



REKRYTOINTIREKISTERI TIETOJÄR- JESTELMÄN KEHITTÄMINEN JA YLLÄ- PITO

Teemu Halmela

Opinnäytetyö
Huhtikuu 2015
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto

TEEMU HALMELA:

Rekrytointirekisteri tietojärjestelmän kehittäminen ja ylläpito

Opinnäytetyö 54 sivua, joista liitteitä 5 sivua
Huhtikuu 2015

Rekrytointirekisteri on Tampereen ammattikorkeakoulun liiketalouden linjalle rakennettu opiskelijoiden rekrytoinnin apuväline. Järjestelmään tallennetaan muun muassa käyttäjien henkilötiedot, heidän vahvuudet sekä suoritettut kurssit, joita vastuhenkilö pystyy myöhemmin selaamaan sekä hakemaan. Sen arkkitehtuuri perustuu MVC-malliin, ja tiedot tallennetaan tietokantaan.

Linux-palvelimella toimiva MVC-arkkitehtuuriin perustuva runko on toteutettu PHP-ohjelmointikielellä. Järjestelmän MySQL-tietokantaa käytetään ORM-kirjaston kautta. Käyttöliittymän ulkoasu on toteutettu HTML:n ja CSS-määrittelyjen avulla ja sen dynaaminen toiminnallisuus JavaScript-kielellä. Git versionhallinta pitää lokia järjestelmän koodin muutoksista sekä auttaa sen käyttöönotossa. Työkalut kuten phpMyAdmin ja Internet-selainten debug-tilat ovat suuressa osassa järjestelmän ylläpitoa ja kehitystä.

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT engineering
Option of Software Engineering

TEEMU HALMELA:

Development and Maintenance of Rekrytointirekisteri Information System

Bachelor's thesis 54 pages, appendices 5 pages

April 2015

Rekrytointirekisteri is an information system developed for Tampere University of Applied Sciences. Its main purpose is to help students of the business administration to get jobs. Personal info, their strengths and courses of the users among other, are being stored in the system. Person in charge is able to browse and search this data. Architecture of the system is based on a MVC-model and data is stored in a database.

MVC-architecture based framework, implemented in PHP programming language is running on a Linux server. MySQL database of the system is being used via an ORM-library. User interface is implemented with HTML and CSS and its dynamic functionality in JavaScript. Git version control logs the changes in the code and helps in deployment of the system. Tools like phpMyAdmin and debug modes of the Internet browsers are in big part of the systems upkeep and development.

Key words: information system, database, PHP, MySQL, JavaScript

SISÄLLYS

1	JOHDANTO.....	6
2	REKRYTOINTIREKISTERIN ESITTELY	8
3	JÄRJESTELMÄN OSAT.....	11
	3.1 Avoin lähdekoodi.....	11
	3.2 Tietojen tallennus.....	12
	3.3 Ohjelmointikielet	12
	3.4 Palvelin	13
	3.5 Yhteenveto	13
4	ARKKITEHTUURI	15
	4.1 MVC	15
	4.2 ORM	16
	4.3 Rakenne.....	17
5	TIETOKANNAN RAKENNE	18
6	JÄRJESTELMÄN RUNGOT	22
	6.1 MVC-malli.....	22
	6.2 Tietokanta	23
	6.3 Käyttöliittymä	23
7	KEHITYSTYÖKALUT	25
	7.1 Kehitysympäristö	25
	7.2 Versionhallinta.....	26
	7.3 Internet-selaimen debug-työkalut	28
8	REKRYTOINTIREKISTERIN TOIMINTA	31
	8.1 Käyttäjien salasanat	31
	8.2 Käyttäjätilit ja niiden hallinta.....	31
	8.3 Tietojen tallennus järjestelmään	33
	8.4 Käyttäjätietojen selaus ja haku.....	38
9	JÄRJESTELMÄN KÄYTTÖÖNOTTO	42
10	JÄRJESTELMÄN YLLÄPITO.....	45
	10.1 Tietokannan hallinta.....	45
	10.2 Palvelimen hallinta.....	46
	10.3 Järjestelmän siirto	46
11	POHDINTA.....	48
	LÄHTEET	49
	LIITTEET	50
	Liite 1. Käyttötapaus – Muokkaa tietoja	50
	Liite 2. Rekrytointirekisterin kirjastot	51

Liite 3. Rekrytointirekisterin osat.....	52
Liite 4. Rekrytointirekisterin rakenne.....	53
Liite 5. Tietokannan ER-kaavio	54

1 JOHDANTO

Rekrytointirekisteri on web-käyttöliittymällä varustettu tietojärjestelmä, joka käyttää tietokantaa tietojen talletukseen ja käsittelyyn. Rekisteri kehitettiin liiketalouden koulutusohjelmalle vuonna 2014 tietotekniikan oppilaiden toimesta. Rekrytointirekisterin vastuuhenkilö Jenni Viinanen (2014) kertoo kuinka sen tarkoituksena on luoda liiketalouden koulutusohjelmalle opiskelijoiden rekrytointiin tarkoitettu väline. Sen päämääränä on auttaa opiskelijoiden työllistymistä sekä tiivistää yhteistyötä yrityselämän ja TAMKin välillä. Opiskelijat voivat halutessaan lisätä tietojaan rekisteriin, josta tallennettuja tietoja voidaan selata kootusti. Tietoja voi järjestää tiettyjen kriteerien mukaan ja sopivan henkilön löytyminen tarjottuun työtehtävään on helpompaa. (Viinanen 2014, 9.)

Internet-selaimessa toimivan tietojärjestelmän etuina, verrattuna natiivi sovellukseen on sen käyttöjärjestelmäriippumattomuus sekä käytön helppous. Käyttäjän ei tarvitse asentaa ylimääräisiä ohjelmia koneelleen vaan järjestelmä sijaitsee jossain palvelimella johon otetaan yhteys omalta koneelta. Tämän vuoksi myös järjestelmän päivittäminen helpottuu ja uudet versiot saadaan käyttäjille saumattomasti.

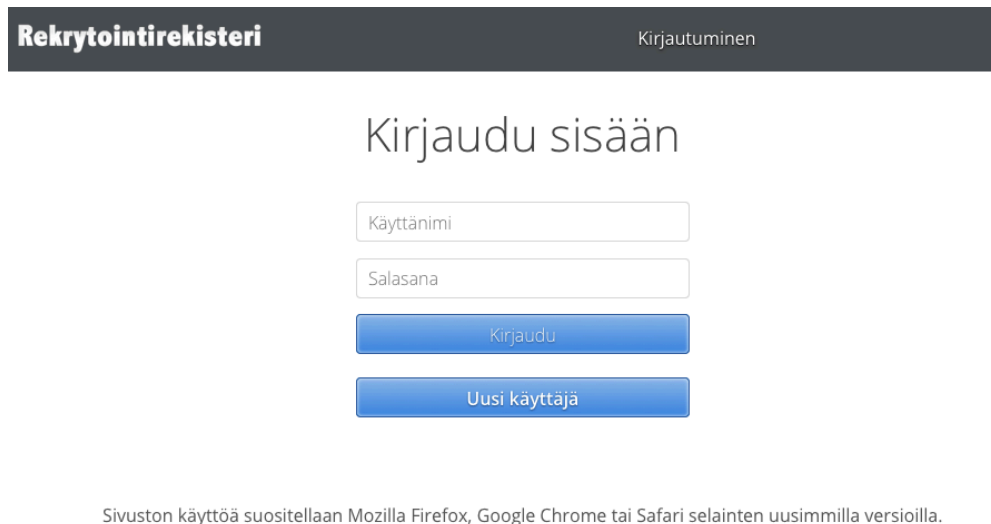
Rekrytointirekisterin, työnimeltään datapankki, järjestelmän kehityksen aloitti Roope Thomsson vuoden 2014 alussa. Allekirjoittanut tuli projektiin mukaan maaliskuussa 2014 Toni Mannisen kanssa. Tällöin järjestelmässä oli tehty alustan ja tekniikoiden valinnat sekä pohja mihin loppuja ominaisuuksia ryhdyttiin rakentamaan. Toukokuussa projektiin tuli mukaan Janne Uusi-Pohjola. Rekrytointirekisteriä rakennettiin tällä porukalla heinäkuuhun saakka. Hanke jatkui syyskuussa 2014 allekirjoittaneen voimin, jolloin järjestelmä otettiin käyttöön ja kehitystä jatkettiin uusilla ja puuttuvilla ominaisuuksilla.

Työn tarkoituksena on kertoa järjestelmän arkkitehtuurista sekä erilaisista tekniikoista ja työkaluista joita järjestelmän kehityksessä ja ylläpidossa käytettiin. Tavoitteena on antaa käsitys verkossa toimivan tietojärjestelmän rakentamiseen ja ylläpitoon tarvittavista toimista sekä työkaluista. Julkaisu keskittyy järjestelmän ohjelmisto puoleen. Läpi käydään järjestelmän ominaisuuksia, josta siirrytään kertomaan millaisista osista se koostuu sekä arkkitehtuuri valinnoista. Tästä siirrytään järjestelmän toteutukseen eli miten tietokanta on suunniteltu ja millaisia työkaluja ja tekniikoita on käytetty. Käyttäjienhallin-

nan, tietojen tallennuksen ja käyttäjien haun toteutusta käydään läpi tarkastelemalla niiden koodia. Lopuksi kerrotaan miten järjestelmän käyttöönotto on implementoitu sekä millaisia tehtäviä ylläpitoon sisältyy.

2 REKRYTOINTIREKISTERIN ESITTELY

Kuten johdannossa todettiin, Rekrytointirekisteri on web-pohjainen tietojärjestelmä. Järjestelmään pääsee osoitteesta <http://rekrytointirekisteri.projects.tamk.fi> missä on järjestelmän etusivu (kuva 1). Järjestelmä on kaikille avoin, ja sinne pystyy kuka tahansa luomaan uuden käyttäjätilin.



Rekrytointirekisteri Kirjautuminen

Kirjaudu sisään

Käyttäjänimi

Salasana

Kirjaudu

Uusi käyttäjä

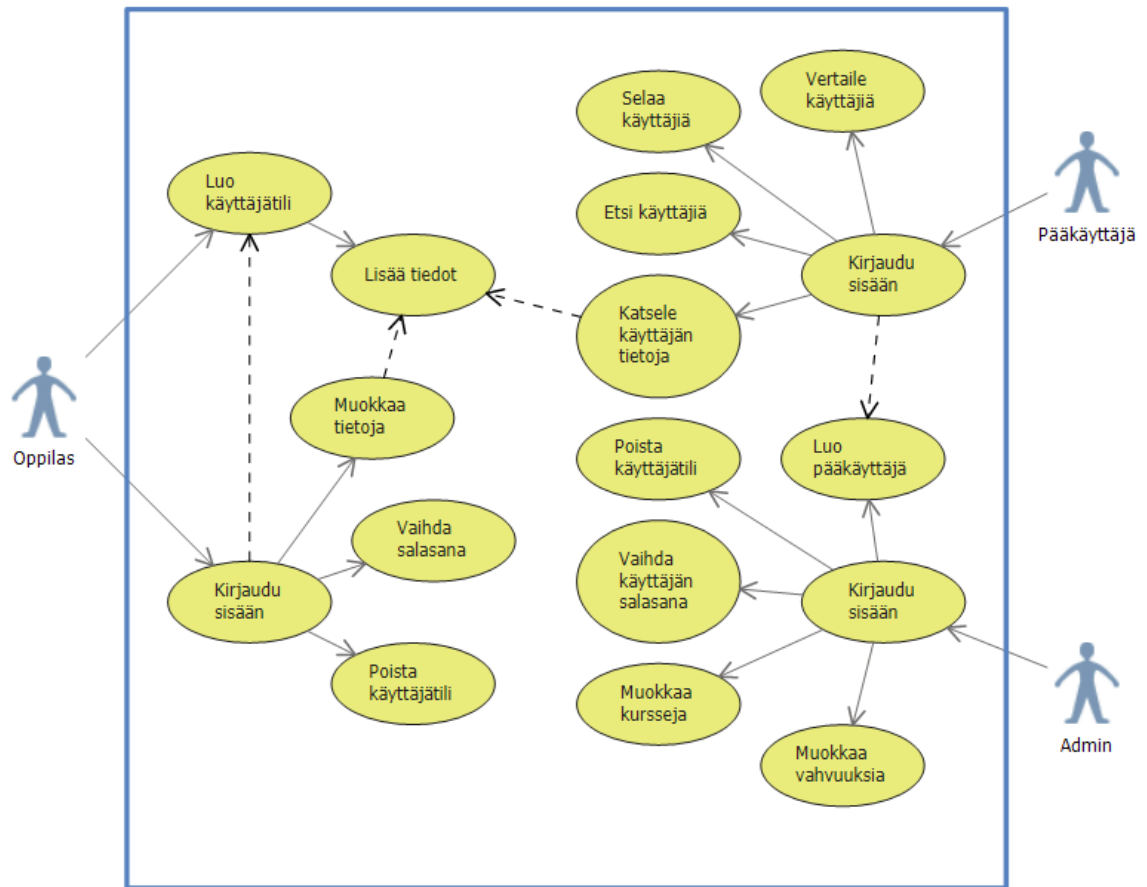
Sivuston käyttöä suositellaan Mozilla Firefox, Google Chrome tai Safari selainten uusimmilla versioilla.

KUVA 1. Järjestelmän aloitussivu.

Normaalikäyttäjät luovat järjestelmään oppilas käyttäjän. Oppilas voi lisätä, muokata tai poistaa omia tietojaan. Oppilas voi tallentaa järjestelmään mm. henkilötiedot, kurssit, tutkinnot ja työkokemuksen. Omaa osaamistaan aiheissa kuten asiakaspalvelu, palkanlaskenta, kirjanpito, tiimityötaidot ja kansainvälisyys voi arvioida 1-5 väliltä. Näiden vahvuuksien sekä muiden vaatimusten perusteella oppilaita jaotellaan sopivuusjärjestykseen tiettyä paikkaa täytettäessä. Koska järjestelmä on suunniteltu liiketalouden linjan tarpeeseen, liittyvät tallennettavat vahvuudet suurilta osin alan asioihin.

Muita järjestelmän käyttäjiä ovat pääkäyttäjä sekä admin käyttäjä. Pääkäyttäjä tilejä pystyy luomaan vain admin käyttäjä, joita voi luoda vain järjestelmän ylläpitäjät. Tarvemmin eri käyttäjien toimintoja ja mitä ne pystyvät järjestelmässä tekemään on kuvattu käyttötapauskaaviossa (kuvio 1). Kuviossa 1 yhtenäisellä viivalla on merkitty kyseisen käyttäjän tehtävät ja poikkiviivalla tehtävät, jotka ovat riippuvaisia toisesta tehtävästä. Esimerkiksi oppilas ei voi kirjautua sisään ellei hän ole luonut käyttäjätiliä. Oppilas ei myöskään voi vaihtaa salasanaa ellei ole kirjautunut sisään. Liitteessä 1 on käyty läpi

oppilas käyttäjän muokkaa tietoja käyttötapaus. Käyttötapauksesta ilmenee eri vaiheet mitä käyttäjä tekee, kun muokkaa omia tietojaan.



KUVIO 1. Rekrytointirekisterin käyttötapauskaavio

Pääkäyttäjä hakee oppilaita kuvan 2 mukaisella käyttöliittymällä. Hakuun voi valita vahvuuksia minkä mukaan tulokset järjestetään sekä erilaisia vaatimuksia mitä käyttäjillä pitää olla. Esimerkiksi pääkäyttäjä voi valita vain tiettyyn opinto-ohjelmaan kuuluvat tai tiettyä paikkaa hakevat henkilöt. Tarkempi haku napin takaa voi vaatimuksiin asettaa kieliä tai kursseja mitä vaaditaan. Näiden ehtojen perusteella järjestelmä järjestää ja näyttää sopivat käyttäjät.

