KEMI-TORNION AMMATTIKORKEAKOULU

Tekijät Internetiin

Tekijäportaalin toteutus CodeIgniter-sovelluskehyksellä

Joona Kauppi & Matias Ylipelto

Tietojenkäsittelyn koulutusohjelma
Web-asiantuntijan suuntautumisvaihtoehto
Tradenomi

TORNIO 2010

TIIVISTELMÄ

Kauppi, Joona & Ylipelto, Matias 2010. Tekijät Internetiin - Tekijäportaalin toteutus CodeIgniter-sovelluskehyksellä. Opinnäytetyö. Kemi-Tornion ammattikorkeakoulu. Kaupan ja kulttuurin toimiala. Sivuja 56. Liitteet 1 - 3.

Opinnäytetyön tavoitteena oli luoda toimeksiantajan verkkokaupan tarpeita vastaamaan pilottiversio tekijäportaalista, joka yhdistää tekijät ja asiakkaat. Toteutuksessa keskeisenä näkökulmana oli palvelun käytettävyys sekä jatkokehityksen mahdollistava suunnittelu.

Työmme eteni konstruktiivisen tutkimusotteen mukaisesti soveltavana tutkimuksena, jossa pyrimme toteuttamaan tekijäportaalin olemassa olevan tiedon pohjalta. Osana tutkimusta oli myös tekijäportaalin toiminnallisuuksien määrittely.

Toteutimme tekijäportaalin käyttäen CodeIgniteria, joka on avoimen lähdekoodin sovelluskehys PHP-ohjelmointikielelle. Suoritimme ohjelmoinnin Eclipse-ohjelmointiympäristössä ja tallensimme lähdekoodin toimeksiantajan palvelimella sijaitsevaan versioarkistoon SVN-versionhallintajärjestelmällä. Suunnittelimme ja toteutimme sovelluksen käyttämän tietokannan viisivaiheisen suunnitteluputken mukaisesti. Käytimme tietokannassa MySQL-hallintajärjestelmää.

Ulkoasun suunnittelussa huomioitiin eri käyttäjät erottamalla käyttäjäryhmien käyttöliittymät toisistaan erilaisin värein. Tekijäportaalin ulkoasun toteutuksessa käytimme 960 Grid System -CSS-rakennekehystä, joka nopeutti ulkoasuun liittyvien ratkaisujen tekemistä.

Opinnäytetyön lopputuloksena syntyi pilottiversio tekijäportaalista, josta puuttui vielä lopullisen version kannalta oleellisia toiminnallisuuksia, mutta joka oli valmis jatkokehitystä varten.

Avainsanat: verkko-ohjelmointi, PHP, relaatiotietokannat, SQL, verkkopalvelut, käytettävyys

ABSTRACT

Kauppi, Joona & Ylipelto, Matias 2010. Moving contractors to the Internet - Implementation of contractor online service with CodeIgniter framework. Bachelor's Thesis. Kemi-Tornio University of Applied Sciences. Business and Culture. Tornio. Pages 56. Appendices 1 - 3.

The goal of our bachelor thesis was to create an online service which meets with our client's need to connect customers with contractors. The focus of the implementation was to create a user friendly online service which could be easily further developed.

Our work was executed with a constructive research method. We aimed to implement the online service based on knowledge that already exists. Defining the service's functionality was also a part of the research.

Contractor online service was built on CodeIgniter which is an open source PHP framework. Coding was done in Eclipse software development environment. Source code was stored and saved remotely to SVN version control system's repository. We designed and implemented the service's database using a five phase method. We used MySQL as the database management system.

While designing the layout, we paid special attention to different user groups by visually separating their user interfaces with different colors. Layout's structure was built on 960 Grid System framework which sped up the implementation progress.

As a result of our thesis the pilot version of contractor online service was born. Some essential features were still missing but the contractor online service was ready for further development.

Keywords: web programming, PHP, relational databases, SQL, Online services, usability

SISÄLTÖ

TIIVISTELMÄ

ABSTRACT

1. JOHDANTO 6
1.1 Opinnäytetyön tausta 6
1.2 Tavoite 7
1.3 Tutkimusmenetelmä 7
1.4 Työvälineet 7
1.5 Käsitteitä 8

2. SUUNNITTELU 11
2.1 Sovelluksen toiminnallisuuksien määrittely 11
2.2 Tietokanta 12
2.2.1 Käsiteanalyysi 13
2.2.2 Tarveanalyysi 16
2.2.3 Normalisointi 16
2.2.4 Taulujen muodostaminen 18
2.2.5 Indeksointi 19
2.3 Käyttöliittymä 20
2.3.1 Perustason käyttölogiikan suunnittelu 20
2.3.2 Käyttöliittymän hahmottaminen 21
2.3.3 Visuaalinen suunnittelu 22

3. CODEIGNITER-SOVELLUSKEHYS 24
3.1 MVC-suunnittelumalli 24
3.2 Tiedon virtaus 24
3.3 Asennus ja kansiorakenne 25
3.4 Luokkakirjasto 28
3.5 Avustajat 30
3.6 Esimerkkisovellus 31
3.6.1 Ohjain 31
3.6.2 Malli 32

3.6.3 Näkymät 33

4 TOTEUTUS 35
4.1 Kehitysympäristö 35
4.2 Ulkoasu 36
4.2.1 960 Grid System 37
4.2.2 Värimaailman muotoutuminen 39
4.2.3 Muuta ulkoasun toteutuksessa huomioitavaa 40
4.3 Testaus 40
4.4 Asiakkaan käyttöliittymä 41
4.5 Tekijän käyttöliittymä 42
4.5.1 Rekisteröityminen 42
4.5.2 Sisäänkirjautuminen 44
4.5.3 Salasanan uusiminen 45
4.6 Ylläpito-osio 45

5. TEKIJÖIDEN MIETTEITÄ 47
5.1 Lopputulos 47
5.2 Yleistä projektista 47
5.3 Projektin eteneminen 48
5.4 Kiitokset 48

LÄHTEET 49

LIITTEET 51

1. JOHDANTO

1.1 Opinnäytetyön tausta

Kaupankäynnin siirtyessä yhä enemmän verkkokauppoihin, asiakkaan ja myyjän välinen henkilökohtainen vuorovaikutus jää vähäiseksi tai jopa olemattomaksi. Tämä saattaa aiheuttaa ongelmia, joita ei ole aiemmin välttämättä edes tiedostettu. Jos esimerkiksi asiakkaan oma ammattitaito ei riitä juuri hankitun tuotteen asennukseen, on fyysisessä kaupassa asioitaessa mahdollista tiedustella henkilökunnalta asiaankuuluvaa asennusapua, mutta verkkokaupassa tätä mahdollisuutta ei ole.

Suorittaessaan keväällä 2010 työharjoittelua toimeksiantaja-yrityksessä toinen tämän opinnäytetyön tekijöistä yhdessä yrityksen henkilökunnan kanssa törmäsi edellä mainittuun ongelmaan, jossa asiakkailla oli tarvetta löytää hankkimalleen tuotteelle sopiva asentaja. Tapauksissa asiakkaalta joko puuttui taito asentaa tuote itse, tai sitten lakiteknisistä asioista johtuen tuotteen asennus itse ilman asiaankuuluvaa lupaa on kiellettyä. Ongelma korostuu verkkokaupassa asioitaessa, kun ostotilanteessa ei ole mahdollista kääntyä suoraan kenenkään puoleen.

Yrityksessä pohdittiin, että asiaa helpottaisi palvelu, jonne verkkokaupan asiakkaita voitaisiin ohjata, kun he tarvitsevat ammattilaisen apua. Tällaisia palveluita on jo olemassa toisten tahojen toteuttamana, mutta näiden palveluiden käyttämistä tähän tarkoitukseen ei koettu toimeksiantaja-yrityksessä tarkoituksenmukaiseksi, vaan asiakkaalle tulisi voida tarjota tekijöiden tiedot ”samalta luukulta”. Ratkaisuksi ongelmaan tarvittaisiin ammattilaisten ja tekijöiden yhteystietoja välittävä palvelu, josta asiakas voisi halutessaan etsiä yrityksiä, jotka voivat toteuttaa asiakkaan tarvitsemia asennuspalveluita. Samalla myös tekijät voisivat tuoda oman osaamisensa esiin palvelussa kaupan asiakkaiden nähtäville.

Yrityksestä kysyttiin kiinnostusta niin sanotun tekijäportaalin tekemiseen opinnäytetyönä. Tarjouduimme tekemään sovelluksen ja yritys otti tarjouksen vastaan.

1.2 Tavoite

Opinnäytetyön tavoitteena oli luoda tekijöitä ja asiakkaita yhdistävä palvelu, jonka kautta he voivat olla yhteydessä toisiinsa. Palvelu toteutettiin tekijäportaalina, eli verkkosivuna, jossa rekisteröityneet tekijät voivat mainostaa omia palveluitaan tiettyä vuosimaksua vastaan. Asiakkaat voivat hakea sivuilta heille sopivaa tekijää toimialojen ja maakuntien perusteella ja ottaa tarvittaessa helposti yhteyttä haluamiinsa tekijöihin palvelun kautta.

Toteutus tuli suorittaa nopeaan sovelluskehitykseen soveltuvalla CodeIgniter-nimisellä PHP-sovelluskehyksellä. Tekijäportaalista oli tarkoitus saada valmiiksi niin sanottu pilottiversio, jota voidaan jatkokehittää opinnäytetyön päättymisen jälkeen. Jatkokehityksen kannalta oli siis kiinnitettävä erityisen paljon huomiota sovelluksen dokumentointiin ja suunnitteluun.

1.3 Tutkimusmenetelmä

Opinnäytetyömme tehtiin soveltavana tutkimuksena käyttäen konstruktiivista tutkimusotetta. Konstruktiiviselle tutkimukselle luonteenomaista on uuden todellisuuden rakentaminen olemassa olevan tiedon pohjalta. Samalla on ratkaistava, millaista uutta todellisuutta halutaan rakentaa. (Järvinen & Järvinen, 102.)

Tarkoituksenamme oli luoda jo olemassa olevan palvelun tueksi sovellus, jota voitaisiin käyttää myös omana irrallisena kokonaisuutenaan. Päädyimme käyttämään konstruktiivista tutkimusotetta parhaiten tarpeitamme vastaavana.

1.4 Työvälineet

Opinnäytetyön työstössä pyrittiin mahdollisimman pitkälle käyttämään ilmaisia vapaan lähdekoodin ohjelmistoja. Ainoana poikkeuksena oli Microsoft Office 2010 -toimisto-ohjelmisto, jota käytettiin dokumenttien kirjoittamiseen.

Työasemina käytimme koulun puolesta saatuja Dell Latitude E6400 -mallisia kannettavia tietokoneita, joissa oli asennettuna Windows XP- sekä Linux Ubuntu -käyttöjärjestelmät.

Sovelluksen suunniteluvaiheessa käytimme käyttötapauskaavioiden piirtämiseen StarUML-ohjelmaa. Tietokannan suunnittelussa ja toteuttamisessa käytimme MySQL Workbench -ohjelmaa. Käyttöliittymän näytöt suunnittelimme Pencil-ohjelmalla. Sivuston grafiikan luomiseen käytimme Gimp-kuvankäsittelyohjelmaa.

Ohjelmointiin käytimme Eclipse-ohjelmointiympäristöä, johon latasimme tuen PHP-kielelle ja SVN-versionhallinnalle. Versioarkistoa säilytimme toimeksiantajan palvelimella. Koodin testaamiseen käytimme työasemille asennettua XAMPP-palvelinohjelmistoa sekä Internet Explorer-, Mozilla Firefox- ja Google Chrome -selaimia.

1.5 Käsitteitä

Aivoriihi on luovan ongelmanratkaisun standardimenetelmä, jolla tuotetaan ideoita ryhmässä. Aivoriihellä on ryhmän vetäjä, joka kirjaa kaikki ryhmän tekemät ehdotukset ylös. Ideoita ei arvioida heti, vaan arviointi jätetään myöhempään ajankohtaan. (Lavonen & Meisalo & al., 2010.)

Avoin lähdekoodi eli **open source** on tapa kehittää ja jaella tietokoneohjelmistoja. Asiakas saa vapaasti käyttää, kopioida, muunnella ja jaella avoimen lähdekoodin ohjelmaa – ilman lisenssimaksuja ja työlästä lisenssien ylläpitoa (Avoin lähdekoodi 2010). Tunnetuimpia avoimeen lähdekoodiin perustuvia ohjelmia ovat Firefox-Internet-selain, Open Office.org -toimisto-ohjelmisto sekä Linux-käyttöjärjestelmä.

CASE-ohjelmisto (lyhenne sanoista Computer Aided Software Engineering) on ohjelmisto, jolla voidaan laatia tietokannan käsitemalli kuvauksineen sekä generoida siitä halutun tietokantajärjestelmän taulujen ja indeksien perustamiskäskyt (Hovi, Huotari & Lähdenmäki 2005, 336).

PHP (lyhenne sanoista *PHP: Hypertext Preprocessor*) on laajalti käytetty palvelimella

suoritettava ohjelmointikieli. PHP-ohjelmointikieltä voidaan myös upottaa suoraan HTML-sivujen sisään.

Sovelluskehys eli framework on ohjelmoinnin nopeuttamiseen tarkoitettu apuväline, ”runko”, jonka päälle ohjelmaa tehdään. Se esimerkiksi sisältää valmiita ohjelman osia joita ei tarvitse kirjoittaa itse uudelleen ja näin ollen nopeuttaa uusien sovellusten kehittämistä. Opinnäytetyömme toteutuksessa käytetään CodeIgniter-nimistä PHP-kielelle tarkoitettua sovelluskehystä.

MySQL on suosittu avoimen lähdekoodin relaatiotietokannan hallintajärjestelmä, joka on saatavissa ilmaiseksi GPL-lisenssillä. MySQL on saatavissa myös kaupallisella lisenssillä, jos GPL ei ole sopiva. (MySQL 2010.)

HTML (lyhenne sanoista *Hypertext Markup Language*) on verkkosivujen rakentamisessa käytettävä kuvauskieli. Se vakiintui Internetin hypertekstin kieleksi 1990-luvun alussa. HTML-kielellä kuvataan kuinka verkkosivut näkyvät, ja kuinka sivut ovat linkitettyinä toisiinsa. (Metsämäki 2000, 34.)

CSS (lyhenne sanoista Cascading Style Sheets) on rakeenteellisten dokumenttien, kuten HTML-tiedostojen esitystavan määrittelevä tyylikieli. Sen avulla voidaan eriyttää HTML-dokumenttien sisältö esitystavasta. (CSS Introduction 2010.)

SQL (Lyhenne sanoista *Structured Query Language*) on lähes kaikkien relaatiotietokantatuotteiden tukema tietokannan määrittely- ja käsittelykieli. SQL-kielellä tehdään taulujen määrittely, tietojen haku, päivitys ja poisto sekä valtuusmääritykset ja tapahtumankäsittelyn ohjausta. (Hovi ym. 2005, 345.)

Subversion lyhennettynä **SVN** on monipuolinen versionhallintajärjestelmä, joka on alkuperin kehitetty korvaamaan CVS-versionhallintajärjestelmä (Concurrent Versions System). SVN mahdollistaa suurien tiedostojoukkojen muokkaamisen hajautetusti verkon yli siten, että kaikkien käyttäjien tiedostot pysyvät ajan tasalla. SVN mahdollistaa myös aikaisempien tiedostoversioiden palauttamisen versioarkistosta. (Apache Subversion Features 2010.)

XAMPP (lyhenne sanoista tai termeistä **X** (merkityksessä cross-platform), **A**pache

HTTP Server, **M**ySQL, **P**HP, ja **P**erl) on ilmainen avoimeen lähdekoodiin pohjautuva palvelinohjelmisto. Sitä kehittää voittoa tavoittelematon Apache Friends -projekti, joka perustettiin vuonna 2002. (Apache Friends 2010.)

