kuin Moodlen käyttäjätaulussa. Tämän yksilöllisen numeron avulla käyttäjien tiedot pystytään yhdistämään näiden kahden taulun välillä.

Taulukko 1: Rakenne tietokantataululle *user*

Nimi	**Tyyppi**	**Ominaisuudet**
id	INT	NOT NULL, PRIMARY KEY
supervisor	INT	
personnelgroup	VARCHAR(255)	
jobdesc	VARCHAR(255)	
unit	VARCHAR(255)	
office	VARCHAR(255)	
costunit	VARCHAR(255)	

Käyttäjätaulun lisäksi täytyi luoda taulu, joka sisältää sovelluksen käyttäjien henkilökohtaiset käyttäjäroolit (taulukko 2). Roolit olisi periaatteessa voinut lisätä *user*-tauluun, mutta niille oli kannattavampaa tehdä oma taulunsa, sillä *user*-taulua saatetaan tulevaisuudessa käyttää myös muissa saman asiakkaan sovelluksissa, joissa rooli ei välttämättä olekaan sama kuin koulutusrekisterissä. Huomaa, että tästä eteenpäin kaikkiin tauluihin on lisätty *report*-etuliite, joka kertoo, että taulut ovat käytössä juuri koulutusrekisterisovelluksessa.

Taulukko 2: Rakenne tietokantataululle *report_role*

Nimi	**Tyyppi**	**Ominaisuudet**
id	INT	NOT NULL, PRIMARY KEY
role	VARCHAR(30)	

Oma taulunsa (taulukko3) luotiin myös kurssien suorituskriteereille, joita asiakasyritysten pääkäyttäjien pitää päästä tallentamaan. *Course*-sarake sisältää id:n Moodlen kurssista, jolle uusi kriteeri halutaan antaa. *Modulename* kertoo, mikä kyseisen kurssin eri tehtävätyypeistä on kyseessä. *Module* on kyseisen kurssimoduulin yksilöllinen id. *Criteria*-sarake taas sisältää kyseiselle tehtävälle annetut suorituskriteerit. Kirjan tapauksessa riittää, että maininta kirjasta löytyy taulusta, mutta tentistä tallennetaan *criteria*-sarakkeeseen tentin minimipistemäärä.

Taulukko 3: Rakenne tietokantataululle *report_course_criteria*

Nimi	**Tyyppi**	**Ominaisuudet**
id	INT	NOT NULL, PRIMARY KEY, auto_increment
course	INT	
modulename	VARCHAR(255)	
module	INT	
criteria	VARCHAR(255)	

Sovelluksen tavoitteena oli pitää kirjaa suoritetuista kursseista ja niiden suorituspäivämääristä. Näitä tietoja varten luotiin taulu *report_course_history* (taulukko 4), jonne tallennetaan käyttäjien kurssihistoria. Taulu sisältää tiedon käyttäjästä, kurssista, jolle käyttäjä on osallistunut, sekä kurssin suorituspäivämäärän. Näiden lisäksi tauluun lisätään myös kurssin nimi, sillä suoritustietojen halutaan säilyvän, vaikka kurssi poistettaisiin Moodlesta. Jos nimeä ei tallennettaisi tauluun, katoaisi se Moodlesta poistettavan kurssin mukana.

Taulukko 4: Rakenne tietokantataululle *report_course_history*

Nimi	**Tyyppi**	**Ominaisuudet**
id	INT	NOT NULL, PRIMARY KEY, auto_increment
user	INT	
course	INT	
coursename	VARCHAR(255)	
passed	DATE	

Kuviossa 5 ovat kaikki sovelluksen tietokantataulut. Kuviosta nähdään eri taulujen suhteet toisiinsa. Keltapohjaiset taulut ovat Moodlen tauluja, ja valkopohjaiset ovat sovellusta varten lisättyjä uusia tauluja. Taulujen välillä olevissa suhteita kuvaavissa viivoissa on käytetty merkkejä, jotka kertovat, moneenko taulun riviin on yhteyksiä viivan toisessa päässä olevista riveistä. Ohessa oleva taulukko 5 selventää eri tilanteita.

Taulukko 5: Tietokantataulujen rivien suhteita kuvaavat merkinnät

Käyttäjä 0...*	Kurssi 0...*	Yhdellä käyttäjällä voi olla 0 tai useita kursseja. Kurssilla voi olla 0 tai useita osallistujia.
Kurssi 1	Tehtävä 0...*	Yhdellä kurssilla voi olla 0 tai useita tehtäviä. Yksittäinen tehtävä liittyy vain yhteen kurssiin.
Käyttäjä (Moodle) 1	Käyttäjä (Koulutusrekisteri) 1	Yhtä Moodlen käyttäjää kohden löytyy yksi rivi sovelluksen omasta käyttäjätaulusta ja päinvastoin.

Kuvio 5: Sovelluksen tietokantakaavio

Tietokanta saattaa ensisilmäyksellä vaikuttaa melko epäselvältä, joten sen rakennetta on hyvä hieman avata. Taulut *mdl_user*, *user* ja *report_role* olivat siis tauluja, joihin on tallennettuna kaikki käyttäjiin liittyvät henkilökohtaiset tiedot. *User* toimii jatkeena

mdl_user-taululle, ja *report_role* sisältää käyttäjän roolin sovelluksessa.