[Muokkaa haku](#)

Vahvuudet

- Asiakaspalvelu
- Myynti
- Projektijohtaminen
- Markkinointi
- Oikeustradenomin tutkinto
- HR-toiminnot
- Rahoitus
- Kansainvälinen kauppa
- Sisäinen laskenta
- Kirjanpito
- Palkanlaskenta
- Yrittäjyys
- Logistiikka
- ERP-järjestelmät
- IT-osaaminen
- Esiintymistaidot
- Paineensietokyky
- Itsenäinen työskentely
- Tarkkuutta vaativat työt

Haku (max 5)

- Viestintä
- Kansainvälisyys
- Johtamistaidot
- Organisointitaidot
- Tiimityötaidot

Vaatimukset

- Harjoittelupaikka
- Opinnäytetyön aihe
- Projektiohjaus
- Työpaikka
- Liiketalous nuoret
- Liiketalous oikeudellinen asiantuntijuus nuoret
- Liiketalous liiketoimintaosaaminen aikuiskoulutus
- Liiketalous oikeudellinen asiantuntijuus aikuiskoulutus
- International business
- Liiketalous yrittäjyys ja tiimijohtaminen
- SAP
- Excel
- Microsoft Word
- PowerPoint
- International Business
- Ympäristöoikeus

[Tarkempi haku](#)
[Sulje](#)

KUVA 2. Hakuehtoien valinta

3 JÄRJESTELMÄN OSAT

3.1 Avoin lähdekoodi

Rekrytointirekisteri tietojärjestelmä on rakennettu avoimen lähdekoodin avulla. Tämä tarkoittaa, että käytetyt kirjastot ja ohjelmistot ovat vapaasti saatavilla. Liitteessä 2 on listattu järjestelmässä käytössä olevia osia, näiden lisenssit, linkit Internet-sivuille ja mistä tämänhetkisiä lähdekoodeja voi selata. Liitteestä 2 nähdään kuinka MIT-lisenssi on käytössä suuressa osassa kirjastoista mitä järjestelmässä käytetään. Kuvassa 3 on MIT-lisenssi. Se mahdollistaa esimerkiksi koodin muuttamisen, kopioinnin, jakamisen ja myynnin kunhan tämä lisenssi sisällytetään siihen. BSD 3-Clause -lisenssi on samantapainen kuin MIT, mutta alkuperäisen tekijänoikeuden haltijan nimeä ei saa käyttää uuden tuotteen markkinoinnissa ilman lupaa. GPLv2 lisenssillä olevat koodit eroavat edellisistä mm. sillä, että siitä johdetuista töistä pitää lähdekoodi tuoda käyttäjien saataville. Tässä käytiin muutaman lisenssin pääkohdat nopeasti läpi, mutta aina kannattaa varmistaa kirjaston tai ohjelman lisenssi ja mitä se oikeuttaa tekemään, erityisesti jos sitä aiotaan käyttää kaupalliseen tarkoitukseen. MIT ja BSD ovat sallivia ja yksinkertaisia lisenssejä, mutta esimerkiksi GPLv2 sisältää useita kohtia mitkä pitää ottaa huomioon, kun koodia lähtee muokkaamaan tai levittämään.

The MIT License (MIT)

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

KUVA 3. MIT-lisenssi

3.2 Tietojen tallennus

Rekrytointirekisterin käyttäjien data tallennetaan relaatiotietokantaan. Relaatiotietokannassa dataa pystytään järjestämään ja etsimään nopeasti indeksointi hyväksi käyttämällä. Tietokannan eri taulujen välisiä yhteyksiä pidetään yllä viiteavaimien avulla ja näin varmistetaan datan oikeellisuus ja mille käyttäjälle se kuuluu. Käyttäjillä on myös mahdollisuus tallentaa tiedostoja kuten omakuva ja cv. Näitä ei tallenneta tietokantaan vaan ne tallennetaan suoraan palvelimen kiintolevyille.

Relaatiotietokantoja on erilaisia ja monille alustoille. Tällaisia ovat mm. PostgreSQL, Microsoft SQL, MySQL, SQLite ja Oracle Database. Näistä Oracle Database ja Microsoft SQL ovat maksullisia, kun muut ovat avoimen lähdekoodin ohjelmistoja. Microsoft SQL tietokannasta on myös saatavilla ilmainen Express versio, jossa on rajoitettu prosessorien ja muistin käyttöä. Microsoft SQL toimii ainoastaan Microsoftin alustoilla, kun muut toimivat myös muilla alustoilla. Rekrytointirekisteri käyttää MySQL-tietokantaa. Se on yksi käytetyimmistä tietokannoista (DB-Engines, 2015) ja on ollut olemassa jo 20 vuotta sekä tuki on erinomainen. TAMKn ympäristössä MySQL on laajasti käytössä ja koulu tarjoaa opiskelijoille MySQL-kannan omaan käyttöön.

3.3 Ohjelmointikielät

Palvelinpään ohjelmointikielenä käytetään PHP:tä. PHP:n manuaalissa (2015) kerrotaan kuinka sen päätarkoituksena on antaa web-kehittäjille mahdollisuus kirjoittaa dynaamisesti luotuja web-sivuja helpolla tavalla (PHP Group, 2015). PHP on skripti kieli eikä sen ajamiseen vaadita erillistä kääntämistä, vaan PHP tulkitsee koodin ajettavaksi. PHP sopii hyvin yhteen järjestelmän muiden komponenttien kanssa ja sen käyttöönotto on helppoa. Muita vaihtoehtoja palvelinpään ohjelmointikieliksi on esimerkiksi Ruby, Python, Go tai ASP.NET.

Selainpuolen dynaaminen toiminnallisuus hoidetaan JavaScript-kieltä käyttäen. Selainpuolella ei ole paljoa valinnanvaraa toteutuksen ohjelmointikielissä, koska web-sivujen esittämiseen on yksi standardi HTML ja dynaamiseen toiminnallisuuteen on nykyään yksi kunnan vaihtoehto. Osan selainpuolen dynaamisesta toiminnasta voisi toteuttaa myös PHP:tä käyttäen. Mutta siirtämällä toiminnallisuutta selaimelle voidaan palveli-

men kuormaa vähentää ja asioiden tekemistä helpottaa. Tässä pitää vain ottaa huomioon, että kaikilla käyttäjillä ei välttämättä ole JavaScript päällä, joko turvallisuussyistä tai heillä on liian vanha selain tukemaan kaikkia ominaisuuksia. Tällöin kannattaa tarkastella järjestelmän kohdekäyttäjiä ja heidän käyttöympäristöänsä ja tehdä tekniikoiden valinnat näiden mukaan.

Rekrytointirekisterin tapauksessa järjestelmä on suunniteltu TAMK:n liiketalouden opiskelijoiden käyttöön, joten heillä kaikilla on pääsy koulun koneille, jossa on nykyaikaiset selaimet, jotka mahdollistavat uusien tekniikoiden käytön. Suunnittelussa ei myöskään ole otettu huomioon mobiililaitteita vaan käyttö on kohdistettu työpöytä koneille. Tämän vuoksi eri mobiilialustojen mahdollisia rajoituksia tuetuissa tekniikoissa ei ole otettu huomioon.

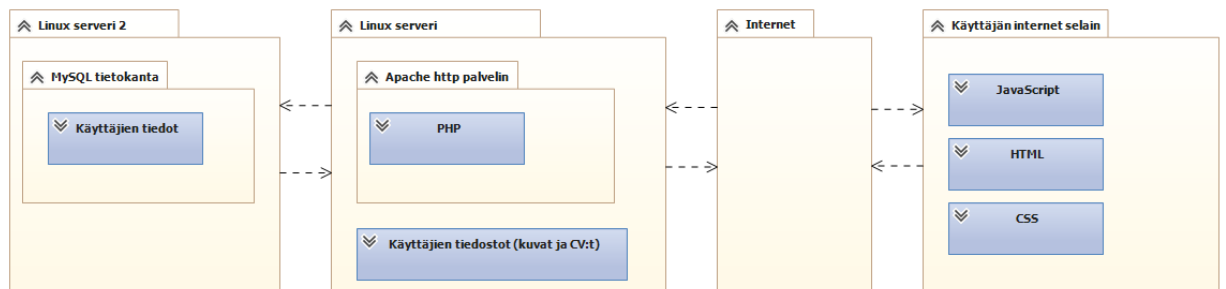
3.4 Palvelin

Palvelinympäristö järjestelmälle on Linux ja ohjelmisto Apache HTTP Server. Nämä tukevat erinomaisesti PHP:n ja MySQL:n käyttöä. Linuxin tilalla voi käyttää vaihtoehtoisesti jotain Unix pohjaista järjestelmää kuten OpenBSD. Apache:n tilalla voi käyttää esimerkiksi Nginx serveriä. PHP on hyvin integroitu Apache:en, mutta jollain toisella kielellä esimerkiksi Pythonia käytettäessä Nginx olisi erinomainen valinta Apache:n tilalle.

3.5 Yhteenveto

Yksinkertaistettu kuva järjestelmän eri osista ja miten ne sijoittuvat, on esitetty kuviossa 2 ja liitteessä 3. Apache ja MySQL ovat omilla Linux-pohjaisilla palvelimilla. Järjestelmän käyttäjä tekee selaimellaan HTTP-kutsuja palvelimelle missä Apache tulkitsee ne ja vastaa kutsutulla HTML-sivulla. Jos kutsu vaatii jotain toimenpiteitä kuten datan hakua tietokannasta, haetaan tieto MySQL-kannasta PHP-koodin avulla. Jos data halutaan näyttää käyttäjälle, päivitetään kutsuttu HTML-sivu haetulla tiedolla ja lähetetään käyttäjän selaimelle.

MySQL-tietokanta voidaan sijoittaa palvelimelle, joka ei ole suorassa yhteydessä Internetiin. Tämä auttaa suojelemaan käyttäjien dataa, koska voidaan määrittellä mitkä koneet pääsevät käsiksi tietokannan tietoihin. Tämä tietenkin edellyttää, että asetukset ja käyttöoikeudet ovat kunnossa. Palvelimien ei tarvitse olla fyysisesti eri laitteilla vaan erottaminen voidaan toteuttaa virtualisointia käyttäen. Virtualisoinnilla voidaan alentaa järjestelmän kustannuksia, koska sama laitteisto voi pyörittää monta eri palvelua, kunhan resurssit ovat riittävät.



KUVIO 2. Rekrytointirekisterin osat

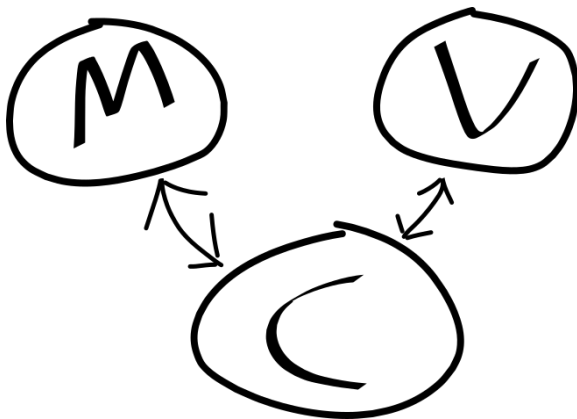
4 ARKKITEHTUURI

4.1 MVC

Rekrytointirekisteri tietojärjestelmän arkkitehtuuri perustuu suurimmaksi osaksi MVC-malliin. MVC-malli on jaettu kolmeen osaan, malli (model), näkymä (view) ja kontrolleri (controller). Järjestelmän osat on jaettu seuraavasti:

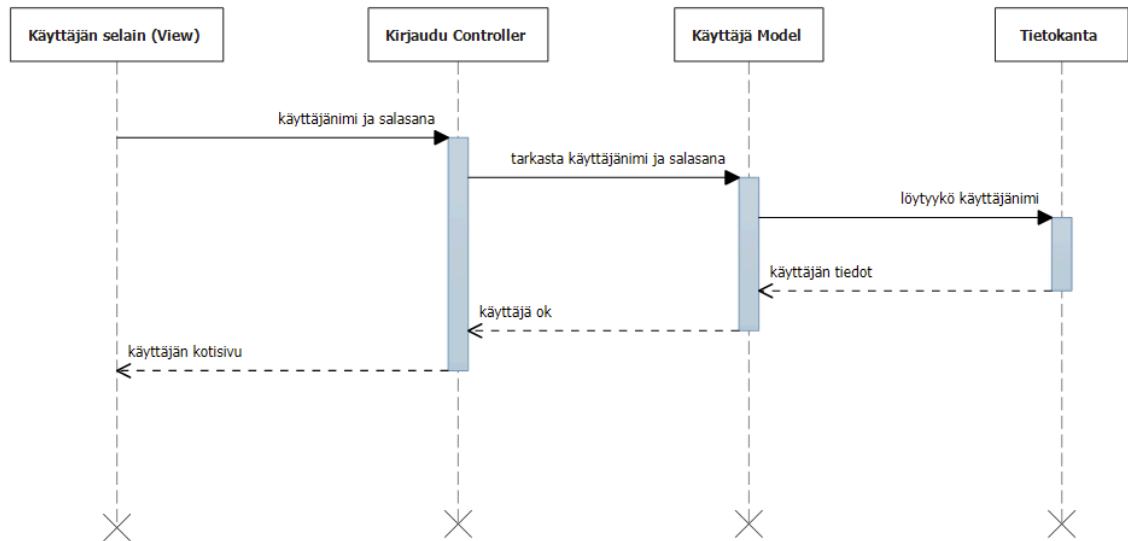
- Tietokannan toiminnot on sijoitettu malli osioon.
- Näkymään laitetaan käyttöliittymän osat.
- Järjestelmän toiminnallisuus menee kontrolleri osioon.

Kuviossa 3 on esitetty miten MVC-mallin osat ovat yhteydessä toisiinsa. Kontrolleri erottaa näkymän ja mallin toisistaan, malli sisältää datan mitä näkymissä halutaan näyttää. Kun näkymässä pitää näyttää käyttäjän tallennettuja tietoja, kontrolleri hakee mallissa olevan datan näkymän käyttöön.



KUVIO 3. MVC-mallin osien yhteydet

Kuviossa 4 on esimerkki miten sisäänkirjautuminen voi tapahtua, kun käytössä on MVC-malli. Ensinnäkin käyttäjä kirjoittaa kirjautumistietonsa sivulle, mistä selain lähettää tiedot oikealle kontrollerille. Tässä tapauksessa kirjautu kontrolleri pyytää käyttäjä mallia tarkastamaan käyttäjänimen ja salasanan. Tämä hakee tietokannasta käyttäjänimellä olevat tiedot. Jos tietoja löytyy malli tarkastaa ovatko kantaan tallennettu ja annettu salasana samoja. Malli palauttaa tiedon onnistuiko vai epäonnistuiko tietojen tarkastaminen. Tilanteessa missä tiedot ovat kunnossa, käyttäjä kirjataan sisään ja ohjataan oikealle sivulle, muuten ilmoitetaan käyttäjälle, että kirjautuminen epäonnistui.



KUVIO 4. Käyttäjän sisään kirjaus

4.2 ORM

Rekrytointirekisterin tietokantaa käytetään ORM (object relational mapper) kirjaston avulla. ORM muuttaa tietokannan taulut olioiksi joita voidaan koodissa käsitellä. Esimerkiksi jos tietokannassa on käyttäjä taulu, luodaan ORM-luokka, joka viittaa tähän tauluun. Luokan attribuutit ovat taulun sarakkeita ja taulujen väliset yhteydet viittaavat sen taulun ORM-luokkaan. Tällä voidaan helpottaa tietokannan kanssa toimimista ja esimerkiksi dynaamisten kyselyjen teko helpottuu, koska se voidaan luoda sopivista palasista. Tästä kerrotaan lisää luvussa 8.

Kuvassa 4 on esitetty miten käyttäjän käyttäjänimi haetaan ensin käyttäen normaalia SQL-kutsua ja sitten sama ORM-kirjastoa käyttämällä. Kuvassa 4 rivillä 3 ORM hakee Käyttäjä olion, jonka id on yksi. Tältä oliolta voidaan sitten pyytää käyttäjänimi (rivi 4). Taustalla ORM luo saman SQL-kutsun kuin mitä rivillä 1. ORM-kirjaston käyttöä käytetään tarkemmin läpi luvussa 8, kun kerrotaan miten Rekrytointirekisterin ominaisuuksia on toteutettu.

```

1  SELECT kayttajanimi FROM kayttaja WHERE id=1;
2
3  $kayttaja = $em->find('Entity\Kayttaja', 1);
4  $kayttaja->get_kayttajanimi();

```

Kuva 4. Käyttäjänimen haku

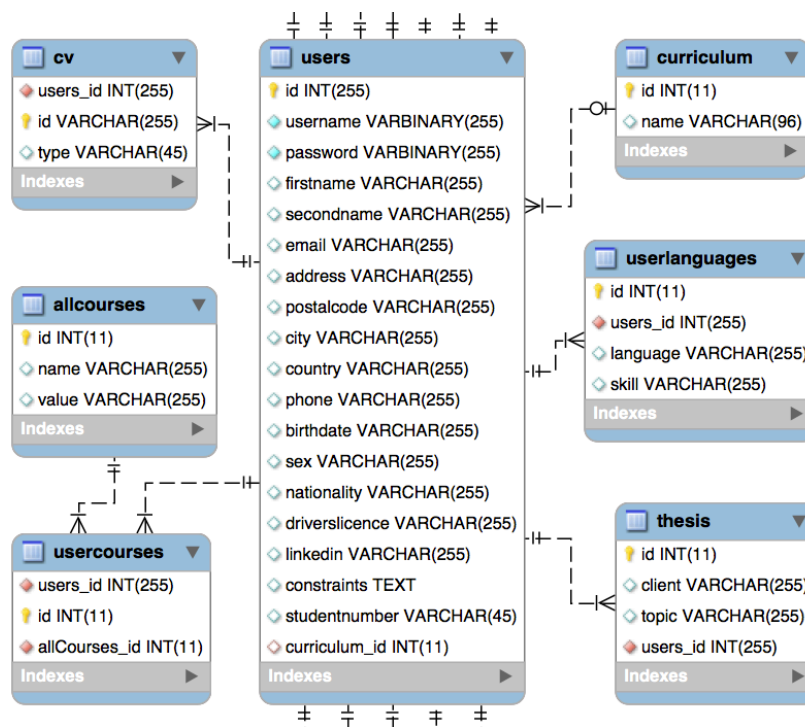
4.3 Rakenne

Kuten todettu Rekrytointirekisteri rakentuu MVC-komponenttien päälle. Tämän mallit on jaettu kahteen osa-alueeseen, entityt ja näitä entityjä käyttäviin luokkiin. Entityt ovat ORM-kirjaston luomia olioita tietokannan tauluista. Näitä olioita sitten manipuloidaan tehdasluokilla. Tehdasluokat sisältävät taulun käsittelyyn liittyvät toimenpiteet, kuten luonnin, muokkauksen ja poiston. Nämä luokat on pyritty luomaan siten, että ne muodostavat loogisia kokonaisuuksia. Esimerkiksi kuvat ja CV:t käsitellään samassa luokassa. Liitteessä 4 on kuvattu millaisista luokista Rekrytointirekisterin on rakennettu. Siitä myös näkyvät kaikki tehdasluokat sekä entityt.