ZIP on tiedon pakkaamiseen käytettävä menetelmä. ZIP-paketissa useampi tiedosto on pakattu yhteen pakettiin vähemmän tilaa vievään muotoon. Näin ollen useasta tiedostosta koostuvaa kokonaisuutta on helpompi levittää. ZIP-tiedostojen pääte on *.zip*. (Helsingin yliopisto 2004.)

2. SUUNNITTELU

2.1 Sovelluksen toiminnallisuuksien määrittely

Ensimmäisenä suunnitteluvaiheena ryhdyimme pohtimaan tekijäportaalin mahdollisia käyttötapauksia eri osapuolten kannalta. Käytimme toiminnallisten vaatimusten listaamisessa työmenetelmänä aivoriihtä. Aloitimme listaamalla toimeksiantajayrityksen henkilöstön kanssa palaverissa kaikkia mahdollisia toiminnallisuuksia, joita valmiissa sovelluksessa toivottiin olevan. Kirjasimme kaikki ideat ylös Excel-taulukkoon. Tässä vaiheessa emme kiinnittäneet vielä liikaa huomiota toiminnallisuuksien tarpeellisuuksiin, vaan annoimme mielikuvituksemme lentää. Kun saimme listattua toiminnallisuuksia tarpeeksi, aloimme jakaa taulukon rivejä eri ryhmiin sen perusteella, kuinka tärkeitä ne olivat tekijäportaalin perustoiminnan kannalta. Tärkeimmiksi luokitellut toiminnallisuudet päätyivät opinnäytetyönä tehtävän pilottiversiosovelluksen vaatimuksiksi ja vähemmän tärkeät säästimme jatkokehitystä varten.

Seuraavaksi aloimme kuvata sovelluksen käyttötapauksia. Käytimme käyttötapauskuvausten kirjoittamisessa hyödyksi eräänlaista kiinteää rakennetta, jossa jokaisesta käyttötapauksesta kirjoitimme ylös seuraavat tiedot: tavoite, edellytykset, onnistuneen suorituksen määritelmä, virhetilanteet, ensisijainen toimija, vaihtoehtoinen toimija ja vapaamuotoinen toiminnallinen kuvaus. Käyttötapauksen toiminnalliseen kuvaukseen saatoimme halutessa kirjoittaa useita tapahtumakulkuja. Numeroimme käyttötapaukset juoksevasti. ”Rekisteröidy palveluun”-niminen käyttötapauskuvaus on liitteenä (Liite 1).

Sovelluksen toiminnallisuuksien muodostaman kokonaisuuden hahmottamiseksi päätimme kuvata sovellusta käyttötapauskaaviolla, johon merkittiin jokainen määritelty käyttötapaus sekä niiden toimijat ja yhteyssuhteet. Käyttötapausten toimijoita eli rooleja olivat tekijä, asiakas, ylläpitäjä ja verkkokauppa. Kaavion käyttötapaukset vastasivat numeroinniltaan ja nimiltään käyttötapauskuvauksia. Käyttötapauskaavio löytyy liitteenä (Liite 2).

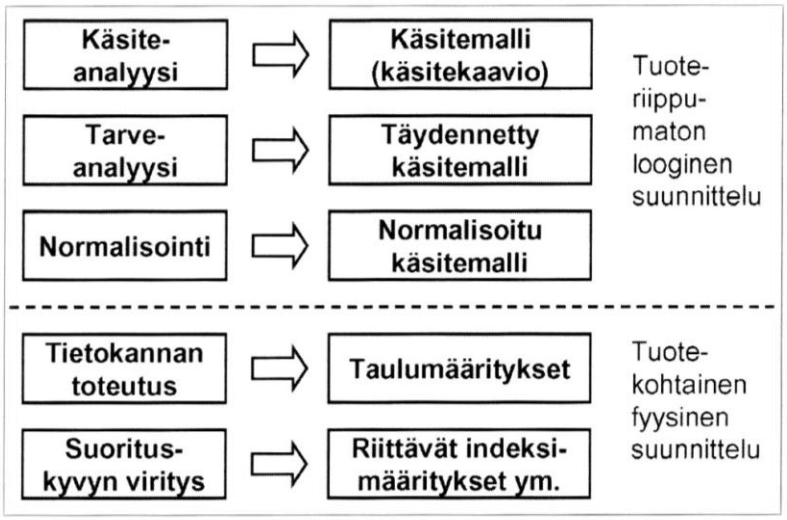
2.2 Tietokanta

Tekijäportaalisovellus tarvitsee toimiakseen räätälöidyn tietokannan, jota se voi itse käsitellä. Sovelluksen tietokannalle pelkkä tietovarastointi ei riitä, koska sovelluksen käyttötapaukset perustuvat operatiiviseen eli tuotannolliseen tietojenkäsittelyyn, jossa tietokannan tietoja on oltava mahdollista päivittää sovelluksen toimesta. Voidaankin sanoa, että tekijäportaalin tietokannan on oltava operatiivinen tietokanta. Sovelluksen on voitava käyttää tietokantaa monella tapaa, kuten tehdä kyselyjä, käsitellä tapahtumia ja ajaa eräajoja. Kaikkia näitä voi tapahtua jopa samanaikaisesti. (Hovi ym. 2005, 15.)

Rakenteeltaan tietokannat voidaan jakaa eri tyyppeihin, joita ovat mm. *verkkomalliset*, *hierarkkiset* ja *relaatiotietokannat*. Nykyisin käytetyistä tietokannoista valtaosa on relaatiotietokantoja. Ne ovat helpompia käyttää ja muuttaa kuin perinteisemmät verkkomalliset ja hierarkkiset tietokannat. (Hovi ym. 2005, 5.)

Päätimme yhdessä toimeksiantajan kanssa toteuttaa tietokannan MySQL-relaatiotietokantaohjelmalla, koska se oli valmiiksi asennettuna toimeksiantajan palvelinympäristössä. Perustelimme tietokantaohjelman valintapäätöstä myös sen maksuttomuudella, suorituskyvyllä, joustavuudella ja monipuolisuudella. MySQL on kaiken tämän lisäksi myös erittäin suosittu relaatiotietokantaohjelma ja sille löytyy oppaita ja kirjallisuutta runsaasti.

Suunnittelimme tietokantaa vaiheittain suunnitteluputkimallin mukaisesti. Suunnitteluputken vaiheet voidaan jakaa kahteen pääryhmään, joista ensimmäinen on *tuoteriippumaton looginen suunnittelu*, jonka vaiheita ovat *käsiteanalyysi*, *tarveanalyysi* ja *normalisointi*. Toinen vaiheryhmä on nimeltään *tuotekohtainen fyysinen käsittely*, jonka vaiheita ovat *tietokannan toteutus* ja *suorituskyvyn viritys*. Viimeistään jälkimmäisen vaiheryhmän kohdalla täytyy olla valittuna tietokannan hallintajärjestelmä eli TKHJ (Hovi ym. 2005, 24). Käytimme tässä projektissa MySQL-hallintajärjestelmää. Kuva 1 havainnollistaa tietokantojen suunnitteluputkea (Hovi ym. 2005, 24).



Kuva 1. Tietokantojen suunnitteluputki

Suunniteluputkessa eteneminen vaiheesta toiseen ei tapahdu ”vesiputousmaisesti” tai ”kuten putkessa”, niin kuin nimi voi antaa ymmärtää, vaan työtä tehdään yleensä spiraalinomaisesti tai iteratiivisesti. (Hovi ym. 2005, 24.) Tämä sopi meidän projektiimme, sillä opinnäytetyön aikana valmistunut tekijäportaalisovellus ei ollut lopullinen versio, vaan niin sanottu pilottiversio. Siinä ei ollut vielä kaikkia toiminnallisuuksia ja ominaisuuksia, joita sovelluksen varalle oli kaavailtu. Jatkokehityksen kannalta oli oleellista, että olemassa olevan tietokannan laajentaminen oli mahdollista.

2.2.1 Käsiteanalyysi

Käsiteanalyysin keskeisin päämäärä on saada aikaiseksi tietokannasta graafinen käsitemalli, josta käy ilmi eri käsitteet ja niiden väliset yhteydet. Käsitemallilla voidaan havainnollistaa tiedot, joita tietokantaan talletetaan. Käsitemalli toimii siis eräänlaisena reaalimaailman kohdealueen ja tietokannan välisenä ikkunana, jonka avulla ihmisen on helpompi ymmärtää tietokantaa.

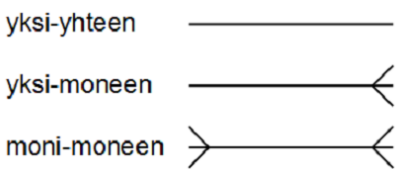
”Käsitemalli kuvataan yleensä graafisena käsitekaaviona eli ER-kaaviona jollakin sovitulla esitystavalla, nk. kuvaustekniikalla eli notaatiolla.” (Hovi ym. 2005, 33.) Käytimme ER-kaavion piirtämisen työkaluna MySQL Workbench open-source ohjelmaa. MySQL Workbench on CASE-ohjelmisto, jolla voimme myöhemmin luoda

MySQL-taulujen ja -indeksien perustamiskäskyt valmistuneen käsitemallin pohjalta. Käsitemallin hahmottamisessa lähdimme liikkeelle vaatimusmäärittelyn käyttötapausten kautta. Pyrimme etsimään luomastamme käyttötapauskaaviosta mahdolliset asiakokonaisuudet omiin käsitteisiinsä. Ensimmäisinä käsitteiksi hahmottui hyvin konkreettisia asioita kuten käyttäjä, toimiala ja tilaus. MySQL Workbench piirsi käsitteet kaavioon laatikoina. Relaatiotietokannoissa käsitteitä kutsutaan yleisesti tauluiksi, mutta emme käytä kyseistä nimitystä vielä tässä vaiheessa, koska etenemme suunniteluputkessa vielä tuoteriippumattoman loogisen suunnittelun alueella.

Jokainen käsite pitää sisällään tietoja eli attribuutteja. Tiedot ovat käsitteen ominaisuuksia. Esimerkkinä jo muodostetulle käyttäjäkäsitteelle tietoja voivat olla nimi, osoite, puhelinnumero ja sähköpostiosoite. Käsitteen tiedoista täytyy muodostua myös yksilöivä tieto eli toisin sanoen perusavain. Perusavaimena oleva tieto ei saa puuttua, koska silloin tietokannasta ei tule Coddin määrittelemien eheyssääntöjen mukainen eikä se tällöin vastaa reaalimaailmaa. (Hovi ym. 2005, 11.) Lisäsimme käsitteille useimmiten perusavaimeksi id-numeron, jonka nimi muodostui käsitteen nimestä, johon lisätään pääte ”_id”, esimerkiksi käyttäjä_id.

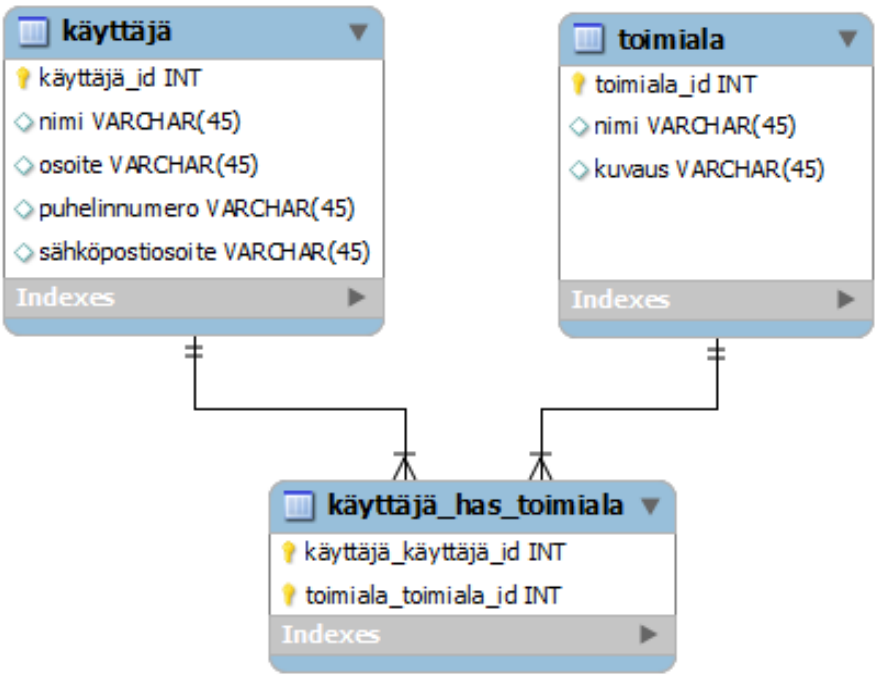
Tiedot tallentuvat käsitteisiin esiintyminä. Kun käsitteen tiedoille annetaan arvot, saadaan aikaiseksi esiintymä. Käyttäjä-käsitteen esiintymiä ovat käyttäjät, joilla kullakin on omat tietonsa. Esiintymät erottuvat toisistaan yksilöivien perusavainten ansiosta.

Käsitteet eivät aina ole itsenäisiä kokonaisuuksia, vaan useimmiten ne liittyvät toisiin käsitteisiin. Käsitemallissa voidaan kuvata käsitteiden välisiä yhteyksiä erilaisilla notaatioilla. MySQL Workbench piirtää yhteydet ns. ”harakanvarvasnotaatiolla”. Nimi notaatiolle tulee siitä, että käsitteiden välille piirtyvät yhteydet näyttävät useimmiten harakanvarpailta. (Hovi ym. 2005, 34.) Harakanvarvasnotaatiossa yhteydet voidaan jakaa kolmeen tyyppiin: yksi-yhteen, yksi-moneen ja moni-moneen. Kuvassa 2 esitetään yhteyksien piirrossymbolit.



Kuva 2. Yhteystyyppien piirrossymbolit

Näistä yksi-yhteen tarkoittaa, että tietyn käsitteen kuhunkin esiintymään liittyy vain yksi toisen käsitteen esiintymä. Yksi-moneen-yhteydessä käsitteen esiintymään voi liittyä toisesta käsitteestä useita esiintymiä. Tämä yhteystyyppi on käsitemalleissa yleisin. Moni-moneen-yhteys voidaan käsittää siten, että esimerkiksi käyttäjä-käsitteen esiintymillä eli käyttäjillä voi olla monta toimialakäsitteen ilmentymää eli toimialaa ja toisin päin. Moni-moneen-yhteyttä voidaan myös käyttää käsitemalleissa. Useimmiten tällaiset yhteydet on kuitenkin syytä purkaa auki muodostamalla jokin välikäsite eli *assosiatiivinen käsite*. (Hovi ym. 2005, 44.) MySQL Workbench muodostaa moni-moneen-yhteyksissä automaattisesti tällaisen välikäsitteen jonka perusavain koostuu liitettyjen käsitteiden perusavaimista. Kuvassa 3 on esimerkki moni-moneen-yhteydestä, joka on purettu kahdeksi yksi-moneen-yhteydeksi välikäsitteen avulla.



Kuva 3. Kahden käsitteen välille muodostettu välikäsite

2.2.2 Tarveanalyysi

Käsiteanalyysin avulla saimme aikaiseksi alustavan käsitemallin, jonka avulla pystyimme siirtymään tarveanalyysiin. Käsitemallimme tietoluettelo vaikutti vielä tässä vaiheessa puutteelliselta. Tarveanalyysillä pyrimme saamaan käsitemallin tarkemmalle tasolle.

Tarveanalyysillä on seuraavat tavoitteet:

- tarkistaa ja pöytätestata tehtyä käsitemallia tiedossa olevilla tietotarpeilla
- täydentää käsitemallia lisäämällä uusia tietoja sekä mahdollisesti myös uusia käsitteitä ja yhteyksiä.
  (Hovi ym. 2005, 80.)