Käyttäjätauluissa olevan henkilönumeron eli *id*-sarakkeen arvon avulla voidaan etsiä *mdl_role_assignments*-taulusta käyttäjälle osoitettuja tehtäviä. Käyttäjät voivat toimia eri tehtävissä eri rooleilla, kuten ylläpitäjänä tai opettajana. Koulutusrekisterin kannalta oleellista on kuitenkin tietää vain, missä tehtävissä käyttäjä on osallistuneena oppilas-roolilla. Tällaiset käyttäjät voidaan tunnistaa *roleid*-sarakkeessa olevasta numerosta viisi.

Taulusta löytyvä *contextid* vastaa *mdl_context*-taulussa olevaa *id*-kenttää. Tässä taulussa määritetään, missä kontekstissa käyttäjälle annettu rooli on voimassa. Eri konteksteja voivat olla esimerkiksi koko Moodle-sivusto, yksittäinen kurssi tai yksittäinen tehtävänanto kurssin sisällä. Kurssia vastaa kontekstitaso (*contextlevel*) 50, ja tällaiselta riviltä löytyvä *instanceid* taas vastaa kurssitaulussa (*mdl_course*) olevan kurssin *id*:tä. (Naakka 2009)

Edellä mainittujen taulujen avulla on siis mahdollista selvittää, missä kurssi-tason tehtävänannoissa kukin käyttäjä on osallistuneena oppilas-tason roolilla. Koulutusrekisterin tehtävänä on helpottaa juuri kurssien edistymisen seurantaa, joten tätä tietoa tarvitaan lähes kaikissa sovelluksen toiminnallisuuksissa.

Oleellista on myös tietää, mitä tehtäviä kuhunkin kurssiin liittyy. Tämä tieto löytyy tauluista *mdl_quiz*, *mdl_book* sekä *mdl_questionnaire*. Näissä tauluissa olevat tiedot ovat kurssimoduuleita eli kursseihin liitettyjä yksittäisiä tehtäviä, joita kurssin osallistujat pääsevät tekemään. Kussakin kurssimoduulitaulussa oleva *course*-sarakkeen arvo vastaa jotakin *mdl_course*-taulussa määritettyä kurssia. Tämän tiedon avulla saadaan selville, mille kursseille eri moduulit on liitetty.

Erilaisia moduuleita on olemassa enemmänkin, mutta sovelluksen ensimmäiseen versioon ei ollut tarvetta ottaa mukaan kuin edellä mainitut kolme moduulia. Nämä moduulit ovat siis *tentti*, *kysely* sekä *kirja*.

Kun käyttäjä suorittaa kyselyä tai tenttiä, tallentuvat tiedot yrityksestä m*dl_quiz_attempts*- ja *mdl_questionnaire_attempts*-tauluihin. Näistä tauluista siis nähdään, joko käyttäjä on esimerkiksi yrittänyt tenttiä, ja paljonko pisteitä hän siitä on saanut. Näitä tietoja käytetään hyväksi, kun sovellus tarkistaa, onko käyttäjä saanut tentistä kurssin läpäisemiseen vaaditun pistemäärän, tai joko käyttäjä on käynyt vastaamassa kyselyyn.

Yhtenä mahdollisena kurssikriteerinä oli myös *book*-moduuli. Tällaisen online-kirjan lukemisesta ei kuitenkaan tallennu tietoa samalla tavalla kuin tentin ja kyselyn suorittamisesta. Tämän vuoksi tieto kirjan lukemisesta täytyy tarkistaa *mdl_log*-taulusta eli Moodlen lokitiedoista. Tässä apuna käytetään *mdl_course_assignments*-taulua.

Tästä taulusta etsitään rivi, jolla *course*-sarake vastaa halutun kurssin id:tä, kurssimoduulina on *book* ja *instance* vastaa halutun kirjan id:tä. Tällaisen rivin id taas vastaa Moodlen lokista löytyvää *cmid*-saraketta (lyhenne sanoista *course module id*). Jos lokista löytyy id:n sisältävä rivi, jossa kurssi, kurssin osallistuja sekä moduulityyppi vastaavat etsittyä tietoa, ja *action*-sarake sisältää toiminnon ”*view*”, tarkoittaa tämä, että käyttäjä on käynyt lukemassa kirjan. (Nieminen 2009)

Moodlen lokiin kirjataan ylös tiedot kirjan lukemisen lisäksi monista muista tapahtumista, eli sen päivitystahti on hyvin nopeaa. Lokin koko ei kuitenkaan ole rajaton, vaan jossakin vaiheessa lokissa olevat tiedot poistuvat tehden tilaa uusille lokimerkinnöille. Tämän vuoksi piti sovelluksen ohelle toteuttaa toiminto, joka tasaisin väliajoin päivittäisi koko kurssihistorian.

Ajastustoimintoa ei voinut toteuttaa sovellukseen itseensä, vaan se lisättiin palvelimelle, jonne sovellus asennettiin. Ajastamiseen käytettiin Cron-nimistä ohjelmaa, johon voi määrittää erilaisia komentoja ajettavaksi haluttuun aikaan. Koska päivitystoiminto käy läpi kaikki Moodlen käyttäjät, vaatii sen suorittaminen paljon aikaa sekä palvelimen resursseja. Tämän vuoksi se ajastettiin suoritettavaksi keskellä yötä, koska tällöin palvelimella ei ole käyttäjiä, joita palveluiden mahdollinen hidastuminen voisi haitata.

Tiettyyn aikaan yöllä palvelin siis suorittaa sovelluksesta löytyvän toiminnallisuuden, joka käy läpi kaikkien käyttäjien eri kurssien tilanteen ja päivittää niihin liittyvät tiedot. Moodlen lokissa olevat tiedot eivät siis ajastuksen ansiosta pääse vanhenemaan ja poistumaan lokista ennen kuin ne on otettu huomioon kurssien suoritushistoriassa.