Liitteen 4 kuva sisältää myös järjestelmän kontrollerit ja näkymät. Kontrolleri on jaettu pääosin eri näkymien mukaan. Kontrollerit hoitavat oikean sivun näyttämisen käyttäjälle, kun tämä menee selaimella tiettyyn osoitteeseen. Ne suorittavat myös muita järjestelmän toimenpiteitä kuten datan manipulointia. Näkymät sisältävät kaikki järjestelmän HTML-sivut. Sivusta ja tilanteesta riippuen kontrolleri lataa jonkin näkymän mikä sitten näytetään käyttäjälle. Tarvittaessa kontrolleri noutaa dataa tietokannasta mallien avulla, mikä sitten asetetaan sivulle.

5 TIETOKANNAN RAKENNE

Järjestelmän päätarkoituksena on tallentaa oppilaiden tietoja. Näiden tietojen perusteella voidaan sitten tehdä päätöksiä onko opiskelijalla riittävät taidot haluttuun tehtävään. Tiedoista kuten henkilötiedot, käyty TAMK kurssit, muut kurssit, työkokemus, vahvuudet, kielitaito, cv ja kuva voidaan suunnitella järjestelmän tietokanta. Tietokannan keskipisteessä on users taulu, joka sisältää käyttäjän henkilötiedot sekä kirjautumistiedot. Tämän taulun avulla kaikki muut tietokannan tiedot liitetään tiettyyn käyttäjään. Kuvioon 5 on koottu erityyppisiä järjestelmän tauluja.



KUVIO 5. Tietokannan esimerkkitaulut

Eri koulutuslinjojen sisältämät kurssit tallennetaan allcourses tauluun. Jos käyttäjä on käynyt jonkin näistä kursseista, löytyy tämä tieto usercourses taulusta. Usercourses viittaa allcourses taulun kurssiin sekä käyttäjään jolla tämä kurssi on. Kuvassa 5 on SQL-kutsut joilla taulut on luotu. Kuvan 5 riveillä 18–20 ja 23–25 on tehty usercourses taulun viiteavain yhteydet users ja allcourses tauluihin, joilla tietty kurssi voidaan liittää tiettyyn käyttäjään. Nämä taulut tuottavat ns. many-to-many suhteen. Eli yhdellä kursilla voi olla monta käyttäjää ja käyttäjällä monta kurssia.

```

1 CREATE TABLE IF NOT EXISTS `allcourses` (
2   `id` INT(11) NOT NULL AUTO_INCREMENT,
3   `name` VARCHAR(255) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NULL DEFAULT NULL,
4   `value` VARCHAR(255) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NULL DEFAULT NULL,
5   PRIMARY KEY (`id`),
6   INDEX `name` (`name` ASC),
7   INDEX `value` (`value` ASC))
8 ENGINE = InnoDB
9 DEFAULT CHARACTER SET = utf8;
10
11 CREATE TABLE IF NOT EXISTS `usercourses` (
12   `users_id` INT(255) NOT NULL,
13   `id` INT(11) NOT NULL AUTO_INCREMENT,
14   `allCourses_id` INT(11) NOT NULL,
15   PRIMARY KEY (`id`),
16   INDEX `fk_table1_users1_idx` (`users_id` ASC),
17   INDEX `fk_userCourses_allCourses1_idx` (`allCourses_id` ASC),
18   CONSTRAINT `fk_table1_users1`
19     FOREIGN KEY (`users_id`)
20     REFERENCES `users` (`id`)
21     ON DELETE CASCADE
22     ON UPDATE CASCADE,
23   CONSTRAINT `fk_userCourses_allCourses1`
24     FOREIGN KEY (`allCourses_id`)
25     REFERENCES `allcourses` (`id`)
26     ON DELETE CASCADE
27     ON UPDATE CASCADE)
28 ENGINE = InnoDB
29 DEFAULT CHARACTER SET = utf8;

```

KUVA 5. Taulujen luonti esimerkki

Taulu curriculum sisältää järjestelmässä olevat opinto-ohjelmat. Käyttäjille on tallennettu users tauluun viittaus mihin opinto-ohjelmaan se kuuluu. Tämän avulla voidaan valita minkä opintosuunnitelman kurssveja näytetään. Curriculum tauluun liittyvät SQL-kutsut ovat kuvassa 6. Rivit 1–5 kuuluvat users taulun create table koodiin ja niillä asetetaan viite curriculum tauluun, mikä luodaan riveillä 7–12. Curriculum taulu on tietokannan ainut johon viitataan users taulussa. Tämä on toteutettu näin, koska käyttäjällä voi olla vain yksi opinto-ohjelma. Tämä on ns. one-to-many suhde eli käyttäjällä voi olla vain yksi opinto-ohjelma, mutta opinto-ohjelmalla voi olla monta käyttäjää.

```

1 CONSTRAINT `fk_users_curriculum1`
2   FOREIGN KEY (`curriculum_id`)
3   REFERENCES `curriculum` (`id`)
4   ON DELETE SET NULL
5   ON UPDATE CASCADE)
6
7 CREATE TABLE IF NOT EXISTS `curriculum` (
8   `id` INT NOT NULL,
9   `name` VARCHAR(96) NULL,
10  PRIMARY KEY (`id`))
11 ENGINE = InnoDB
12 DEFAULT CHARACTER SET = utf8;

```

KUVA 6. Curriculum taulu

Taulut cv, userlanguages ja thesis edustavat kannan yleisintä taulutyyppiä one-to-many. Curriculum taulusta nämä eroavat siten, että taulun sisällä viitataan johonkin käyttäjään

(users_id) mille nämä tiedot kuuluvat. Toisin kuin curriculum taulussa jossa viittaus oli users taulussa. Osa näistä tauluista kuuluisi olla one-to-one suhteella, eli yhdellä käyttäjällä voi olla esimerkiksi vain yksi cv. Nämä on epähuomiossa toteutettu one-to-many suhteella ja tätä yhden taulun rajoitusta pitää ylläpitää enemmän koodin avulla. Thesis taulu, johon tallennetaan käyttäjän opinnäytetyön tiedot, on luotu kuvan 7 kutsulla.

```

1 CREATE TABLE IF NOT EXISTS `thesis` (
2   `id` INT(11) NOT NULL AUTO_INCREMENT,
3   `client` VARCHAR(255) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NULL DEFAULT NULL,
4   `topic` VARCHAR(255) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NULL DEFAULT NULL,
5   `users_id` INT(255) NOT NULL,
6   PRIMARY KEY (`id`),
7   INDEX `fk_thesis_users1_idx` (`users_id` ASC),
8   CONSTRAINT `fk_thesis_users1`
9     FOREIGN KEY (`users_id`)
10    REFERENCES `users` (`id`)
11    ON DELETE CASCADE
12    ON UPDATE CASCADE)
13 ENGINE = InnoDB
14 DEFAULT CHARACTER SET = utf8;

```

KUVA 7. Thesis taulun luonti

Koska thesis taulun avain on id (rivi 6), jota aina kasvatetaan, kun lisätään uusi tieto, pystytään käyttäjälle lisäämään useampia thesis tietoja. Jos tämä haluttua estää tietokannan tasolla voidaan taulu muuttaa one-to-one suhteeksi. Thesis taululle tämän muutoksen voi tehdä vaihtamalla taulun avaimeksi users_id. Taulun avaimet pitää olla yksilöllisiä eikä samaa avainta pysty tallentamaan useampaan kertaan. Kun siis avaimeksi vaihdetaan käyttäjä taulun id, voi käyttäjälle lisätä vain yhden thesis tiedon. One-to-one tapauksessa thesis taulun luonti on kuvan 8 mukainen. Koodi on muuttunut vain vähän. Id rivi on poistettu ja avaimeksi asetetaan users_id (rivi 5). Users_id:n indeksointi on voitu poistaa, koska se tehdään automaattisesti avaimille. SQL-kielellä ei voi suoraan kertoa millaisen suhteen taulujen välille haluaa. Se pitää tehdä edelliseen tapaan valitsemalla sopiva avain ja viite tauluun.

```

1 CREATE TABLE IF NOT EXISTS `thesis` (
2   `client` VARCHAR(255) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NULL DEFAULT NULL,
3   `topic` VARCHAR(255) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NULL DEFAULT NULL,
4   `users_id` INT(255) NOT NULL,
5   PRIMARY KEY (`users_id`),
6   CONSTRAINT `fk_thesis_users1`
7     FOREIGN KEY (`users_id`)
8     REFERENCES `users` (`id`)
9     ON DELETE CASCADE
10    ON UPDATE CASCADE)
11 ENGINE = InnoDB
12 DEFAULT CHARACTER SET = utf8;

```

KUVA 8. Thesis taulun luonti one-to-one suhteella

Loput tietokannan tauluista on tehty samaan tapaan kuin kuviossa 5 olevat. Koko tietokannan kuvaus on liitteessä 5. Siitä mainitsemisen arvoinen on admin taulu, jolla ei ole yhteyksiä mihinkään muuhun tauluun. Siitä tarkistetaankin vain admin käyttäjän kirjautumistiedot, eikä sillä tarvitse olla yhteyttä muihin tietoihin. Toinen erilainen on autosave taulu, joka on ainut one-to-one suhteella oleva taulu. Se on viimeisiä kantaan lisättyjä tauluja, minkä lisäyksen aikaan on huomattu, että miten se kannattaa toteuttaa.

Tietokannan suunnittelussa olisi voinut tehdä asioita toisin. Esimerkiksi usercourses taulun id on turha ja users_id ja allcourses_id olisi voinut yhdistää avaimeksi ja tehdä kurssien tauluista kunnan many-to-many suhteen omaavat taulut. Toinen parannus olisi jos käyttäjätilin tiedot eli käyttäjänimi ja salasanan erottaisi henkilötiedoista sekä edellä mainittu taulujen muuttaminen one-to-one suhteeksi. Nämä selkeyttäisi kannan toimintaa ja ylläpitoa.

6 JÄRJESTELMÄN RUNGOT

Järjestelmän kaikkien osien toteutus itse olisi varmasti hyvä idea opiskelun kannalta, mutta yleisesti on suositeltavaa käyttää jotain valmista runkoa. Valmiit rungot voivat tarjota paljon helpotusta järjestelmän rakentamiseen ja pystyvät antamaan esimerkiksi tietoturvaominaisuuksia mitä ei itse tiedä tarvitsevana tai osaa edes toteuttaa. Rungon valinnassa kannattaa olla tarkkana. Pitää esimerkiksi huomioida, että runko on riittävä omiin käyttötarkoituksiin, mitä muut ovat siitä mieltä ja onko sitä päivitetty viime aikoina eli onko sen kehitys aktiivista. Hyvä dokumentointi on myös erittäin tärkeä ja helpottaa käyttöönottoa varsinkin jos oma kokemus ei ole valtava. Näillä valinnoilla pyritään estämään mahdolliset ongelmat ja epäsovaisuudet tulevaisuudessa.

6.1 MVC-malli

Rekrytointirekisterin MVC-malli on toteutettu käyttäen CodeIgniter-alustaa. CodeIgniter:n Internet-sivuilla kerrotaan kuinka se tarjoaa erinomaista suorituskykyä ja pientä tilantarvetta (CodeIgniter, 2015.) Se on toteutettu käyttäen PHP-kieltä ja Rekrytointirekisterissä oleva versio 2.1.4 käyttää omaa lisenssiä. Uusin 3.0 versio käyttää MIT-lisenssiä, joka antaa enemmän vapauksia koodin käyttöön.

CodeIgniter hoitaa kaiken MVC-mallin toiminnallisuuden ja kehittäjälle jää vain näiden ominaisuuksien käyttäminen. Se tarjoaa myös valmiita turvallisuustoimia kuten syötteiden putsauksia ja validointia sekä turvaa XSS-hyökkäyksiä vastaan. Michal Zalewski (2012) kertoo kuinka XSS (cross site scripting) hyökkäyksessä haitallista JavaScript koodia päästetään tahattomasti HTML-koodin sisään. Näin hyökkääjä saa haltuunsa sivuston toiminnan ja ulkoasun. (Zalewski 2012, 71.)

Muita vaihtoehtoja rungoksi on esimerkiksi Laravel (<http://laravel.com>) tai Symfony (<http://symfony.com>). Molemmilla on hyvä maine ja ne käyttävät MIT-lisenssiä. Valinta näiden kahden ja CodeIgniter:n välillä voikin olla vaikeaa. Jos mahdollista, kannattaa runkoja kokeilla ennen lopullisen päätöksen tekoa ja valita se mikä itselle tuntuu parhaimmalta. Tämä on yksi avoimen lähdekoodin ohjelmistojen vahvuus, sillä niitä pystyy kokeilemaan ilman maksua.

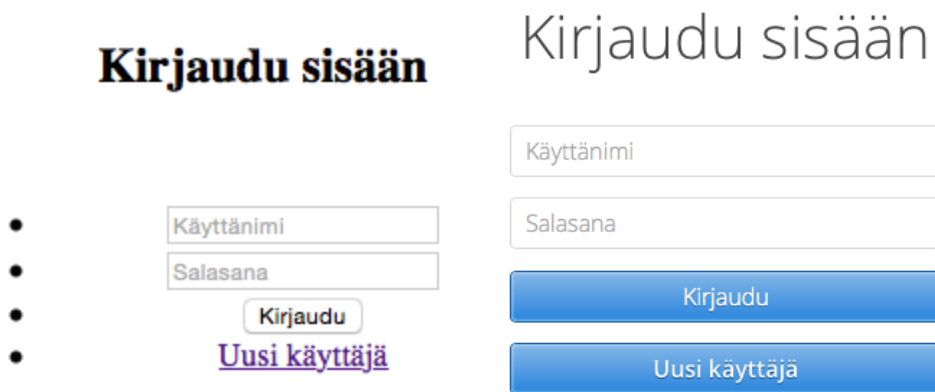
6.2 Tietokanta

Rekrytointirekisteri käyttää Doctrine ORM-kirjastoa, mikä on toteutettu PHP-ohjelmointikielellä ja käyttää MIT-lisenssiä. MySQL:n lisäksi Doctrine:a voi käyttää mm. PostgreSQL ja Microsoft SQL tietokantojen kanssa. Doctrine:n kaikkia ominaisuuksia, kuten automaattisia skeeman (engl. schema) luonteja ja päivityksiä ei ole käytetty Rekrytointirekisterin kehityksessä tai ylläpidossa. Skeeman ja entityjen avulla Doctrine voi vertailla tietokannan tilaa ja luoda tarvittavat SQL-kutsut, joilla se saadaan päivitettyä (Symfony, 2015). Tällä tavalla pystyy esimerkiksi lisäämään uuden sarakkeen kantaan, kirjoittamatta ollenkaan SQL-komentoja. Näitä ominaisuuksia ei ole osattu hyödyntää kunnolla, mutta ne voivat olla iso apu tietokannan ylläpidossa.

6.3 Käyttöliittymä

Rekrytointirekisterin käyttöliittymän rakentamisessa käytetään apuna Gumby framework nimistä CSS-runkoa. Se on käytännössä kasa CSS-tyylimäärittelyjä joilla helpotetaan modernin näköisten Internet-sivujen tekoa. Gumby sisältää mm. ruudukko järjestelmän mikä helpottaa eri elementtien sijoittelua haluttuun kohtaan. Kuvassa 9 on vasemmalla puolella kuva ilman CSS-määrittelyjä ja oikealla puolella Gumby:n määrittelyjen kanssa.