Tarveanalyysissa käytimme apuna luomaamme käyttötapauskaaviota. Kävimme käyttötapaukset läpi kohta kohdalta ja tarkistimme, löytyivätkö kaikki sovelluksen tietokannan tarvitsemat käsitteet ja niiden tiedot käsitemallista. Lisäsimme tässä vaiheessa vielä useita käsitteitä ja tarkensimme käsitteiden tiedot, mikäli havaitsimme niissä puutteita. Loimme yhteydet käsitteiden välille sillä tavalla kuin sovelluksen kannalta näimme ne tarpeelliseksi.

Tarveanalyysin tuloksena saimme tarkennetun tietokannan käsitemallin, joka vastasi melko tarkasti niitä tarpeita, joita valmis sovellus tulisi tarvitsemaan. Emme tosin olleet vielä tässä vaiheessa suunnitelleet kaikkia käyttöliittymän ikkunoita, mutta koska tietokannan suunnitteluputkessa ei edetä vesiputousmallin mukaan, pystyisimme palaamaan vielä tarveanalyysin pariin myös myöhemmissä vaiheissa, mikäli tietokannasta löytyisi puutteita.

2.2.3 Normalisointi

Normalisointi on tuoteriippumattoman loogisen suunnittelun viimeinen vaihe. Normalisoinnin on kehittänyt E. F. Codd vuonna 1972. Codd on kehittänyt myös relaatiokantateorian. (Hovi ym. 2005, 86.) Käsiteanalyysin ja tarveanalyysin avulla päästään jo melko hyvin normalisoituun käsitemalliin. Normalisointi on silti hyvä

suorittaa tietojen toistamisen minimoimiseksi.

Normalisointiteoriassa käsitteille määritellään useita eri *normaalimuotoja*, joista yleisimmät ja alkuperäiset ovat ensimmäinen, toinen ja kolmas normaalimuoto. (Hovi ym. 2005, 86.) Jokaisella normaalimuodolla on oma perussääntönsä. Kävimme läpi nämä kolme normaalimuotoa ja tarkistimme, että kaikki tietokannan käsitteet ovat näiden sääntöjen mukaisia.

Ensimmäisen normaalimuodon perussäännön mukaan toistuvat ryhmät ja moniarvoiset sarakkeet poistetaan (Hovi ym. 2005, 88). Ensimmäisen normaalimuodon saavuttamiseksi voi joutua jakamaan käsitteitä useammiksi käsitteiksi, jos käsitteen tieto saa useita arvoja samassa esiintymässä, tai jos esiintymälle joutuu tallentamaan useita kertoja samankaltaista tietoa. Useimmiten jakautuneet käsitteet yhdistetään toisiinsa yksi-moneen-liitoksella. Kyseistä liitosta voi kutsua myös isä-lapsi-yhteydeksi, jolloin yksi käsite toimii isänä, jolla voi olla monta lasta, mutta lapsilla voi olla vain yksi isä.

Toisen normaalimuodon säännön mukaan kaikkien tietojen on oltava funktionaalisesti riippuvia koko perusavaimesta, jos käsitteessä on moniosainen perusavain. Toisessa normaalimuodossa tarkastellaan käsitteiden tietojen funktionaalista riippuvuutta perusavainten suhteen. Funktionaalista riippuvuutta on se, kun tiettyä käsitteen tietoa kohti on korkeintaan yksi joku muu tietty tieto. Esimerkiksi postitoimipaikka-käsitteen perusavain on postinumero ja käsitteen muut tiedot ovat funktionaalisesti riippuvaisia siitä. Samalla postinumerolla ei voi olla useita postitoimipaikkojen nimiä. Postitoimipaikan nimellä voi tosin olla monta postinumeroa, jolloin postinumero ei ole funktionaalisesti riippuvainen postitoimipaikan nimestä. Jos perusavain koostuu useasta osasta, täytyy jokaisen tiedon, joka ei ole osa perusavainta, olla funktionaalisesti riippuvainen koko perusavaimesta, eikä vain osasta siitä. Käsitteet, joiden perusavain koostuu vain yhdestä osasta, ovat toisessa normaalimuodossa, jos kaikki tiedot ovat perusavaimesta funktionaalisesti riippuvaisia. (Hovi ym. 2005, 90-92.)

Kolmannen normaalimuodon säännön mukaan jokaisen tiedon pitää olla fyysisesti riippuvainen vain perusavaimesta (Hovi ym. 2005, 93). Kolmannessa normaalimuodossa käsitteen tiedot eivät saa olla fyysisesti riippuvaisia tiedoista, jotka eivät ole osa perusavainta. Rikoimme tietoisesti tätä normalisoinnin sääntöä käyttäjä-käsitteen kohdalla. Käyttäjä-käsitteessä olevista osoitetiedoista postitoimipaikka on

funktionaalisesti riippuvainen postinumerosta, vaikka postinumero ei ole osa perusavainta. Käyttäjä-käsite ei siis ole kolmannessa normaalimuodossa. Tietokannan käytössä tästä on kuitenkin hyötyä, sillä käyttäjätietojen käsittelyssä vältetään yksi ylimääräinen liitos postitoimipaikka-käsitteeseen. Tällaista käsitteiden tietojen tahallista toistamista kutsutaan *denormalisaatioksi* (Hovi ym. 2005, 95).

2.2.4 Taulujen muodostaminen

Kun siirryimme tuotekohtaisen fyysisen suunnittelun puolelle, aloimme kutsua käsitemallien objekteja eri nimillä. Valitsemamme tietokannan hallintajärjestelmä MySQL käyttää tietokannan kuvaamisessa relaatiotietokantojen termejä. Relaatiotietokannoissa objekteja on kahdenlaisia: tauluja ja sarakkeita. Käsitteitä kutsutaan siis tauluiksi ja tietoja sarakkeiksi. Esiintymiä kutsutaan riveiksi, mutta yhteyksille ei ole suoraa vastinetta, vaan niistä tulee viiteavain-sarakkeita tauluihin. (Hovi ym. 2005, 104.)

Käytimme käsitemallin piirtämiseen MySQL Workbench CASE-välinettä. Kyseinen ohjelma osaa luoda käsitemallista valmiit taulujen ja indeksien luontikäskyt MySQL hallintajärjestelmälle, joten taulujen muodostaminen on hyvin helppoa.

Päätimme muuttaa taulujen ja sarakkeiden nimet tässä vaiheessa englannin kielelle, koska emme voineet olla varmoja, ymmärtääkö sovelluksen jatkokehittäjä suomen kieltä. Taulujen ja sarakkeiden uudelleen nimeäminen jo valmiiseen sovellukseen voi olla hyvin työlästä.

Valitsimme jokaiselle sarakkeelle oman tietotyypin sen perusteella, minkälaista tietoa se tallentaa. Tietotyyppi määrää, minkä muotoista tietoa kenttään pystyy tallentamaan ja miten tieto esitetään käyttäjälle myöhemmin (Heinisuo 2004, 83). Perusavaimien ja lukuja varastoivien sarakkeiden tietotyypiksi laitoimme usein INT(11), totuusarvoa käsitteleville sarakkeille TINYINT(1), postinumerolle MEDIUMINT(5), päivämäärille DATE, *muokattu*- ja *luotu*-sarakkeille DATETIME, pitkähköjä vapaamuotoisia tekstejä varten TEXT ja lyhyempiä vapaamuotoisia tekstejä varten VARCHAR() tyypin.

MySQL Workbench osaa tehdä monet taulukon luomiseen liittyvät asiat automaattisesti,

mutta tarkistimme kaiken varalta vielä muutamia seikkoja. Ensimmäiseksi asetimme kaikille perusavaimena oleville id-numeroille AUTO_INCREMENT- ja NOT NULL -määreet, jotta id-numero muodostuisi automaattisesti aina kun uusi rivi lisätään tauluun. Perusavaimille on oltava joka tapauksessa NOT NULL -määre ja se on aina hyvä tarkistaa muutenkin.

Seuraavaksi tarkastimme yksi-moneen- eli isä-lapsi-yhteyksien viiteavaimet. Viiteavaimen on aina oltava näissä yhteyksissä siinä taulussa, joka on yhteyden lapsena. Viiteavaimeksi tulee isä-taulun perusavain. (Hovi ym. 2005, 105.) Asetimme myös erikseen lapsitauluille viite-eheyssäännöiksi vyörytyssääntöjä (ON DELETE CASCADE), jolloin tietokantajärjestelmä poistaa automaattisesti isätaulun riviin kuuluvat lapsitaulun rivit, kun isätaulun rivi päätetään poistaa. Viite-eheyssääntöjen takia tauluja ei voi perustaa satunnaisessa järjestyksessä, sillä isä-taulujen on oltava perustettuna ennen lapsi-taulujen perustamista. (Hovi ym. 2005, 111.) Meidän ei kuitenkaan tarvinnut huolehtia tästä erikseen, koska MySQL Workbench osaa järjestää taulujen luontikäskyt oikeaan järjestykseen.

2.2.5 Indeksointi

Kun tietokantaan lisätään tietoja, MySQL tallentaa ne oletuksena siihen järjestykseen, jossa ne syötetäänkin. Kun etsitään tietoa tietokannasta, taulun rivejä joudutaan käymään läpi järjestelmällisesti. Tällainen menettely voi olla hyvin hidasta etenkin, jos tietokannassa tauluilla on hyvin paljon rivejä. (Laaksonen 2010.)

”Indeksi on taulun oheen tallennettava hakemistorakenne, josta voi tarkistaa nopeasti, missä päin taulua on rivejä, joiden kentät vastaavat hakuehtoja.” (Laaksonen 2010.) Indeksoinnilla voidaan nopeuttaa tietokantakyselyjä, koska tällöin ei tarvitse käydä koko taulua läpi. Taululle voidaan perustaa indeksejä heti kun se on luotu, tai vaihtoehtoisesti myöhemmin. Sillä ei ole väliä onko taulussa rivejä vai ei. (Hovi ym. 2005, 158.)

MySQL Workbench muodostaa automaattisesti indeksit taulujen perusavaimille ja viiteavaimille, kun ohjelmaa käsketään luomaan käsitemallista taulujen perustamiskäskyt. Perusavaimen indeksi nopeuttaa perusavaimen arvoon perustuvia

kyselyitä ja viiteavainten indeksit nopeuttavat liitosten tekemistä.

Indeksejä on järkevää luoda taulujen kentille, jotka esiintyvät säännöllisesti sovelluksen kyselyissä (Laaksonen 2010). Tekijäportaalisovellukseen oli määrä toteuttaa hakutoiminto, jolla voi etsiä tekijöitä erilaisilla hakuparametreilla, kuten vapaasanahaulla. Tällöin tietokantakyselyn tulokset rajataan niihin tekijäprofiileja sisältävän taulun riveihin, joissa yrityksen nimi -sarakkeen arvo sisältää hakuparametrin sisällön. Sanahakujen nopeuttamiseksi lisäsimme tekijäprofiileja sisältävään tauluun indeksin yrityksen nimi -kentälle.

2.3 Käyttöliittymä

Käyttöliittymä on se auton, tietokonesovelluksen, television tai minkä tahansa muun tuotteen osa, jonka avulla käyttäjä käyttää tuotetta. Menestymisen kannalta käyttöliittymän hyvä toimivuus on oleellinen asia varsinkin sellaisille tuotteille, joille löytyy useita muita vastaavia vaihtoehtoja. Huonosti suunniteltu käyttöliittymä saattaa vaikeuttaa tarpeettomasti tuotteen käyttöä, turhauttaa käyttäjän ja pahimmillaan karkottaa tämän pysyvästi. Hyvä käytettävyys on äärimmäisen tärkeää verkkotuotteessa, jonka käyttäminen on täysin vapaaehtoista ja vaihtoehtoja on paljon. (Sinkkonen & Nuutila & Törmä 2009, 17.)

2.3.1 Perustason käyttölogiikan suunnittelu

Kuten ihmisten välisissä suhteissa, myös verkossa toimivissa palveluissa on tärkeää, millaisen ensivaikutelman antaa itsestään. Tekijäpalvelun käyttöliittymän käyttölogiikan suunnittelussa lähtökohtana oli tehdä siitä käytettävyydeltään mahdollisimman helppo ja yksinkertainen, koska kohderyhmä koostuu enimmäkseen rakennusalan ammattilaisista, joilla ei välttämättä ole pitkäaikaista ja syvällistä kokemusta verkossa sijaitsevien palveluiden käyttämisestä. Jos palvelu vaikuttaa heti ensisilmäyksellä monimutkaiselta ja liian vaikealta käytettäväksi, asiakas siirtyy muihin palveluihin tai hylkää koko ajatuksen.

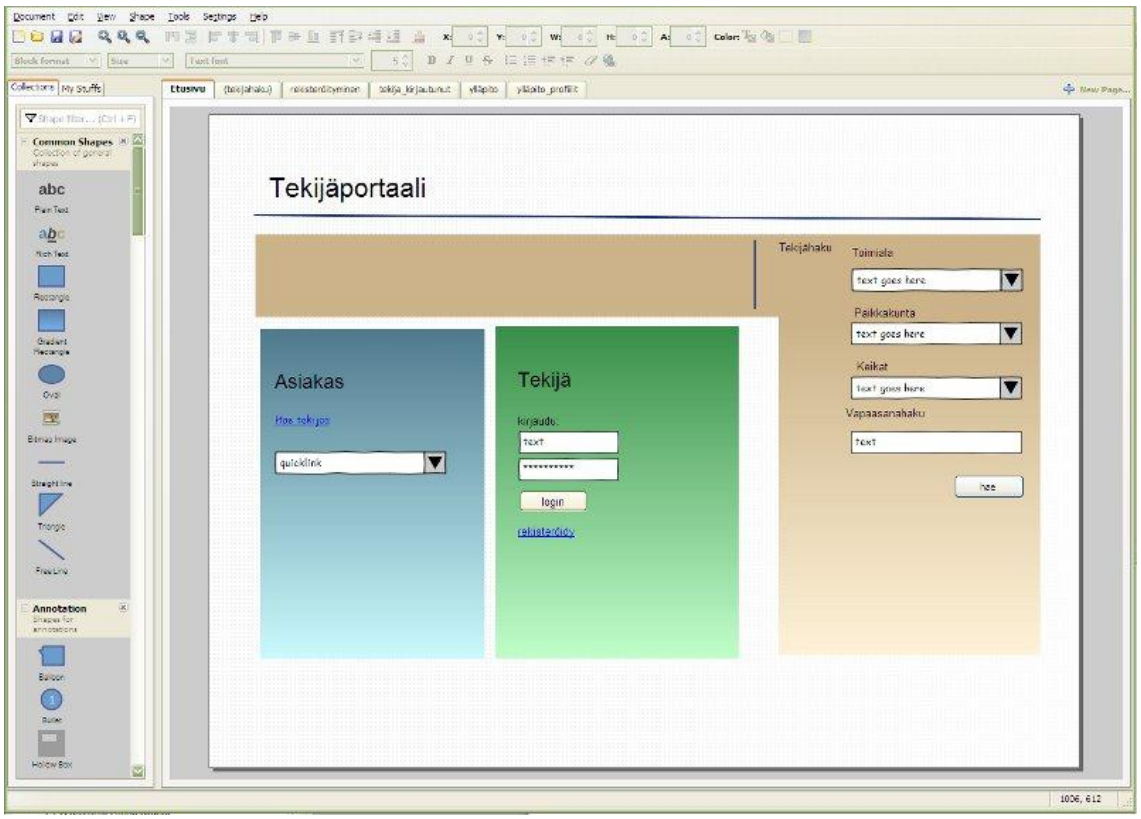
Palvelun tulee olla niin sanotusti yhteensopiva ihmisen kanssa, jolloin verkkopalvelun

täytyy sopia ihmisen fyysisiin piirteisiin, kognitiivisiin ominaisuuksiin ja muihin erilaisiin tarpeisiin ja tehtäviin. Huomion arvoisia ovat myös erilaiset virhetilanteet. Jos jokin asia on mahdollista tehdä väärällä tai muuten suunnittelusta poikkeavalla tavalla, on varmaa, että hyvin monet näin tekevät. (Sinkkonen ym. 2009, 19.)