5.4 Kirjautumisistunnot

Sovellus oli tarkoitus integroida Moodlen oheen niin kiinteästi, että loppukäyttäjä ei huomaa siirtymistä Moodlen ja sovelluksen välillä. Käytännössä tämä tarkoittaa, että kertaalleen Moodleen kirjauduttuaan voi käyttäjä siirtyä Moodlen ja sovelluksen välillä ilman, että hänen täytyy siirtyessään kirjautua aina uudelleen sisään. Tämä toteutettiin käyttämällä hyväksi Moodlen kirjautumisistuntoja.

Käyttäjän kirjautuessa sisään WWW-pohjaiseen sovellukseen, tallennetaan tieto onnistuneesta kirjautumisesta tavallisesti niin sanottujen evästeiden avulla. Evästeet ovat pieniä tiedostoja, jotka selain tallentaa käyttäjän tietokoneelle. Ne voivat itsessään sisältää kaikki tarvittavat kirjautumiseen liittyvät tiedot, tai sitten tiedot voidaan tallentaa palvelimelle, jolloin eväste sisältää ainoastaan istunnon tunnuksen. Tämä tunnuksen avulla voidaan käyttäjä yhdistää palvelimella oleviin tietoihin, eikä mitään tiedoista tarvitse tallentaa käyttäjän omalle tietokoneelle.

Moodle käyttää kirjautumisistuntoihin jälkimmäistä menetelmää, eli tiedot kirjautumisista tallennetaan palvelimelle. Koska koulutusrekisterisovellus sijaitsee samalla palvelimelle kuin Moodle, pystyy sovellus käyttämään hyväkseen näitä Moodlen jo valmiiksi tallentamia tietoja saadakseen selville, kuka sovellukseen haluaa päästä sisään.

Oletuksena Moodle tallentaa kirjautumistiedot palvelimella olevaan tiedostoon, mutta asetuksia säätämällä ne on mahdollista saada tallentumaan vaihtoehtoisesti myös Moodlen tietokantaan. Tietoihin käsiksi pääseminen oli helpompaa tietokannan kautta, joten

asetukset säädettiin tämän mukaiseksi. Istuntodata on tallennettuna molempiin paikkoihin serialisoituna ja URL-koodattuna.

Serialisointi on tekniikka, jolla voidaan generoida PHP-muuttujia tallennettavaan muotoon. Se konvertoi muuttujat merkkijonoiksi, joihin jää varsinaisen arvon lisäksi talteen myös muuttujan tyyppi ja rakenne (The PHP Group, 2009). URL-koodaus taas muuttaa merkkijonoiksi URL-osoitteissa olevat erikoismerkit, jotka alkuperäisessä muodossaan saattaisivat olla yhteen sopimattomia HTML-koodin kanssa.

Purkamalla tietokannasta löytyvästä merkkijonosta nämä kaksi toimintoa, päästään käsiksi Moodlen luomaan objektiin, joka sisältää muun muassa kirjautuneen henkilön yksilöllisen käyttäjätunnuksen ja käyttäjä-id:n, joiden avulla käyttäjä voidaan tunnistaa. Sekä serialisointiin että URL-koodaukseen löytyvät valmiit funktiot PHP:sta, mutta jostakin syystä Moodlessa oli käytetty serialisointiin itse toteutettua funktiota, mikä vaati Moodlen lähdekoodin tutkimista, ja koodauksen purkamiseen tehdyn funktion kopioimista alkuperäisessä muodossaan koulutusrekisterin koodiin.

Koulutusrekisterin kirjautumisprosessi toimii siis seuraavasti: siirtyessään Moodlesta sovellukseen, sovellus tarkistaa, löytyykö käyttäjän koneelta Moodlen tekemää evästettä. Jos eväste löytyy, etsii sovellus palvelimelta tätä vastaavan istuntodatan, avaa sen luettavaan muotoon ja ottaa sovelluksen käyttöön datasta löytämänsä käyttäjätiedot. Näiden tietojen avulla voidaan tarkistaa käyttäjän id:tä vastaava käyttäjärooli, ja tämän perusteella valita näytettäväksi vain kaikki kyseiselle roolille sallittu sisältö.

Jos evästettä kuitenkaan ei löydy, tarkoittaa tämä, että käyttäjä ei ole siirtynyt sovellukseen Moodlen kautta, vaan esimerkiksi kirjoittamalla sen osoitteen suoraan selaimen osoiteriville. Tätä tilannetta varten piti sovellukseen toteuttaa myös aivan oma kirjautumissivu, jotta kirjautuminen onnistuisi myös tässä tilanteessa. Jos evästettä siis ei löydy, ohjataan käyttäjä sovelluksen omalle kirjautumissivulle.

Koska sovellus on samalla palvelimella ja käyttää samaa tietokantaan kuin Moodle, pystyy se tarkistamaan käyttäjän antaman tunnus-salasana-parin myös itsenäisesti suoraan tietokannasta, ja aloittamaan näiden perusteella oman kirjautumisistunnon.

Tarvetta omalle kirjautumissivulle on myös, jos istunnon aikana tapahtuu aikakatkaisu. Tämä on turvallisuustoimenpide, joka katkaisee istunnon, jos käyttäjä on liian pitkään passiivisena. Sen tarkoituksena on välttää esimerkiksi tilanne, jossa käyttäjä lähtee pois tietokoneelta ja unohtaa istunnon aktiiviseksi. Tällöin kuka tahansa paikalle sattuva voisi ilman aikakatkaisua käyttää hyväkseen istuntoa ja päästä käsiksi käyttäjän tietoihin.