Gumby framework ei ole ainut CSS-runko ja vaihtoehtoisesti voi käyttää esimerkiksi alun perin Twitter:n kehittämää Bootstrap runkoa (<http://getbootstrap.com>). Gumby:n valinta onkin tehty hätäisesti ja järjestelmän kehityksen aikana on huomattu, että dokumentointi on puutteellinen. Esimerkiksi ruudukon käyttö voi jossain tilanteissa aiheuttaa päänvaivaa. Myös käyttäjäkunta on melko pieni joten ratkaisujen löytäminen ongelmiin voi osoittautua hankalaksi. Koska Bootstrap:stä ei ole kokemusta, on vaikea sanoa olisiko se ratkaissut osan ongelmista.



KUVA 9. Gumby esimerkki

Käyttöliittymän dynaaminen toiminnallisuus on toteutettu käyttäen jQuery JavaScript-runkoa. JQuery helpottaa esimerkiksi HTML elementtien valintaa, lisäystä ja tapahtumien käsittelyä. Kuvassa 10 on vertailtu miten normaali JavaScript ja jQuery eroavat toisistaan, kun lisätään uusi elementti olemassa olevaan elementtiin. Kuvasta 10 nähdään kuinka jQuery tiivistää lisäyksen neljästä yhteen riviin.

```

1 // Normaaali javascript
2 var newDiv = document.createElement('div');
3 newDiv.className = 'field';
4 newDiv.innerHTML('Tämä on uusi div elementti');
5 document.getElementById('test').appendChild(newDiv);
6
7 // jquery
8 $('#test').append('<div class="field">Tämä on uusi div elementti</div>');
```

KUVA 10. JavaScript vertailu

JQuery:llä elementtejä luotaessa täytyy kiinnittää huomiota HTML:n oikeellisuuteen. Jos tarkastellaan kuvan 11 esimerkkiä, huomataan, että lisäykset muuttuvat nopeasti monimutkaisiksi. Varsinkin heittomerkkien käytössä täytyy olla tarkkana, ettei tule virheitä. Unohtunutta tai väärässä paikassa olevaa heittomerkkiä ei välttämättä heti huomaa ja se voi aiheuttaa outoja ongelmia, kun input-elementin arvoa yritetään muuttaa.

```

1 $('#moreTamkForm').append('<div class="field">' +
2   '<input class="input five columns" id="moreTamkCourse' +
3   moreTamkCounter + '" name="moreTamkCourse' + moreTamkCounter + '" type="text"/>' +
4   '<input class="input one columns push_one" id="moreTamkOp' +
5   moreTamkCounter + '" name="moreTamkOp' + moreTamkCounter + '" type="text"/>' +
6   '<div style="display: none"><input id="moreTamkid' +
7   moreTamkCounter + '" name="moreTamkid' + moreTamkCounter + '" value="0"></input></div>' +
8   str + '</div>');
```

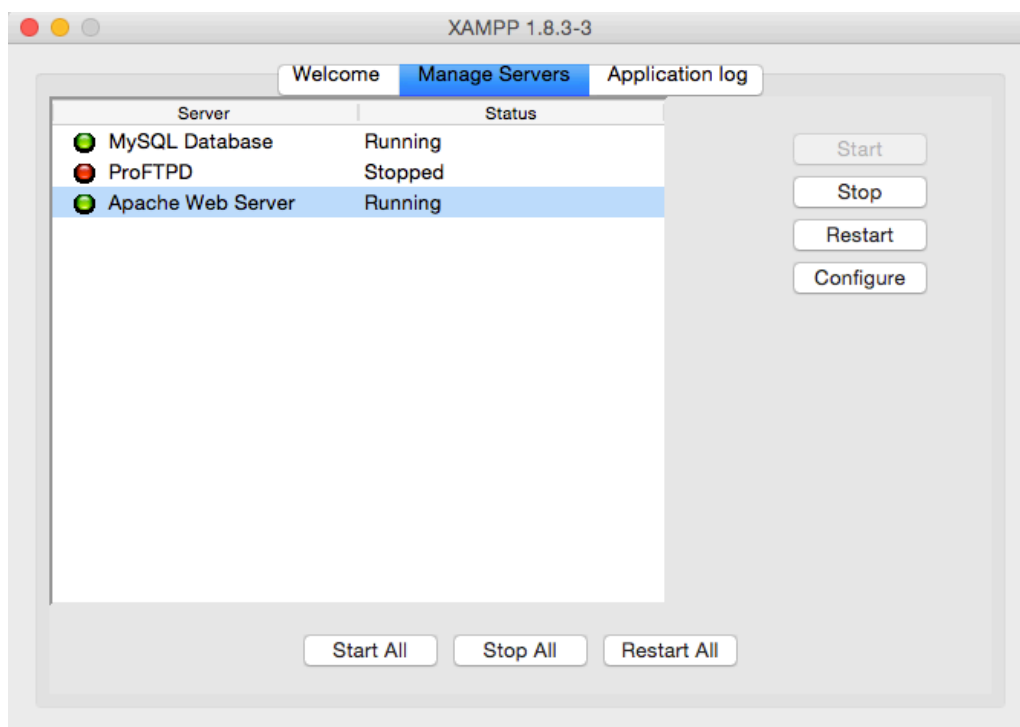
KUVA 11. jQuery esimerkki

7 KEHITYSTYÖKALUT

7.1 Kehitysympäristö

Kehitysympäristön tärkeitä osia on mm. palvelin missä on Apache, PHP ja MySQL. Nämä on käytännössä ainoat tarvittavat osat millä järjestelmää pystytään kehittämään ja testaamaan. Jos käytössä on Linux tai Unix ympäristö näiden käyttöönotto on triviaalia, mutta Windows käyttöjärjestelmässä tämä on vaikeampaa. Tässä apuun tulee paketti nimeltä XAMPP. Sillä pystyy asentamaan Apache:n PHP-tuella ja MySQL-palvelimen helposti omalle koneelle. Paketin on kehittänyt Apache Friends -projekti ja se tarjoaa asennuspaketit Windows, Linux ja Mac OS X käyttöjärjestelmille.

XAMPP:n käyttö on helppoa, ja asennuksen jälkeen Apache:n ja MySQL:n pystyy käynnistämään mukana tulleella managerilla (kuva 12). Apache:n asetuksia pääsee muokkaamaan configure napilla, ja siellä voi määrittellä mm. käytettävän portin. Sieltä löytää myös Apache:n asetustiedoston missä voi tarkemmin määrittää palvelimen asetuksia. MySQL asetusten muokkaaminen tapahtuu samalla tavalla valitsemalla MySQL listasta.



KUVA 12. XAMPP application manager

XAMPP ei ole täydellinen vaihtoehto kehitysympäristölle, mutta ottaen muutaman asian huomioon on se helppo ja toimiva ratkaisu. Rekryointirekisterin kehityksessä on huomattu, kuinka kehitys ja tuotanto järjestelmien alustojen on hyvä olla mahdollisimman samanlaisia. Jos järjestelmät eroavat tulee ongelmia esimerkiksi ohjelmistojen eri versioiden kanssa. On hyvä valita XAMPP:sta versio, joka sisältää samat versiot PHP:stä ja MySQL:stä mitä tuotantoympäristössä on. Eri yhdistelmiä on paljon, mutta lähelle ainakin pääsee. Ongelmia voi myös tulla, koska Windowsin ja OS X:n tiedostojärjestelmät eivät välitä kirjainkoosta, mutta Linux järjestelmä välittää. Esimerkiksi tiedostot data.txt ja Data.txt ovat samat tiedostot Windowsilla ja OS X:llä, mutta ovat eri tiedostoja Linuxissa.

Toinen hyvä vaihtoehto kehitysympäristöksi on asentaa virtuaalikoneeseen tuotantoympäristöä vastaavat ohjelmistot. Tämä ei ole yhtä helppoa kuin XAMPP:n asentaminen ja vaatii enemmän osaamista Linuxin käytössä, mutta tällä pystytään varmistamaan kehitys- ja tuotantoympäristöjen samankaltaisuus. Kun ympäristön on saanut valmiiksi voi siitä tehdä levykuvan, joka levitetään muille kehittäjille ja varmistetaan, että kaikki toimii samassa ympäristössä.

7.2 Versionhallinta

Versionhallinta kuuluu ohjelmistokehittäjän tärkeimpiin työkaluihin. Versionhallinnan hyödyt tulevat nopeasti esille, varsinkin tilanteissa missä lähdekoodia muokkaa useampi henkilö. Pääosin versionhallinta pitää kirjaa mitä muutoksia tiedostoihin on tehty ja nämä muutokset pystyy tallentamaan kommentin kera. Muutoksia voi tarkastella tai muuttaa myöhemmin. Kun muutoksia tallennetaan hyvien kommenttien kera, voidaan esimerkiksi vuosia myöhemmin selvittää miksi ja kuka on kirjoittanut jonkin koodirivin. Versionhallinta pystyy automaattisesti yhdistämään monen kehittäjän tekemät muutokset tiedostoon, ellei ole muokattu samaa riviä. Tässä tapauksessa syntyy konflikti, joka käyttäjän pitää ratkaista. Tämä tapahtuu valitsemalla joko omat tai toisen muutokset, tai muokkaamalla käsin rivit halutuiksi.

Versionhallinta ei tee jokaiselle muutokselle omaa kopiota tiedostosta vaan tallentaa pelkästään muuttuneen osan. Täten versionhallinnan käyttö ei lisää paljoa levynkäyttöä. Tämä toimii yleisesti vain koodi- ja tekstitiedostoille. Kaikki versionhallinnat eivät so-

vellu binääritiedostoille tai muille tiedostomuodoille, koska ne joutuvat tekemään kopioita koko tiedostosta sen muuttuessa.

Rekrytointirekisterin lähdekoodi on Git versionhallintaohjelman alla. Kaikki tiedostot ja muutokset on tallennettu tähän Git varastoon (repository), mikä sisältää kaiken tiedon mitä Git tarvitsee ylläpitääkseen historiaa. Git versionhallintaan tallennetaan muutoksia tekemällä committeja. Commitille annetaan viesti, johon voi kirjata mitä tai miksi commit on tehty. Kuvassa 13 on muutama esimerkki Rekrytointirekisteriin tehdyistä committeista. Niissä näkyy commit viestin lisäksi tekijä ja aika milloin se on tehty, sekä merkkijono mistä commitin tunnistaa. Varastoon työnnetään (push) muutoksia eli tehdyt commitit tallennetaan palvelimelle. Palvelimen varastosta vedetään (pull) muutoksia omaan paikalliseen varastoon. Varasto voi sisältää eri haaroja (branch). Nämä haarat ovat itsenäisiä kopioita varaston senhetkisestä tilasta. Haaraan voi tehdä committeja vaikuttamatta toisten haarojen toimintaa. Esimerkiksi jos halutaan toteuttaa uusi ominaisuus, voidaan sitä varten tehdä oma haara johon työ tehdään. Näin uuden ominaisuuden kehittäminen ei vaikuta muun järjestelmän toimintaan. Kun ominaisuus on saatu valmiiksi, voidaan commitit liittää (merge) alkuperäiseen haaraan.

Git on hajautettu versionhallintajärjestelmä. Hajautettu versionhallinta tarkoittaa, että järjestelmän lähdekoodia voi säilyttää monessa paikassa samaan aikaan. Keskitetyssä versionhallinnassa tiedostot sijaitsevat jossain määrättyssä sijainnissa mihin kaikki muutokset menevät, kun taas hajautetussa muutoksia ei tarvitse aina tallentaa tiettyyn paikkaan. Muutokset voi esimerkiksi tallentaa lokaalisti ja vasta myöhemmin lähettää palvelimelle mistä muutkin kehittäjät pääsevät niihin käsiksi. Jos esimerkiksi lataa Git varaston omalle koneelleen ulkoiselta palvelimelta, siihen ei enää tarvitse olla yhteydessä. Tämä lokaali kopio sisältää kaiken historian mitä tämä kyseinen haara palvelimellakin sisältää ja siihen voi vapaasti tehdä muutoksia. Keskitetty versionhallinta kuten SVN tallentaa aina commitit palvelimelle. Hajautettu versionhallinta mahdollistaa erilaisen työnkulun. Kehittäjä pystyy esimerkiksi tekemään useita pieniä committeja ja myöhemmin yhdistämään nämä suuremmaksi kokonaisuudeksi, joka lähetetään palvelimelle. Tämän voi tehdä, koska kehittäjä on tehnyt vain lokaaleja muutoksia joita kukaan muu kehittäjä ei ole nähnyt tai käyttänyt. Tällainen historian muuttaminen on helppoa ja sallittua, edellyttäen ettei muutoksia ole vielä lähetetty palvelimelle.

```

commit da1493b0e95aedad32d2116d03a85478f59fac8b
Author: Teemu Halmela <teemu.halmela@eng.tamk.fi>
Date: Tue Oct 21 18:38:20 2014 +0300

    Korjaus vahvuuksien päivitykseen

    Oli jäänyt poistamatta kielten osuus vahvuuksien päivityksen
    käsittelyssä.

commit 17d4e4dcb62fc843f21975f011b68b18de896d0a
Author: Teemu Halmela <teemu.halmela@eng.tamk.fi>
Date: Mon Oct 20 15:32:44 2014 +0300

    Tarkempi hakuun kielet haetaan kannasta

    Hakee kymmenen kieltä joita on eniten kannassa.

commit 15cc8c9c0a7a7544995da00c88dba124265f2a39
Author: Teemu Halmela <teemu.halmela@eng.tamk.fi>
Date: Mon Oct 20 14:58:54 2014 +0300

    Muutettu: filttareiden kurssit haetaan kannasta

    Nyt tarkemmassa haussa on kaikki kurssit mitä kannasta löytyy

commit a8dab697ba1880711334ab664deec4bda55090a2
Author: Teemu Halmela <teemu.halmela@eng.tamk.fi>
Date: Wed Oct 15 20:02:01 2014 +0300

    Lisätty datan muokkaus aikaisempi koulutus sivulle

    Nyt datan muokkaaminen toimii, sekä tietoja voi poistaa välistä.

commit 02a6545f46328308fa4b06c5b0379389f5d0c52e
Author: Teemu Halmela <teemu.halmela@eng.tamk.fi>
Date: Mon Oct 13 15:47:44 2014 +0300

    Korjaus käyttäjienhallintaan

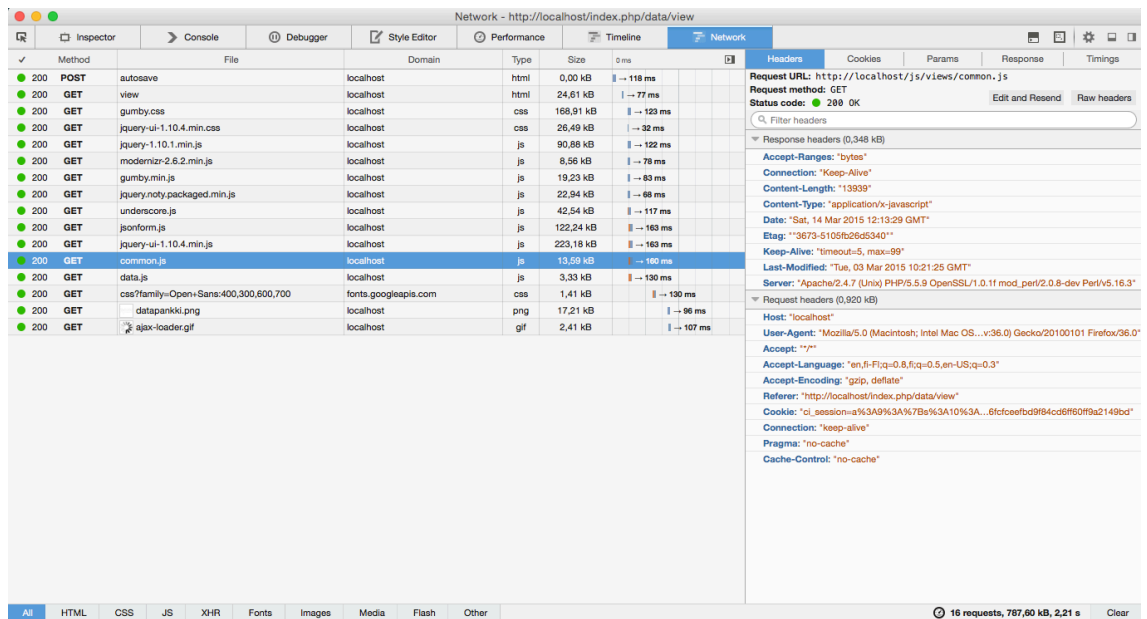
    Vaihdettu käyttäjän tietojenhaku oikeaan funktioon, joka toimii vanhalla
    php versiolla.