Palvelun ulkoasua ja käyttölogiikkaa lähdimme suunnittelemaan perustuen aiemmin laadittuihin käyttötapauskuvauksiin ja -kaavioihin, joista tässä opinnäytetyössä toteutettavat osat määriteltiin suunnittelun varhaisimmassa vaiheessa. Hyvin suunniteltu on puoliksi tehty, kuuluu myös vanha viisaus, ja tässäkin tapauksessa käyttöliittymän toteuttamista helpotti huomattavasti selkeät ja perusteellisesti loppuun asti mietityt kuvaukset palvelun toiminnallisista ominaisuuksista, joiden tekstimuotoinen tieto tarvitsi vain konkretisoida web-sivuston muotoon.

2.3.2 Käyttöliittymän hahmottaminen

Ennen varsinaisen toimivan web-sivuston ohjelmointia palvelun eri osista luotiin mockup-malleja, eli hahmotelmia valmiiden käyttötapausten perusteella, sekä kaavailtiin tulevan käyttöliittymän rakennetta. Tähän käytettiin Pencil-työkaluohjelmaa, jolla käyttäjälähtöisten prototyyppien tekeminen eri näkymistä oli vaivatonta. Ohjelma sisältää kirjaston erilaisia valmiita web-sivuille tyypillisimpiä elementtejä, jotka vedä & pudota -periaatteella voitiin asetella tyhjälle pohjalle tarpeiden mukaisesti. Oheisessa kuvassa (kuva 4.) on nähtävissä ruudunkaappaus Pencil-ohjelman käyttöliittymästä.



Kuva 4. Pencil-ohjelman käyttöliittymä

Suunnittelussa piti ottaa huomioon eri käyttäjäryhmät, joita olivat siis asiakkaat, tekijät ja ylläpitäjät. Näiden lisäksi suunnittelupöydällä oli erikseen kaikille yhteinen etusivu, jonka kautta edellä mainitut ryhmät pääsevät käyttämään portaalin eri osia. Ulkoasun tuli olla yhtenäinen lukuun ottamatta ylläpidon näkymää, jolle riitti yksinkertaistetumpi käyttöliittymä.

Kaiken suunnittelun pohjana oli pyrkimys toimivaan informaatioarkkitehtuuriin. Hyvä ja tehokas informaatioarkkitehtuuri on sellainen, jossa liikkuessaan ihmiset kokevat koko ajan olevansa lähestymässä etsimäänsä tietoa, ja toisaalta myös koko ajan ovat selvillä sijainnistaan, vaikka olisivat useamman klikkauksen päässä aloituspaikasta. (Sinkkonen ym. 2009, 184)

2.3.3 Visuaalinen suunnittelu

Visuaalinen ulkonäkö viestittää kahta asiaa käyttäjälle. Ensimmäinen näistä on verkkosivujen sisällön esittäminen. Ulkonäön pitää auttaa huomaamaan, jäsentämään ja ymmärtämään asiat, jotka pitää huomata ja ymmärtää. Tätä kutsutaan *visuaaliseksi*

käytettävyydeksi. Toinen tehtävä on välittää käyttäjälle sivujen kokonaisilmeeseen liittyvä viesti: palvelun brändi, kokonaisilme, tunnelma ja persoonallisuus. Visuaalisen suunnittelun tuloksena sivustolle muodostuu yleisilme joka kuvastaa brändiä myös syvemmällä tasolla. Silmäilemällä on mahdollista saada vaikutelma siitä, onko palvelu esimerkiksi konservatiivinen, hauska tai taiteellinen. Ulkoasun tulee vastata palvelua jotta käyttäjä jaksaa katsoa palvelua siinä merkityksessä johon se on tarkoitettu. Visuaalinen suunnittelu itsessään on niin mittava projekti, että yleensä se toteutetaan ostopalveluna, jolloin suunnittelutyö ulkoistetaan asiaan perehtyneille graafikoille ja käyttökokemusasiantuntujoille. (Sinkkonen ym. 2009, 242.)

Opinnäytetyömme pääpainon ollessa muualla kuin visuaalisessa suunnittelussa emme käyttäneet kovinkaan paljon aikaa visuaalisuuden suunnitteluun. Jokaiselle käyttäjäryhmälle tuli kuitenkin suunnitella omanlaisensa käyttöliittymä heidän tarpeittensa mukaisesti. Kunkin näkymät päätettiin värikoodata. Esimerkiksi kaikilla asiakkaalle kuuluvilla sivuston osilla on näkyvissä sininen teema. Värikoodaus helpottaa sivustolla navigointia ja pitää yleisilmeen selkeänä. Huomiota tuli kiinnittää myös itse käyttäjäryhmien tietotaitoon. Liika informaatiotulva yhdessä paikassa varmasti säikyttää Internetiä ainoastaan satunnaisesti selailevat, kun taas ylläpitoon kuuluvat henkilöt löytävät suurestakin tietomäärästä helposti tarvitsemansa.

3. CODEIGNITER-SOVELLUSKEHYS

Heti projektin alussa päätimme, että tekijäportaalin toteutus tehdään CodeIgniter-sovelluskehyksellä. CodeIgniter on avoimen lähdekoodin sovelluskehys PHP-ohjelmointikielelle ja sen on kehittänyt *EllisLab*-niminen yritys. Se soveltuu nopeaan ohjelmistokehitykseen ja sisältää runsaan kirjaston valmiita ohjelmistokomponentteja, joita ohjelmistokehittäjä voi hyödyntää rakentaessaan dynaamisia, turvallisia ja tietokantaa käyttäviä Internetsivuja. (EllisLab - products 2010.)

3.1 MVC-suunnittelumalli

CodeIgniter noudattaa malli-näkymä-ohjain-arkkitehtuuria (MVC, model-view-controller), jonka perusajatus on erottaa käyttöliittymä varsinaisesta sovelluslogiikasta ja -datasta (Koskimies & Mikkonen 2005, 142).

MVC-arkkitehtuurin mukainen sovellus koostuu kolmentyyppisistä osista: mallit, näkymät ja ohjaimet. Näillä osilla on toisistaan poikkeavat tehtävät. Malli vastaa varsinaisen tieto-objektin tai tietokannan operaatioista. Käytännössä se pitää sisällään esimerkiksi valmiit funktiot tietokanta-operaatioiden suorittamiseen. Ohjain kutsuu tarvittaessa mallin funktioita tietojen hakemiseksi tai päivittämiseksi sekä vastaa ohjaavasta logiikasta, kuten minne sovelluksen käyttäjä ohjataan virhetilanteissa, tai kun jokin tieto on päivitetty. Ohjain kutsuu lopulta näkymää ja välittää sille tarvittavat muuttujat. Näkymäosa koostuu tekijäportaalissa HTML-kuvauskielestä, johon voidaan halutessa lisätä dynaamisesti sisältöä PHP-koodin avulla.

3.2 Tiedon virtaus

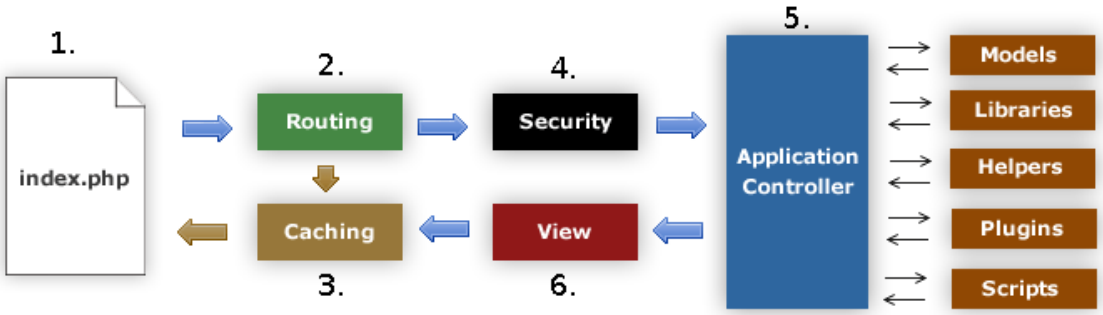
CodeIgniterin sovellukset suunnitellaan MVC-arkkitehtuurin mukaisesti, mutta tieto ei virtaa ainoastaan ohjelmoidun sovelluksen osissa. Sovelluksen ollessa käynnissä tieto kulkeutuu myös tiettyjen sovelluskehyksen osien läpi. Kuva 5 havainnollistaa miten tieto tyypillisesti virtaa sovelluskehyksen sisällä (CodeIgniter User Guide 2010a).

1. *index.php*-niminen tiedosto toimii etuohjaimena sovelluskehykselle ja

määrittää sen käyttämät resurssit.

2. Reititystarkastaja tutkii HTTP-pyyntöä ja päättää mitä sille tulee tehdä.
3. Jos pyydetty tieto on välimuistissa, se lähetetään suoraan selaimelle, eikä jatketa normaalin tietovirran mukaisesti.
4. Tietoturvallisuuden saavuttamiseksi HTTP-pyyntö sekä muut käyttäjän lähettämät tiedot suodatetaan ennen sovelluksen ohjaimen lataamista.
5. Sovelluksen ohjain lataa mallit, luokkakirjaston, avustaja-funktiot, ohjelmalisäkkeet ja muut tarvittavat resurssit kyseessä olevan pyynnön käsittelemiseen.
6. Lopullinen näkymä luodaan ja lähetetään selaimeen. Jos välimuistin hyödyntäminen on otettu käyttöön, näkymä tallennetaan välimuistiin, jolloin toistuvat pyynnöt voidaan ladata suoraan sieltä.

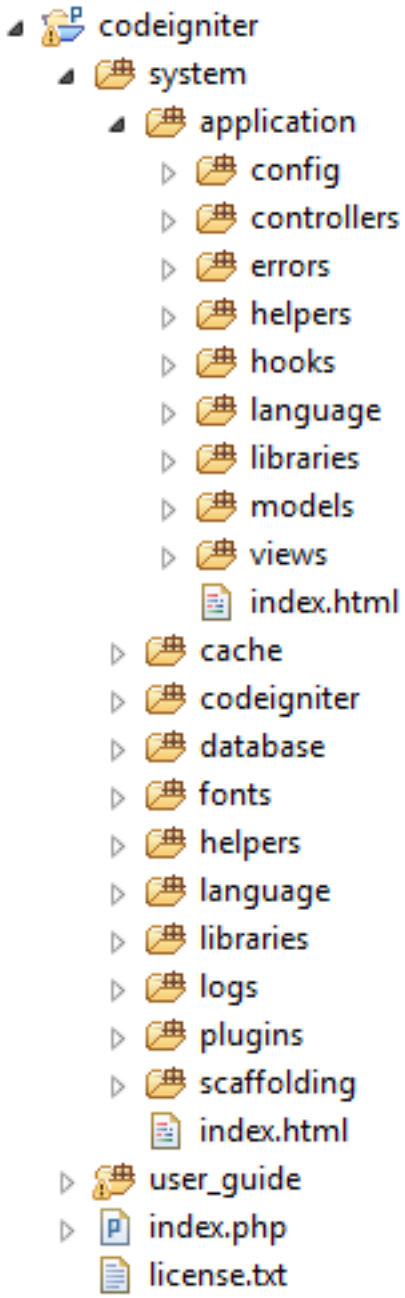
(CodeIgniter User Guide 2010a).



Kuva 5. Tiedon virtaus CodeIgniterissa

3.3 Asennus ja kansiorakenne

Päätimme asentaa palvelimelle tekijäportaalia varten CodeIgniterin version 1.7.2, sillä se oli päätöksentekohetkellä uusin vakaa versio. Lähdimme liikkeelle asentamalla sovelluskehyksen ensin paikallisesti työasemille. Aloitimme lataamalla ZIP-pakatun CodeIgniterin sovelluskehyksen omalta viralliselta kotisivulta. Purimme pakatun tiedoston työaseman XAMPP-palvelinohjelmiston *htdocs*-kansioon, joka on oletuksena XAMPP-palvelimen ajohakemisto. Kuva 6 havainnollistaa CodeIgniterin kansiorakennetta sen jälkeen, kun ZIP-paketti oli purettu.



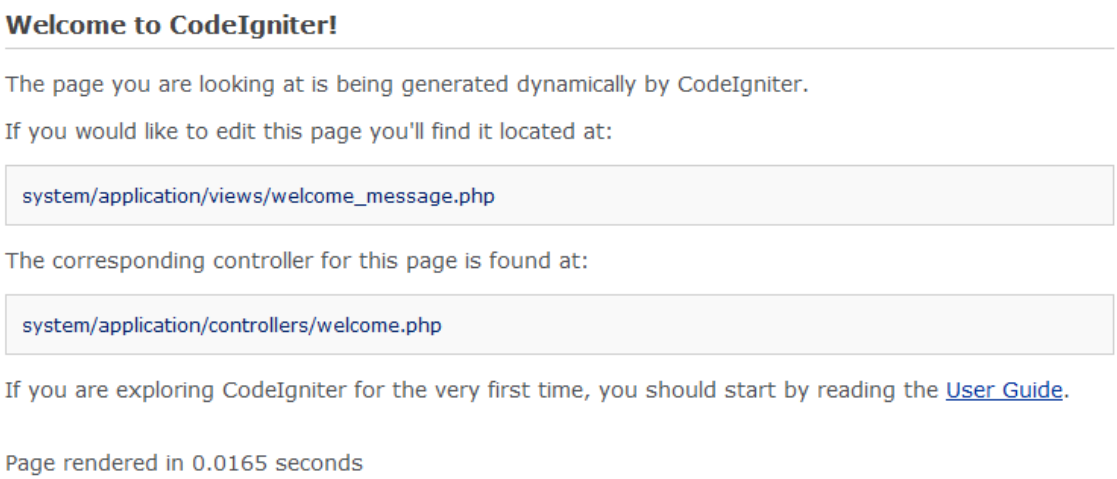
Kuva 6. CodeIgniterin kansiorakenne

Puretun kansion juuressa ovat *system*- ja *user_guide*-kansiot alikansioineen sekä *index.php*- ja *license.txt*-tiedostot. *License.txt* on tekstitiedosto, joka sisältää CodeIgniterin lisenssitiedot. *Index.php* on CodeIgniterin etuohjain, jossa on määriteltynä sovelluskehyksen käyttämät resurssit. CodeIgniterin mukana tuleva käyttöopas eli User Guide sijaitsee *user_guide*-kansiossa. Itse CodeIgniter-sovelluskehys sijaitsee *system*-kansiossa. Toimiakseen sovelluskehys ei tarvitse käyttöopasta eikä lisenssitietoja sisältävää tekstitiedostoa, joten siirsimme nämä pois

XAMPP-palvelinympäristöstä.

System-kansion alla sijaitseva *application*-kansio on se, jonne sovellus ohjelmoidaan. *Application*- eli sovelluskansio voidaan nimetä uudelleen ja siirtää minne tahansa palvelimella. Tällöin täytyy sovelluskehyksen etuohjaimeen määritellä uudestaan sovelluksen polku. Työasemilla päätimme säilyttää oletuksena olevan sovelluskansion.

Seuraavaksi muokkasimme sovelluksen *config*-kansiossa sijaitsevia *config.php* ja *database.php* -tiedostoja, jotka sisältävät sovelluksen ympäristökohtaisia muuttujia. *Config.php*-tiedostoon vaihdoimme *$config['base_url']*-muuttujaan arvon ”http://127.0.0.1/”, joka viittaa XAMPP-palvelimen *htdocs*-kansioon. *Database.php*-tiedosto sisältää tietokantayhteyden luomiseen tarvittavat muuttujat, kuten isäntäpalvelimen osoitteen, tietokannan tyypin, käyttäjänimen, salasanan ja tietokannan nimen. Muokkasimme muuttujia vastaamaan työasemille luotujen MySQL-tietokantojen asetuksia. Tämän jälkeen testasimme CodeIgniteria menemällä selaimella osoitteeseen ”http://127.0.0.1/codeigniter/”, jolloin selaimeen aukesi sovelluskehyksen oletusnäkymä (Kuva 7.).



