OULUN SEUDUN
AMMATTIKORKEAKOULU

Tiina Runtti

BLOGI-LISÄOSAN KEHITTÄMINEN RAMSES CMS -JÄRJESTELMÄÄN

BLOGI-LISÄOSAN KEHITTÄMINEN RAMSES CMS -JÄRJESTELMÄÄN

Tiina Runtti
Opinnäytetyö
Kevät 2011
Tietojenkäsittelyn koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä(t): Tiina Runtti
Opinnäytetyön nimi: Blogi-lisäosan kehittäminen Ramses CMS -järjestelmään
Työn ohjaaja(t): Jouni Juntunen
Työn valmistumislukukausi ja -vuosi: Kevät 2011 Sivumäärä: 32 + 15 liitesivua

---

Tämän opinnäytetyön toimeksiantaja on Faarao Oy. Faarao Oy:n eräs tuote on Ramses CMS -järjestelmä, joka on helppokäyttöinen julkaisujärjestelmä. Ramses CMS -järjestelmä sisältää useita lisäosia, joita ovat muun muassa uutis-, google kartat- ja galleria-lisäosat. Asiakkaiden pyynnöstä Faarao Oy halusi saada helppokäyttöisen blogi-lisäosan järjestelmäänsä. Tämän työn tavoitteena on kehittää toimiva blogi-lisäosa Ramses CMS -järjestelmään, jonka ominaisuudet vastaavat toimeksiantajan vaatimuksia.

Ramses CMS -järjestelmä on kehitetty CodeIgniter-ohjelmalla, joka hyödyntää MVC-arkkitehtuurimallia. Tämän vuoksi työssä käsitellään ohjelmistoarkkitehtuuria yleisesti, MVC-arkkitehtuurisuunnittelumallia sekä CodeIgniter-ohjelmistoa. Raportissa esitetään MVC-arkkitehtuurimallin mukaisesti, kuinka blogi-lisäosa kehitettiin.

Työn tuloksena syntyi toimiva blogi-lisäosa Ramses CMS -järjestelmään, josta se voidaan liittää Internet-sivuille. Toimeksiantaja voi myydä blogi-lisäosaa asiakkailleen ja halutessaan kehittää lisää ominaisuuksia siihen. Lisäosa on ohjelmoitu englanniksi ja kommentoitu perusteellisesti. Lisäosaan käytettyjä funktioita, kuten tietokantakyselyitä, voidaan hyödyntää tarpeen mukaan myös muiden lisäosien kehittämisessä.

Asiasanat: ohjelmistoarkkitehtuuri, MVC, codeigniter

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Technology

Author(s): Tiina Runtti
Title of thesis: Developing a blog plug in for Ramses CMS
Supervisor(s): Jouni Juntunen
Term and year when the thesis was submitted: Spring 2011
Number of pages: 32 + 15

The client of this thesis is the software company Faarao. One of Faarao's products is the Ramses CMS. It includes several additional components like among others Google Maps plug-in, Gallery plug-in and News plug-in. The wish of Faarao's customers is to get a simple blog component for the Ramses CMS. Therefore the aim of this project is to develop a blog plug-in for the Ramses CMS which fit with the client's requirements.

Ramses CMS is developing their products with CodeIgniter which takes advantage of MVC architecture model. Therefore, this thesis is explaining the software architecture in general, the MVC architecture design model and the CodeIgniter software. Additionally this report explains how the blog plug-in was developed using the MVC architecture model. The entire plug-in has been completely commented and developed in English.

As a result of this project, the blog plug-in can now be used in the Ramses CMS. The client Faarao likely sell the blog plug-in to their customers and later develop the additional features for the blog plug-in. Some additional functions, example database queries, may be used for further development of other plug-ins.

Keywords: software architecture, MVC, codeigniter

SISÄLLYS

1 JOHDANTO

Ohjelmistoarkkitehtuureja on pidetty ainoastaan korkeamman tason suunnitteluna. Vasta 1990-luvun loppupuoliskolla ne ovat yleistyneet ohjelmistotekniikassa. Nykypäivänä ohjelmistoarkkitehtuurin merkitys ohjelmistotekniikassa kasvaa kokoajan. Ohjelmistojen jatkuva monimutkaistuminen, kasvaminen ja uudelleenkäytön merkityksen kasvaminen ovat vaikuttaneet siihen, että ohjelmistojen luominen tapahtuu yhä enemmän arkkitehtuuritason ohjelmiston ja toteutusmekanismien pohjalta. Vain arkkitehtuuritasolla tunnistettavat järjestelmän osat ovat mielekkäitä työn jakamiseen. Hyvä arkkitehtuuri jakaa ohjelmiston osiin, joten ohjelmisto on helppo testata erillisinä osina ja sen ylläpito on yksinkertaisempaa. (Koskimies & Mikkonen 2005, 15–16.)

Ohjelmiston hahmottamiseen käytetään apuna usein arkkitehtuurityyliä. Kerrosarkkitehtuuri on yksi yleinen arkkitehtuurityyli, mutta tietynlainen toteutustapa on vakiintunut joillekin sovelluksille. (Koskimies & Mikkonen 2005, 125.) ”Toteutusteknisesti sovellusaluekohtaista arkkitehtuurityyliä noudatettava järjestelmä voidaan rakentaa palveluiden ja jonkinlaisen kerrostamisen varaan” (Koskimies & Mikkonen 2005, 125).

Eräs arkkitehtuurityyli on MVC (Model-View-Controller), joka on yksi laajasti käytetty arkkitehtuurimalleista. MVC-arkkitehtuuri kehiteltiin Xerox PARC Smalltalk-80 -ohjelmaa kehittäessä. MVC-arkkitehtuurimalli ei vaadi tiettyä ohjelmointikieltä, sen vuoksi se on erittäin suosittu ja käyttökelpoinen arkkitehtuurimalli. Siitä on tullut erittäin suositeltu malli Sun J2EE -alustalle ja se on kasvattanut suosioitaan myös ColdFusion- ja PHP-kehittäjien keskuudessa. (Kotek 2002, hakupäivä 26.04.2011.)

2 TAUSTA

Opinnäytetyön toimeksiantaja on oululainen ohjelmistoyritys Faarao Oy, jossa tekijä on suorittanut opintoihin kuuluvan ammattiharjoittelun. Faarao Oy on Internet-pohjaisiin ohjelmistoihin erikoistunut yritys, jonka palveluilta ovat Internet-sivustot, verkkokaupparatkaisut sekä intra- ja extranetratkaisut. Faarao Oy:n eräs tuote on Ramses CMS -järjestelmä. Ramses CMS -järjestelmä on kehitetty yhteistyössä Oulun yliopiston tietojenkäsittelytieteiden laitoksen kanssa. Kehitystyön taustalla on runsas määrä käyttäjäkokemuksia vastaavista järjestelmistä sekä omalta asiakaskunnalta tulleita toiveita ja palautteita. Näiden pohjalta Faarao Oy on kehittänyt oman julkaisujärjestelmän, johon on otettu mukaan hyvänä pidettyjä toimintoja muualta ja kehitetty heikkona pidettyjä ominaisuuksia.

Ramses CMS -järjestelmä on helppokäyttöinen julkaisujärjestelmä, jolla voidaan julkaista Internet-sivustoja. Järjestelmä sisältää useita lisäosia, esimerkiksi uutiset-, kartat-, kuvagalleria ja sää-lisäosat. Käyttäjä voi esimerkiksi tallentaa kartat-lisäosassa Google Maps -karttoja, jonka jälkeen ne voidaan lisätä Internet-sivuille. Ramses CMS -järjestelmä sisältää palautuspisteitä. Jos käyttäjä tekee virheen sivustoa rakentaessa, tehdyt muutokset voidaan kumota. Ramses CMS -järjestelmässä on eri käyttäjätasoja, jonka vuoksi sen käyttäjät voivat olla henkilöitä aina ammattilaisista loppukäyttäjiin. Esimerkiksi alan ammattilaiset pääsevät muokkaamaan sivustojen graafista ulkoasua, kun taas peruskäyttäjät voivat kehittää oman Internet-sivustonsa helposti. Ramses CMS -järjestelmään voidaan kehittää jatkuvasti asiakkaiden toiveiden perusteella uusia lisäosia.

Ramses CMS -järjestelmä on ohjelmoitu CodeIgniter-ohjelmalla, jonka on kehittänyt EllisLab-niminen yritys. CodeIgniter on avoimen lähdekoodin sovelluskehys, joka sisältää useita luokkia ja avustajia, joiden avulla voidaan tehdä turvallisesti ja nopeasti dynaamisia Internet-sivustoja. CodeIgniter pohjautuu MVC-arkkitehtuurimalliin. MVC-arkkitehtuurimallissa sovelluslogiikka erotetaan käyttöliittymästä. (EllisLab 2011, hakupäivä 17.03.2011.)

Tämän opinnäytetyön tarkoituksena on kehittää helppokäyttöinen blogi-lisäosa Ramses CMS -järjestelmään, jota Faarao Oy voi myydä Ramses CMS -järjestelmän lisäosana asiakkailleen. Faaraon asiakkaat ovat toivoneet helppokäyttöistä blogia. He ovat käyttäneet jotain muita blogi-ohjelmia kuten Wordpress- ja Blogspot-ohjelmia, mutta he kaipasivat blogi-ohjelmaa integroituna Ramses CMS -järjestelmään. Faaraon asiakkaat käyttävät jo Ramses CMS -järjestelmää, jonka vuoksi heidän on helppo opetella käyttämään blogi-lisäosaa jo entuudestaan tunnetulla järjestelmällä. Aihe on siis toimeksiantajalle ajankohtainen ja tärkeä, koska toimeksiantaja tulee hyötymään blogi-lisäosasta.