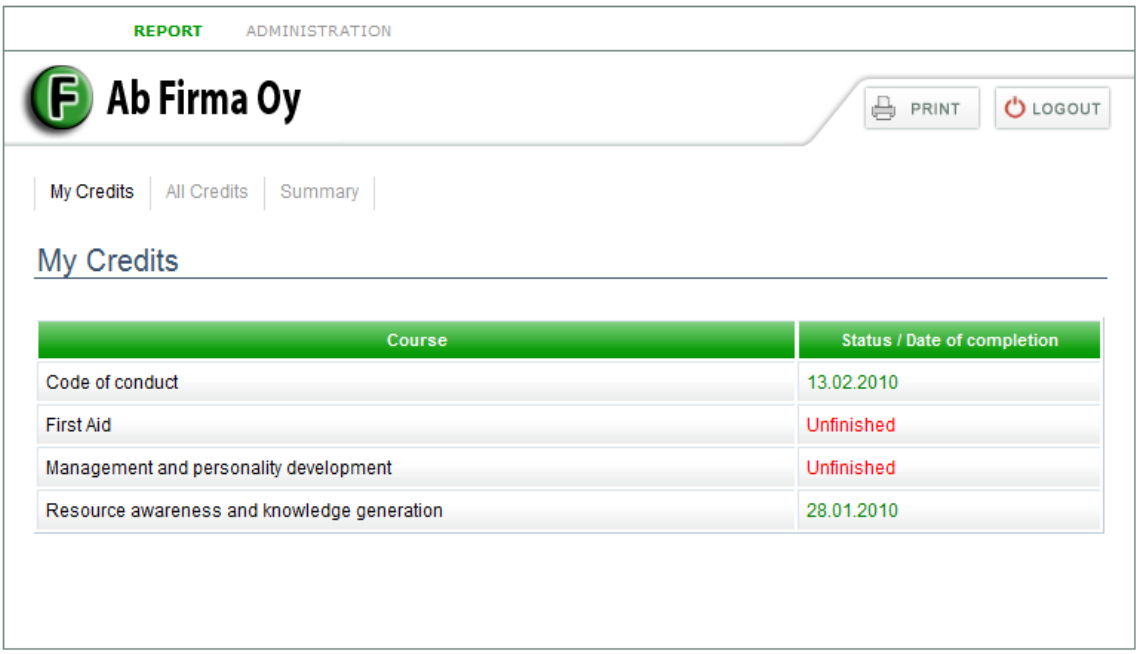
Oman kirjautumissivun ansiosta käyttäjän ei aikakatkaisun jälkeen tarvitse palata Moodleen, kirjautua siellä sisään, ja sitten siirtyä uudelleen sovellukseen, vaan hän voi palata suoraan jatkamaan siitä, mihin jäi ennen aikakatkaisun tapahtumista.

5.5 Roolit ja resurssit

Edellisessä alaluvussa käsiteltiin käyttäjien tunnistusta. Tunnistaminen on tarpeellista, koska sovelluksen eri käyttäjillä on erilaisia rooleja, joista jokaisella on oikeuksia vain tiettyihin tietoihin ja toiminnallisuuksiin. Zend Frameworkin yhteydessä näistä käytetään termiä *resurssi*. Jokaisen sivulatauksen yhteydessä sovellus tarkistaa kirjautumisistunnon tiedoista käyttäjän roolin ja generoi näkyviin vain kyseiselle roolille sallitut resurssit. Sovelluksen loppukäyttäjinä ovat yritysorganisaatiot, joten roolit piti määrittää tällaiselle organisaatiolle sopiviksi.

Rooleja tuli yhteensä kolme. Nämä roolit ovat *peruskäyttäjä* (employee), *esimies* (supervisor) sekä *pääkäyttäjä* (key-user). Peruskäyttäjän roolia käyttävät yrityksen tavalliset työntekijät, ja esimies-roolia käyttävät luonnollisesti työntekijöiden esimiehet. Kaikkein eniten oikeuksia sisältävät pääkäyttäjän oikeudet annetaan vain yritysten HRD-henkilöille, koska heidän täytyy olla perillä koko yrityksen henkilöstön tiedoista.