Kuva 7. CodeIgniterin oletusnäkymä

Saimme asennettua CodeIgniterin työasemillemme onnistuneesti. Päätimme asentaa sovelluskehyksen myös toimeksiantajan vuokraaman webhotellin palvelimelle siten, että sovellusta pääsisi käyttämään vain sovelluskehitystä varten luodun verkkotunnuksen kautta. Muutimme CodeIgniterin kansiorakennetta siten että etuohjain ja sovellus-kansio siirrettiin verkkotunnuksen käyttämään ajokansioon. *System*-kansion

sijoitimme verkkotunnuksen ajohakemiston ulkopuolelle erilliseen */home/[käyttäjänimi]/frameworks/ci/CodeIgniter_1.7.2/*-hakemistoon. Loimme */home/[käyttäjänimi]/frameworks/ci/*-hakemistoon symbolisen linkin nimeltä *current*, joka osoittaa *CodeIgniter_1.7.2*-kansioon, jossa asentamamme sovelluskehyksen *system*-kansio sijaitsee. Tämä mahdollistaa sen että jatkossa järjestelmä on helppo päivittää, kun uusi versio CodeIgniterista on ilmestynyt. Tällöin tarvitsee muokata vain symbolista linkkiä osoittamaan uudempaan *system*-kansioon.

Toimiakseen webhotellin palvelimella CodeIgniterin ympäristökohtaiset tiedostot täytyi vielä muokata. Sovelluksen *config*-kansion *config.php* ja *database.php* -tiedostot muokkasimme vastaamaan palvelimen ja tietokannan asetuksia. Tämän lisäksi etuohjaimeen täytyi määrittää *system*-kansion poluksi */home/[käyttäjänimi]/frameworks/ci/current/system*.

3.4 Luokkakirjasto

CodeIgniter sisältää varsin runsaan kirjaston valmiita ohjelmistokomponentteja. Nämä ohjelmistokomponentit ovat *luokkia*. Luokilla on omia toimintoja eli *metodeja* sekä muuttujia, joita kutsutaan luokkien yhteydessä *attribuuteiksi*. Luokkien ilmentymiä kutsutaan *olioiksi*. Yhdestä luokasta voi luoda rajattoman määrän olioita. Olioiden attribuuttien arvot voivat vaihdella, mutta attribuuttien nimet ja metodit ovat saman luokan olioilla yhtenäiset. Luokka voidaan käsittää kaavana, jonka avulla olioita luodaan. (PHP-opas 2010.)

CodeIgniterin valmiit luokat ovat tehty helpottamaan yleisten ja toistuvien toimintojen luomista. Näitä luokkia voi myös halutessaan laajentaa tai korvata kokonaan sovelluksen omilla luokilla. Myös aivan uusien luokkien lisääminen sovelluksiin on mahdollista.

CodeIgniterin nimeämiskäytännön mukaan luokkien ja luokka-tiedostojen nimet tulee olla identtiset lukuun ottamatta tiedostonimen *.php*-päätettä. Poikkeuksena ovat sovelluskehyksen alkuperäiset luokat, joiden alustuksessa käytetään usein etuliitettä ”CI_”, vaikka tiedostonimissä kyseistä etuliitettä ei ole. Luokkien alustukset ja tiedostonimet tulee alkaa myös isolla alkukirjaimella. CodeIgniterissa käytettävien

kirjasto-luokkien perusrakenne on seuraavanlainen:

```
1  <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
2  class Jokuluokka {
3      function joku_funktio()
4      {
5      }
6  }
7  ?>
```
(CodeIgniter User Guide 2010b.)

Jos sovelluksessa halutaan laajentaa jotain sovelluskehyksen perusluokista, on luotava sovelluksen *library*-hakemistoon uusi *.php*-päätteinen tiedosto, jonka nimi muodostetaan aliluokkaa merkitsevästä etuliitteestä ja laajennettavan luokan tiedostonimestä. Aliluokkaa merkitsevä etuliite määritellään sovelluksen *config*-kansion *config.php*-tiedostossa ja on oletuksena ”MY_”. Luotuun tiedostoon alustetaan uusi luokka, jonka nimi muodostuu aliluokan etuliitteestä ja laajennettavan luokan nimestä. Luokan alustuksen yhteydessä tulee myös merkitä sen luokan periminen, jota uusi luokka laajentaa. Seuraavassa esimerkki CodeIgniterin *Email*-luokkaa laajentavan luokan alustamisesta:

```
1  class MY_Email extends CI_Email {
2
3  }
```
(CodeIgniter User Guide 2010b.)

Kun sovelluksessa halutaan korvata CodeIgniterin alkuperäisen kirjaston luokka, täytyy sekä tiedosto, että luokan alustaminen nimetä täsmälleen samalla tavalla kuin alkuperäisessä luokassa. Uusi luokka sijoitetaan sovelluksen *library*-hakemistoon. Seuraavassa esimerkki *Email*-luokan korvaavan luokan alustamisesta:

```
1  class CI_Email {
2
3  }
```
(CodeIgniter User Guide 2010b.)

Oma luokan luominen sovellukseen onnistuu hyvin pitkälti samaan tapaan kuin korvaavan luokan luonti, mutta uudelle luokalle ei saa antaa samaa nimeä, joka on jo käytössä olemassa olevilla kirjaston luokilla.

Lisäsimme tekijäportaaliin uusiksi luokiksi *Auth*-, *Template*- ja *Messages*-luokat. Auth-luokka sisältää metodeja sisään- ja uloskirjautumiseen sekä käyttäjien todentamiseen. *Template*-luokka mahdollistaa HTML-mallinteiden eli näkymäpohjien käyttämisen, kun ohjain lataa näkymää. *Messages* on Vijay Mahrran ja Sheikh Ahmedin luoma luokka, joka mahdollistaa palautetekstien tallentamisen istuntotietoihin sekä niiden lataamisen sieltä. *Messages*-luokka on *LGPL* eli *Lesser General Public License* -lisenssin alainen, mikä tarkoittaa, että luokkaa voi vapaasti käyttää kaupallisissa sovelluksissa. *Messages*-luokka on vapaasti ladattavissa CodeIgniterin wiki-sivustolta. (CodeIgniter Wiki 2010.)

Laajentaviksi luokiksi lisäsimme *MY_Controller* ja *MY_Model* -luokat. *MY_Model*-luokkaan lisäsimme käytännöllisiä ja malleissa useasti toistuvia tietokannan käsittelyyn liittyviä metodeja. *MY_Controller*-luokkaan lisäsimme muodostinmetodin, joka lataa ohjaimen attribuutteihin merkityt mallit automaattisesti. Lisäsimme Luokkaan myös *Template*-luokan metodeja hyödyntävän metodin, joka käsittelee HTML-mallinteinta automaattisesti.

Määritimme CodeIgniterin lataamaan jokaisen sivunlatauksen yhteydessä automaattisesti *Database*-, *Session*-, *Template*- ja *Messages*-luokat muokkaamalla sovelluksen *config*-hakemiston *autoload.php*-tiedostoa. Pystyimme käyttämään automaattisesti ladattujen luokkien metodeja sovelluksen koodissa ilman erillistä latauskomentoa.

3.5 Avustajat

Avustajien eli *helper*-funktioiden tarkoituksena on helpottaa ohjelmointia. Jokainen avustajatiedosto koostuu ryhmästä funktioita, jotka edustavat jotain tiettyä kategoriaa. Esimerkiksi *url*-avustaja sisältää funktioita sivulinkkien luomiseen, *form*-avustajalla saadaan luotua lomakkeen elementtejä ja *file*-avustajalla voidaan yksinkertaistaa tiedostojen käsittelyä palvelimella. (CodeIgniter User Guide 2010c.)

Toisin kuin useimmat muut CodeIgniterin elementit, avustajat eivät ole toteutettu olio-ohjelmoinnin käytännön mukaisesti. Avustajat eivät ole luokkia, vaan ryhmä aliohjelmia eli proseduureja, joita voidaan kutsua sovelluksesta. CodeIgniterin alkuperäisiä

avustajia voi myös laajentaa tai korvata kuten luokkiakin. Samoin omien avustajien luominen on mahdollista. Sovelluskohtaiset avustajat tallennetaan sovelluksen *helpers*-hakemistoon. (CodeIgniter User Guide 2010c.)

Määritimme CodeIgniterin *url*- ja *form*-avustajat latautumaan automaattisesti muokkaamalla sovelluksen *config*-hakemiston *autoload.php*-tiedostoa. Tällöin pystyimme käyttämään näiden avustajien funktioita ilman latauskomentoja.

3.6 Esimerkkisovellus

Kun harjoittelimme CodeIgniterin käyttöä, teimme useita testisovelluksia, joissa kokeilimme sovelluskehyksen eri ominaisuuksia. Yhtenä tällaisena testisovelluksena loimme pienen esimerkkisovelluksen, jonka tarkoituksena oli tulostaa luettelo rekisteröityneistä tekijäprofiileista siten, että tekijän nimeä klikkaamalla voitiin avata uusi näkymä, johon sovellus tulosti valitun tekijän tietoja. Kyseisen sovelluksen tekovaiheet havainnollistavat hyvin pitkälti sitä tapaa, jolla kehitimme Tekijäportaalia. Käymme seuraavaksi läpi tämän esimerkkisovelluksen luomisvaiheet. Esimerkkisovelluksen ohjaimen, mallin ja molempien näkymien lähdekoodit ovat liitteenä (Liite 3).

3.6.1 Ohjain

Yleensä aloitimme sovelluksen toimintojen toteuttamisen luomalla kullekin toimintokokonaisuudelle sopivan ohjaimen. Teimme esimerkkisovellukselle ohjaimen, joka perii *MY_Controller*-luokan ja annoimme ohjaimelle nimen *Esimerkki*. Sivun oletusotsikon määrittelevälle attribuutille asetimme arvon ”Profiilit”. Muodostinmetodissa latasimme *html*- ja *typography*-avustajafunktiot käytettäväksi.

Koska esimerkkisovelluksessa oli kaksi toimintoa, teimme ohjaimeen kaksi metodia, joissa molemmissa kutsuttiin lopussa *MY_Controller*-luokan *load_action_view*-metodia, joka asetti *Esimerkki*-ohjaimen metodin nimeä vastaavan näkymän HTML-mallinteen sisälle. Näistä kahdesta metodista ensimmäiselle annoimme nimeksi *index*, jota käytetään yleisesti ohjainten oletusmetodin nimenä. Toiselle metodille annoimme nimen

view. Päätimme käyttää *index*-metodia tekijäluettelon ja *view*-metodia yksittäisen tekijän tietojen esittämiseen.

Asetimme *view*-metodin vastaanottamaan tekijän profiilin id-numeroa merkitsevän parametrin URI-tunnisteesta. Parametrin avulla metodi tiesi minkä profiilin tietoja haluttiin hakea tietokannasta. Asetimme parametrille säännöiksi, että sen täytyy olla ensinnäkin määritelty ja että sen arvon on oltava numeerinen. Muussa tapauksessa sovellus ohjaisi käyttäjän takaisin *index*-metodiin. Tämän lisäksi teimme säännön, jonka mukaan sovellus ohjaisi käyttäjän takaisin myös niissä tapauksissa, joissa id-numeron avulla profiilia ei löydetä tietokannasta. Ennen näkymän kutsumista sivun oletusotsikko korvattiin arvolla ”Yhteenveto”.

3.6.2 Malli

Ohjaimen metodit tarvitsivat vielä sopivat tiedot, joita välittää omille näkymilleen. Muodostimme tätä sovellusta varten uuden *Esimerkkimalli*-nimisen mallin, joka perii *MY_Model*-luokan. Annoimme mallille taulun nimeä merkitsevän attribuutin arvoksi ”profiles” ja taulun id-saraketta merkitsevän attribuutin arvoksi ”profile_id”. Näiden attribuuttien ansiosta mallin funktioihin ei tarvinnut kirjoittaa tietokannan tietoja enää erikseen, sillä pystyimme käyttämään tietokantakyselyissä asetettujen attribuuttien arvoja.

Esimerkkisovellus tarvitsi toimiakseen kahta erilaista tietokantakyselyä. Ensimmäisen kyselyn tuloksena oli määrä saada kaikkien tekijöiden tiedot. Toisen kyselyn avulla piti saada yhden tietyn tekijän tiedot. Muodostimme tekemäämme malliin näille kyselyille omat funktionsa. Ensimmäisen nimeksi annoimme *findAll* ja toisen *findById*. Käytimme molempien funktioiden kyselyissä CodeIgniterin oman *Database*-luokan metodeja.

Jos malleja olisi tässä esimerkkisovelluksessa enemmän, viisaampi paikka sijoittaa nämä funktiot lienee MY_Model-luokka, sillä kaikki MY_Model-luokan perivät mallit saavat käyttöönsä isäntäluokkansa julkiset funktiot. Tämä tarkottaa sitä, että jokainen malli pystyisi käyttämään samoja *findAll* ja *findById*-funktioita omilla attribuuteillaan.

Pystyäksemme kutsumaan mallin funktioita ohjaimen metodeista lisäsimme *Esimerkki-*

ohjaimen *$use_models* attribuuttiin arvon ”Esimerkkimalli”. Asetimme ohjaimen metodit tallentamaan kutsuttujen mallin funktioiden palautusarvot muuttujiin.

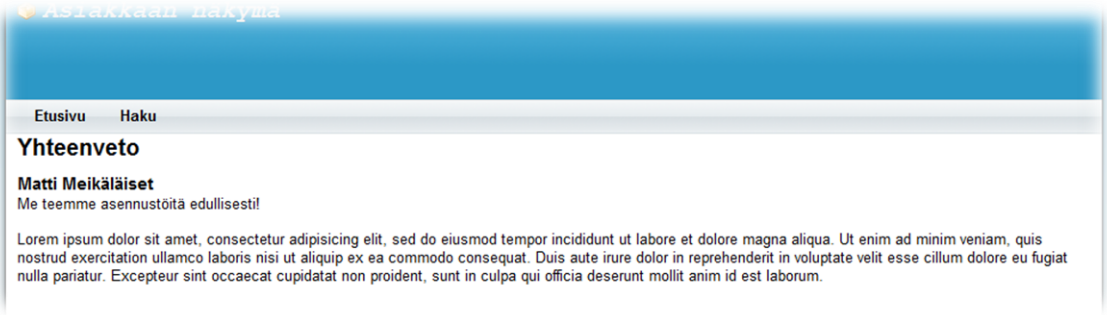
3.6.3 Näkymät

MY_Controller-luokan *load_action_view*-metodi etsii sovelluksen *views*-hakemistosta sellaista alikansiota, jolla on sama nimi kuin ohjaimella, josta näkymää on kutsuttu. Tässä onnistuttuaan metodi valitsee näkymäksi löydetystä alikansiosta sen tiedoston, jolla on sama nimi kuin ohjaimen metodilla lukuun ottamatta *.php*-tiedostopäätettä.

Loimme esimerkkisovellusta varten *esimerkki*-nimisen alikansion sovelluksen *views*-hakemistoon. Tallensimme alikansion sisälle kaksi näkymätiedostoa nimeltä *index.php* ja *view.php*. *Index.php* näkymälle välittynyt muuttuja oli moniulotteinen taulukko, joka sisälsi kaikki tietokannasta löytyneet profiilitiedot. Lisäsimme *index.php*-näkymään foreach-silmukan, joka kävi koko taulukon läpi ja generoi jokaiselle profiilitiedolle oman rivin ul-luetelmaelementtiin. Jokainen rivi koostui *url*-avustajalla tehdystä linkistä, jossa linkin osoite viittasi *view*-metodiin ja linkin tekstinä käytettiin tekijäyrityksen nimeä (Kuva 8).

Kuva 8. Esimerkkisovelluksen *index.php*-näkymä

Yksittäisen tekijäprofiilin näyttävälle *view.php*-näkymälle välittynyt muuttuja oli yksiulotteinen taulukko. Tähän näkymään emme lisänneet silmukkaa, vaan laitoimme näkymän tulostamaan taulukkomuuttujan kentistä tekijäyrityksen nimen ja kuvauksen. Tulostuvan tekstin ulkoasuun vaikutimme html- ja typography-avustajilla (Kuva 9).