Ohjelmointikielenä on pääasiallisesti PHP, mutta työssä käytetään myös HTML- ja JavaScript-kieliä käyttöliittymän toteutuksessa. Blogi-lisäosan tietokanta suunnitellaan MySQL Workbench -ohjelmalla ja se toteutetaan SQL-kielellä. MySQL Workbench -ohjelman avulla suunnitellusta tietokannasta voidaan generoida taulujen SQL-kieliset luontilauseet automaattisesti. Tietokannan hallintatyökaluna käytetään phpMyAdmin-työkalua. Blogi-lisäosan tarvitsemat taulut lisätään Ramses CMS -järjestelmän tietokantaan, joka sisältää jo aiempaan toiminnallisuuteen liittyvät taulut. Edellä mainittuja työkaluja käytetään, koska ne ovat toimeksiantajan päivittäisessä käytössä. Työkalut ovat perustyökaluja ilman erikoisia hienouksia, mutta kaikki työkalut ovat avoimen lähdekoodin ohjelmistoja, jonka vuoksi ne sopivatkin erinomaisesti pienelle yritykselle.

Blogiin tulee kehittää erilaisia toimintoja, joita käsitellään Ramses CMS -järjestelmässä. Näitä toimintoja ovat: blogin kirjoitus, johon voidaan myös liittää kuvia, blogin muokkaaminen ja poistaminen sekä ajastettu julkaisu. Ramses CMS -järjestelmässä pitää voida myös lisätä, muokata, ja poistaa kategorioita, avainsanoja sekä kommentteja. Kommenteissa täytyy olla asetus, josta voidaan valita, julkaistaanko kommentit heti, vai pitääkö ne hyväksyä ennen julkaisemista. Internet-sivuilla pitää pystyä näkemään yksittäisen blogin, kaikki blogit kerralla sekä blogin historian. Internet-sivulla blogien kategoriat ja avainsanat pitää olla listattuna sivun reunalla. Kun kategoriaa tai avainsanaa klikataan, sen kategorian tai avainsanojen sisältämät blogit tulevat Internet-sivulle näkyviin. Blogin kehitysvaiheessa toimeksiantajalta tuli pyyntö blogien arkistoinnille. Blogit pitää pystyä jaottelemaan vuoden mukaan ja ne pitää voida nähdä Internet-sivuilla.

Kun blogi-lisäosa halutaan lisätä joillekin Internet-sivuille Ramses CMS-järjestelmässä, haluttu lisäosa valitaan sivusto-osassa ”lisää plugin”-painikkeella. Käyttäjä valitsee haluamansa lisäosan ja haluamansa käyttöliittymä vaihtoehdon. Blogi-lisäosaa lisättäessä käyttäjä voi valita kuinka monta blogia tulee Internet-sivuille esille.

Tästä opinnäytetyöstä on rajattu testaus ja Internet-sivujen käyttöliittymäsuunnittelu kokonaan pois. Faarao Oy yrityksen graafikot voivat muokata myöhemmin käyttöliittymätiedostot haluamaansa muotoon. Myös Blogi-lisäosan hallintopuolen käyttöliittymää ei käsitellä tässä opinnäytetyössä ollenkaan. Ramses CMS -järjestelmä sisältää useita lisäosia, joten Blogi-lisäosan hallintopuolen käyttöliittymä tehdään samalla tyylillä kuin muut lisäosat.

3 MVC-ARKKITEHTUURI

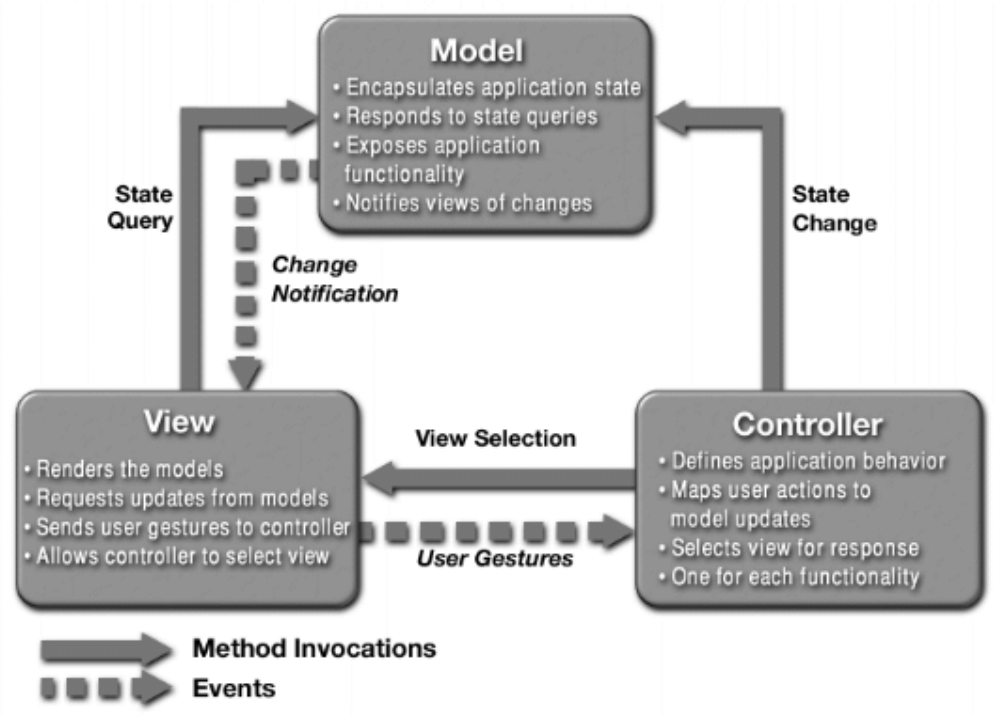
Usein sana arkkitehtuuri liitetään fyysisiin rakennuksiin. Sovelluksen arkkitehtuuri voidaan ajatella samalla tavalla kuin minkä tahansa normaalin rakennuksen arkkitehtuuri. Mitään rakennusta ei rakennettaisi ilman arkkitehtuurisuunnittelua. Sovellusten arkkitehtuureissa pätee sama asia. Hyvä arkkitehtuuri ottaa huomioon ohjelmiston käytettävyyden, muunneltavuuden, turvallisuuden, kannettavuuden ja samanaikaisuuden. Pienet Web sovellukset voivat toimia ilman arkkitehtuurimallia, mutta kun on kyse isommasta projektista, tulee koodista hyvin epäselvä ilman mallia. (Bass, Clements & Kazman 1998, 27.)

Ohjelmistoarkkitehtuuri ei ainoastaan jaa järjestelmän osia, vaan se kuvaa myös näiden osien suhteita ja niiden kehittymistä. Suhteet ovat yleensä ajonaikaisia, joten arkkitehtuuri sisältää myös järjestelmän rakenteen ja käyttäytymisen. Arkkitehtuuri sisältää myös ohjelman suorituksen aikaisia rakenteita, ei pelkästään koodin rakennetta. (Koskimies & Mikkonen 2005, 18.) Koskimiehen ja Mikkosen mukaan (2005, 17) arkkitehtuurin voidaan ajatella olevan järjestelmän perustuslaki. Sitä ei saa muuttaa, ainoastaan vain erittäin painavilla perusteilla. Sitä on myös noudatettava, kun järjestelmää rakennetaan.

MVC-arkkitehtuurimalli pohjautuu olio-ohjelmoinnin mukaiseen ajattelutapaan. Siinä sovelluslogiikka erotetaan käyttöliittymästä, minkä johdosta ohjelmiston kehitys ja ylläpito yksinkertaistuvat. (McArthur 2008, 201.)

MVC-arkkitehtuuri sisältää kolme komponenttia: mallin, näkymän sekä ohjaimen. Malli sisältää järjestelmän tiedot. Se sisältää yleensä tietokannan käsittelylauseita. Näkymä on sovelluksen käyttöliittymä. Se lähettää käyttäjän syötteet ohjaimelle sekä sallii ohjaimen valita tietyn näkymän. Ohjain taas yhdistää nämä komponentit. Se määrittää ohjelman kulun, valitsee tietyn näkymän ja välittää tietoa mallille. Kuten kuviosta 1 huomataan, näkymässä annetut käyttäjän syötteet kulkeutuvat ohjaimelle. Ohjain ottaa vastaan käyttäjän syötteen ja välittää sen mallille. Malli vastaanottaa tiedot ja tekee mahdolliset muutokset, esimerkiksi sovelluksen tietokantaan. Ohjain jatkaa ohjelman kulkua ja antaa

tiedot suoritusten onnistumisesta näkymälle. (Sun Microsystems 2000, hakupäivä 17.03.2011.)



KUVIO 1. MVC-arkkitehtuurin komponentit ja niiden suhteet (Sun Microsystems 2000, hakupäivä 17.03.2011).

MVC-arkkitehtuurimallia noudattavan sovelluksen tiedostorakenne on selkä. Malli-, näkymä- ja ohjaintiedostot tallennetaan omiin kansioihinsa, josta ne ovat helposti löydettävissä ja päivitettävissä. (Sun Microsystems 2000, hakupäivä 17.03.2011.) Koskimiehen ja Mikkosen mukaan (2005, 124) MVC-arkkitehtuurin mukaiset järjestelmät on helppo siirtää toiselle graafiselle alustalle.

Malli-komponentti sisältää usein tietokantakyselyitä. Näitä kyselyitä voidaan isossa projektissa käyttää uudelleen helposti, mikä vähentää koodin määrää huomattavasti. (Sun Microsystems 2000, hakupäivä 17.03.2011.) Samoja komponentteja voidaan kutsua eri käyttöliittymiltä, koska malli palauttaa tiedon ilman alustamista. Yleensä

käyttöliittymätiedostot ovat HTML-tiedostoja, mutta ne voivat olla myös muita tiedostoja kuten esimerkiksi Macromedia Flash -tiedostoja. (Kotek 2002, hakupäivä 26.04.2011.)

MVC-arkkitehtuuri on erinomainen arkkitehtuurimalli, kun ohjelmisto halutaan kehittää projektimaisesti. Eri asiantuntijat voivat vastata eri osa-alueista. Graafikot ja käyttöliittymäsuunnittelijat voivat vastata näkymä-tiedostojen kehittämisestä. Tietokantasuunnittelijat voivat keskittyä mallin kehittämiseen. Ohjaimen voivat suunnitella ohjelmistokehittäjät sekä järjestelmäsuunnittelijat. (McArthur 2008, 202.)