```

KUVA 13. Commit esimerkkejä

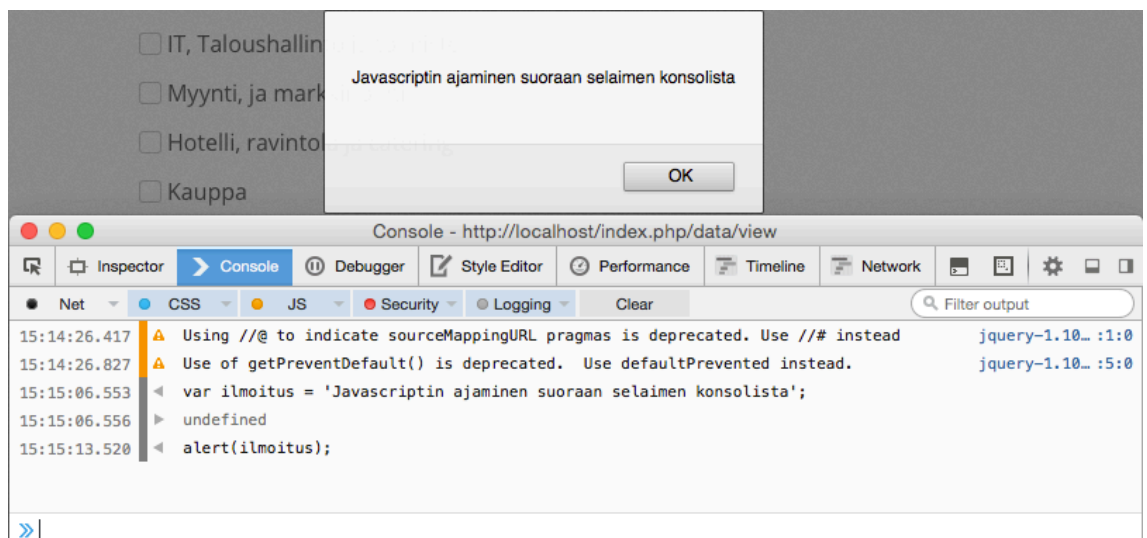
7.3 Internet-selaimen debug-työkalut

Nykyaikana Internet-selaimissa on hyvät debug-työkalut sisäänrakennettuna. Niiden opettelu onkin tärkeä osa web-kehitystä. Kuvassa 14 on Mozilla Firefox Internet-selaimen debug-työkalu. Siinä on Network välilehti valittuna mikä näyttää tehdyt HTTP-kutsut. Kutsuista näkee mm. polun, tyyppin, koon ja kutsuun kuluneen ajan. Kutsun valitsemalla näkee tarkempia tietoja (kuvassa 14 oikealla) kuten otsikko tiedot, keksit sekä palvelimelta saadun vastauksen. Kutsuja pystyy myös muokkaamaan ja lähettämään uudelleen. Muokkaamalla kutsuja pystyy esimerkiksi testaamaan miten järjestelmä reagoi, kun sille lähetetään erilaisia parametreja. Koska HTTP-kutsujen muokkaus ja generointi on näinkin helppoa, ei järjestelmien toteutuksessa saa luottaa asiakkaan päässä tehtyyn datan validointiin. Kaikki mitä palvelimelle lähetetään pitää tarkastaa ja varmistaa oikeaksi, koska sille tuleva data ei ole rajoitettu vain käyttöliittymässä määriteltyihin osiin.



KUVA 14. Firefox-selaimen debug-työkalut

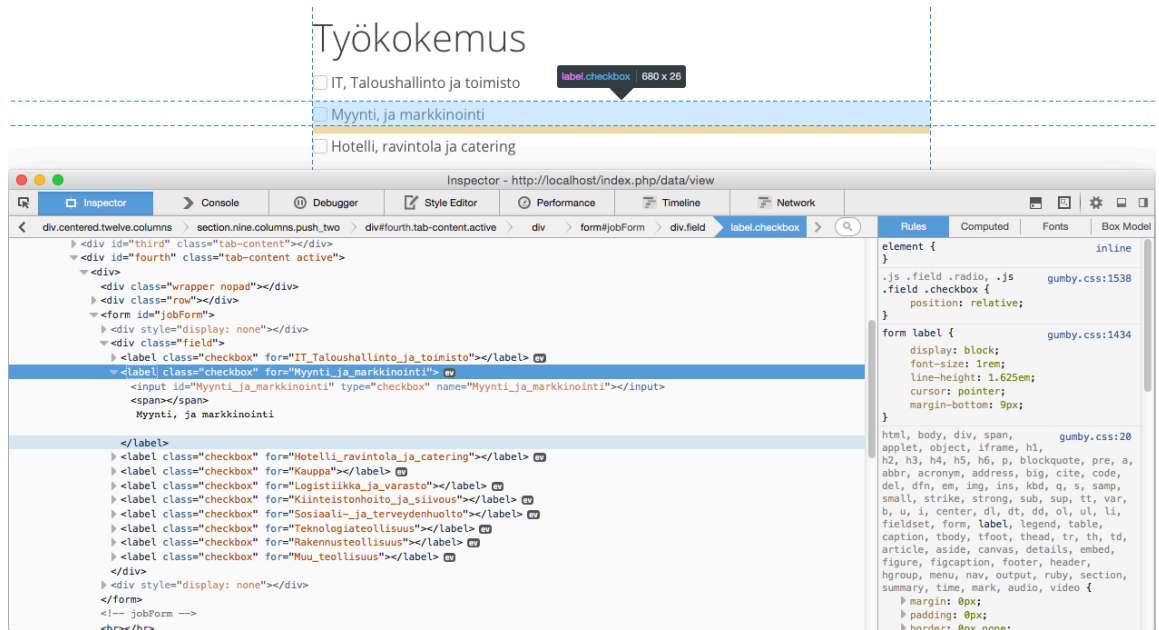
Muita debug-työkalujen tärkeitä toimintoja on mm. konsoli (kuva 15). Konsoli näyttää jos sivulla on tapahtunut virheitä. Myös omia tulostuksia voi laittaa järjestelmän tilasta konsoliin käyttäen JavaScript:iä. Konsoli mahdollistaa JavaScript-koodin ajamisen suoraan ilman, että järjestelmän tiedostoja tarvitsee muokata. Kuvassa 15 konsolissa luodaan muuttuja ilmoitus, johon asetetaan tekstinpätkä. Tämä muuttuja sitten näytetään alert ikkunassa.



KUVA 15. Debug-konsoli

Inspector-työkalulla (kuva 16) voi katsoa sivun HTML-koodia. Se mahdollistaa elementin valinnan ja näyttää suoraan tämän koodin sekä CSS-määrittelyt. Inspector on erinomainen työkalu, kun sivu näyttää jonkin elementin väärin. Sillä on helppo löytää mah-

dolliset elementtien puuttuvat loppu tagit tai väärin luodut elementin parametrit. Uusissa Firefox-selaimissa inspector näyttää myös elementtiin liitettyt JavaScript tapahtumat.



KUVA 16. Firefox-selaimen inspector-työkalu

8 REKRYTOINTIREKISTERIN TOIMINTA

8.1 Käyttäjien salasanat

Käyttäjät autentikoidaan käyttäjänimen ja salasanan mukaan, jotka tallennetaan tietokantaan. Ennen tallennusta salasanat ajetaan bcrypt KDF-funktion kautta käyttäen satunnaista suolaa. KDF-funktio (key derivation function) tuottaa johdetun avaimen käyttäen pohja avainta eli salasanaa sekä parametreja suola ja iteraatiomäärä (Kaliski 2000, 8). KDF tuottaa kuvan 17 mukaisia merkkijonoja. Nämä avaimet on johdettu samasta salasanana1234 merkkijonosta. Satunnaisen suolan ansiosta, mikä sisältyy avaimeen, bcrypt tuottaa erilaisen tuloksen vaikka sisääntulo on sama. Iteraatiomäärällä voi vaikuttaa kuinka paljon resursseja avaimen tekoon käytetään. Kaliski (2000) kertoo miten iteraatiomäärän tarkoituksena on tehdä avainten laskennasta vaikeampaa. Tämä estää hyökkääjää laskemasta suuria määriä avaimia kerralla vaikuttamasta suuresti yksittäisen avaimen laskentaan. (Kaliski, 2000, 8.) Kuvan 17 avaimista ei pysty suoraan päättelemään mikä alkuperäinen salasana on. Käyttäjän salasana pystytään tarkastamaan vertaamalla KDF:n tulosta tietokantaan tallennettuun avaimeen. Tällä tavalla tehtäessä järjestelmään ei tarvitse tallentaa salasanajoja selkokielisinä.

```
$2y$10$VhkiwCVfJNtISIVbu/6nz.FRdgUuMoBg8kv4ByUg09DaL/92woSRi
$2y$10$BSDDLih6/X4cZ7ZwuYAf.eojlBIWr6WHYOMNfm9fqp78Kvb0QoIn.
$2y$10$cvcwBrIKgabqGCjx81f8s0KPD6uTT5qAaC02Dt.tfBuHFxX5mkiWi
$2y$10$NGh.SfDhyVTveW9xUnt1oux0TpVWn5Fq/2AVwn5mr2wAk/52JgcgC
$2y$10$SgpLYBZar1KwzfUB2Bsgoe016LbXCAGf9hHxVoVoFfAJto9IYqQrG
```

KUVA 17. Salasanan avaimia

8.2 Käyttäjätilit ja niiden hallinta

Rekrytointirekisterissä on kolme erilaista käyttäjäryhmää: oppilas, pääkäyttäjä ja admin. Oppilas pystyy lisäämään ja poistamaan omia tietojaan. Pääkäyttäjä voi selata ja hakea järjestelmään tallennettujen oppilaiden tietoja. Admin käyttöoikeuksilla voi poistaa käyttäjiä, vaihtaa näiden salasanan ja lisätä pääkäyttäjä tilejä. Eri käyttäjäryhmät on erotettu toisistaan eikä niillä ole pääsyä muiden näkymiin. Järjestelmässä pääkäyttäjä on hämäävästi nimetty company nimellä. Joten tulevissa koodiesimerkeissä, kun viitataan pääkäyttäjään, on se company nimellä. Järjestelmän käyttäjien autentikointi tapahtuu

salasanan ja keksien avulla. Käyttäjän kirjautumisen onnistuessa keksiin asetetaan minimaaliset käyttöoikeudet sille on järjestelmään. Tässä käytetään apuna CodeIgniter:n sisäänrakennettuja sessionhallinta ominaisuuksia.

Kun käyttäjä kirjautuu järjestelmään, tarkastetaan sen kirjautumistiedot `check_database` funktiossa (kuva 18). Rivillä kolme haetaan post datasta login niminen tieto. Tämä tehdään CodeIgniter:n kautta, joka tekee XSS-filtteröinnin tulevalle datalle. Rivillä neljä kutsutaan loginhandler mallia ja sen `userlogin` funktiota. `Userlogin` funktio tarkastaa käyttäjänimen ja salasanan tietokannasta kuvan 19 koodilla.

```

1 <?php
2 function check_database($password){
3     $login = $this->input->post('login');
4     $result = $this->loginhandler->userlogin($login, $password);

```

KUVA 18. Kirjautuminen

```

1 <?php
2 function userLogin($username, $password){
3     $em = $this->doctrine->em;
4     $user = $em->getRepository('Entity\Users')->findOneBy(array('username' => $username));
5     if(isset($user) && password_verify($password, $user->getPassword())){
6         return $user;
7     }else{
8         return false;
9     }
10 }

```

KUVA 19. Kirjautumistietojen tarkastus

Kuvassa 19 rivillä neljä haetaan Doctrine ORM-kirjastoa apuna käyttäen `user` olio jolla on käyttäjänimi `username`. Rivillä viisi tarkastetaan löytyikö käyttäjä ja täsmääkö sen salasana. Jos tämä onnistuu, palautetaan kyseisen käyttäjän olio tai epäonnistuessaan palautetaan `false` (rivi 8). `password_verify` funktio (rivi 5) on PHP:n oma funktio jolla pystytään tarkastamaan onko annettu salasana (`password`) sama kuin kantaan tallennettu salasanan avain. Kantaan tallennettu avain on luotu PHP:n `password_hash` funktiolla, joka käyttää luvussa 8.1 esiteltyä `bcrypt` KDF-funktiota. Käyttäjänimen ja salasanan tarkastuksen jälkeen `check_database` funktio (kuva 18) jatkuu käyttäjätason tarkistuksella (kuva 20).


```

6  if($result){
7  $company = $this->loginhandler->checkIfCompanyLogin($result);
8  if($company){
9  $sess_array = array(
10     'id' => $result->getId(),
11     'login' => $result->getUsername(),
12     'user' => 'company'
13     );
14  }else{
15  $sess_array = array();
16  $sess_array = array(
17     'id' => $result->getId(),
18     'login' => $result->getUsername(),
19     'curriculum' => $result->getCurriculum(),
20     'user' => 'student'
21     );
22  }
23  $this->session->set_userdata('logged_in', $sess_array);
24  return TRUE;
25  }else{
26  $this->form_validation->set_message(
27     'check_database', '<li class="danger alert">Tarkasta käyttäjätunnus ja salasana.</li>');
28  return false;
29  }
30 }

```

KUVA 20. Käyttäjätason tarkastus

Jos userLogin funktiolta (kuva 19) saatiin onnistunut tulos (kuva 20, rivi 6), tarkastetaan kuuluuko käyttäjä pääkäyttäjiiin (rivi 7). CheckIfCompanyLogin tarkastaa tietokannasta kuuluuko käyttäjä tähän ryhmään. Jos se palauttaa true asetetaan käyttäjälle pääkäyttäjän oikeudet (rivi 12), muuten asetetaan se oppilas käyttäjäksi (rivi 20). Kirjautumisessa tallennetaan myös käyttäjän id, käyttäjänimi sekä oppilaan tapauksessa sen opintolinja. Rivillä 23 nämä tiedot tallennetaan kekseihin käyttäen CodeIgniter:n session luokkaa. Check_database funktion tuloksen mukaan käyttäjä ohjataan, joko oppilaan tai pääkäyttäjän sivulle. Kirjautumisen epäonnistuessa käyttäjä ohjataan etusivulle virheen kera, missä kehoitetaan tarkastamaan käyttäjätunnus ja salasana (rivi 27).

8.3 Tietojen tallennus järjestelmään

Kun käyttäjä on luonut uudet tilin Rekrytointirekisteriin, siirretään hänet sivulle (kuva 21) missä hän voi täyttää haluamansa tiedot ja tallentaa ne järjestelmään. Sivun on jaettu välilehtiin helpottamaan tietojen välillä navigointia. Eri välilehdet ja näiden sisältämät kokonaisuudet on jaettu omiin HTML-lomakkeisiin. Esimerkiksi henkilötiedot ja kieli-taito ovat omissa lomakkeissaan. Tällä tavalla kaikkia sivun lomakkeita ei ole pakko lähettää kerralla vaan ne voidaan tallentaa osissa. Tätä käytetään hyväksi tietojen muokausvaiheessa.

Rekryointirekisteri
Kirjautuminen

Profiili
TAMK opinnot
Aikaisempi koulutus
Työkokemus
Vahvuudet
Lisätiedot

Tiedot tallennetaan väliaikaisesti vaihtaessasi välilehteä. Voit jatkaa täyttämistä myöhemmin kirjautumalla uudestaan sisään. Tietojasi käytetään hakuun vasta, kun olet itse tallentanut ne lisätiedot välilehdellä löytyvällä napilla.