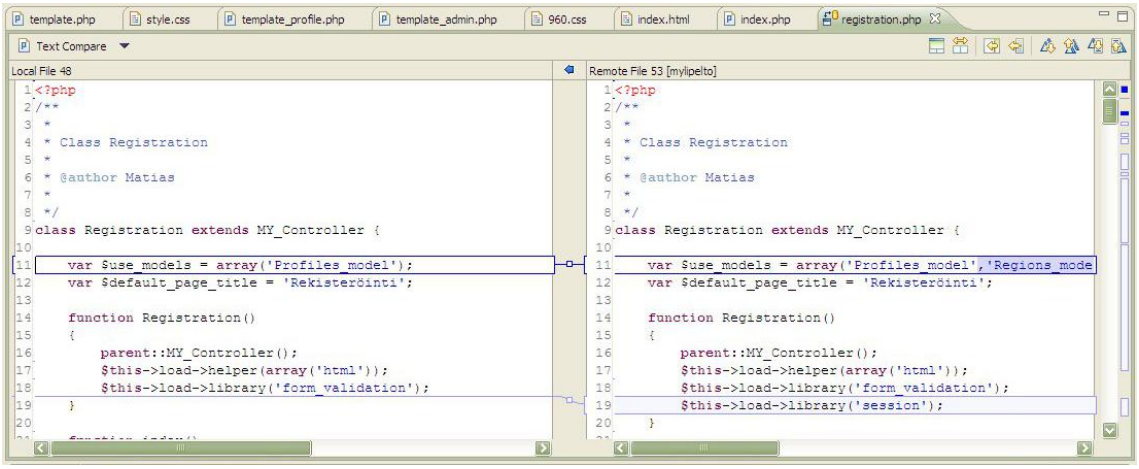
Kuva 9. Esimerkkisovelluksen *view.php*-näkymä

4 TOTEUTUS

4.1 Kehitysympäristö

Käytimme ohjelmointiin *Eclipse*-nimistä vapaan lähdekoodin ohjelmointiympäristöä, joka lienee paremmin tunnettu Java- kuin PHP-kielen ohjelmointiympäristönä. Latasimme työasemille *PHP Development Tools* eli *PDT*-jakelun Eclipsestä, jossa on valmiina monia PHP-kielen kirjoittamista helpottavia ominaisuuksia, kuten PHP-kielen syntaksin värikoodaus ja sisältörakenteiden muodostamisen avustustoiminto. Asennettua Eclipse-ohjelmointiympäristöä on mahdollista vielä laajentaa tukemaan muita ohjelmointikieliä. Myös erilaisia kehitystyökaluja on saatavilla moneen käyttötarkoitukseen.

Laajensimme työasemien Eclipse-asennuksia asentamalla tuen *SVN*-versionhallintajärjestelmälle. Saimme tässä projektissa käyttää toimeksiantajan webhotelliin asennettua versioarkistoa. Pystyimme versionhallintajärjestelmän avulla jakamaan ohjelmointitöitä keskenämme siten, että molempien työtiedostot pysyivät ajan tasalla. Pystyimme myös palauttamaan aikaisempia tiedostoversioita tarpeen tullen. Jokaisen uuden versiopäivityksen yhteydessä merkitsimme version kommenttikenttään lyhyesti muutamalla lauseella päivityksen sisällön. Kirjoitimme versioarkiston kommentit englanniksi. Jos olimme muokanneet samaa tiedostoa yhtä aikaa, niin pystyimme yhdistämään tehdyt muutokset Eclipsen vertailueditorilla. Pystyimme käyttämään vertailueditoria myös tiedostojen versioiden välisten muutosten tarkkailuun. Kuvassa 10 vertailueditoriin on aukaistu työaseman paikallinen ja versioarkiston uusin versio *registration*-ohjaimen tiedostosta. Eclipse merkitsee versioiden väliset muutokset värien ja viivojen avulla.



Kuva 10. Eclipse-ohjelmointiympäristön vertailueditori

Pystyimme testaamaan SVN-versionhallintajärjestelmän kautta ladattuja tiedostoja välittömästi, koska olimme määrittäneet Eclipsen käyttämään työtilanaan työasemille asennetun XAMPP-palvelinohjelmiston *htdocs*-kansiota. CodeIgniteri-sovelluskehystä ei kokonaisuudessaan säilytetty versioarkistossa, vaan ainoastaan sen *application*-hakemisto sekä sovelluksen käyttämät *img*-, *css*- ja *js*-hakemistot olivat määriteltynä versionhallinnan piiriin. Myös *application/config*-hakemiston *config.php*- ja *database.php*-tiedostot jätettiin pois versionhallinnasta, koska ne sisälsivät ympäristökohtaisia muuttujia.

4.2 Ulkoasu

Pilottiversion lopullista ulkoasua pääsimme toteuttamaan, kun sivuston informaatioarkkitehtuuri oli tarvittavan tarkasti selvillä ja tiesimme mitä elementtejä se tuli sisältämään. Pilottiversion ulkoasu muotoutui iteroiden tekemiemme rautalanka- ja niistä muotoutuneiden mockup-mallien pohjalta. Iteratiivinen kehittäminen tarkoittaa tuotteen tai palvelun kehittämistä sykleinä siten, että joka kierroksella analysoidaan suunnitelmien tila, suunnitellaan lisää yksityiskohtia ja toteutetaan suunnitelmasta prototyyppi, jonka käytettävyys arvioidaan tai testataan. Testin tulokset toimivat pohjana seuraavalle suunnittelukierrokselle. Tätä jatketaan, kunnes tuotteen käytettävyys on hyvä ja toiminnallisuus riittävä. (Sinkkonen ym. 2004.)

4.2.1 960 Grid System

Rakenteen pohjana käytimme 960 Grid System -rakennekehystä. Englanninkielinen termi *framework* on suurpiirteisyytensä vuoksi tässä tapauksessa sovellettavissa suomenkieliseksi termiksi *rakennekehys* termin *sovelluskehys* asemesta. Tässä tapauksessa rakennekehys on kuvaavampi sana, koska 960 Grid System ei varsinaisesti ole täysverinen ohjelmointiin tarkoitettu sovelluskehys, vaan HTML-sivujen ulkoasun nopeaan muodostukseen käytettävä työkalu. Nimensä mukaisesti se perustuu ruudukkopohjaiseen 960 pikseliä leveään järjestelmään. Luku 960 on jaollinen luvuilla 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 32, 40, 48, 60, 64, 80, 96, 120, 160, 192, 240, 320 ja 480. Näin ollen se on erittäin joustava toteutettaessa ulkoasuun liittyviä ratkaisuja. (960 Grid System 2010.)

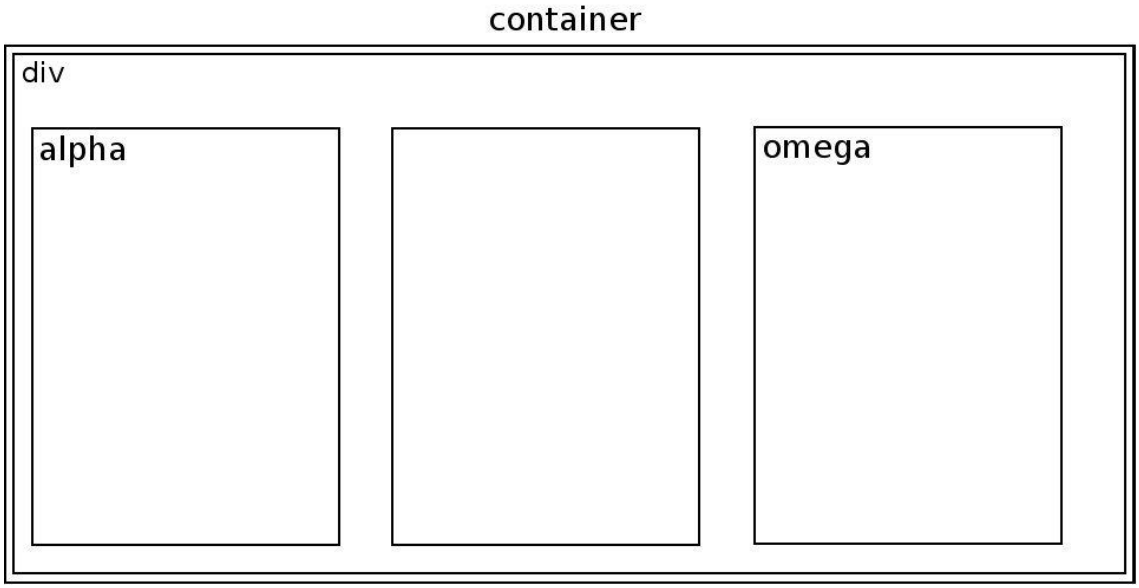
960 Grid System itsessään koostuu CSS-tiedostosta johon on määritelty valmiita tyylejä div-elementtejä varten. Tyylejä voidaan käyttää esimerkiksi seuraavalla tavalla:

```
1   <div class="container_12">
2     <div class="grid_7 prefix_1">
3       <div class="grid_2 alpha">
4         ...
5       </div>
6         <div class="grid_3">
7             ...
8         </div>
9       <div class="grid_2 omega">
10        ...
11      </div>
12    </div>
13    <div class="grid_3 suffix_1">
14      ...
15    </div>
16  </div>
```

(960 Grid System 2010.)

Rivillä yksi määritellään, kuinka moneen osaan jaettua ruudukkoa halutaan käyttää. *Container*-divin sisään asetetaan kaikki muu sisältö. Tässä tapauksessa sivu on jaettu vaakatasossa 12 pylvääseen joiden yhteenlaskettu leveys, 10 pikselin marginaaleineen sekä oikealla että vasemmalla, on 960 pikseliä. Rivillä kaksi asetetaan sisältöelementin luokaksi *grid_7* joka tarkoittaa että elementin leveys on seitsemän yksikköä kahdestatoista. Tämän lisäksi luokalla *prefix_1* saadaan yhden yksikön verran

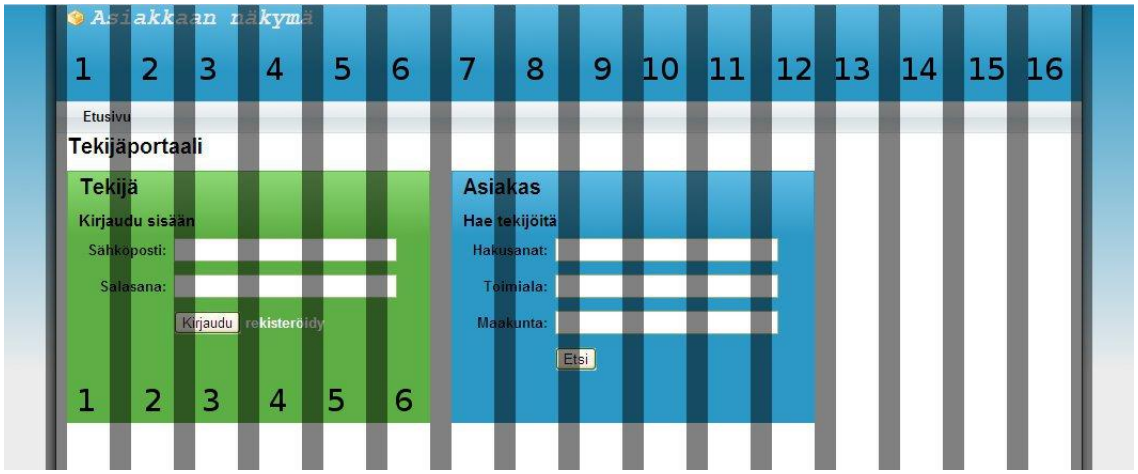
ylimääräistä tilaa ennen elementtiä ja vastaavasti elementin jälkeen luokalla *suffix_1* rivillä yhdeksän. Mikäli saman elementin sisällä on useampia lapsi-elementtejä, niistä ensimmäinen merkitään luokalla *alpha* ja viimeinen luokalla *omega*, kuten riveillä kolme ja yhdeksän on tehty. Kuva 11 osoittaa lapsi-elementtien sijoittumista *alpha*- ja *omega*-määrityksin.



Kuva 11. *Alpha*- ja *omega*-määriteltyjen elementtien sijoittuminen divin sisällä

Pylväisiin jaettu rakenne tekee sisällön asettelusta helppoa ja toimivaa, koska kaikkien elementtien koko täsmää tiettyä määrää pylväitä. 960 pikseliä jaettaessa pienempiin osiin on mahdollista muodostaa parillinen määrä täsmälleen samankokoisia elementtejä, ja näin pystytään varmistamaan että sivusto näkyy oikein. Jos sivuston leveys on jokin hankalasti jaollinen luku, sisältöelementtien koon määritteleminen on hankalampaa, ja on mahdollista, että osa sisältöelementeistä menee sisällön sisältävän elementin yli.

Ohessa kuva työstämme (kuva 12.), jossa käytettiin 16 pylvään levyistä rakennetta. Ulkoasun päällä on havainnollistava ruudukko ja pylväät on numeroitu parhaan mahdollisen selkeyden saavuttamiseksi. Mustat pystypalkit kuvaavat pylväiden välissä olevaa marginaalia.

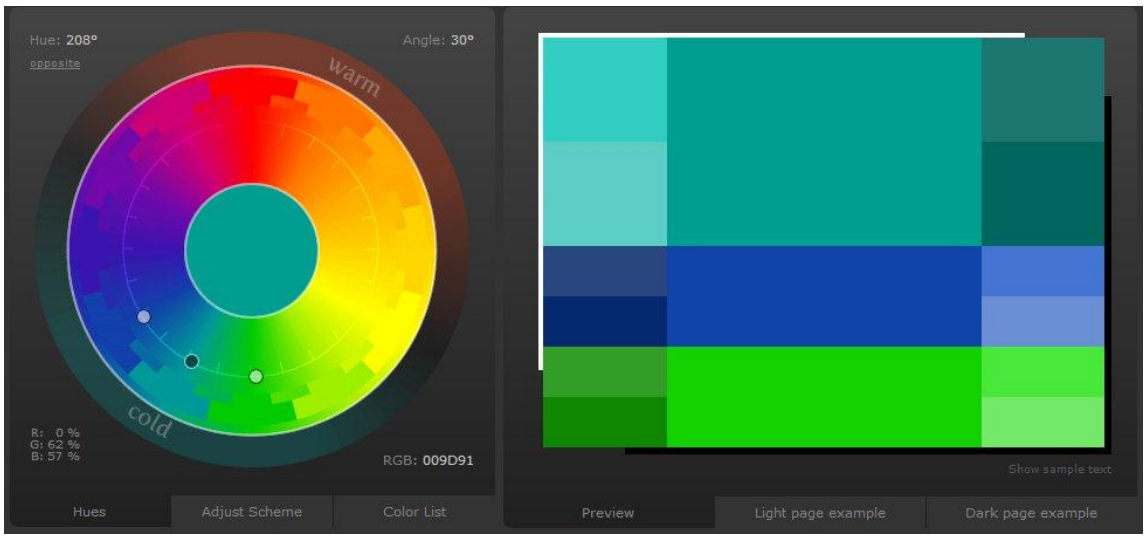


Kuva 12. Sivuston päälle on asetettu ruudukko havainnollistamaan rakennetta

960 Grid System -CSS-rakennekehyksen kehittäjän omilta kotisivuilta ladattava paketti sisältää CSS-tyylitiedostojen lisäksi erilaisia pohjia esimerkiksi Gimp- ja Adobe Photoshop -kuvankäsittelyohjelmia varten. Suunnittelussa nämä myös tulostettavaksi kelpaavat pohjat tulevat tarpeeseen, mutta varsinaista rakennetta varten tarvitsimme paketista ainoastaan yhden ”960.css”-nimisen tiedoston, jonka koko on varsin kohtuulliset 3,2 kilotavua. (960 Grid System, 2010.)