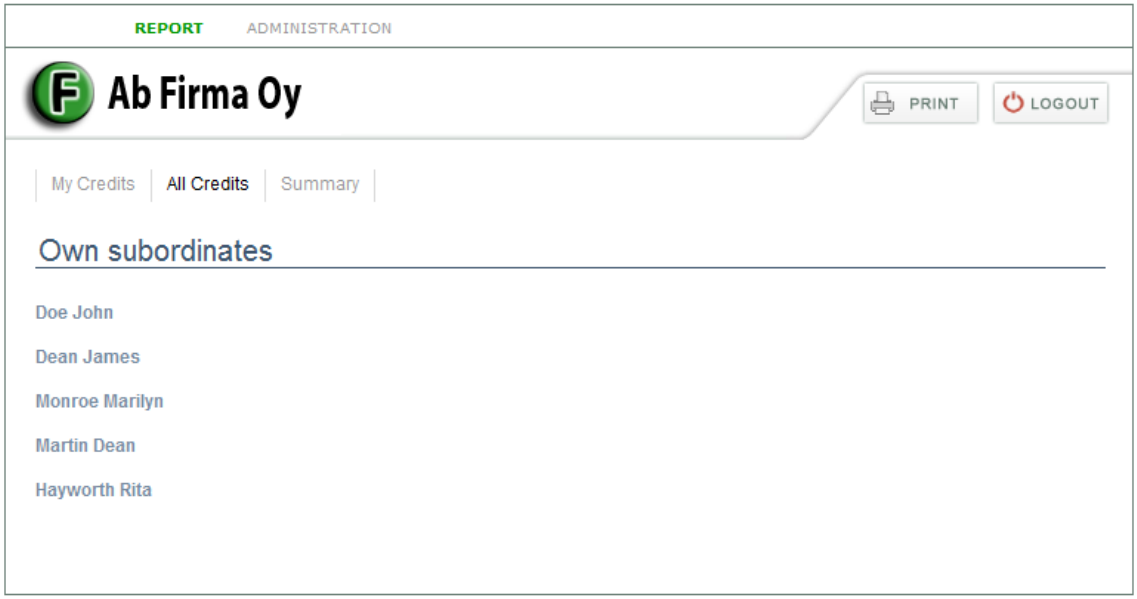
Eri käyttäjien pääsy eri resursseihin toteutettiin piilottamalla tietyt sivut eri rooleilta. Tämä toteutettiin Zend Frameworkin pääsylistatoiminnoilla, jotka piilottavat automaattisesti sivun navigaatiosta linkit, joihin käyttäjän roolilla ei ole pääsyoikeutta. Kuviossa 6 on näkyvissä sovelluksen navigaatio kokonaisuudessaan. Ylhäällä on esillä päänavigaatio, ja yrityksen logon alla raporttisivun alanavigaatio.



Kuvio 6: Käyttäjän oma kurssihistoria

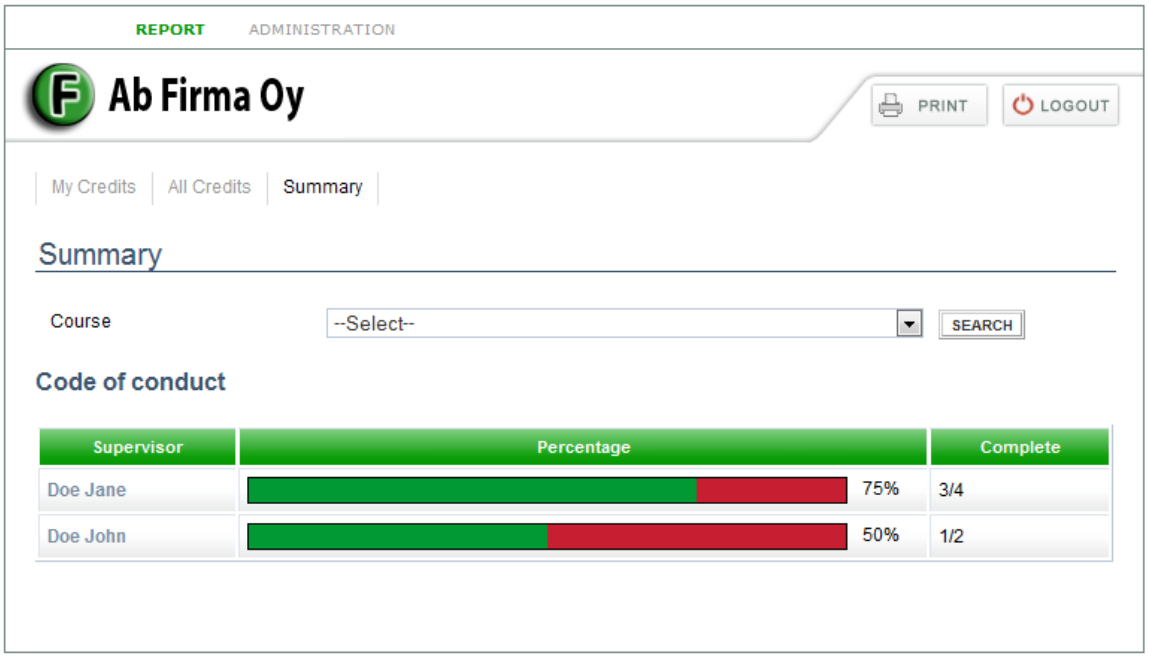
Peruskäyttäjä pääsee sovelluksessa käsiksi vain omaan kurssihistoriaansa, joten käytettäessä sovellusta tällä roolilla, ei käyttäjälle ole ollenkaan näkyvissä muita linkkejä kuin *Report* ja tämän alla linkki *My Credits*. Kurssihistoriassa on listattuna kaikki ne kurssit, joille käyttäjä on osallistuneena. Toisessa sarakkeessa näkyy joko päivämäärä, jolloin kurssi on suoritettu tai maininta siitä, että kurssi on vielä kesken.

Esimiesroolilla kirjautuneena oleva käyttäjä pääsee oman kurssihistoriansa lisäksi näkemään *All Credits* -sivun (kuvio 7), jolla on listattuna kaikki henkilön alaiset. Listassa olevien henkilöiden nimet toimivat linkkeinä, joista esimies pääsee näkemään alaistensa kurssihistorian samalla tavalla kuin omansakin.