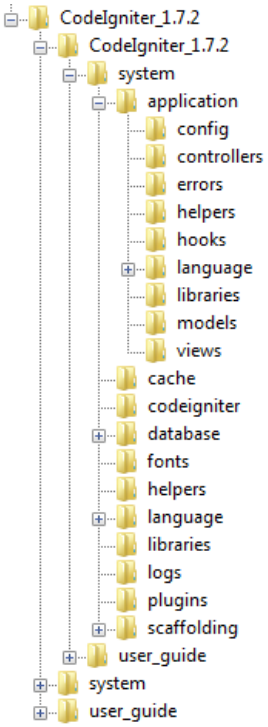
MVC-arkkitehtuuriin liittyy myös ongelmia. Koskimiehen ja Mikkosen mukaan (2005, 144) MVC-arkkitehtuurissa näkymä- ja ohjainluokat ovat toisiinsa yhteydessä niin kiinteästi, että niitä voi olla hankala käyttää uudelleen, jos ne ovat irrallaan toisistaan. MVC-arkkitehtuurin käyttäminen ei ole aina helppoa ja sen käyttäminen edellyttää huolellista suunnittelua. MVC-arkkitehtuuria käytettäessä täytyy miettiä, miten eri osat ovat vuorovaikutuksissa toisiinsa. MVC-arkkitehtuurin käyttäminen pienissä tai jopa keskisuurissa sovelluksissa voi olla turhaa, koska MVC-arkkitehtuuria käyttäessä tarvitsee hallita enemmän tiedostoja, kuin pienen sovelluksen kehittämiseen oikeasti tarvittaisiin. (Kotek 2002, hakupäivä 26.04.2011.) Toisaalta, jos sovellusta halutaan kehittää myöhemmin, olisi se hyvä jo alusta asti kehittää jonkin arkkitehtuurin mukaisesti.

4 CODEIGNITER

CodeIgniter on avoimen lähdekoodin ohjelmisto dynaamisten Web-sovellusten kehittämiseen, joka pohjautuu MVC-arkkitehtuuriin. Se suosii yksinkertaisia ratkaisuja ja sen avulla voidaan välttää monimutkaisuutta. CodeIgniter on selkeästi ja perusteellisesti dokumentoitu. Järjestelmä vaatii vain erittäin pieniä kirjastoja, joten se on hyvin kevyt ohjelmisto, jonka vuoksi myös erittäin suorituskykyinen. Muut tarvittavat kirjastot ladataan dynaamisesti käyttäjän pyynnöstä. (EllisLab 2011, hakupäivä 17.03.2011.)

4.1 CodeIgniter-ohjelman rakenne ja toiminta

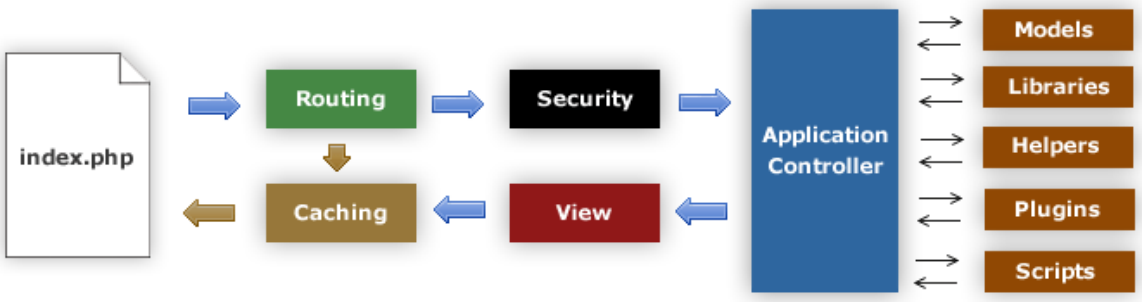
CodeIgniter voidaan ladata ilmaiseksi CodeIgniter-ohjelman omalta kotisivultaan. Ohjelma ladataan ZIP-pakettina, joka puretaan palvelimelle. Tällöin saadaan kuviossa 2 esitetyn mukainen kansiorakenne.



KUVIO 2. CodeIngiter-ohjelman kansiorakenne

Itse CodeIgniter-ohjelma sijaitsee system-kansion sisällä. Config-kansion sisältä löytyy esimerkiksi config.php, autoload.php sekä database.php -tiedostot. Esimerkkejä config-tiedostoista löytyy liitteestä 1. Config-kansion tiedostot ovat hyvin kommentoitu, joten niitä on erittäin helppo muokata omaan käyttöön sopivaksi. Config-tiedostossa määritellään ohjelmiston ympäristömuuttujat. Määriteltäviä asioita ovat index-sivun, oletuskielen sekä aikamuotojen määrittäminen. Autoload-tiedostossa määritellään, mitkä avustajat, luokat ja kielet alustetaan heti, kun sovellus käynnistetään. Database-tiedostossa määritellään tietokantayhteydet. Kun kaikki tarvittavat asetukset on tallennettu config-kansioon, ohjelmointi on nopeampaa, koska tarvittavia kirjastoja ja avustajia ei tarvitse erikseen enää ladata, eikä tietokantayhteyksistä ei tarvitse enää huolehtia, kun ne on kerran asennettu oikein.

Kuviosta 3 nähdään kuinka CodeIgniter käsittelee HTTP-pyyntöjä ja miten eri osat kommunikoivat toistensa kanssa. CodeIgniter-ohjelmassa täytyy olla index.php tiedosto määriteltynä, jota tarvitaan CodeIgniter-ohjelman ajamiseen. Router eli reititin tutkii HTTP-pyyntöä ja välittää sen eteenpäin oikeaan ohjaimeen. Jos Cache eli välimuistin tiedosto on olemassa, se lähetetään suoraan selaimeen, eikä sivua tarvitse rakentaa ohjaimen, mallin ja näkymän avulla dynaamisesti. HTTP-pyynnöt ja käyttäjän syötteet suodatetaan aina turvallisuuden vuoksi ennen kuin sovelluksen ohjain on ladattu. Ohjain lataa mallin, ydinkirjastot, avustajat sekä muut tarvittavat resurssit käyttäjän pyynnöstä. Tämän jälkeen luodaan näkymä, joka lähetetään selaimelle. Näkymä tallennetaan välimuistiin, jos sen käyttö on sallittu. Tällöin toistuvat pyynnöt voidaan ladata suoraan välimuistista. (EllisLab 2011, hakupäivä 17.03.2011.)



KUVIO 3. CodeIngiter-ohjelman tietovirtaus (EllisLab 2011, hakupäivä 17.03.2011).

Tavallisissa PHP-sovelluksissa kaikki pyynnöt tulevat tietylle tiedostolle. Selaimen osoitteeksi tulee näissä sovelluksissa /polku/tiedostolle/tiedosto.php. CodeIgniter-ohjelman MVC-arkkitehtuurin mukaiset sivut ovat normaalisti toteutettu keskitetylle PHP-skriptille, joka käsittelee kaikki Internet-sivuston pyynnöt. Tällöin selaimen osoitteeksi tulee /controllerin_nimi/funktion_nimi. (EllisLab 2011, hakupäivä 17.03.2011.) Kuviossa 5 on esitetty esimerkki URL-osoitteesta.

```
<?php
class Blog extends CI_Controller {
           public function index()
           {
                      echo 'Hello World!';
           }
           public function comments()
           {
                      echo 'Look at this!';
           }
}
?>
```

KUVIO 4. CodeIngiter-ohjelman ohjain-esimerkki (EllisLab 2011, hakupäivä 26.04.2011).

Esimerkiksi, ohjelman ohjaimeen (katso kuvio 4.) on tehty luokka nimeltä blog, jossa on funktio nimeltä comments. Kun käyttäjä haluaa tulostaa ”Hello World”-sanoman, hän syöttää URL-osoitteeksi kuvion 5 mukaisen osoitteen. Jos käyttäjä haluaa taas tulostaa ”Look at this”-sanoman, hän syöttää URL-osoitteeksi kuvion 6 mukaisen osoitteen.

```
example.com/index.php/blog/index
```

KUVIO 5. URL-osoitteen esimerkki (EllisLab 2011, hakupäivä 26.04.2011).

```
example.com/index.php/blog/comments
```

KUVIO 6. URL-osoitteen esimerkki (EllisLab 2011, hakupäivä 26.04.2011).

4.2 Luokkakirjasto

CodeIgniter sisältää useita luokkia, joita voidaan ladata ohjelman eri osiin. Form validation -luokka on lomakkeiden validointia varten. HTML-lomakkeen tietoturva-asioissa epäonnistutaan monesti ja se vaatii myös huomattavan määrän koodia. Form validation -luokan avulla käyttäjän syötteen tarkistaminen on helppoa ja nopeaa ja se vähentää huomattavasti koodin määrää. Esimerkiksi, jos käyttäjä syöttää käyttäjätunnuksen, sille voidaan antaa Form validation -luokan avulla määrittelyt, että käyttäjätunnus voi olla tietyn mittainen tai se sallii vain joitain tiettyjä merkkejä. Se voi myös muotoilla käyttäjän syötteitä. (EllisLab 2011, hakupäivä 17.03.2011.)

```
$this->form_validation->set_rules('username', 'Username', 'required|
min_length[4]');
$this->form_validation->set_rules('password', 'Password', 'required| alpha');
$this->form_validation->set_rules('email',    'Email',    'required|
valid_email');
$this->form_validation->set_rules('url',      'Url',
'valid_ip|prep_url');
```

KUVIO 7. Form validation -luokan syötetarkistus esimerkki

Ohjelman ohjaimessa määritellään muuttujat, joihin halutaan tehdä syötetarkistus ja parametreihin annetaan haluttu syötetarkistus. Kuviossa 7 muuttujalle username on annettu kaksi syötetarkistusta. Ensimmäisenä ”required”, joka palauttaa virheen jos annettu käyttäjätunnus on tyhjä. Seuraavana on min_leght[4], joka palauttaa virheen, jos käyttäjätunnus on merkkimäärältään lyhempi kuin neljä. Password-muuttujalle on annettu syötetarkistus ”alpha”. Jos password-muuttuja sisältää muita merkkejä kuin kirjaimia, se palauttaa virheen. Email-muuttuja taas sisältää syötetarkistuksen ”valid_email”, joka palauttaa virheen, jos email-muuttuja ei sisällä validia sähköpostiosoitetta. URL-muuttujan syötetarkistus tarkistaa, että URL-osoite on validi. ”Prep_url”-tarkistus muokkaa käyttäjän syötettä. Jos käyttäjä ei ole syöttänyt osoitteen alkuun ”http://”, ”prep_url”-tarkistus, lisää sen automaattisesti. Käyttäjän syöttäessä virheellistä tietoa annetaan siitä ilmoitus, ja

käyttäjä ohjataan ohjaimessa määrättyyn näkymään vasta kun kaikki syötteet on annettu oikein. (EllisLab 2011, hakupäivä 17.03.2011.)