4.2.2 Värimaailman muotoutuminen

Suunnitteluvaiheessa päätimme pyrkiä yksinkertaisuuteen ja selkeyteen, joten päädyimme käyttämään muutamaa perusväriä erottamaan toimijoiden näkymät toisistaan. Suunnitteluvaiheessa paletista valittiin suurpiirteisesti muutama väri. Toteutusvaiheessa värejä ja niiden yhteensopivuutta hiottiin Internetistä löytyvällä *Color Scheme Designer* -apuohjelmalla. Kuvassa 13 Color Scheme Designerin värivalitsin ja värien esikatselu (Stanicek 2010).



Kuva 13. Kuvakaappaus Color Scheme Designerin värivalitsimesta

4.2.3 Muuta ulkoasun toteutuksessa huomioitavaa

Vaikka 960 Grid System piti huolen ulkoasun toimivuudesta, täysin sen varaan ei voinut laskea. Koska ihmiset selaavat Internetiä monilla eri selaimilla ja nykyään myös suurenevassa määrin erilaisilla mobiililaitteilla, kaikkien selainten kanssa yhteensopivaa sivustoa on mahdotonta tehdä. Ongelmia voi tulla esimerkiksi PNG-kuvaformaatin kanssa, joka toimii hyvin uusimmissa selaimissa, mutta vanhemmilla on ongelmia kuvien läpinäkyvien osien kanssa. GIF-kuvaformaatti on laajemmin tuettu, mutta jo kohtuullisen iäkäs formaatti ja sen varjopuolena on heikompi tarkkuus. On joko yritettävä tehdä sivusto mahdollisimman yhteensopivaksi mahdollisimman monen selaimen kanssa, tai arvioitava mikä voisi mahdollisesti olla sivuston kohderyhmän käytetyin selain ja edetä sen mukaan. Opinnäytetyön tekohetkellä pyrimme käyttämään vakiintuneita menetelmiä HTML-rakenteen ja -tyylin suhteen.

4.3 Testaus

Tekijäportaalin tapauksessa testaus suoritettiin yksikkötestauksena. Sillä tarkoitetaan hyvin matalan tason testausta, jossa varmistutaan metoditasolla siitä, että koodi toimii ohjelmoijan tarkoittamalla tavalla. (Ketterät käytännöt 2010). Aina yhden isomman kokonaisuuden tultua valmiiksi, testasimme sen toimivuuden käytännössä kaikilla mahdollisilla mieleen tulleilla tavoilla, joilla palvelua olisi mahdollista käyttää.

Mahdollisten virhetilanteiden tapauksissa etsimme koodista virheelliset kohdat ja korjasimme ne. Toistimme tätä niin kauan, kunnes emme enää kyenneet havaitsemaan virheitä.

4.4 Asiakkaan käyttöliittymä

Tekijäportaalin asiakkaita varten teimme hakulomakkeen, jonka avulla asiakas voi etsiä tekijöitä toimialan, maakunnan ja hakusanojen perusteella. Sovellus näyttää haun tulokset listana lomakkeen vieressä. Hakutuloksista tekijän nimeä klikkaamalla käyttäjä voi tarkastella tekijän profiilia tarkemmin. Tekijän profiilisivu sisältää tekijän rekisteröimän yrityksen tiedot ja mahdolliset profiilikuvat. Kuvassa 14 havainnollistetaan tietojen asettelu profiilissa.

Kuva 14. Asiakkaalle näkyvä tekijän profiilisivu

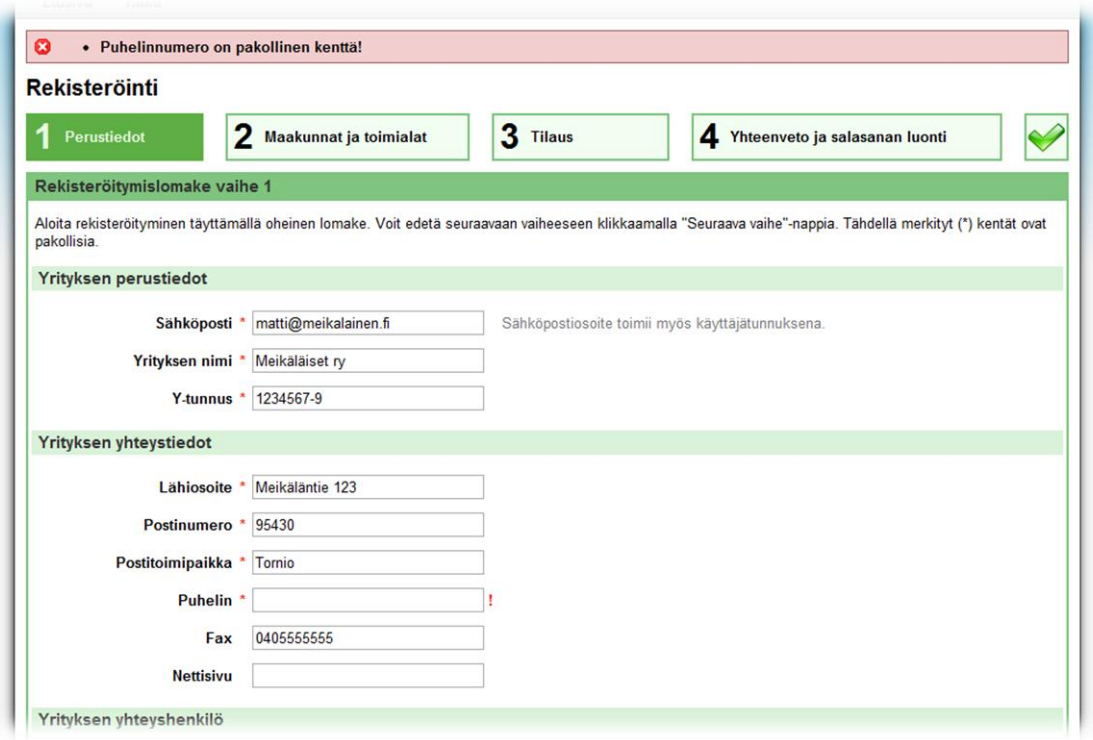
Tekijäprofiilin sivulla on myös yhteydenottolomake, jolla asiakas voi lähettää tekijälle sähköpostia. Mikäli asiakas haluaa ottaa yhteyttä tekijään lomakkeen avulla, asiakkaan tulee syöttää lomakkeen kenttiin oma nimi, sähköpostiosoite ja vapaamuotoinen viesti. Ennen viestin lähettämistä sovellus tarkistaa sähköpostiosoitteen kirjoitusasun oikeellisuuden ja että nimi- ja viesti-kentät ovat täytettynä. Jos yksi tai useampi kenttä ei läpäise tarkistusta, tulostetaan asiakkaan näkymään virheilmoitus, jossa luetellaan lomaketietojen virheet. Lisäksi virheelliset kentät merkitään punaisella huutomerkillä. Syötetyt tiedot säilytetään lomakkeen kentissä. Kuvassa 15 on esimerkki lähetystä lomakkeesta, joka sisälsi virheen.

Kuva 15. Virheellisen kentän osoittaminen lomakkeessa

4.5 Tekijän käyttöliittymä

4.5.1 Rekisteröityminen

Voidakseen käyttää tekijäportaalia tekijän oli ensin rekisteröidyttävä palveluun. Teimme rekisteröinnistä monivaiheisen lomakkeen, jossa tekijän oli täytettävä kaikki tarpeelliset tiedot omasta yrityksestään sekä valittava haluamansa tilaus hinnastosta. Tilauksia oli erikestoisia ja -hintaisia. Jokaisen vaiheen kohdalla sovellus tarkisti syötetyt tiedot ennen kuin se antoi tekijän edetä seuraavaan vaiheeseen. Täytetyt välivaiheet tallentuivat istuntotietoihin. Osa lomakkeiden kentistä oli pakollisia ja osassa pakollisten kenttien tiedoista tarkistettiin vielä niiden sisällön oikeellisuus. Pidimme periaatteena että järjestelmä ei antanut tekijän edetä, jos tekijän syöttämissä tiedoissa oli sellainen sähköpostiosoite tai Y-tunnus, joka oli jo jonkun toisen profiilin käytössä. Kuvassa 16 tekijä saa virheilmoituksen täyttämättömästä pakollisesta kentästä.



Kuva 16. Tekijää huomautetaan puutteellisista tiedoista

Lomakkeen viimeisessä vaiheessa tekijälle tulostettiin vielä hänen syöttämänsä tiedot, joita sovellus ohjeisti tarkistamaan. Jos tekijä huomasi syöttämissään tiedoissa virheitä, pystyi hän palaamaan vielä aikaisempiin vaiheisiin. Mikäli tekijä oli tyytyväinen syöttämiinsä tietoihin, tarvitsi hänen enää kirjoittaa itselleen salasana. Salasanan oli oltava vähintään 6- ja enintään 16-merkin pituinen ja sovellus kysyi sitä kahteen kertaan. Täytettyään salasana-kentät tekijä pystyi viimeistelemään rekisteröintinsä klikkaamalla ”rekisteröidy palveluun”-painiketta. Sovellus tarkisti vielä salasana-kentät ja tulosti tekijälle virhesanoman sekä kysyi salasanoja uudestaan, jos hänen kirjoittamansa salasana ei ollut oikean mittainen tai jos se oli toisessa kentässä kirjoitettu erilailla. Kun salasana oli onnistuneesti syötetty, sovellus loi tekijälle oman profiilin, jota hän pystyi käyttämään välittömästi.

Tekijäprofiilin salasanaa ei tallennettu tietokantaan sellaisenaan, vaan ensin se suolattiin ja suolatusta arvosta laskettiin sha-1-algoritmilla niin kutsuttu lyhenne eli *hash*. Suolauksella tarkoitamme alkuperäisen tiedon yhdistämistä muihin tietoihin siten, että tuloksena syntyy merkkijono, jonka väliin alkuperäinen tieto piilotetaan. Suolaukseen käytimme CodeIgniterin omaa *$config['encryption_key']*-muuttujaa sekä paria muuta määritettyä muuttujaa. Kun suolatusta salasanasta laskettiin vielä sha-1-lyhenne, saatiin aikaiseksi kryptattu salasana, jonka sovellus tallensi profiilin tietoihin.

4.5.2 Sisäänkirjautuminen

Kun tekijä oli onnistuneesti rekisteröitynyt, pystyi hän kirjautumaan palveluun sisäänkirjautumislomakkeen kautta. Tekijän täytyi kirjoittaa sisäänkirjautumislomakkeeseen oma sähköpostiosoite ja salasana. Vastaanotettuaan tiedot lomakkeelta sovellus tarkisti tietokannan tiedoista löytyikö kyseisellä sähköpostilla tekijäprofiilia. Jos tekijä löytyi, järjestelmä laski suolatun sha-1-algoritmilyhenteen syötetystä salasanasta ja vertasi sitä tietokannan profiilin salasana-kentän arvoon. Jos molemmat tiedot täsmäsivät tietokannan tietojen kanssa, tekijä kirjattiin sisään palveluun. Onnistuneen sisäänkirjautumisen yhteydessä tekijän istuntoon tallennettiin profiilin yhteyshenkilön nimi, käyttäjätunnus ja profiili-id-numero sekä muuttujat *role* ja *user_token*. Istunnon *role*-muuttujan arvoksi asetettiin ”profile”. *User_token*-muuttujan arvoksi laskettiin sha-1-lyhenne käyttäen tekijäprofiilin tietokannan tietojen arvoista yhdistettyä merkkijonoa.

Jokaisen vain *profile*-roolille tarkoitetun sivun latauksen yhteydessä ohjaimen metodi tarkasti, että istuntotietojen profiili-id-numero vastasi avatun sivun vastaavaa numeroa. Sovellus ei antanut tekijöiden katsoa muuta kuin omaa profiiliaan. Istuntotiedoista tarkistettiin myös että *user_token*-muuttujan arvo täsmäsi tietokannan arvoista laskettua sha-1-lyhennettä. Uloskirjautumisen yhteydessä tekijän istuntotiedoista poistetaan sisäänkirjautumista koskevat tiedot.

Sisäänkirjautunut tekijä ohjattiin sivulle, joka tulosti tekijän tiedoista muodostetun yhteenvedon. Tästä näkymästä tekijä pystyi tarkastelemaan nykyisen tilauksen tilannetta ja syöttämiään tietoja. Näkymästä oli myös mahdollista edetä profiilitietojen muokkaamista varten tarkoitetuille sivuille. Sisäänkirjautunut tekijä pystyi lataamaan myös profiiliin yrityksensä logon sekä yhden profiilikuvan.

Tilausten hallintasivulla tekijälle tulostettiin tiedot tehdyistä tilauksista. Tekijäprofiili tulisi näkyville tekijähauissa vasta, kun tekijä on maksanut tekemänsä tilauksen. Emme ehtineet luoda opinnäytetyön aikataulun puitteissa valmista maksutoimintoa tekijäportaalin pilottiversioon, joten tilausten hallintasivun ”maksa tilaus”-linkki ohjasi vain tyhjään ohjaimen metodiin. Myös kaavailtu uuden tilauksen tekeminen puuttui

opinnäytetyön aikana tehdystä tekijäprofiilin pilottiversiosta.

4.5.3 Salasanan uusiminen

Tapauksissa, joissa tekijä olisi joistain syystä unohtanut salasanansa, sen uusiminen oli mahdollista siihen tarkoitetun lomakkeen avulla. Lomakkeelle pääsi sisäänkirjautumislomakkeen yhteydessä olevan ”Unohditko salasanasi?”-linkin kautta. Lomake kysyi tekijältä sähköpostiosoitetta, joka toimi tekijän käyttäjätunnuksena. Jos syötettyä sähköpostiosoitetta ei löytynyt tietokannan profiilitiedoista, sovellus ilmoitti virheellisestä sähköpostiosoitteesta. Mikäli sähköpostiosoite löytyi tietokannasta, sovellus lähetti tähän osoitteeseen viestin, jossa oli linkki salasanan uusimiseen tarkoitetulle sivulle. Sovelluksen lähettämä linkki sisälsi kryptatun avaimen, jonka avulla voitiin varmistaa se, ettei kuka tahansa voinut päivittää käyttäjän salasanaa, vaan se oli mahdollista ainoastaan klikkaamalla viestissä ollutta linkkiä. Linkin kryptausavain oli voimassa vain lyhyen ajan, joten linkkiäkään ei voinut käyttää rajattomasti. Linkistä sovellukseen siirryttäessä sovellus tarkisti oliko kryptausavain oikeanlainen ja voimassaoleva. Mikäli tämä tarkistus meni läpi, näytettiin tekijälle lomake, jossa pyydettiin tekijää kirjoittamaan uusi salasana ja salasanan vahvistus. Kun uusi salasana oli syötetty onnistuneesti, sovellus vaihtoi uuden salasanan tekijäprofiiliin käyttäen samaa kryptausmenetelmää kuin rekisteröinnissä. Tekijä ohjattiin lopussa sisäänkirjautumislomakkeelle, jossa hänelle tulostettiin viesti onnistuneesta salasanan vaihdosta.

4.6 Ylläpito-osio

Loimme tekijäportaaliin erillisen sisäänkirjautumisen ylläpitoa varten. Ylläpitäjän sisäänkirjautumislomakkeelle pääsi vain ylläpidon tiedossa olevan url-osoitteen kautta. Url-osoite on tehty hankalasti arvattavaksi, eikä sinne ole olemassa linkkejä sivustolla. Pyrimme näin hankaloittamaan asiattomien pääsyä ylläpitäjän kirjautumislomakkeelle. Ylläpitäjän tunnus ja salasana oli tallennettuna eri tauluun tietokannassa kuin tekijöiden tiedot. Sisäänkirjautuneen ylläpitäjän istuntotietoihin tallennettiin role- ja user_token-muuttujat. Role-muuttujan arvoksi asetettiin ”admin” ja *User_token*-muuttujan arvoksi laskettiin sha-1-lyhenne usealla tietokannan tiedolla suolatusta ylläpitäjän

käyttäjätunnuksesta.