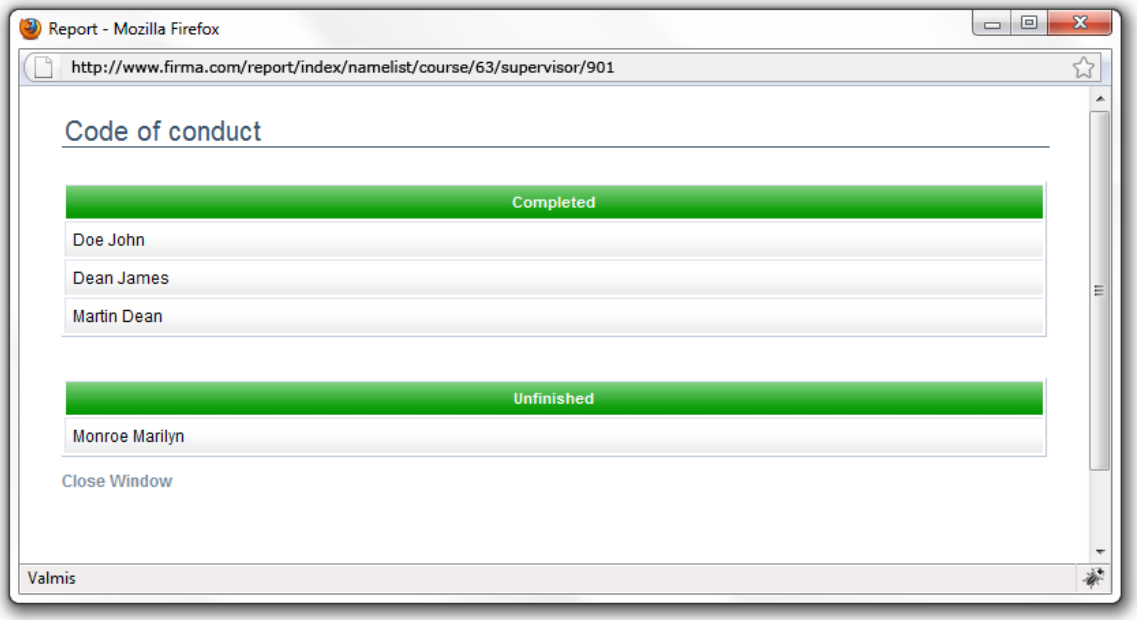
Kuvio 7: Lista esimiehen alaisista

Kuviossa 8 olevan *Summary*-sivun tarkoituksena on näyttää yhteenvetoja esimiehittäin alaisten kurssien suoritustilanteesta. Sivulla on valintalaatikko, josta pääsee valitsemaan haluamansa kurssin. Valinnan jälkeen avautuu lista, jossa on esimiehittäin sekä graafinen, prosentuaalinen sekä lukumäärällinen esitys siitä, kuinka moni henkilön alaisista on suorittanut valitun kurssin.



Kuvio 8: Yhteenveto kurssin suoritustilanteesta esimiehittäin

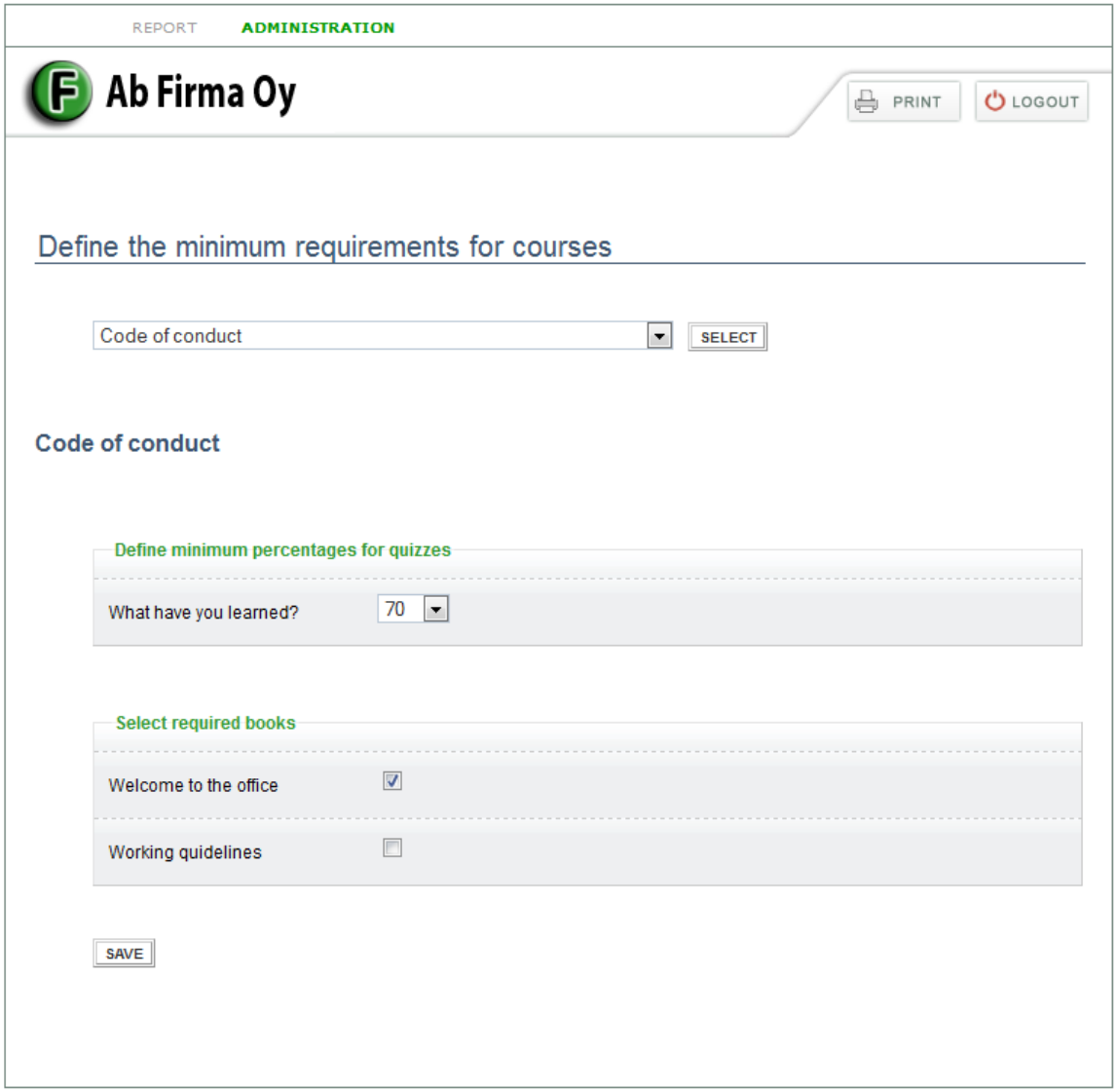
Myös tällä sivulla henkilöiden nimet toimivat linkkeinä, joista pääsee näkemään tarkempaa tietoa. Linkeistä aukeaa kuviossa 9 näkyvä nimilista, jossa on kerrottuna, ketkä alaisista ovat suorittaneet kurssin, ja keillä se on vielä kesken. *Key-user*-rooli pääsee näkemään kyseisen listan jokaisen esimiehen kohdalta, mutta esimiehillä aktiivisena on vain linkki heidän omien alaistensa listaukseen.