CodeIgniter-ohjelman Active Record -luokka on varsin käytännöllinen tietokantaa käsittelevien lauseiden muodostamiseen. Tämä malli mahdollistaa tiedon hakemisen, lisäämisen ja päivittämisen tietokantaan pienellä koodimäärällä. Suurin hyöty käyttämällä Active Record -luokkaa on se, että sen avulla voidaan luoda kyselyitä tietokannasta riippumatta, koska kyselyn syntaksi syntyy tietokantakohtaisen sovittimen avulla. Active Record -luokkaa käyttäessä, tietokantaan lisättävä, muokattava tai poistettava tieto on tallennettava ensin taulukko- tai objektimuotoon. Tämän jälkeen kyseinen taulukko tai objekti voidaan syöttää yhdellä lauseella tietokannalle. Kuviossa 8 esitetään esimerkki tiedon lisäämisestä tietokantaan. (EllisLab 2011, hakupäivä 17.03.2011.)

```
$data = array(
   'user_name' => 'john' ,
   'password'  => 'test' ,
   'email'     => 'john@email.com'
);
$this->db->insert('mytable', $data);
```

KUVIO 8. Esimerkki tiedon lisäämisestä tietokantaan (EllisLab 2011, hakupäivä 17.03.2011).

4.3 Avustajat

Avustajat ovat nimensä mukaisesti koodin ohjelmoinnin apuna. CodeIgniter sisältää useita avustajia; URL-avustaja auttaa linkkien tekemisissä, Form-avustaja lomakkeiden luonnissa, Cookie-avustaja asettaa ja lukee evästeitä, File-avustaja auttaa tiedostojen käsittelyssä. CodeIgniter-ohjelmassa avustajat eivät ole kirjoitettu olio muodossa, toisin kuin useimmissa sovelluksissa. Ne ovat yksinkertaisia, staattisia metodeja. Jokainen avustaja suorittaa yhtä tiettyä tehtävää, eivätkä ne ole riippuvaisia muista funktioista. (EllisLab 2011, hakupäivä 07.04.2011.)

CodeIgniter-ohjemisto käyttää “Auto-load”-ominaisuutta, joka lataa käyttäjän haluamat kirjastot, avustajat ja mallit aina, kun ohjelma on käynnissä. Käyttääkseen tätä

ominaisuutta, käyttäjän täytyy lisätä haluamansa kirjastot avustaja mallit CodeIgniter-ohjelman application/config/autoload.php-tiedostoon. (EllisLab 2011, hakupäivä 26.04.2011.)

Array-avustaja nimensä mukaisesti sisältää funktioita, jotka ovat yhteydessä taulukkoihin. Tämän opinnäytetyön toteutukseen Array-avustaja helpottaa ohjelma koodin ohjelmointia, kun halutaan lähettää ohjaimesta käyttäjän syöttämät tiedot mallille taulukkomuotoisena. Active Record -luokka tarvitsee myös tiedon taulukkomuotoisena tietokantaan lisäystä, muokkausta taikka poistoa varten. Taulukon syntaksi on esitetty kuviossa 8. (EllisLab 2011, hakupäivä 07.04.2011.)

Form-avustaja sisältää funktioita, jotka ovat yhteydessä lomakkeen luonnissa. Ensimmäiseksi lomaketta tehdessä se täytyy avata. Se avaa suoraan sille ohjaimen funktiolle lomakkeen, jonka nimi lomakkeen avauksessa mainitaan. Form-avustajaa käyttäessä lomake ohjelmoidaan PHP-kielellä. Form-avustajan avulla lomakkeen avaaminen on esitetty kuviossa 9. (EllisLab 2011, hakupäivä 07.04.2011.)

```
echo form_open('name_of_the_controller/name_of_the_function');
```

KUVIO 9. Esimerkki lomakkeen avaamisesta PHP-kielellä

Form-avustaja sisältää kaikki lomakkeen elementin sisällä tarvittavat syöte-elementit, kuten esimerkiksi: tekstikentän, pudotuslistan, valintanapit, valintaruudut sekä painikkeet. Kuviossa 10 on esitetty tavallisen tekstikentän tekeminen HTML-kielellä. Kuviossa 11 on taas esitetty kuinka sama tekstikenttä tuotettaisiin Form-avustajia käyttäen. Aina Form-avustajaa käyttäessä, tiedot ohjelmoidaan ensin taulukko muotoon, jonka jälkeen taulukko tulostetaan Form-avustajan funktioiden avulla. (EllisLab 2011, hakupäivä 07.04.2011.)

```
<input type="text" name="username" id="username" value="john"
maxlength="50" size="50" style="width:50%" />
```

KUVIO 10. Esimerkki tekstikentän luomisesta HTML-kieltä käyttäen

```
$data = array(
              'name'        => 'username',
              'id'          => 'username',
              'value'       => 'john',
              'maxlength'   => '50',
              'size'        => '50',
              'style'       => 'width:50%',
            );
echo form_input($data);
```

KUVIO 11. Esimerkki tekstikentän luomisesta PHP-kielellä Form-avustajaa käyttäen

URL-avustaja avustaa nimensä mukaisesti URL-osoitteissa. URL-avustajalla voidaan helposti muokkailla URL-osoitteita sekä ottaa tietoa niistä. URL-avustajalla on useita eri funktioita, joita ovat muun muassa uri_string() ja url_title(). Uri-string()-funktio palauttaa URL-osoitteen osina. Jos selaimella on osoite “http://www.example.com/index.php/blog/edit_blog/15”, uri-string avustaja palauttaisi tiedon String-tyypisenä ”/blog/comments/123”. Url_title()-funktion avulla, voidaan muokata tekstiä, jotta se kävisi muodoltaan URL-osoitteelle. Esimerkiksi erikoismerkit eivät ole sallittuja merkkejä URL-osoiterivillä. Jos käyttäjällä on String-tyyppinen $title-niminen muuttuja, joka sisältää "What's that blog?"-tekstin, url_title()-funktion palauttaisi url_title($title)-funktiolla ”Whats-that-blog”, jolloin tätä tekstiä voitaisiin käyttää URL-osoitteessa. (EllisLab 2011, hakupäivä 20.4.2011.)

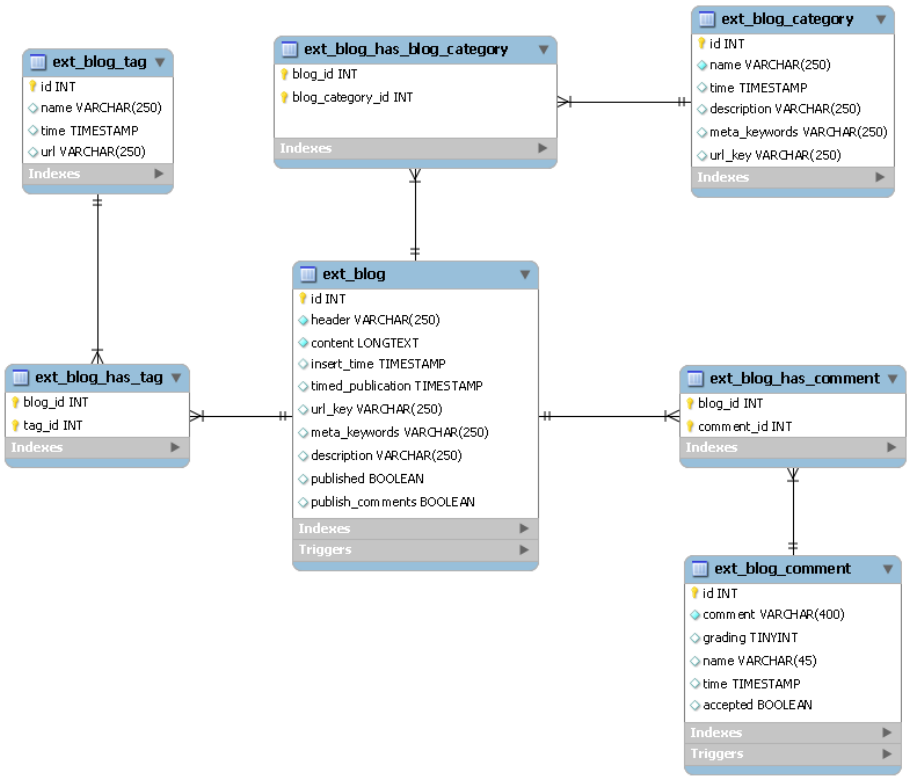
5 BLOGI-LISÄOSAN KEHITTÄMINEN

CodeIgniter sisältää siis Controller-, Model-, View- ja Extension-kansiot . Kansiorakenne kokonaisuudessaan on esitelty kuviossa 2. Blog-niminen luokka toimii blogi-lisäosan ohjaimena. Blogi-lisäosan osoitteeksi on asetettu ”cms5.ramses.fi/blog”. Ohjaimen nimi täytyy olla Blog, muuten CodeIgniter ei löytäisi sitä. Blog_model-luokka toimii blogin mallina. Näkymätiedostoja on monta. Ne ovat käyttöliittymätiedostoja blogin hallinto-osiolle. Ohjain kutsuu aina tarvittaessa Blog_model-luokasta tarvittavia funktioita ja lähettää tiedot käyttöliittymätiedostoille. Extension-luokassa on Blog_ext-tiedosto, joka Internet-sivujen käyttöliittymää varten.