Ylläpitäjällä oli oikeus muokata kaikkien profiilien tietoja sekä määrittää käytettävissä olevat toimialat ja voimassa oleva hinnasto. Vaikka tekijän käyttöliittymästä puuttui vielä tilausten tekemiseen ja maksamiseen vaadittavat toiminnot, pystyi ylläpitäjä lisäämään yksittäiselle tekijälle näkyviä tarjouksia tilauksista sekä merkitsemään tekijän maksamattomat tilaukset maksetuiksi ja voimassaoleviksi.

5. TEKIJÖIDEN MIETTEITÄ

5.1 Lopputulos

Projektin lopputuloksena syntyi asiakkaita ja erilaisia asennustöitä tekeviä tekijöitä yhdistävän tekijäportaalin pilottiversio. Tekijän oli mahdollista luoda palveluun oma profiili ja asiakkaat pystyivät tekemään tekijöistä hakuja erilaisin ehdoin. Ylläpitäjä pystyi muokkaamaan tekijöiden profiilitietoja sekä heidän tilauksiaan.

Kaikkia pilottiversioon kaavailtuja toiminnallisuuksia emme opinnäytetyön aikataulun puitteissa ehtineet toteuttamaan, mutta toisaalta pilottiversion ei ollut tarkoituskaan olla valmis julkaistavissa oleva tuote. Huomioimme jatkokehityksen mahdollisuuden suunnittelussa ja toteutuksessa. Teimme esimerkiksi tietokantaan valmiita tauluja ja taulujen sarakkeita, joita on mahdollista hyödyntää myöhemmin.

5.2 Yleistä projektista

Kaiken kaikkiaan tekijäportaalin toteuttaminen oli mielenkiintoinen projekti. Opinnäytetyötä tehdessä tuli opittua paljon uusia ohjelmistokehitykseen liittyviä menetelmiä. Esimerkiksi CodeIgniter ja versionhallintajärjestelmä olivat toiselle tämän opinnäytetyön tekijöistä täysin uusia asioita projektia aloitettaessa, mutta tulivat työn edetessä tutummiksi. Koulun puitteissa näitä kumpaakaan ei oltu sivuttu millään tavoin, joten tekijäportaalia toteutettaessa olemme oppineet paljon uutta arvokasta tietoa tulevaisuutta silmälläpitäen.

CodeIgniter-sovelluskehyksen käyttö projektissa oli varmasti yksi tärkeimmistä asioista lopputuloksen onnistumisen kannalta. Sillä säästyi turhalta samojen asioiden toistuvalta koodaamiselta ja se helpotti paljon muun muassa tietokantakyselyiden muodostamista. Ohjelmoinnin helpottumisen myötä saimme lyhyehkössä ajassa aikaiseksi sovelluksen, jonka toteuttamiseen ilman sovelluskehystä olisi mennyt huomattavasti pitempi aika. Samalla se vahvisti käsitystä siitä, ettei kaikkea välttämättä kannata alkaa koodata itse puhtaalta pöydältä, vaikka taitoa olisikin.

5.3 Projektin eteneminen

Projektin aikana olimme yhteydessä toimeksiantajaamme säännöllisin väliajoin Skype-pikaviestimen avulla. Mielestämme tämä oli erittäin toimiva ratkaisu etenkin, kun välimatka meidän ja toimeksiantajan välillä oli useita satoja kilometrejä. Säännöllisillä etäpalavereilla säästyimme turhautumisilta ja samalla pysyimme ajan tasalla projektin etenemisestä.

Toimeksiantajamme oli vahvasti mukana projektissa ja saimme heiltä hyvää ohjausta sekä rakentavaa palautetta. Toimeksiantajamme kuunteli myös meidän ideoitamme ja ehdotuksiamme projektin toteutuksen suhteen ja oli muutenkin kaikin puolin yhteistyöhaluinen.

5.4 Kiitokset

Haluamme lopuksi kiittää ensinnäkin toimeksiantajaamme opinnäytetyön aiheesta ja tästä ainutlaatuisesta mahdollisuudesta olla mukana tämänkaltaisessa projektissa, joka oli kaikin puolin opettavainen ja antoisa kokemus. Lisäksi kiitos kuuluu ohjaavalle opettajallemme Yrjö ”Ykä” Koskenniemelle, ystäville, jotka estivät vaipumasta synkkyyteen epätoivon hetkinä, sekä sukulaisille ja tutuille, jotka oikolukivat raporttiamme – ja toki haluamme kiittää myös toisiamme kunnialla valmiiksi saadusta opinnäytetyöstä. Kiitos.

LÄHTEET

Painetut

Heinisuo, Rami 2004. PHP ja MySQL - Tietokantapohjaiset verkkopalvelut. Gummerus Kirjapaino Oy, Jyväskylä.

Hovi, Ari & Huotari, Jouni & Lahdenmäki, Tapio 2005. Tietokantojen suunnittelu & indeksointi. Docendo Finland Oy. Porvoo.

Järvinen, Pertti & Järvinen, Annikki 2000. Tutkimustyön metodeista. Opinpajan kirja, Tampere.

Koskimies, Kai & Mikkonen, Tommi 2005. Ohjelmistoarkkitehtuurit. Gummerus Kirjapaino Oy, Jyväskylä.

Metsämäki, Markku 2000. Verkkopalvelun suunnittelu. Oy Edita Ab, Helsinki.

Sinkkonen, Irmeli & Nuutila, Esko & Törmä, Seppo 2009. Helppokäyttöisen verkkopalvelun suunnittelu. Kariston Kirjapaino Oy, Hämeenlinna.

Painamattomat

960 Grid System 2010. Luettu 9.11.2010. <http://960.gs/>

Apache Friends 2010. Luettu 15.11.2010. <http://www.apachefriends.org/en/index.html>

Apache Subversion Features 2010. Luettu 2.11.2010. <http://subversion.apache.org/features.html>

Avoin lähdekoodi 2010. Luettu 11.11.2010. <http://www.coss.fi/abc/avoin-lahdekoodi>

CodeIgniter User Guide 2010a. Creating Libraries. Luettu 16.11.2010. <http://codeigniter.com/user_guide/overview/appflow.html>

CodeIgniter User Guide 2010b. Application Flow Chart. Luettu 16.11.2010. <http://codeigniter.com/user_guide/general/creating_libraries.html>

CodeIgniter User Guide 2010c. Helper Functions. Luettu 18.11.2010. <http://codeigniter.com/user_guide/general/helpers.html>

CodeIgniter Wiki 2010. Messages. Luettu 16.11.2010. <http://codeigniter.com/wiki/Messages/>

CSS Introduction 2010. Luettu 8.11.2010. <http://www.w3schools.com/css/css_intro.asp>

EllisLab - products 2010. Luettu 15.11.2010. <http://ellislab.com/products/>

Helsingin yliopisto 2004. Tiedostojen paketointi. <http://www.ling.helsinki.fi/kit/2005s/clt130/luento7.shtml>

Ketterät käytännöt 2010. Yksikkötestaus. Luettu 19.11.2010. <http://www.ketteratkaytannot.fi/fi-FI/Kaytannot/Yksikkotestaus>

Laaksonen, Antti 2010. Oppaat: MySQL ja PHP. Osa 10 – Lisätietoa. Luettu 1.11.2010. <http://www.ohjelmointiputka.net/opas.php?tunnus=mysqlphp10>

Lavonen & Meisalo & al. 2010. Luovan ongelmaratkaisun työtavat. Luettu 2.11.2010. <http://www.edu.helsinki.fi/malu/kirjasto/lor/>

MySQL 2010. MySQL Community Edition. Luettu 9.11.2010. <http://www.mysql.com/products/community/>

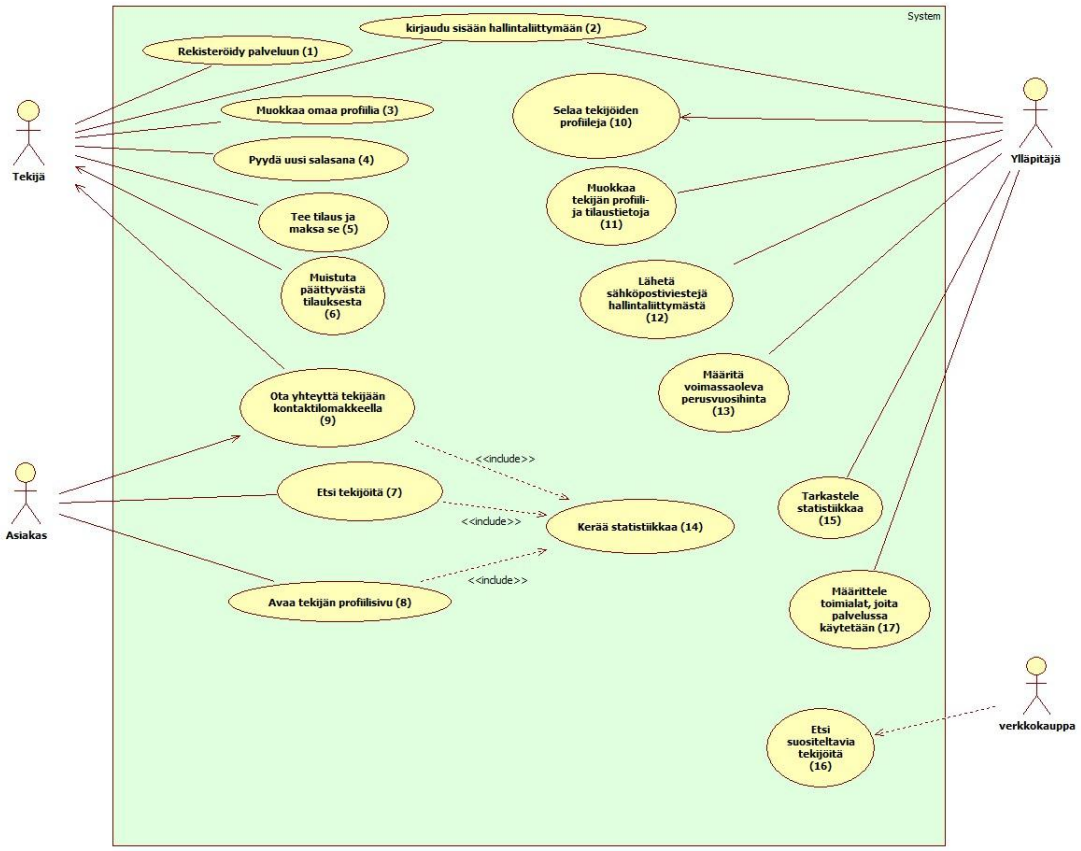
PHP-opas 2010. Luettu 16.11.2010. <http://wiki.mureakuha.com/wiki/PHP-opas>

Stanicek, Petr 2010. Color Scheme Designer. Luettu 17.11.2010. <http://colorschemedesigner.com/>

1 – Rekisteröidy palveluun

Versio	**Versio**	**Pvm**	**Tekijä**	**Muutokset**
	1.00	30.09.10	Matias	Ensimmäinen versio
Tavoite	Tekijä voi rekisteröityä palveluun omatoimisesti ilman ylläpidon tukea			
Edellytykset	-			
Onnistunut suoritus	Tekijän profiili tallentuu palveluun.			
Virhetilanteet	**Virhe**	**Tulos**	**Tuloksen ehto**	
	Järjestelmä ei hyväksy tekijän syöttämiä tietoja	Järjestelmä ilmoittaa tekijälle virheellisistä tiedoista ja pyytää korjaamaan ne.		
Ensisijainen toimija/rooli	Tekijät			
Toissijaiset toimijat				
TOIMINNALLINEN KUVAUS				
Päätoimintatapa	1. Tekijä klikkaa rekisteröitymislomakkeelle johtavaa linkkiä. 2. Järjestelmä avaa sivun, jossa on lomake. 3. Tekijä täyttää lomakkeen omilla tiedoillaan ja tallentaa tiedot 4. Järjestelmä tallentaa tiedot tietokantaan ja lähettää sähköpostiviestin ylläpitäjän osoitteeseen uudesta profiilista 5. Käyttötapaus onnistuneesti suoritettu			
Vaihtoehtoinen toimintatapa	4. Järjestelmälle annetut tiedot olivat virheellisiä 4a1. Järjestelmä näyttää tekijälle uudestaan saman lomakkeen ja ilmoittaa tekijälle mitkä annetuista tiedoista olivat virheellisiä 4a2. Tekijä muokkaa lomakkeen tietokenttiä. 4a2a. Tekijä korjaa omat tietonsa ja lähettää tiedot uudestaan 4a2b. Tekijä ei rekisteröidy ja poistuu toiminnosta. 4a3. Käyttötapaus onnistuneesti suoritettu			
Avoimet asiat	Ylläpitäjä saa sähköpostiviestin uudesta rekisteröityneestä tekijästä ja tekijän maksusta. Selvitettävä profiilin sisältö ja eri kenttiin sovellettavat validointimekanismit.			

KÄYTTÖTAPAUSKAAVIO Liite 2



Esimerkki-ohjain

```php
<?php
class Esimerkki extends MY_Controller {

    var $use_models = array('Esimerkkimalli');
    var $default_page_title = 'Profiilit';

    function Esimerkki()
    {
        parent::MY_Controller();
        $this->load->helper(array('html','typography'));
    }

    function index()
    {
        // load data from database and set it for view file
        $view_data['profiles'] = $this->Esimerkkimalli->findAll();
        // load html template and the esimerkki/index.php view file.
        $this->load_action_view($view_data);
    }

    function view($id=null)
    {
        // redirect back to index page if id parameter isn't set
        // or isn't numeric value
        if( !isset($id) OR !is_numeric($id) )
            redirect(site_url('esimerkki/index'));

        // find a row that has id number that matches $id parameter
        $profile = $this->Esimerkkimalli->findById($id);

        // redirect back to index page if records were not found
        if(empty($profile))
           redirect(site_url('esimerkki/index'));

        // set data for view file
        $view_data['profile'] = $profile;

        // set different page title and headline to html template
        $view_config = array('title' => 'Yhteenveto');

        // load html template and the esimerkki/view.php view file.
        $this->load_action_view($view_data,$view_config);
    }
}
/* End of file esimerkki.php */
/* Location: ./system/application/controllers/esimerkki.php */
```

Esimerkkimalli-malli

```php
<?php
class Esimerkkimalli extends MY_Model {
    var $tbl_name = 'profiles';
    var $tbl_id_column = 'profile_id';

    function Esimerkkimalli()
    {
        parent::MY_Model();
    }

    /**
     *
     * Returns all rows from model's table.
     */
    function findAll()
    {
        $query = $this->db->get($this->tbl_name);
        return $query->result_array();
    }

    /**
     *
     * Returns a row with given id from model's table
     * @param $id
     */
    function findById($id)
    {
        $id_column = $this->tbl_name.'.'.$this->tbl_id_column;
        $query = $this->db->get_where($this->tbl_name, array($id_column => $id), 1);
        return $query->row_array();
    }
}
/* End of file esimerkkimalli.php */
/* Location: ./system/application/models/esimerkkimalli.php */
```

Näkymä *Esimerkki*-ohjaimen *index*-metodille

```
<div class="grid_16">
<?php
if(!empty($profiles)):?>
    <ul><?php
    foreach($profiles as $p):?>
        <li><?php
        echo anchor('esimerkki/view/'.$p['profile_id'],$p['company_name']);?>
        </li><?php
    endforeach;?>
    </ul><?php
endif;?>
</div>
<?php
/* End of file index.php */
/* Location: ./system/application/views/esimerkki/index.php */
```

Näkymä *Esimerkki*-ohjaimen *view*-metodille

```
1 <div class="grid_16">
2 <?php
3 echo heading($profile['company_name'],3);
4 echo auto_typography($profile['company_description']);
5 ?>
6 </div>
7 <?php
8 /* End of file view.php */
9 /* Location: ./system/application/views/esimerkki/view.php */
```