Kuvio 9: Yhteenvedosta aukeava nimilista

Edellä mainituissa näkymissä on nähtävillä aina ajantasainen tieto, sillä sovellus päivittää jokaisella sivunlatauksella kurssien tilanteen. Se tarkistaa tietokannasta, mille kursseille käyttäjä on ilmoittautuneena, mitä pakollisia tehtäviä kurssiin kuuluu, ja joko käyttäjä on suorittanut tämä tehtävät. Mikäli uusia suorituksia tai osallistumisia löytyy, tallennetaan ne talteen kurssihistoriaan.

Raporttiominaisuuksien lisäksi sovelluksesta pääsee myös määrittämään, mitä kursseihin kuuluvia tehtäviä eri kursseilla pitää suorittaa ennen kuin ne katsotaan suoritetuiksi. Tämä on mahdollista ylläpitosivulla (kuvio 10), jonne on pääsy vain key-user-roolilla. Sivulta pääsee valitsemaan kurssin, jonka jälkeen sovellus listaa kaikki kyseisellä kurssilla olevat erilaiset tehtävät, ja antaa käyttäjän määrittää pakolliset osiot tai esimerkiksi tentin prosentuaalisen minimipistemäärän.



Kuvio 10: Kurssin pakollisten suorituskriteerien määrittäminen

Kriteerien tallentamisen jälkeen sovellus käyttää tietoja hyväkseen aina tarkistaessaan kurssien suoritustilannetta. Jokainen määritetty kriteeri tarkistetaan erikseen, ja kurssi lasketaan suoritetuksi vasta, kun kaikki kriteerit on hyväksytysti suoritettu.

6 Tulokset

Ohjelmistoprojektin tuloksena syntyi Mediamaisterin tuotevalikoimaan uusi sovellus, joka saavutti kaikki asiakasyritysten toiveet. Sovellus helpotti merkittävästi yritysten henkilöstöjohtajien työtä, sillä nyt kaikkia kursseja pääsi seuraamaan helposti yhdestä paikasta. Sovellus antoi selkeän ja havainnollisen kuvan kurssien edistymisestä kokonaisuutena, mutta antoi myös tarkempia tietoja yksittäisistä kursseista ja niiden osallistujista.

Sovelluksen avulla pääsi myös määrittämään kurssikohtaisesti aktiviteettimoduulit, jotka olivat pakollisia kurssin hyväksytyn suorituksen kannalta. Tämä toi lisäarvoa varsinkin kirja-moduulia ajatellen, sillä Moodlessa ei ollut valmiina ominaisuutta, jolla olisi voinut seurata, ovatko käyttäjät käyneet lukemassa kurssiin liittyvän kirjan.

Sovellus toi lisäarvoa myös yritysten työntekijöille, jotka pääsivät sovelluksen kautta näkemään oman koulutushistoriansa. Vastaavasti hyötyivät esimiesroolissa olevat, jotka pääsivät oman kurssihistoriansa lisäksi näkemään myös alaistensa tilanteen. Koska kurssihistoria toteutettiin omaksi itsenäiseksi tietovarastokseen, säilyvät tiedot suoritetusta kurssista tallessa, vaikka varsinainen kurssi poistettaisiinkin Moodlesta.

Hankalinta sovellusta tehdessä oli keksiä, miten jakaa erilaiset toiminnallisuudet omiksi Zend Frameworkin malleikseen. Lopulta kaikelle löytyi kuitenkin oma paikkansa. Yksi malli suoritti pakollisten kurssimoduulien hallinnoinnin, ja tämän tallentamia tietoja käytti hyväkseen malli, jonka tehtävänä oli tarkistaa yksittäisten moduulien suoritustilanteita. Suoritustilanteiden tarkastamista taas käytti hyväkseen malli, joka päivitti historiaa tarpeen mukaan joko henkilöittäin, esimiehittäin tai kursseittain. Näiden prosessin tuottamista historiatiedoista taas loi erilaisia yhteenvetoja ja raportteja siihen tarkoitettu oma mallinsa.