Henkilötiedot

* vaaditut kentät

Etunimi *

Sukunimi *

Koulun email *

Opiskelijanumero *

Lähiosoite *

Postinumero *

KUVA 21. Tietojen syöttö

Kun käyttäjä on täyttänyt tietonsa ja painaa tallenna nappia, lähetetään tiedot ajax-kutsulla palvelimelle jossa ne tarkastetaan ja tallennetaan tietokantaan. Data lähetetään kuvan 22 JavaScript-koodilla. Rivillä kuusi määritellään mitä dataa halutaan lähettää kutsussa mukana. Siihen valitaan kaikki lomakkeet mitä sivulla on ja ne muutetaan sopivaan muotoon käyttäen apuna jQuery:n serialize funktiota. Tämä data lähetetään kontrollerin save funktiolle (rivi 5). Jos tallennus onnistuu ja kontrolleri palauttaa ”Success” viestin (rivi 10), siirretään käyttäjä toiselle sivulle (rivi 12) missä hän voi jatkaa omalle kotisivulle. Virheen sattuessa (rivi 14) näytetään käyttäjälle kontrollerin lähettämä virheviesti (rivi 16). Virheviesti näytetään Noty-kirjaston avulla.

```

1 function submitAllForms(){
2   if(!errorCheck_data()){
3     $.ajax({
4       type: "POST",
5       url: "save",
6       data: $("#courseForm, #educationForm, #trainingForm, #jobForm, #languageForm, #strengthForm,
7           #studentForm, #refereeForm, #thesisForm, #workexperienceForm, #hakuForm, #moreTamkForm")
8       .serialize(),
9       success: function(data){
10        var result = jQuery.parseJSON(data);
11        if(result.status == "Success"){
12          window.onbeforeunload = null;
13          window.location.replace($('div[data-base]').data('base') + 'index.php/data/view/result');
14        }
15        else if(result.status == "Error"){
16          $.noty.closeAll();
17          noty({text: result.msg, type: 'error'});
18        }
19      }
20    });
21    return false;
22  };

```

KUVA 22. Datan lähetys palvelimelle

Kuvassa 23 on save funktio, joka ottaa lähetetyn datan vastaan. Kuvassa 23 rivillä kuusi asetetaan kaikki kutsussa ollut data userData muuttujaan. Tämä data annetaan datafactory mallin saveUserData funktiolle (rivi 8), joka hoitaa tietokantaan tallennuksen. SaveUserData antaa poikkeuksen jos jokin epäonnistuu tallennuksessa. Tällaisessa tapauksessa selaimelle palautetaan poikkeuksen mukana ollut virheviesti, joka näytetään käyttäjälle (rivit 17–20).

```

1  <?php
2  function save(){
3      if(!$this->session->userData('userId')){
4          redirect('home/logout');
5      }
6      $userData = $this->input->post(NULL, TRUE);
7      try{
8          $this->datafactory->saveUserData($userData);
9          $autosave = $this->autosavefactory->getUsersAutosave($this->session->userData('userId'));
10         if($autosave != null){
11             $this->autosavefactory->deleteAutosave($autosave->getId()->getId());
12             $this->session->unset_userdata('autosave');
13         }
14         $result['status'] = "Success";
15         echo json_encode($result);
16     }
17     catch(Exception $e){
18         $result['status'] = "Error";
19         $result['msg'] = $e->getMessage();
20         echo json_encode($result);
21     }
22 }

```

KUVA 23. Kontrollerin save funktio

Lyhennetty versio saveUserData funktiosta on esitetty kuvassa 24. Datan tallennuksessa käytetään hyväksi tietokannan transaktiota. Eli mitään muutoksia ei tehdä tietokantaan ennen kuin tiedetään, että kaikki muutokset saadaan varmasti tallennettua. Koska tiedot on jaettu moneen tietokannan tauluun, on tällä tavalla hyvä varmistaa, ettei tauluihin tallenneta dataa mikä ei ole oikein. Jos samaan transaktioon haluaa lisätä useammassa eri funktiossa tehtäviä toimia, täytyy Doctrine:n:n entitymanager (em) instanssi antaa jokaiselle funktiolle. Doctrine tekee em:n kautta kaikki tietokantaan tehtävät toimenpiteet. Koska kaikki data tallennetaan yhdessä transaktiossa, pitää sama em millä se on aloitettu antaa funktioille, jotka tallentavat datan. Kuvassa 24 rivillä 3 haetaan tämä em ja transaktio aloitetaan rivillä neljä. Jos virheitä ei tule tallennetaan data kantaan commit käskyllä rivillä 11. Jos jokin toimenpide aiheuttaa poikkeuksen otetaan se kiinni (rivi 13) ja peruutetaan kaikki muutokset mitä yritettiin tehdä (rivi 14).

```

1 <?php
2 public function saveUserData($userData){
3     $this->em = $this->doctrine->em;
4     $this->em->getConnection()->beginTransaction();
5     try{
6         $this->saveStudent($userData);
7         $this->saveEducation($userData);
8         $this->saveLanguage($userData);
9         $this->saveThesis($userData);
10        $this->em->flush();
11        $this->em->getConnection()->commit();
12    }
13    catch(Exception $e){
14        $this->em->getConnection()->rollback();
15        throw $e;
16    }
17 }

```

KUVA 24. Transaktion käynnistys

Eri osien tallennukset on jaettu omiin funktioihinsa selkeyttämään ohjelman rakennetta. Esimerkiksi saveThesis funktio on kuvan 25 mukainen. Rivillä kolme tarkastetaan, että käyttäjän antamassa datassa on jotain tallennettavaa. Rivillä 4 tämä data annetaan thesisfactory:n addThesis funktiolle (kuva 26), joka tekee itse tietokantaan tallennuksen. Kuvassa 26 rivillä neljä tarkastetaan onko funktiolle annettu em vai pitääkö hakea uusi. Jos em sisältää jotain, kuuluu kutsu johonkin transaktioon, mutta muuten kutsu on erillinen muista ja sille voidaan hakea oma em (rivi 5). Rivillä kahdeksan haetaan kannasta käyttäjä, jolla on annettu id. Jos tätä ei löydy heitetään poikkeus (rivi 10). Riveillä 12–14 asetetaan thesis olioon haluttu data. Rivillä 15 kerrotaan em:lle, että tämä olio halutaan tallentaa tietokantaan. Jos transaktiota ei ole manuaalisesti laitettu päälle, niin em hoitaa tallennuksen rivillä 16. Mutta tässä tapauksessa rivillä 16 ei tehdä mitään, koska transaktio on käynnistetty manuaalisesti.

```

1 <?php
2 function saveThesis($userData){
3     if($userData['topic'] != ''){
4         $this->thesisfactory->addThesis($this->session->userdata('userId'),
5             $userData['client'],
6             $userData['topic'], $this->em);
7     }
8 }

```

KUVA 25. Datafactory:n saveThesis funktio

```

1 <?php
2 public function addThesis($userId, $client, $topic, &$em = NULL){
3     try{
4         if($em == NULL){
5             $em = $this->doctrine->em;
6         }
7         $thesis = new Entity\Thesis;
8         $user = $em->find('Entity\Users', $userId);
9         if(!$user){
10            throw new Exception('User not found.');
```

KUVA 26. Opinnäytetyön tallennus tietokantaan

Doctrine tallentaa oletuksena kaiken transaktion avulla, kun kutsutaan flush funktiota. Tässä tapauksessa on haluttu ottaa käyttöön manuaalinen tapa. Tällä tavalla on pyritty selventämään, että kaikki data mitä saveUserData käyttäjältä saa, tallennetaan samassa paketissa. Näin myös funktioita jotka tallentavat datan tietokantaan voidaan uudelleen käyttää. Tämä parantaa ylläpidettävyyttä, koska funktioista ei ole tarvinnut tehdä kahta eri versiota. Tätä samaa funktiota käytetään, kun tallennetaan dataa minkä käyttäjä lisää myöhemmin, mutta tällöin se ei kuulu mihinkään transaktioon. Muu datan tallennus on toteutettu samaan tyyliin thesis tallennuksen kanssa. Vain käytetyt oliot eli taulut ja saatu data on erilaista.

Kun käyttäjä muokkaa tietojaan näytetään hänelle sama näkymä kuin tietojen syötössä. Sivulle ladataan vain eri JavaScript tiedostot sekä HTML-koodin sekaan laitettun PHP:n avulla määritellään mitä erilaista sivulla näkyy. Esimerkiksi muokkaus näkymässä, jokaisella välilehdellä on tallenna nappi mikä ei näy tietojen syötössä. Nappi näytetään kuvan 27 koodilla. Rivillä yksi tarkastetaan ollaanko sivulla millä nappi pitää näyttää. Jos ollaan, lisätään rivit 2–4 HTML-koodiin mikä näytetään. Page muuttuja (rivi 1) asetetaan kontrollerilla halutuksi, kun sivuja ladataan.

```

1 <? if($page == 'student'){ ?>
2     <div class="pretty medium primary btn three columns">
3         <a onClick="javascript: submitStrength();">Tallenna</a>
4     </div>
5 <? } ?>
```

KUVA 27. Tallenna napin näyttö

8.4 Käyttäjätietojen selaus ja haku

Pääkäyttäjä oikeudet omaavalla on mahdollisuus selata ja katsella järjestelmään tallennettujen käyttäjien tietoja. Tämä tapahtuu omassa näkymässään (kuva 28). Kuvassa 28 vasemmalla on listattu tallennetut käyttäjät ja tietyn voi hakea etsimällä nimen mukaan sen yllä olevasta kentästä. Käyttäjien vahvuuksia pystyy myös vertailemaan erillisessä vertailunäkymässä (kuva 29). Vertailunäkymässä voi valita muutamasta erilaisesta versiosta, jotka näyttävät saman tiedon, mutta hieman eri tavalla.

Rekrytointirekisteri Etusivu Oppilaat pääkäyttäjä Kirjaudu ulos

Muokkaa haku
Tallenna sivu pdf muodossa
Vertaile Lisää vertailuun

Etsi opiskelija

Ahtu Tiainen
Arni Kippo
Reimar Snickare
Salle Narvala
Kristelle Kavalunas
Nii Hautakoski
Sinikukka Alssén
Conan Vehanius
Aleksanderi Keisu
Hamlet Pokilainen
Lisää

Kommentit Merkit: 0

Henkilötiedot Tallenna

Tamk opinnot

Kurssi
Lisää

Tutkinnot

Tutkinto	Oppilaitos	Valmistusmisvuosi	Suoritettu %
----------	------------	-------------------	--------------

Kurssit

Nimi	Järjestäjä	Kesto	Valmistusmisvuosi
------	------------	-------	-------------------

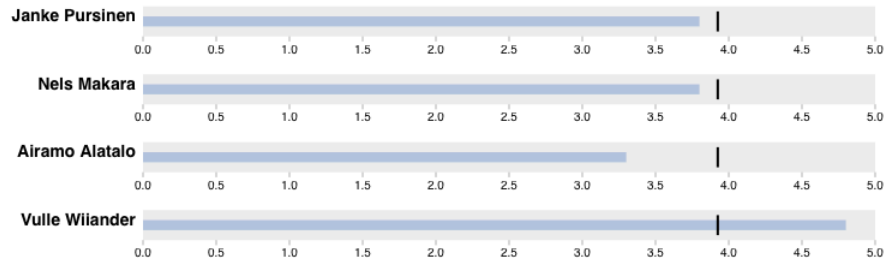
Kieliosaaminen

Kieli	Taito	Kieli	Taito
-------	-------	-------	-------

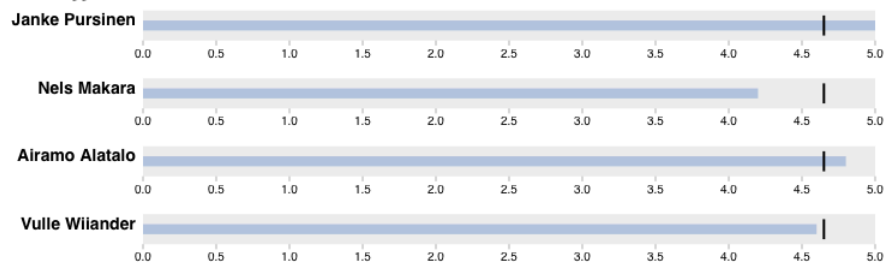
KUVA 28. Tietojen selauksen näkymä



Viestintä



Kansainvälisyys



KUVA 29. Vertailunäkymä

Luvussa 2 esiteltiin tietojen hakuominaisuus ja sen käyttöliittymä (kuva 2). Tämä käyttäjien haku tapahtuu käyttäen apuna Doctrine:n query builder ominaisuutta. Query builder:lla pystyy luomaan dynaamisesti SQL-kutsuja, käyttäen erilaisia SQL-kielen osia, kuten select ja where. Tämä helpottaa haun luontia, koska siitä pystytään helposti lisäämään tai poistamaan osia hakukriteerien mukaan.

Pohjana toimii kuvan 30 mukainen haku. Kuvan 30 rivillä 3 luodaan Doctrine:n query builder olio ja riveillä 5-7 siihen lisätään parametrit. Tämä vastaa kuvan 31 mukaista SQL-kutsua. Koska tämä hakee kaikki käyttäjät, pitää hakuun lisätä käyttäjiä rajoittavia parametreja. Esimerkiksi tiettyjen kurssien omaavat käyttäjät lisätään hakuun kuvan 32 mukaan.

```

1 <?php
2 $em = $this->doctrine->em;
3 $qb = $em->createQueryBuilder();
4
5 $qb ->select(array('u.id'))
6     ->from('Entity\Users', 'u')
7     ->groupBy('u.id');
```

KUVA 30. Query builder haku

```
1 SELECT u.id FROM Users u GROUP BY u.id;
```

KUVA 31.

```
1 <?php
2 if(!empty($courses)){
3     $courseQuery = $em->createQueryBuilder();
4     $courseQuery->select('u2.id')
5         ->from('Entity\Users', 'u2')
6         ->from('Entity\Allcourses', 'c2' )
7         ->from('Entity\Usercourses', 'uc2')
8         ->andWhere($courseQuery->expr()->andX(
9             $courseQuery->expr()->eq('u2.id', 'uc2.users'),
10            $courseQuery->expr()->eq('c2.id', 'uc2.allcourses'),
11            $courseQuery->expr()->in('c2.name', ':coursesName')))
12         ->groupBy('u2.id')
13         ->having($courseQuery->expr()->gte($courseQuery->expr()->count('u2.id'), count($courses)));
14     $courseQuery->setParameter('coursesName', $courses);
15     $query = $courseQuery->getQuery();
16     $courseResult = $query->getResult();
17     if($courseResult){
18         $qb ->andWhere($qb->expr()->in('u.id', ':courseId'))
19         ->setParameter('courseId', $courseResult);
20     }
21 }
```

KUVA 32. Kurssien rajaus

Koska query builder ei tue alikutsuja pitää nämä osat tehdä erillisinä kutsuina. Kuvassa 32 rivillä 3 luodaan uusi query builder olio, johon asetetaan parametrit riveillä 4–13. Riveillä 5–7 valitaan mistä tauluista haetaan. Riveillä 8–11 asetetaan taulujen väliset yhteydet. Riveillä 9 ja 10 olevat eq-funktio vastaa yhtäsuuruusoperaattoria. Rivillä 13 valitaan vain käyttäjät joilla on kaikki vaaditut kurssit. Tässä käytetään apuna valittujen kurssien lukumäärää. Rivillä 14 asetetaan ”:coursesName” kohtaan valitut kurssit. Tämä parametrusointi auttaa estämään injektioit, koska Doctrine hoitaa setParameter funktiossa tarvittavat toimenpiteet injektioiden estoon. Rivillä 16 haku toteutetaan ja tulokset tallennetaan. Jos sopivia käyttäjiä löytyi (rivi 17) lisätään ne (rivit 18–19) aikaisemmin luotuun hakuun. Esimerkiksi jos haussa on ollut yksi kurssi luo kuvan 32 mukainen Doctrine haku kuvan 33 SQL-kutsun.

```
1 SELECT u0_.id AS id0 FROM users u0_, allcourses a1_, usercourses u2_
2 WHERE u0_.id = u2_.users_id AND a1_.id = u2_.allCourses_id
3 AND a1_.name IN ('Yrityksen_perustoiminnot_Toimiva_firma')
4 GROUP BY u0_.id HAVING COUNT(u0_.id) >= 1;
```

KUVA 33. Kurssirajauksen SQL-versio

Muut vaatimukset kuten hakutoive tai kielet lisätään hakuun samalla tavalla. Näistä luodun haun tuloksesta saadaan käyttäjät, jotka listataan pääkäyttäjälle. Käyttäjät näytetään vahvuuksien yhteenlasketun summan mukaan suurimmasta alkaen. Tähän summaan lisätään vielä, kursseista saadut lisäpisteet. Lisäpisteen saa jos käyttäjällä on suoritettu kurssi jolle on asetettu kategoriaksi jokin valituista vahvuuksista.

Tätä järjestelyä varten tarvitaan vielä yksi haku, missä yhdistetään vahvuuksien summa ja lisäpisteet. Tässä käytetään apuna SQL:n union käskyä. Union yhdistää monen select käskyn tulokset yhdeksi (MySQL Manual, 2015). Doctrine:n query builder ei tue union käskyä joten se on toteutettu käyttäen normaalia SQL-kieltä, mutta se ajetaan Doctrine:n kautta kuvan 34 mukaisesti.

```

1  <?php
2  $sql = "SELECT qq.id, qq.firstname, qq.secondname, SUM(qq.Summa) AS Summa FROM" .
3        "(select u.id, u.firstname, u.secondname, COUNT(c.value) AS Summa " .
4         "FROM users u, usercourses uc, allcourses c WHERE u.id=uc.users_id " .
5         "AND c.id=uc.allCourses_id AND c.value IN (" . $strengthParameters . ") " .
6         "AND u.id IN (" . $id . ") " .
7         "GROUP BY u.id " .
8         "UNION " .
9         "SELECT u.id, u.firstname, u.secondname, SUM(us.value) AS Summa " .
10        "FROM users u, allstrengths s, userstrengths us " .
11        "WHERE u.id=us.users_id " .
12        "AND us.allStrengths_id=s.id " .
13        "AND s.name IN(" . $strengthParameters . ") " .
14        "AND u.id IN (" . $id . ") " .
15        "GROUP BY u.id) AS qq " .
16        "GROUP BY qq.id " .
17        "ORDER BY Summa DESC";
18
19  $connection = $em->getConnection();
20  $query = $connection->prepare($sql);
21  $query->execute();
22  $result = $query->fetchAll();

```

KUVA 34. Vahvuuksien SQL-kutsu

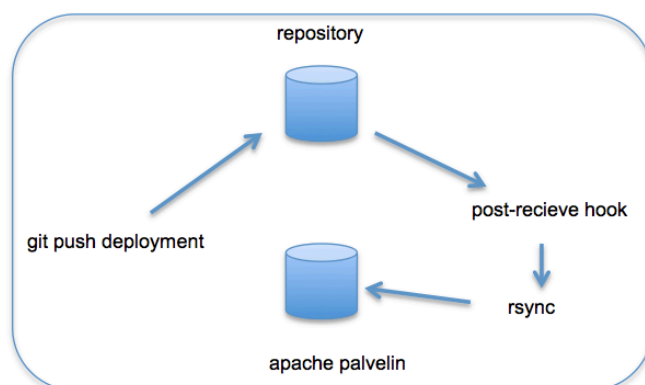
Kuvassa 34. riveillä 2-7 haetaan valittujen vahvuuksien yhteenlaskettu summa käyttäjille. Riveillä 9-15 haetaan lisäpisteet käyttäjille. Nämä osat yhdistetään union käskyllä (rivi 8) ja järjestetään summan mukaan laskevasti (rivi 17). Tämä kutsu annetaan Doctrine:lle (rivi 20), joka suorittaa kutsun (rivi 21). \$id muuttujien paikalle laitetaan rajauksessa saadut käyttäjät ja \$strengthParameters kohtaan tulee valitut vahvuudet. Kuvan 34 haku palauttaa käyttäjien id:t, jotka vastasivat hakuparametreja.