5.1 Blog-lisäosan tietokanta

Blogi-lisäosaa varten täytyi luoda seitsemän uutta taulua Ramses CMS -järjestelmään. Taulujen luontilauseet löytyvät liitteestä 2. Lisäosaan tarvittiin taulut blogien, blogien kategorioiden, kommenttien ja avainsanojen tallentamista, muokkausta ja poistamista varten. Blog-taulu on päätaulu, joka on yhteydessä kategoria, kommentti ja avainsana taulujen kanssa. Kaikissa tauluissa on myös aika-kenttä, joka on TIMESTAMP-muotoa. Tätä hyödynnetään ainoastaan blog-taulussa, mutta ne on lisätty jokaiseen tauluun mahdollista myöhempää käyttöä varten toimeksiantajan pyynnöstä. Aikaleimaan tallentuu tietueen päivittämisen yhteydessä sen hetkinen päivämäärä ja kellonaika. Tietokantaan ohjelmoitiin myös Triggereitä, mutta toteutuksen aikana huomattiin, että Ramses CMS -järjestelmässä käyttäjällä olisi pitänyt olla enemmän oikeuksia näiden käyttöön, joten ne jätettiin kokonaan pois tästä työstä. Triggerit oli tehty kommenttien arvostelemista sekä ajastettua julkaisua varten. Jos käyttäjä antoi kommentin ilman arvosanaa, järjestelmä olisi lisännyt automaattisesti jonkin arvosanan tietokantaan. Tämä ei ollut tärkeä ominaisuus, joten se jätettiin kehittämättä kokonaan. Jos käyttäjä lisäsi blogin ja jätti ajastetun julkaisun tyhjäksi, järjestelmä olisi tallentanut triggerin avulla ajastettuun julkaisu -tietueeseen sen hetkisen päivämäärän ja kellonajan. Tämä voitiin kuitenkin toteuttaa helpommin käyttämällä aiemmin mainittua TIMESTAMP-ominaisuutta.

Blogil-lisäosaan tarvittavat taulut ja niiden attribuutit näkyvät kuviossa 12. Taulujenyhteyksiin on jouduttu tekemään aina yksi ylimääräinen taulu, joka yhdistää kaksi taulua. Kuten kuviosta 12 nähdään, ext_blog-taulu yhdistyy kolmeen eri tauluun ja jokaiselle niistä on täytynyt tehdä yksi ylimääräinen taulu, joka yhdistää ne.



KUVIO 12. Blogi-lisäosan tietokanta

5.1 Controller

Controller eli ohjain on vastuussa blogin ylläpidon käytettävissä olevien komentojen toiminnallisuudesta. Ohjaimessa on luokka nimeltä Blog.php. Luokalla täytyy olla aina alustaja. Alustajassa ladataan kaikki mallit, avustajat, kirjastot ja kielitiedostot mitä kyseinen luokka tulee käyttämään. Kyseiset tiedostot olisi myös voitu alustaa CodeIgniter-ohjelman autoload-luokassa, mutta tässä työssä tiedostot on alustettu ohjaimen alussa,

toimeksiantajan tapaisesti. Blog-luokassa käytetään Blog_model-mallia sekä array- ja form_validation-avustajia.

Blog-luokassa on funktio index(), jota ohjelma kutsuu aina ensimmäiseksi. Index() -funktiossa kutsutaan malli tiedostosta funktiota joka palauttaa kaikki blogit, mitä tietokantaan on tallennettu. Ohjain vastaanottaa tiedot ja lähettää ne näkymään. Liitteessä 3 on esitetty index()-funktio kokonaisuudessaan. Ohjain siis kutsuu aina tarvittavia funktioita käyttäjän toimintojen perusteella. Tärkeimpiä funktioita tässä työssä oli add_blog()-, edit_blog()- sekä delete_blog()-funktiot, blog-lisäosan perustoimintoja varten. Ohjaimeen ohjelmoitiin myös timed_publication()-funktio (katso liite 4.), jolla tarkistettiin, että ajastetun julkaisun päivämäärä ei ole menneisyydessä. Tälle tarkastukselle tehtiin oma funktio, koska tarkastusta jouduttiin tekemään useampaan kertaan ohjaimen eriosissa.

5.2 Model

Model eli malli-komponentissa sijaitsee kaikki tietokantakyselyt Ramses CMS -järjestelmän tietokantaan. Toimeksiantajalla on konfiguraatio-tiedostossa kaikki tietokantayhteydet jo valmiina. Mallissa käytetään CodeIgniter-ohjelman Active Record -luokkaa.

Kun käyttäjä avaa Ramses CMS -järjestelmän blogi hallinto-osion, ohjain kutsuu heti index()-funktiota. Tässä funktiossa kutsutaan funktiota get_all() Blog_model-mallista. Tämä funktio löytyy kokonaisuudessaan liitteestä 5. Get_all()-funktiolle annetaan kutsussa parametrina taulun nimi, jota tarvitaan funktion tietokantakyselyssä. Funktio tekee tietokantakyselyn Ramses CMS -järjestelmän tietokantaan parametrina olevan taulun perusteella. Tietokantakysely tehdään Active Record -luokan avulla. Jos tietokantakyselyn tulos ei ole tyhjä, se muutetaan taulukko-muotoon result_array()-funktiolla ja se tallennetaan $query-nimiseen muuttujan. Tämä funktio on CodeIgniter-ohjelman tietokanta-luokassa määriteltynä valmiina. Jos tietokantakyselyn tulos on tyhjä, funktio palauttaa FALSEN ohjaimelle. Muutoin muuttuja $query palautetaan ohjaimelle. Ohjain vastaanottaa tiedot $blogs-muuttujaan ja lähettää uudestaan blog_model-mallille. Tällä

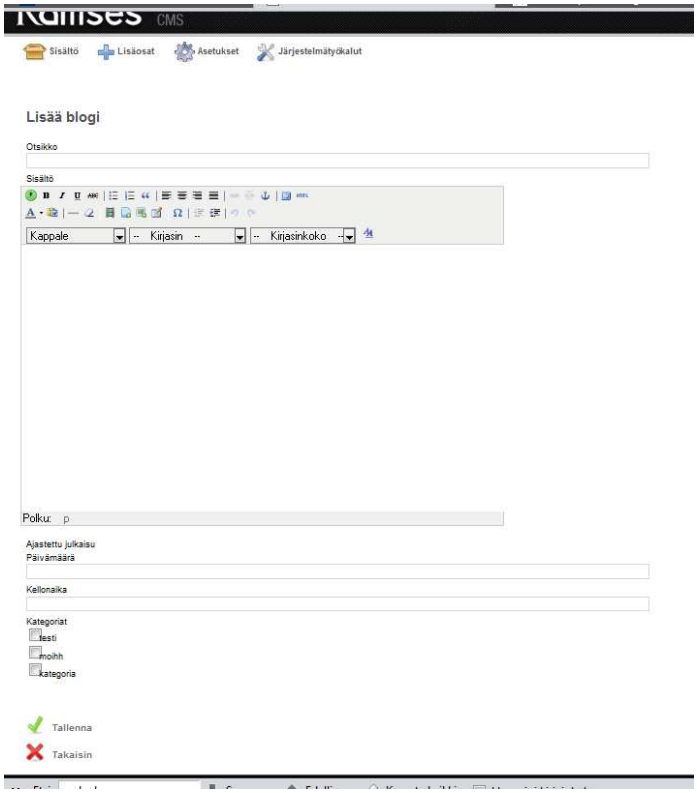
kertaa kutsutaan funktiota make_file_table(). Parametrina annetaan tyyppi eli blog, $blogs-muuttuja, ohjaimen nimi- ja otsikkotiedot. Blog_model-malli tekee foreach-funktiolla Ramses CMS -järjestelmään sopivan taulukon ja palauttaa sen ohjaimelle $blogs_table -muuttujaan. Ohjain lähettää tämän muuttujan näkymälle.

5.3 View

View eli näkymä sisältää kaikki blogin ylläpidon käyttöliittymätiedostot. Ohjain tallentaa kaikki näkymälle tarvittavat muuttujat yhteen taulukko-muotoiseen muuttujaan $data. Tämä muuttuja lähetetään näkymään, joka on määritelty ohjaimessa. Liite 6 sisältää $data-muuttujan tiedot ja sen lähetyksen näkymälle. Tämä näkymä-tiedosto löytyy kokonaisuudessaan liitteestä 6. Nyt käyttäjä näkee kuvion 13 mukaisen käyttöliittymän Ramses CMS -järjestelmässä. Kategoriat, avainsanat ja kommentit ovat listattuna hallinto-osiossa samalla tavalla kuin blogit kuvion 13 mukaisesti.

KUVIO 13. Blogi-lisäosan hallinto-osion etusivu näkymä

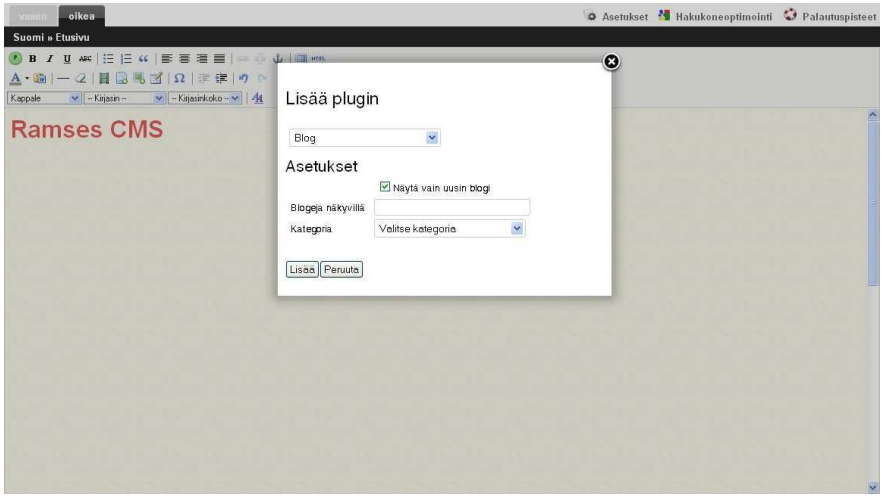
Kuviossa 14 on esitetty käyttöliittymä blogin lisäämistä varten. Samantapaista käyttöliittymää käytetään myös blogin muokkaamiseen, kategorioiden, avainsanojen sekä kommenttien lisäämiseen ja muokkaamiseen.