Kun eri toiminnallisuudet ovat jaettuina näin eri malleihinsa, on niitä tarvittaessa helppo päivittää ja kehittää pidemmälle. Yksi kehitysmahdollisuus sovellukselle olisi esimer-

kiksi uusien kurssimoduuleiden lisääminen. Tämä olisi helppoa, sillä uusi moduuli tarvitsisi lisätä mukaan vain kurssimoduulien hallinnasta vastaavaan malliin sekä malliin, joka tarkistaa moduuleiden suoritustilanteen. Muut mallit ovat täysin riippumattomia näiden kahden mallin toiminnasta, joten niihin ei tarvitsisi tehdä minkäänlaisia muutoksia.

Toinen mieleen tuleva kehitysmahdollisuus olisi monipuolisemman seurannan tekeminen. Seurannassa voisi esimerkiksi olla oma sivunsa, joka näyttäisi kurssikohtaisesti, montako pakollista kurssimoduulia kukin kurssin osallistujista on suorittanut. Tällaista toiminnallisuutta varten ei tarvitsisi toteuttaa kuin uusi malli, joka generoisi raportteja asiasta. Kurssimoduuleiden suoritustilanteet tarkistava mallihan olisi jo valmiiksi olemassa.

Sovelluksessa olisi siis myös potentiaalia kasvaa tarkemmaksi ja monipuolisemmaksi. Joitakin rajoituksia sen käyttömahdollisuuksissa kuitenkin on. Sovellus todennäköisesti olisi hieman liian tehoton ja yksinkertainen käytettäväksi oppilaitoksissa, koska niissä kurssit ovat usein huomattavan monimutkaisia, ja usein kurssille on sekoitettu keskenään Moodlessa tapahtuvaa ja muuta opetusta/materiaalia. Lisäksi opiskelijoiden ja kurssien määrä ja vaihtuvuus ovat huomattavasti suuremmat, minkä vuoksi niitä tulisi päästä hallinnoimaan paljon monipuolisemmin.

Yritysorganisaatioiden käyttötarpeita ajatellen sovelluksesta tuli kuitenkin kaiken kaikkiaan sopivan kattava ja toimiva tapa seurata koulutuksia. Koska sovellus on suoraan yhteensopiva kaikkien versiota 1.7 uudempien Moodle-versioiden kanssa, on se myös hyvin yleisluontoinen ratkaisu ja helposti otettavissa käyttöön useammallakin asiakkaalla.

Myös opinnäytetyö oli onnistunut. Zend Frameworkin perusidea ja -toiminta kävivät selväksi, ja lisäksi käsiteltiin riittävän tarkasti myös koulutusrekisterisovelluksen rakennetta ja toimintaa. Lisäksi saatiin hyvä kuva siitä, miten sovelluksen loppukäyttäjät hyötyivät sovelluksesta.

Lähteet

Allen, Rob, Lo, Nick & Brown, Steven 2009. Zend Framework in Action. Manning Publications Co.

Berners-Lee, Tim 1998. Cool URIs don't change. [www-sivu][viitattu 28.12.2009]. http://www.w3.org/Provider/Style/URI

Cole, Jason 2005. Using Moodle. O'Reilly Community Press.

Evans, Cals 2008. Guide to programming with Zend Framework. Marco Tabini & Associates

Moodle.org 2008. MoodleDocs [www-sivu] [viitattu 13.3.2010]. http://docs.moodle.org/fi/Aktiviteetit-lohko

Naakka, Jaakko 2009. IT-softare specialist. Sähköpostiviesti 3.6.2009. Mediamaisteri Group Oy.

Nieminen, Arto 2009. IT-softare specialist. Haastattelu 10.12.2009. Mediamaisteri Group Oy.

Rice, William H. 2006. Moodle – E-Learning Course Development. Packt Publishing Ltd.

Sanastokeskus TSK ry 1999. Selain. [www-sivu][viitattu 7.2.2010]. http://www.tsk.fi/tepa

Sanastokeskus TSK ry 2001. Eväste. [www-sivu][viitattu 7.2.2010]. http://www.tsk.fi/tepa

Sanastokeskus TSK ry 2007. XHTML. [www-sivu][viitattu 7.2.2010]. http://www.tsk.fi/tepa

Sanastokeskus TSK ry 2009. SOAP. [www-sivu][viitattu 7.2.2010]. http://www.tsk.fi/tepa

The PHP Group 2009, PHP: Hypertext Preprocessor. [www-sivu][viitattu 11.10.2009]. http://www.php.net/

Wikipedia 2009, Ohjelmistokehys. [www-sivu] [viitattu 9.12.2009]. http://fi.wikipedia.org/wiki/Ohjelmistokehys

Wikipedia 2010a, Aliohjelma. [www-sivu] [viitattu 25.3.2010]. http://fi.wikipedia.org/wiki/Aliohjelma

Wikipedia 2010b, PDF. [www-sivu] [viitattu 7.2.2010]. http://fi.wikipedia.org/wiki/PDF

Wikipedia 2010c, RSS. [www-sivu] [viitattu 7.2.2010]. http://fi.wikipedia.org/wiki/Rss

Wikipedia 2010d, Tietokanta. [www-sivu] [viitattu 25.3.2010]. http://fi.wikipedia.org/wiki/Tietokanta

Wikipedia 2010e, XML-RPC. [www-sivu] [viitattu 22.2.2010]. http://en.wikipedia.org/wiki/XML-RPC

Wikipedia 2010f, Moodle. [www-sivu] [viitattu 11.3.2010]. http://fi.wikipedia.org/wiki/Moodle

Wikipedia 2010g, Front Controller pattern. [www-sivu] [viitattu 2.1.2010]. http://en.wikipedia.org/wiki/Front_Controller_pattern

Zend Technologies Ltd. 2010. Zend Framework: About [www-sivu][viitattu 15.2.2010]. http://framework.zend.com/about/numbers