9 JÄRJESTELMÄN KÄYTTÖNOTTO

Järjestelmän uuden version saattaminen käyttäjille on tärkeässä osassa web-pohjaista järjestelmää. Käyttäjien kannalta parasta on, että päivitys tapahtuu ilman käyttökatkoja ja muutenkin huomaamattomasti. Tämä käyttöönotto-prosessi on hyvä hoitaa aina samalla tavalla, jotta mahdollisilta virheiltä vältyttäisiin.

Rekrytointirekisterin käyttöönotto-prosessi on pyritty pitämään helppona. Koska järjestelmä ei vaadi kääntämistä voidaan yksinkertaisesti korvata muuttuneet tiedostot palvelimella. Jotta tämä tapahtuisi varmasti ja siirrettävät tiedostot ovat oikeat ja menevät oikeaan paikkaan, käytetään apuna paria työkalua. Nämä työkalut ovat Git versionhallinnan git-hooks niminen toiminto sekä rsync ohjelma. Git-hooks toiminnolla voi ajaa skriptejä, kun jokin toiminto suoritetaan. Rsync on tiedostojen kopiointiohjelma jolla pystytään pitämään kaksi eri sijaintia ajan tasalla. Rsync kuuluu Linux järjestelmien perustyökaluihin ja löytyy oletuksena monesta järjestelmästä. Git-hooks ja rsync yhdistämällä voidaan järjestelmä päivittää helposti ja aina samalla tavalla. Koska järjestelmän uusi versio päivitetään versionhallinnan kautta, tulee myös kaikki muutokset dokumentoitua.

Rekrytointirekisterin Git varasto sisältää deployment haaran, missä on tuotannon tietokannan sijainnit, salasanat ja muut asetukset. Varastoon on asetettu post-receive hook, joka ajetaan aina, kun sinne työnnetään jotain. Tämä hook tarkastaa onko haara deployment ja jos on, lähettää se uuden version oikeaan paikkaan rsync ohjelmalla. Post-receive hook on normaali bash-skripti, jonka Git ajaa. Kuvio 6 havainnollistaa miten tämä tapahtuu.



KUVIO 6. Rekrytointirekisterin käyttöönotto-prosessi

Kuvassa 35 on Rekrytointirekisterissä käytössä oleva post-receive hook, jolla päivitetään uusi versio järjestelmästä. Kuvassa 35 rivillä 4 parsitaan haara mihin työnnetään. Rivillä 17 otetaan uusi versio versionhallinnasta väliaikaiseen kansioon. Tässä annetaan git komennolle work-tree parametri millä määrätään mihin kansioon tiedostot halutaan ladata. Git-dir parametrilla määritetään missä kansiossa varasto sijaitsee. F parametrilla varmistetaan, että mahdolliset lokaalit muutokset yli kirjoitetaan. Tällä pidetään huoli, että versio on täysin sama mitä varastossa on. Koska tiedostot kopioidaan aina uuteen kansioon (rivi 9) ei f parametri ole pakollinen. Esimerkiksi tilanteessa missä varasto haetaan suoraan palvelimen kansioon mistä ne tarjotaan, tämä parametri on hyvä olla ja sisältyykin mahdollisten muutosten takia skriptiin. Rivillä 18 siirretään uusi versio rsync ohjelmalla oikeaan sijaintiin. Tälle annetaan muutamia parametreja joilla valitaan mm. mitkä tiedostot jätetään siirrosta pois ja mitä tiedostoja ei saa poistaa. Delete parametri kertoo rsync:lle, että lähdehakemiston tiedostot joita ei ole kohdehakemistossa pitää poistaa. Tämän vuoksi osa palvelimen tiedostoista pitää suojella, ettei niitä poisteta. Tähän käytetään filter parametreja. A parametri on ns. archive moodi mikä sisältää hyödyllisiä parametreja tiedostojen siirtoon. Z parametri pakkaa siirrossa käytetyt tiedostot ja v parametri lisää ohjelman tulostuksen määrää. Riveillä 19–24 tarkastetaan tuliko virheitä. Muuttuja \$? sisältää viimeiseksi ajatun komennon virhekoodin.

```

1  #!/bin/bash
2  while read oldrev newrev refname
3  do
4      branch=$(git rev-parse --symbolic --abbrev-ref $refname)
5      if [ "$branch" == "deployment" ]; then
6          echo "Deploying new version..."
7          repodir="$HOME/repo-rekrytointirekisteri.git"
8          tempdir="$HOME/temp_repo/depoy-$$"
9          if mkdir $tempdir
10         then
11             #do nothing if success
12             echo "${tempdir} created."
13         else
14             echo "Error creating temp directory."
15             exit
16         fi
17         git --work-tree=$tempdir --git-dir=$repodir checkout -f deployment
18         rsync -avz --delete --exclude=*.gitignore --exclude=*.DS_Store --exclude=tests/ --filter=
          'P uploads/**' --filter='P old_json/**' --filter='P logs/**' --filter='P php-cgi/**'
          --filter='P phpMyAdmin' --filter='P favicon.ico' --filter='P robots.txt' --filter='P
          cache/**' "${tempdir}/datapankki/" $HOME/public_html
19         status=$?
20         if [ "${status}" -ne "0" ]; then
21             echo "Error: something when't wrong and we got exit code ${status}."
22         else
23             echo "Everything went okay."
24         fi
25         rm -rf $tempdir
26     fi
27 done

```

KUVA 35. Post-receive git-hook

Tässä tavassa on muutama huono puoli. Se ei ota huomioon tietokannan muutoksia ja ne tehdään järjestelmään käsin. Myöskään edellisen version palautukseen ei ole tehty automaattista toimintaa vaan se pitää palauttaa versionhallinnan kautta. Näiden puutteiden korjaaminen onnistuu esimerkiksi seuraavilla ehdotuksilla. Tietokannan päivitykseen voi käyttää Doctrine:n skeema ja migration ominaisuuksia, millä se pystyy automaattisesti päivittämään tietokannan oikeaan muotoon. Yleinen käytäntö edellisen version palautukseen on käyttää symbolista linkkiä. Uutta versiota siirrettäessä säilytetään vanha versio ja uudet siirretään näiden rinnalle. Siirron jälkeen linkki osoitetaan uuteen versioon. Jos se huomataan toimimattomaksi, siirretään linkki takaisin vanhaan. Tämä menettely estää myös ongelmat mitä sattuu, kun käyttäjä pyytää tiedostoja mihin on päivitys käynnissä.

10 JÄRJESTELMÄN YLLÄPITO

Rekrytointirekisteriä on päivitetty ja uusia ominaisuuksia lisätty useasti sen alkuperäisen käyttöönoton jälkeen. Lisätty on mm. tuki useammalle opinto-ohjelmalle, joka on aiheuttanut muutoksia myös tietokantaan. Järjestelmä on ollut hyvin joustava muutoksille eikä minkään osan toimintaa ole tarvinnut muuttaa suuresti.

10.1 Tietokannan hallinta

Tietokannan hallinnassa arvokas apu on ollut phpMyAdmin-työkalu (kuva 36). Tämä työkalu on kirjoitettu PHP:llä ja sitä voi pyörittää vaikka samalla palvelimella missä sivustokin on. Sillä pystyy mm. selaamaan tietokannan tauluja, muuttamaan niiden arvoja ja ajamaan SQL-komentoja. Käyttäjien dataa sisältävän tietokannan käsittelyssä pitää olla tarkkana, ettei menetetä dataa. Komentojen vaikutus on tärkeää varmistaa testikannassa ennen niiden ajoa tuotantokäytössä olevassa kannassa. Tietokannasta onkin hyvä tehdä varmuuskopioita etenkin ennen muutoksia. Varmuuskopiointi onnistuu kätevästi phpMyAdmin export toiminnon kautta.

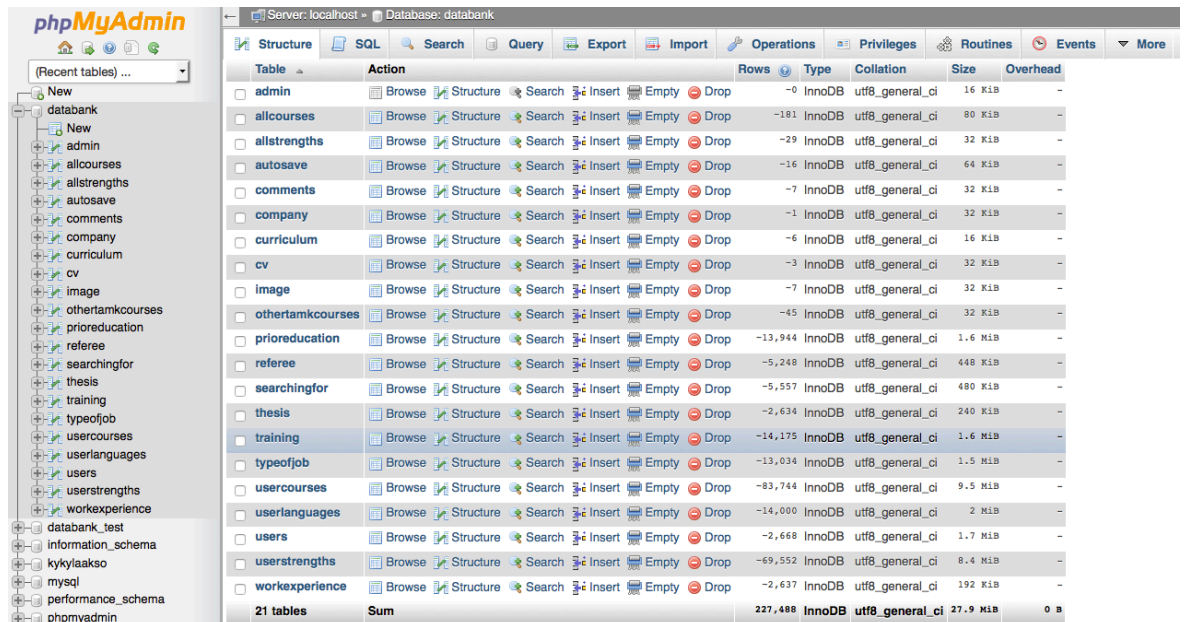


Table	Action	Rows	Type	Collation	Size	Overhead
admin	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8_general_ci	16 K1B	-
allcourses	Browse Structure Search Insert Empty Drop	-181	InnoDB	utf8_general_ci	80 K1B	-
allstrengths	Browse Structure Search Insert Empty Drop	-29	InnoDB	utf8_general_ci	32 K1B	-
autosave	Browse Structure Search Insert Empty Drop	-16	InnoDB	utf8_general_ci	64 K1B	-
comments	Browse Structure Search Insert Empty Drop	-7	InnoDB	utf8_general_ci	32 K1B	-
company	Browse Structure Search Insert Empty Drop	-1	InnoDB	utf8_general_ci	32 K1B	-
curriculum	Browse Structure Search Insert Empty Drop	-6	InnoDB	utf8_general_ci	16 K1B	-
cv	Browse Structure Search Insert Empty Drop	-3	InnoDB	utf8_general_ci	32 K1B	-
image	Browse Structure Search Insert Empty Drop	-7	InnoDB	utf8_general_ci	32 K1B	-
othertamkcourses	Browse Structure Search Insert Empty Drop	-45	InnoDB	utf8_general_ci	32 K1B	-
prioreducation	Browse Structure Search Insert Empty Drop	-13,944	InnoDB	utf8_general_ci	1.6 M1B	-
referee	Browse Structure Search Insert Empty Drop	-5,248	InnoDB	utf8_general_ci	448 K1B	-
searchingfor	Browse Structure Search Insert Empty Drop	-5,557	InnoDB	utf8_general_ci	480 K1B	-
thesis	Browse Structure Search Insert Empty Drop	-2,634	InnoDB	utf8_general_ci	240 K1B	-
training	Browse Structure Search Insert Empty Drop	-14,175	InnoDB	utf8_general_ci	1.6 M1B	-
typeofjob	Browse Structure Search Insert Empty Drop	-13,034	InnoDB	utf8_general_ci	1.5 M1B	-
usercourses	Browse Structure Search Insert Empty Drop	-83,744	InnoDB	utf8_general_ci	9.5 M1B	-
userlanguages	Browse Structure Search Insert Empty Drop	-14,000	InnoDB	utf8_general_ci	2 M1B	-
users	Browse Structure Search Insert Empty Drop	-2,668	InnoDB	utf8_general_ci	1.7 M1B	-
userstrengths	Browse Structure Search Insert Empty Drop	-69,552	InnoDB	utf8_general_ci	8.4 M1B	-
workexperience	Browse Structure Search Insert Empty Drop	-2,637	InnoDB	utf8_general_ci	192 K1B	-
21 tables	Sum	227,488	InnoDB	utf8_general_ci	27.9 M1B	0 B

KUVA 36. phpMyAdmin tietokannan hallinta

10.2 Palvelimen hallinta

Rekrytointirekisteri pyörii Linux ympäristön päällä, mutta käyttöönotto-prosessin ja phpMyAdmin:n avulla ei palvelimelle tarvitse normaaleissa päivitystilanteissa ottaa yhteyttä. Mutta virhetilanteissa tai jos pitää muuttaa esimerkiksi post-receive skriptiä, on hyvä osata käyttää myös Linux komentoriviä. Tärkeitä komentoja on mm. perus navigointi ja listauskomennot: `cd` ja `ls`, sekä tiedostojen kopiointi ja siirtokomennot: `cp` ja `mv`. Komentorivillä toimiva tekstieditori kuten `vim` tai `emacs` esimerkiksi asetustiedostojen muokkaukseen on myös hyvä osata.

Tärkeässä osassa on `ssh`:n käyttö. `Ssh` ohjelmalla otetaan yhteys palvelimelle. Yhteyden muodostamista helpottamaan voi ottaa käyttöön julkisen avaimen autentikointi. Kun käytetään julkista avainta, ei palvelimelle kirjautuessa tarvitse antaa salasanaa vaan tunnistautuminen tapahtuu avainten avulla. Esimerkiksi tilanteissa, kun työnnetään tai haetaan versionhallinnasta tietoa, avainten käyttö helpottaa huomattavasti. Kun käytetään avaimia, käyttäjä luo salaisen ja julkisen avaimen. Julkinen avain laitetaan palvelimelle johon halutaan yhteys. Seuraavalla kerralla palvelimelle yhdistäessä `ssh` tarkastaa, täsmääkö salainen ja julkinen avain toisiinsa. Jos avaimet hyväksytään, käyttäjä kirjataan automaattisesti sisään. Avaimia ei koskaan lähetetä serverin ja käyttäjän välillä vaan niillä tehdään laskutoimituksia, jolla varmistetaan, että käyttäjällä on oikea salainen avain, mikä kuuluu julkiseen avaimeen. Avainten vertailu tehdään esimerkiksi Diffie-Hellman-avaimenvaihtoprotokollalla.