KUVIO 14. Lisää blogi -käyttöliittymä

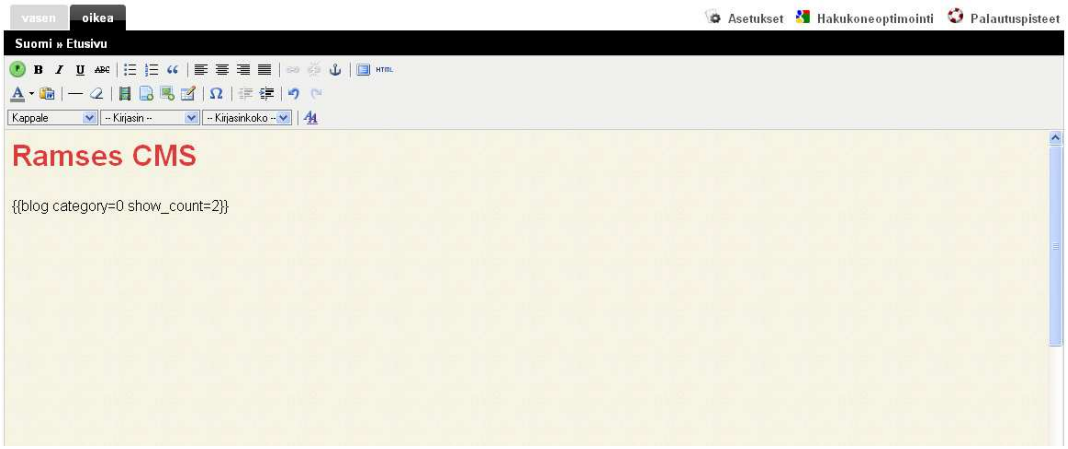
5.4 Extension

Kun blogi-lisäosa halutaan lisätä jollekin Internet-sivulle Ramses CMS-järjestelmässä, käyttäjä lisää lisäosan klikkaamalla ”Lisää plugin” -painiketta järjestelmän sisältö-osiossa. Käyttäjälle avautuu lomake, joka on esitetty kuviossa 15. Tällä lomakkeella valitaan, millainen näkymä blogi-lisäosasta halutaan asiakkaan Internet-sivuille. Esimerkiksi käyttäjä voi valita, että näytetään vain uusin blogi, jolloin sen valintaruutu valitaan lomakkeelta. Tällöin Internet-sivuille ilmestyy uusin blogi ja sen kommentit. Kuviossa 15 valitaan kaksi uusinta blogia. Tekstikenttään ”Blogeja näkyvillä” on siis syötetty numero 2. Internet-sivulle ilmestyy tällöin kaksi uusinta blogia.



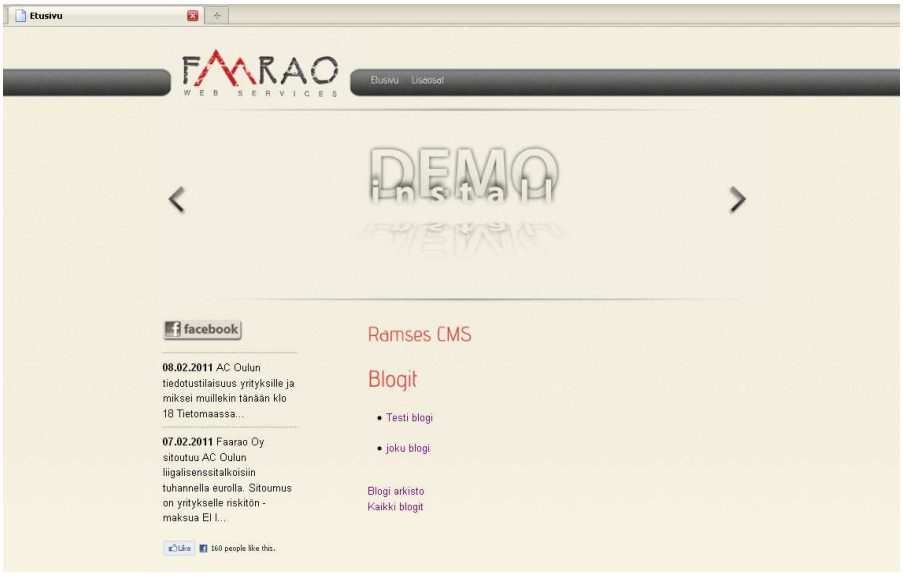
KUVIO 15. Lisää Blog-lisäosa

Kun haluttu näkymä on valittu, Ramses CMS -järjestelmän sivulle tulee teksti, joka sisältää halutun lisäosan nimen sekä parametrit, joiden mukaan saadaan haluttu näkymä Internet-sivuille. Kuviossa 16 nähdään nyt lisäosan nimi sekä parametrit. Kuviossa 15 valittiin, että nähdään kaksi uusinta blogia, nyt tämä tieto tulee parametriksi nimellä show_count.



KUVIO 16. Lisää Blog -lisäosa parametreina

Kun käyttäjä nyt tallentaa tämän sivun, kyseiselle Internet-sivulle ilmestyy kaksi uusinta blogia. Kuten kuviosta 17 huomataan, tässä esimerkissä nähdään ainoastaan blogin otsikot, joita klikkaamalla nähdään itse blogi-kirjoitus.



KUVIO 17. Kaksi uusinta blogia Internet-sivulla

Blogi-lisäosan lisäämiseen Internet-sivulle tarvitaan Extension-komponenttia CodeIgniter-ohjelmasta. Extension-komponentti sisältää Blog_ext-luokan. Käyttäjä valitsee siis ”Lisää plugin”-painikkeelta (katso kuvio 15) haluamansa lisäosan sekä valitsee haluamansa näkymän. Tällä toiminnolla saadaan tiedot Blog_ext luokkaan. Tässä esimerkissä parametreissa on tällä hetkellä ”show_count = 2”. Luokka käsittelee ensin render()-funktion. Tämä tarkistaa, mitä käyttäjän antama parametri sisältää. Tässä tapauksessa se sisältää tietoa show_count muuttujassa (kuvio 18). Kun render()-funktio on löytänyt tiedon, se palauttaa show_count muuttujan blogs_by_insert()-funktiolle.

```
public function render()
{
     if (!empty($this->show_count)) {
     return $this->blogs_by_insert($this->show_count);
     }
```

KUVIO 18. Kuvaus render()-funktiosta

Nyt ohjelma menee blogs_by_insert()-funktioon, jossa parametrina on käyttäjän antama show_count-muuttujan arvo. Blogs_by_insert()-funktio löytyy kokonaisuudessaan liitteessä 7. Kyseisessä funktiossa on ensimmäisenä metodikutsu sovelluksen Blog_model-luokkaan. Metodissa blogs_by_insert_time($blog_numbers) haetaan tietokannasta

Blog_ext luokasta tässä tapauksessa kaksi uusinta blogia. Tämä funktio löytyy kokonaisuudessaan liitteessä 8.

Nyt kaksi uusinta blogia on $blogs-muuttujassa. Tämän jälkeen kutsutaan käyttöliittymätiedostoa Ramses CMS -järjestelmästä nimeltä blogs.php. Tiedosto löytyy liitteestä 9. Tämä tiedosto on lisäosan käyttöliittymätiedosto Internet-sivulle. Tiedosto on HTML-tyyppinen. Siinä luetaan $blogs-muuttujan tiedot, jolloin kaikki tiedot ovat valmiina ja Internet-sivu näkyy käyttäjälle kuvion 17 mukaisesti.

Toimeksiantajan pyynnöstä työn kehitysvaiheessa blogi-lisäosalle kehitettiin myös arkistointisivu. Arkistointisivulta löytyy kaikki blogit listattuna lisäämispäivämäärän mukaan (katso kuvio 19). Blogit on jaoteltu vuoden mukaan. Tämän toteuttamiseen Blog_ext-luokkaan täytyi tehdä uusi funktio archive(). Archive()-funktiossa kutsutaan funktiota Blog_model-tiedostosta, joka palauttaa kaikki blogit insert_time-muuttujan eli blogin lisäämisen päivämäärän mukaan laskevassa järjestyksessä. Tämän jälkeen käsitellään insert_time-muuttujaa, jotta saadaan kaikki vuodet omaan muuttujaan, jolloin blogeja on lisätty. Nämä vuodet tulevat Internet-sivuille listattuna käyttäjälle, jolloin hän voi vuotta klikkaamalla selata sen vuoden sisällä lisättyjä blogeja. Ominaisuuden kehittämiseen on käytetty Internet-sivujen käyttöliittymässä JavaScript-kieltä. Arkiston käyttöliittymätiedosto löytyy liitteestä 10.

KUVIO 19. Blogin arkistointisivu

Lopuksi vielä ohjelmoitiin kuvion 20 mukainen käyttöliittymä, jossa käyttäjä voi lukea blogeja kategorian tai avainsanojen mukaan. Tämä oli hieman haasteellisempi toteuttaa, koska tässä tarvittiin kahta pluginia samalle sivulle. Oikealle on listattuna kategoriat ja avainsanat, joita klikkaamalla sivun vasemmalla osalle ilmestyy klikatun kategorian tai avainsanan blogit. Kategoriat ja avainsanat ovat linkkejä, joiden osoitteen perään on lisätty kyseisen kategorian tai avainsanan nimi. Tämän vuoksi saadaan oikeat blogit sivun vasemmalle puolelle. Nimessä saattaa olla välilyöntejä tai muita merkkejä, jotka ei ole sallittu URL-osoiterivillä. Tämän vuoksi tässä käytetään CodeIgniter-ohjelman uri-title()-funktiota, joka muokkaa tekstin sallittuun muotoon.