10.3 Järjestelmän siirto

Rekrytointirekisteri siirrettiin toiselle palvelimelle, kun se oli ollut käytössä noin kuukauden. Tämä toimenpide aiheutti muutamia ongelmia. Esimerkiksi PHP:n versionumero laski pykälällä 5.3:een. PHP:n versiossa 5.4 esiteltiin toiminto `JsonSerializable`, jolla pystytään määrittelemään miten `json_encode` funktio muuttaa olion json muotoon. Tätä käytettiin hyväksi järjestelmässä, kun `user` olion tietoja lähetettiin selaimelle. Oliota oli muokattu niin, ettei käyttäjänimeä ja salasanaa aina lähetetty selaimelle, kun oliota lähetettiin json muodossa. Koska `JsonSerializable`:a ei enää voinut käyttää tehtiin uusi funktio (kuva 37), joka palauttaa vain halutut `user` taulun tiedot.

```

1 <?php
2 public function getUserInfo($id){
3     try{
4         $em = $this->doctrine->em;
5         $q = $em->createQuery("SELECT u.id, u.firstname, u.secondname,
6             u.email, u.address, u.postalcode,
7             u.country, u.city, u.sex, u.phone,
8             u.nationality, u.birthdate, u.driverslicence,
9             u.constraints, u.linkedin, u.studentnumber
10            FROM Entity\Users u WHERE u.id=:id");
11         $q->setParameter("id", $id);
12         return $q->getResult();
13     }
14     catch (Exception $e){
15         throw $e;
16     }
17 }

```

KUVA 37. Uusi käyttäjätietojenhaku

Järjestelmän siirto tehtiin siten, ettei tietoa hukkuisi eikä vanhaan järjestelmään jää dataa mitä ei ole uudessa. Jotta tämä onnistuu, siirron ajaksi vanha järjestelmä otettiin pois käytöstä. Koska kaikki CodeIgniter toiminnot menevät yhden index.php tiedoston kautta, järjestelmä saadaan helposti pois päältä. Tämän voi tehdä vaikka siirtämällä tai uudelleen nimeämällä index.php tiedosto. Jos käyttäjiä haluaa informoida käyttökatkosta, tiedoston voi korvata esimerkiksi ilmoituksella, että järjestelmä on poissa käytöstä.

Tietokannan datan siirrossa käytettiin phpMyAdminin export ja import toimintoja. Export ominaisuudella voi valita esimerkiksi mitkä taulut kopioidaan ja otetaanko pelkkä data tai myös taulujen luontiin tarvittavat käskyt. Se luo kaikesta taulun datasta insert käskyt joilla tieto saadaan siirrettyä. Export tekee käskyt taulujen aakkosjärjestyksen mukaan. Tästä aiheutuu ongelmia jos jokin taulu sisältää viitteitä toisiin tauluihin ja taulun data yritetään tallentaa uuteen kantaan ennen kuin viitteet on tallennettu. Jotta viitteet eivät ole ongelma voi taulut järjestää käsin oikeaan järjestykseen, mutta se on hankala ratkaisu varsinkin jos järjestelmä sisältää paljon tauluja ja dataa. Toinen vaihtoehto on ottaa viitteiden tarkastus pois datan siirron ajaksi. Sen voi tehdä asettamalla tietokannan FOREIGN_KEY_CHECKS asetus nollassi. Tämä on turvallista tehdä tässä tilanteessa, koska tiedetään, että datan viite-eheys on kunnossa. Asetus pitää vain muistaa laittaa takaisin päälle, kun datan siirto on tehty. Toinen ongelma mitä datan siirrossa voi tulla, on tiedoston kokorajoitus mitä import ottaa vastaan. Export mahdollistaa tietojen pakkauksen mikä voi auttaa tällaisessa tilanteessa. Jos pakkaus ei riitä datan voi esimerkiksi siirtää taulu kerrallaan. Datat siirto suoraan komentoriviltä on myös mahdollista ja tällöin kokorajoituksia ei ole. Järjestelmän muut osat siirrettiin muokkaamalla post-receive skripti osoittamaan uuteen järjestelmään. Versionhallinnan varasto siirrettiin myös samalle palvelimelle järjestelmän kanssa, mikä aiheutti mm. muutoksia skriptissä oleviin polkuihin.

11 POHDINTA

Rekrytointirekisteri ei ole erityisen monimutkainen tietojärjestelmä. Pääasiassa se on Internet-lomake, jonka tiedot tallennetaan tietokantaan. Näitä tietoja sitten voidaan katsella kootusti tai hakea tiettyjen kriteerien mukaan. Tietojen haku onkin kenties järjestelmän monimutkaisin osuus. Yksinkertaisuudestaan huolimatta on projektin rakentaminen ollut haastavaa ja opettavaista.

Järjestelmää kehittäessä on oppinut paljon ja näin jälkeempäin huomaa asioita joita olisi voinut tehdä toisella. Esimerkiksi kurssilistaus ja vahvuudet luodaan dynaamisesti JavaScript:n avulla, kun ne voisi luoda palvelimella PHP:llä. Koska ne luodaan käyttäjän selaimella pitää myös niiden sisältämät tiedot lisätä JavaScript:llä. Tämä monimutkistaa sivuston sisältämän JavaScript:n rakennetta, lisää turhaa tiedon siirtoa ja hidastaa sivujen latausta. Järjestelmä onkin vahvasti riippuvainen JavaScript:stä eikä toimi oikein ilman sitä.

Luvussa 5 esitettiin parannuksia järjestelmän tietokantaan. Näiden toteuttaminen vaatii isoja muutoksia tietokannan rakenteeseen joiden tekeminen on haastavaa, kun se sisältää käyttäjien dataa. Datan oikeellisuus pitää jotenkin varmistaa muutosten jälkeen ja siitä voi aiheutua seisona-aikaa järjestelmään. Muun muassa näiden vuoksi toimivaa systeemiä ei välttämättä kannata lähteä muuttamaan jos välitöntä tarvetta tai hyötyä muutoksille ei ole. Käyttäjänimen ja salasanan erotus users taulusta ei tuo mitään hyötyä tällä hetkellä, mutta esimerkiksi tilanteessa missä käyttäjien autentikointi pitää siirtää omalle palvelimelle kovan rasituksen vuoksi, pitää tämä erotus tehdä.

Projektissa käytetyt tekniikat eivät ole parhaita eikä kaikkia asioita ole tehty alan suositusten mukaisesti. Toisaalta työssä esitetyillä tekniikoilla pääsee hyvin alkuun uuden järjestelmän rakentamisessa. Huomiota kannattaa kiinnittää tietokannan rakenteeseen, koska data on järjestelmän ydin, sekä työkalujen toimivuuteen. On ikävää kehittää järjestelmää, kun työkalut ei toimit. Teoksen tekniikoiden avulla Rekrytointirekisteri on rakennettu toimivaksi järjestelmäksi, mikä tekee sen mitä pitää. Sillä on paljon kasvuvaraa, ja tulevaisuus näyttää valoisalta järjestelmän osalta

LÄHTEET

CodeIgniter. CodeIgniter Web Framework. 2015. Luettu 20.1.2015.
<http://www.codeigniter.com>

DB-Engines. DB-Engines Ranking. 2015. Luettu 22.3.2015.
<http://db-engines.com/en/ranking>

Kaliski, B. 2000. PKCS #5: Password-Based Cryptography Specification. Version 2.0. RSA Laboratories. Luettu 11.3.2015
<https://www.ietf.org/rfc/rfc2898.txt.pdf>

PHP Group. PHP Manual: Preface. 2015. Luettu 15.4.2015.
<http://php.net/manual/en/preface.php>

Symfony. Databases and Doctrine. 2015. Luettu 4.2.2015.
<http://symfony.com/doc/current/book/doctrine.html#creating-the-database-tables-schema>

Viinanen, J. 2014. Tampereen ammattikorkeakoulun liiketalouden yksikön rekrytointipalvelun tuotteistaminen. Liiketalouden koulutusohjelma. Tampereen ammattikorkeakoulu. Opinnäytetyö.

Zalewski, M. 2014. The Tangled Web. A guide to securing modern web applications. San Francisco: No Starch Press, Inc.

LIITTEET

Liite 1. Käyttötapaus – Muokkaa tietoja

Nimi: Muokkaa tietoja

Suorittaja: Oppilas

Esiehdot: Oppilas on luonut käyttäjätilin, oppilas on lisännyt tietonsa.

Kuvaus: Oppilas avaa Rekrytointirekisterin etusivun selaimellaan. Hän kirjaa oman käyttäjätilinsä tiedot käyttäjänimi ja salasana kenttiin. Suorittaja kirjautuu järjestelmään painamalla kirjaudu nappia.

Selaimeen tulee näkyviin oppilas näkymän etusivu. Oppilas painaa muokkaa tietoja linkkiä ja oppilaan aikaisemmin tallentamat tiedot tulevat näkyviin. Hän painaa välilehteä minkä tietoja haluaa muokkaa tai painamalla seuraava nappia siirtyy haluamaansa välilehteen. Oppilas muokkaa, poistaa tai lisää haluamansa kentän tietoja. Tämän hän tekee niin monta kertaa kunnes kaikki halutut kohdat välilehdellä on muokattu halutuiksi.

Suorittaja painaa tallenna nappia. Sivulle tulee näkyviin tallennus onnistui ilmoitus. Hän siirtyy seuraavalle välilehdelle minkä tietoja haluaa muuttaa ja toistaa edelliset toimenpiteet kunnes kaikki halutut tiedot on muokattu.

Poikkeukset: Kirjautuminen epäonnistuu: kirjautumistiedot ovat väärät.

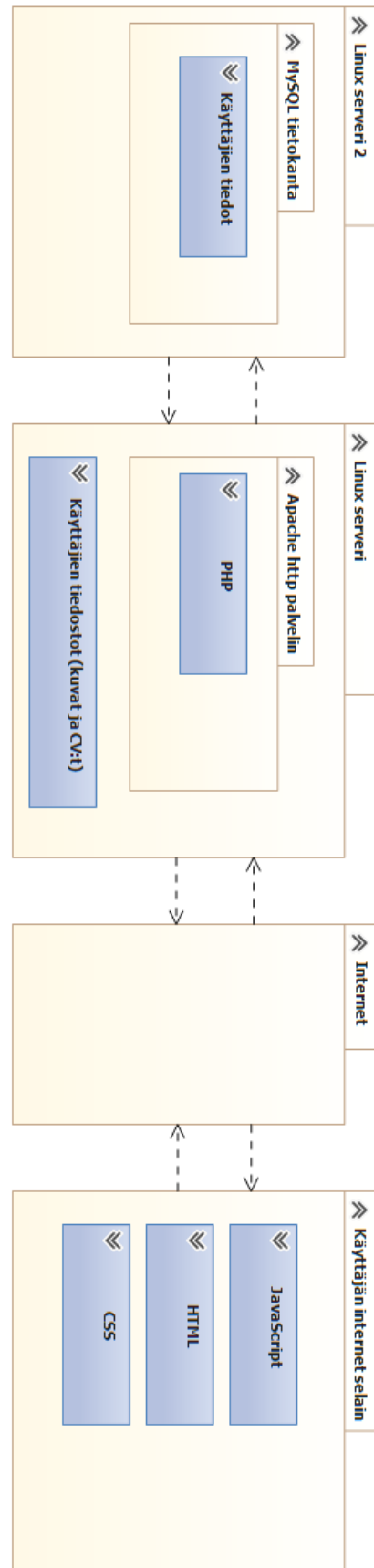
Tietojen tallennus epäonnistuu: kenttiin syötetään vääränmuotoista dataa, vaadittuja kenttiä ei ole täytetty.

Lopputulokset: Oppilas on muokannut tietonsa haluamikseen ja ne on tallennettu järjestelmään.

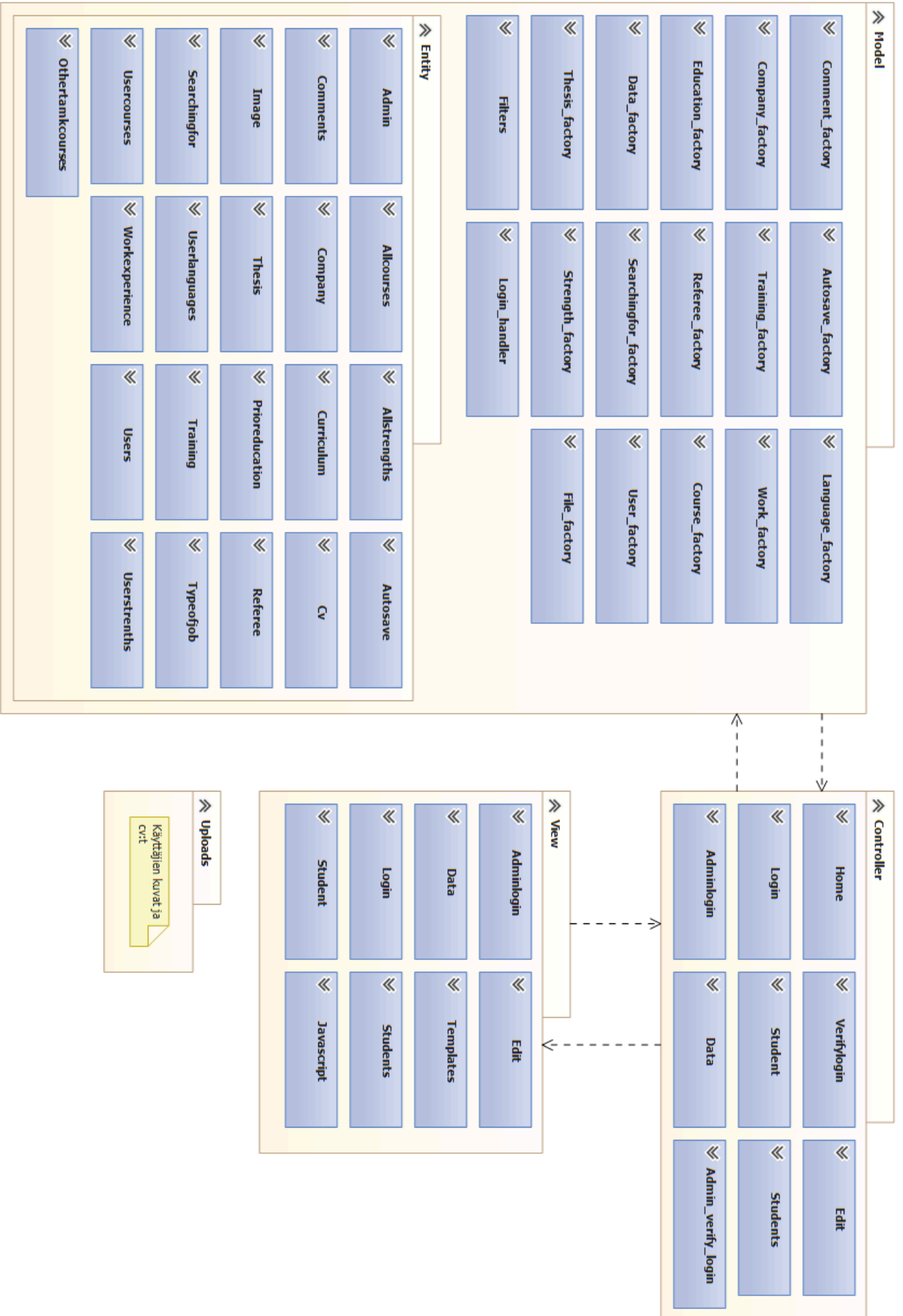
Liite 2. Rekrytointirekisterin kirjastot

Nimi	Versio	Lisenssi	Linkki	Koodi
CodeIgniter	2.1.4	Oma	http://www.codeigniter.com	https://github.com/bcit-ci/CodeIgniter/
Doctrine	2.3.2	MIT	http://www.doctrine-project.org/	https://github.com/doctrine/doctrine2
Gumby Framework	2.6	MIT	http://gumbyframework.com	https://github.com/GumbyFramework/Gumby
jQuery	1.10.1	MIT	http://jquery.com	https://github.com/jquery/jquery
jQuery UI	1.10.2	MIT	http://jqueryui.com	https://github.com/jquery/jquery-ui
JSON Form		MIT		https://github.com/joshfire/jsonform
jsPDF	1.0.178-git	MIT		https://github.com/MrRio/jsPDF
Modernizr	2.6.2	MIT	http://modernizr.com	https://github.com/Modernizr/Modernizr
Underscore.js	1.5.2	MIT	http://underscorejs.org	https://github.com/jashkenas/underscore
D3.js	3.4.1	BSD 3-Clause	http://d3js.org	https://github.com/mbostock/d3
Noty	2.2.2	MIT	http://ned.im/noty/	https://github.com/needim/noty
CIUnit		MIT		https://github.com/Celc/CIUnit
PHPUnit	4.2.6	BSD 3-Clause	http://www.phpunit.de/	https://github.com/sebastianbergmann/phpunit
MySQL		GPLv2	https://www.mysql.com	https://code.launchpad.net/~mysql/mysql-server/
Apache		Apache versio 2	http://httpd.apache.org	http://svn.apache.org/repos/asf/httpd/httpd/
PHP		PHP	http://php.net	https://github.com/php/php-src
phpMyAdmin		GPLv2	http://www.phpmyadmin.net/home_page/index.php	https://github.com/phpmyadmin/phpmyadmin
Git		GPLv2	http://www.git-scm.com	https://github.com/git/git

Liite 3. Rekrytointirekisterin osat



Liite 4. Rekrytointirekisterin rakenne



Liite 5. Tietokannan ER-kaavio