KUVIO 20. Blogit kategorioittain

6 POHDINTA

Blogi-lisäosan kehittäminen Ramses CMS -järjestelmään oli hyvin ajankohtainen sekä mielenkiintoinen aihe. Päätavoitteena oli kehittää toimiva Blog-lisäosa tietyin vaatimuksin, jossa onnistuin hyvin, koska sain kaikki tarvittavat vaatimukset täytettyä. Työ oli suhteellisen helppo aloittaa, koska suoritin ammattiharjoitteluni toimeksiantajan yrityksessä. Tunsin työnohjaajan entuudestaan ja olin tehnyt vastaavia helpompia töitä jo harjoitteluni aikana. Opinnäytetyö tekemistä helpotti se, että tunsin jo entuudestaan Ramses CMS -järjestelmän.

Kehittäessäni blogi-lisäosaa ymmärsin yhä paremmin, kuinka käytännöllinen MVC-arkkitehtuurin mukainen CodeIgniter-ohjelmisto onkaan. Ohjelmisto on ensinnäkin hyvin selkeä, koska kaikki tiedostot ovat omissa kansioissaan. Ramses CMS -järjestelmä sisältää paljon koodia ja ilman MVC-mallin mukaista järjestelyä koodit voisi olla todella sekaisin. Pystyin kutsumaan omaa malli-tiedostoa monta kertaa muualtakin kuin ohjaimesta. Opin, että kaikkea ei todellakaan kannata tehdä uudelleen vaan kannattaisi aina tehdä funktioita, joita voi käyttää uudelleen eri lisäosissa. Itse ohjelmoin kehittämässäni blog_model-mallissa funktiot niin, että parametreissa oli aina taulun nimi. Näin ollen funktiota voidaan käyttää myöhemmin uudestaan eri tauluille, koska taulu määritellään jo funktioin kutsussa, ei itse funktiossa. Itse hyödyin tästä paljon, koska tässä työssä tuli paljon samankaltaisia tietokantakyselyitä. Käytin myös muiden lisäosien malli-tiedostoja mahdollisuuksien mukaan. Koodin testaaminen oli myös helppoa, koska funktiota pystyi testaamaan helposti erikseen.

Työn toiminnallista osuutta oli erittäin mielenkiintoista tehdä. Opin paljon uusia asioita PHP- ja JavaScript-ohjelmoinnista. Opin käyttämään PHP-ohjelmoinnissa käytettäviä valmiita funktioita, joita pysyin hakemaan PHP-manuaalista. Esimerkiki date()-funktioita käytin todella paljon, josta oli paljon esimerkkejä PHP-manuaalin-sivuilla. JavaScript-ohjelmoinnista löytyi myös paljon esimerkkejä jQuery:n Internet-sivuilta. Opin, kuinka erilaisia efektejä voidaan tehdä ja miten tekstikentästä voidaan ottaa tietoa ja tallentaa se muuttujaan JavaScript-kielen avulla. Tätä ominaisuutta käytin työssäni melko paljon. Sain

myös hyvin kokemusta ohjelmistosuunnittelusta. Opin myös tätä opinnäytetyötä kirjoittaessa asiatekstin laatimista. Koska opinnäytetyölläni oli yritys toimeksiantajana, opin myös työelämäkäytäntöjä, joita tuli vastaan työn tekemisen aikana.

Työn suunnittelu sujui melko vaivattomasti. Työn kehittämisen aikana ongelmien ilmaannuttua sain aina apua työnohjaajalta, joten työn tekeminen sujui melko nopeasti. Toimeksiantajalta tuli välillä erilaisia ohjeita, koska työn ohjaajia oli kaksi. Mutta loppujen lopuksi pääsimme aina yhteysymmärrykseen ja muutenkin kommunikointi pienessä yrityksessä sujui erittäin hyvin välillämme. Toimeksiantajalla ei ollut vaatimusmäärittelyä aloittaessani blogi-lisäosan suunnittelua. Vaatimusmäärittely olisi aina hyvä tehdä, koska sen jälkeen on helpompi keskittyä ohjelman kehittämiseen. Työn aikana joutui miettimään esimerkiksi, mitä merkkejä mikäkin muuttuja sai sisältää. Jos nämä olisi valmiiksi mietitty vaatimusmäärittelyyn, työ olisi sujunut vieläkin jouhevammin. Kehittämisen aikana tulleet ideat hidastivat hieman työn tekemistä. Ajattelenkin, että ohjelmistokehityksessä pätee sanonta ”Hyvin suunniteltu on puoleksi tehty”.

Tästä työstä tulee hyötymään toimeksiantajan lisäksi myös minä itse. Toimeksiantaja saa myytävän tuotteen ilmaiseksi. Toimeksiantaja voi myös kehittää entisestään blogi-lisäosaa. Itselleni hyötyä tästä työstä on se, että aihe oli tarpeeksi vaativa ja mielenkiintoinen ja toimeksiantajan yritys on mahdollinen työnantajani lähitulevaisuudessani. Tämän vuoksi oli todella hyvä, että tein ammattiharjoitteluni ja opinnäytetyöni samalle yritykselle. Kyselin jo harjoitteluni aikana mahdollista aihetta opinnäytetyöhön, jotta saisin paremman mahdollisuuden päästä työelämään opiskelujeni jälkeen.

LÄHTEET

Bass, L., Clements, P. & Kazman, R. 1998. Software architecture in practice. Massachusetts : Addison-Wesley.

EllisLab. 2011. Active_record : CodeIgniter User Guide. Hakupäivä 17.03.2011 http://codeigniter.com/user_guide/database/active_record.html.

EllisLab. 2011. Array Helper : CodeIgniter User Guide. Hakupäivä 07.04.2011 http://codeigniter.com/user_guide/database/active_record.html.

EllisLab. 2011. Auto-loading Resources: CodeIgniter User Guide. Hakupäivä 26.04.2011 http://codeigniter.com/user_guide/general/autoloader.html.

EllisLab. 2011. CodeIgniter at a Glance : CodeIgniter User Guide. Hakupäivä 17.03.2011 http://codeigniter.com/user_guide/overview/at_a_glance.html.

EllisLab. 2011. Form Helper : CodeIgniter User Guide. Hakupäivä 07.04.2011 http://codeigniter.com/user_guide/helpers/form_helper.html.

EllisLab. 2011. Form Validation : CodeIgniter User Guide. Hakupäivä 17.03.2011 http://codeigniter.com/user_guide/libraries/form_validation.html.

EllisLab. 2011. Helper functions : CodeIgniter User Guide. Hakupäivä 07.04.2011 http://codeigniter.com/user_guide/general/helpers.html.

EllisLab. 2011. URL Helper : CodeIgniter User Guide. Hakupäivä 26.04.2011 http://codeigniter.com/user_guide/helpers/url_helper.html.

Koskimies, K. & Mikkonen, T. 2005. Ohjelmistoarkkitehtuurit. Helsinki: Talentum.

Kotek, B. 2002. MVC design pattern brings about better organization and code reuse. Hakupäivä 26.04.2011 http://www.techrepublic.com/article/mvc-design-pattern-brings-about-better-organization-and-code-reuse/1049862.

McArthur, K. 2008. Pro PHP: Patterns, Frameworks, Testing and more. Berkeley, CA : Apress.

Sun Microsystems. 2000. Java BluePrints - J2EE Patterns. Hakupäivä 17.03.2011 http://java.sun.com/blueprints/patterns/MVC-detailed.html.

LIITTEET

LIITE 1 – Esimerkkejä Config-tiedostoista

Config.php

```
<? php  if ( ! defined('BASEPATH')) exit('No direct script access allowed');

/*---------------------
 Base Site URL

URL to your CodeIgniter root. Typically this will be your base URL,
WITH a trailing slash:
*/
$config['base_url']          = "http://example.com/";

/*---------------------
 Index File

 Typically this will be your index.php file, unless you've renamed it to
 something else. If you are using mod_rewrite to remove the page set this
 variable so that it is blank.
*/

$config['index_page'] = "index.php";

?>
```

Autoload.php

```
<?php  if ( ! defined('BASEPATH')) exit('No direct script access allowed');

/*---------------------
Auto-load Libraries
These are the classes located in the system/libraries folder
 or in your system/application/libraries folder.
*/
$autoload['libraries'] = array(‘database’, ‘session’);

$autoload['helper'] = array(‘url’,’file’);
?>
```

Database.php

```
<?php  if ( ! defined('BASEPATH')) exit('No direct script access allowed');


$active_group = "default";
$active_record = TRUE;

$db['default']['hostname'] = "localhost";
$db['default']['username'] = "";
$db['default']['password'] = "";
$db['default']['database'] = "";
$db['default']['dbdriver'] = "mysql";
$db['default']['dbprefix'] = "";
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = TRUE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = "";
$db['default']['char_set'] = "utf8";
$db['default']['dbcollat'] = "utf8_general_ci";


/* End of file database.php */
/* Location: ./system/application/config/database.php */
?>
```

LIITE 2 – Taulujen luontilauseet

```
-- -----------------------------------------------------
-- Table `ext_blog`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `ext_blog` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT ,
  `header` VARCHAR(250) NOT NULL ,
  `content` LONGTEXT NOT NULL ,
  `time` TIMESTAMP NULL ,
  `timed_publication` TIMESTAMP NULL ,
  PRIMARY KEY (`id`) )
ENGINE = MyISAM
DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------
-- Table `ext_blog_category`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `ext_blog_category` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT ,
  `name` VARCHAR(250) NOT NULL ,
  `time` TIMESTAMP NULL ,
  PRIMARY KEY (`id`) )
ENGINE = MyISAM
DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------
-- Table `ext_blog_has_blog_category`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `ext_blog_has_blog_category` (
  `blog_id` INT NOT NULL ,
  `blog_category_id` INT NOT NULL ,
  PRIMARY KEY (`blog_id`, `blog_category_id`) ,
  INDEX `fk_blog_has_blog_category_blog_category1` (`blog_category_id` ASC) ,
  CONSTRAINT `fk_blog_has_blog_category_blog`
    FOREIGN KEY (`blog_id` )
    REFERENCES `ext_blog` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_blog_has_blog_category_blog_category1`
    FOREIGN KEY (`blog_category_id` )
    REFERENCES `ext_blog_category` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = MyISAM
DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------
```

```
-- Table `ext_blog_tag`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `ext_blog_tag` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT ,
  `name` VARCHAR(250) NULL ,
  `time` TIMESTAMP NULL ,
  PRIMARY KEY (`id`) )
ENGINE = MyISAM
DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------
-- Table `ext_blog_has_tag`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `ext_blog_has_tag` (
  `blog_id` INT NOT NULL ,
  `tag_id` INT NOT NULL ,
  PRIMARY KEY (`blog_id`, `tag_id`) ,
  INDEX `fk_blog_has_tag_tag1` (`tag_id` ASC) ,
  CONSTRAINT `fk_blog_has_tag_blog1`
    FOREIGN KEY (`blog_id` )
    REFERENCES `ext_blog` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_blog_has_tag_tag1`
    FOREIGN KEY (`tag_id` )
    REFERENCES `ext_blog_tag` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = MyISAM
DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------
-- Table `ext_blog_comment`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `ext_blog_comment` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT ,
  `comment` VARCHAR(400) NOT NULL ,
  `grading` TINYINT NULL ,
  `name` VARCHAR(45) NULL ,
  `time` TIMESTAMP NULL ,
  PRIMARY KEY (`id`) )
ENGINE = MyISAM
DEFAULT CHARACTER SET = utf8;
-- -----------------------------------------------------
-- Table `ext_blog_has_comment`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `ext_blog_has_comment` (
  `blog_id` INT NOT NULL ,
```

```
  `comment_id` INT NOT NULL ,
  PRIMARY KEY (`blog_id`, `comment_id`) ,
  INDEX `fk_blog_has_comment_comment1` (`comment_id` ASC) ,
  CONSTRAINT `fk_blog_has_comment_blog1`
    FOREIGN KEY (`blog_id` )
    REFERENCES `ext_blog` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_blog_has_comment_comment1`
    FOREIGN KEY (`comment_id` )
    REFERENCES `ext_blog_comment` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = MyISAM
DEFAULT CHARACTER SET = utf8;
```

LIITE 3 – Index()-funktio

```
/**
 * Index function
 * @var   array    $blogs              all blogs
 * @var   array    $categories         all categories
 * @var   array    $tags               all tags
 * @var   string   $blog_table         blogs in the correct table
 * @var   string   $categories_table   categories in the correct table
 */
public function index()
{
  $blogs        = $this->Blog_model->get_all('blog');
  $categories = $this->Blog_model->get_all('category');
  $tags         = $this->Blog_model->get_all('tag');



  $blog_table          = $this->Blog_model->make_file_table('blog', $blogs, 'blog',
                           $this>Blog_model->get_headings('blog'));

  $categories_table = $this->Blog_model->make_file_table('category', $categories, 'blog',
                           $this->Blog_model->get_headings('category'));

  $tags_table          = $this->Blog_model->make_file_table('tag', $tags, 'blog',
                          $this>Blog_model->get_headings('tag'));


  $data       = array("blog_table"      => $blog_table,
                      "categories_table"    => $categories_table,
                      "tags_table"          => $tags_table,
                      "category"            => "category",
                      "tag"                 => "tag");

  $data       = array_merge($data,$this->Resource_model->getSiteChanger());

  $this->template->write_view('content', $this->config->item('theme')."/blog", $data);
  $this->template->render();
}
```

LIITE 4 – Timed_publication()-funktio

```
/**
   * Function for checking that the timed publication is valid
   * @param string $time
   * @param string $day
   * @var   date   $timed_publication   timed publication in correct mode for checking the
time that is not in the past
   * @var   date   $current_date        current date from the server
   */
  public function timed_publication($time, $day)
  {
    // TIME
    $time  = $time.':00';
    $dot   = array(":");
    $line  = array("/");
    $time  = str_replace($dot, $line, $time);
    list($hour, $minute, $second) = split('[/.-]', $time);


    // DAY
    $dot   = array(".");
    $line  = array("-");
    $day   = str_replace($dot, $line, $day);


    // Formating the date and time to the correct format
    $date = new DateTime($day);
    $date->setTime($hour, $minute, $second);
    $timed_publication = $date->format('Y-m-d H:i:s') ;


    // Taking the current time
    $current_date = time();
    $current_date = date('Y-m-d H:i:s');


    // Checking if the timed publication is before than the current date, timed publication
date must be bigger than the current date
    if ($current_date > $timed_publication)
      return false;
    else
      return $timed_publication;
  }
```

LIITE 5 – Get_all()-funktio

```
/**
 * Returns all from table
 * @param string $type tablename
 */

public function get_all($type) {

  if ($type == 'category')
  {
    $this->_altdb->select("id, name, time");
    $this->_altdb->order_by("id", "asc");
    $this->_altdb->from("ext_blog_".$type);

    $query      = $this->_altdb->get();
    $query = $query->result_array();
  }

  else if ($type == 'blog')
  {
    $this->_altdb->select("id, header, insert_time, timed_publication");
    $this->_altdb->order_by("id", "asc");
    $this->_altdb->from("ext_blog");

    $query = $this->_altdb->get();
    $query = $query->result_array();
  }

  else if ($type == 'tag')
  {
    $this->_altdb->select("id, name");
    $this->_altdb->order_by("id", "asc");
    $this->_altdb->from("ext_blog_".$type);

    $query  = $this->_altdb->get();
    $query  = $query->result_array();
  }

  else if ($type == 'comment')
  {
    $this->_altdb->select("name");
    $this->_altdb->order_by("id", "asc");
    $this->_altdb->from("ext_blog_".$type);

    $query  = $this->_altdb->get();
    $query  = $query->result_array();
  }
```

```
  // Checking if there is any rows
  if(empty($query)) return false;
     return $query;
}
```

LIITE 6 – Blog.php-käyttöliittymätiedosto

```
<script type="text/javascript">
 $(document).ready(function() {
 });

</script>



<div id="tabs">
  <div id="cntLinks">
    <img src="<?=base_url()?>gfx/icons/newspaper_add.png" />  <a href="<?=
base_url().index_page()."blog/add_blog" ?>"><?= $this->lang>line('add_new_blog')?></a>

    <img src="<?=base_url()?>gfx/icons/newspaper_add.png" />  <a href="<?=
base_url().index_page()."blog/add_category_or_tag/$category" ?>"><?= $this->lang-
>line('add_new_category')?></a>

    <img src="<?=base_url()?>gfx/icons/newspaper_add.png" />  <a href="<?=
base_url().index_page()."blog/add_category_or_tag/$tag" ?>"> <?= $this-
>lang>line('add_new_tag')?></a>

  </div>

  <ul>
    <li><a href="#blogs"><?= $this->lang->line('blogs')?></a></li>
    <li><a href="#categories"><?= $this->lang->line('categories')?></a></li>
    <li><a href="#tags"><?= $this->lang->line('tags')?></a></li>
  </ul>

  <div class="blackBar"></div>
  <div id="blogs"><?= $blog_table ?></div>
  <div id="categories"><?= $categories_table ?></div>
  <div id="tags"><?= $tags_table ?></div>


</div>
```

LIITE 7 – Blogs_by_insert($show_count)-funktio luokassa Blog_ext

```
function blogs_by_insert($show_count)
  {
    $blogs = $this->CI->Blog_model->blogs_by_insert_time($show_count);

    ob_start();
    include $this->views_path . "plugins/blog/" . $this->language . "/blogs.php";
    $rendered = ob_get_clean();
    return $rendered;
  }
```

LIITE 8 – Blogs_by_insert($blog_numbers)-funktio luokassa Blog_model

```
public function blogs_by_insert_time($blog_numbers)
{
  $query = $this->_altdb->query("SELECT *
                          FROM `ext_blog`
                          Order by insert_time DESC
                          LIMIT 0,".$blog_numbers);

  $query = $query->result_array();
  return $query;

}
```

LIITE 9 – Internet-sivun käyttöliittymätiedosto blogs.php

```
<?php
if (!empty($blogs)) {
?>
    <h1>Blogit</h1>
    <ul>
    <?php

        foreach ($blogs as $blog) {

            ?>
                <li><a href="<?= "/fi/blogi/nayta/".$blog['id']."/".$blog['header'] ?>"><?=
$blog['header']?> </a></li>
                <br />
            <?php

        }
    ?>
    </ul>
<?php
}

?>
<a href="<?= "/fi/blogi/uusi" ?>">Uusi blogi</a>
<br />
<a href="<?= "/fi/blogi/arkisto" ?>">Blogi arkisto</a>
<br />
<a href="<?= "/fi/blog" ?>">Kaikki blogit</a>
```

LIITE 10 – Archive.php-käyttöliittymätiedosto

```
<script type="text/javascript">
  $(document).ready(function() {

    $('a').click(function() {
      var value = "#blog_"+$(this).attr("id");
      $('#years').children().hide("slow");
      $(value).show("slow");

    });

    $('[name=month]').change(function () {
          var value = $("#months").val();
          $('[name=select_month]').val(value);
          $("#archive").submit();

    });

  });


</script>

<h2>Blogi arkisto</h2>

<?php
    foreach ($years as $year) {
    ?>
        <a href="#blog_<?=$year?>" id="<?=$year?>"> <?=$year?> </a>
    <?php
    }
?>
<br />
<br />
<div id = "years">
<?php foreach ($years as $year) { ?>
    <div id="blog_<?=$year?>" <?php ($year == date('Y')) ? $value ="" : $value ="none"; ?>
style="display:<?=$value?>" >
    <h3> <?=$year?> </h3>

    <ul>
    <?php
        foreach ($blogs as $blog) {

            if (substr($blog['insert_time'], 0, 4) == $year) {
                $date = new DateTime($blog['insert_time']);
```

```
            $blog_date = $date->format('H:i:s Y-m-d');
            ?>
            <li> <?=$blog['header']; ?>  ( <?= $blog_date; ?> )</li>

        <?php
            }
        }
    ?>
    </ul>

    </div>

<?php
}
?>
</div>


<a href="<?= "/" ?>"> >> Takaisin</a>
```