Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Toni Sissala

Opinnäytetyö

CakePHP-ohjelmointikehyksen käyttö verkkosovelluskehityksessä

Työn ohjaaja Lehtori Harri Hakonen
Työn tilaaja Syrjä Interactive
Huhtikuu 2010 Tampere

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä	Toni Sissala
Työn nimi	CakePHP-ohjelmointikehyksen käyttö verkkosovelluskehityksessä
Sivumäärä	81
Valmistumisaika	Huhtikuu 2010
Työn ohjaaja	Lehtori Harri Hakonen
Työn tilaaja	Syrjä Interactive

TIIVISTELMÄ

Opinnäytetyön toimeksiantaja on tamperelainen www-tuotantoon keskittynyt ohjelmistotuotantoyritys Syrjä Interactive. Työn tarkoituksena on tutkia CakePHP-ohjelmointikehystä, jotta toimeksiantaja saisi kattavamman kuvan käyttämästään ohjelmointikehyksestä. Toimeksiannon mukaisesti opinnäytetyöhön liittyy myös verkkosovellus, jonka toteutuksessa on käytetty tutkittavaa ohjelmointikehystä sekä siihen liittyviä ohjelmointikäytäntöjä.

Työhön liittyy keskeisesti myös MVC(Model-View-Controller)-ohjelmistoarkkitehtuuri, jota CakePHP-ohjelmointikehys käyttää. Työ avaa lukijalle kyseisen arkkitehtuurin tarkoituksen, käytön ja problematiikan periaatteita.

Paras tietolähde aiheeseen perehtymistä ajatellen on Internet. Se tarjoaa kattavaa tietoa ohjelmointikehyksestä ja MVC-arkkitehtuurista. Hyvää, käytettävää ja ajankohtaista tietoa löytyy myös ohjelmistokehittäjien verkkopäiväkirjoista, joissa on usein esitetty jokin ongelma ratkaisuesimerkkeineen. Internetiä lähteenä käytettäessä tulee kuitenkin muistaa olla erityisen kriittinen kaikkea lukemaansa kohtaan.

Perehtyminen CakePHP-ohjelmointikehykseen ja MVC-arkkitehtuuriin vaatii olio-ohjelmointi-, Internet-ohjelmointi-, PHP- ja SQL-kokemusta, niin teorian kuin käytännönkin tasolla. Käsittelen asioita kuitenkin mahdollisimman käytännönläheisesti ja havainnollisesti, jotta lukijan ennakkotiedon määrällä olisi mahdollisimman vähän merkitystä.

Avainsanat mvc, arkkitehtuuri, ohjelmointikehys, kehitystyökalu, verkkosovelluskehitys

Tampere University of Applied Sciences
Business Information Systems

Author	Toni Sissala
Thesis	Producing web applications with CakePHP-development framework
Pages	81
Graduation time	April 2010
Thesis supervisor	Lecturer Harri Hakonen
Co-operating Company	Syrjä Interactive

ABSTRACT

Syrjä Interactive is a programming company, which focuses mainly on web-programming. It is located in Tampere. It acts as the co-operating company for this thesis. The purpose of the thesis is to research CakePHP web application framework, which is used by Syrjä Interactive on a number of their projects. The research is done to give Syrjä Interactive more comprehensive knowledge of the framework. The thesis also includes a web-program developed using the CakePHP framework. It contains programming practices and procedures that are to be used with CakePHP.

MVC(Model-View-Controller)-architecture is crucially important to the thesis, because CakePHP implements it. The thesis gives the reader information about the principles of the architecture's purpose, use and problems.

The best source of information, considering the topic, is the Internet. It offers inclusive information about the framework and MVC-architecture. Information which is good, usable and up-to-date, can also be found from the web-blogs contributed by developers. These often include programming problems that have been solved by the developer. When using Internet as a source of information, one must remember to be particularly critical towards the given information.

Orienting oneself to understand the CakePHP framework and MVC architecture requires knowledge about object-orientated programming, web-programming, PHP programming and SQL commands. The thesis describes the topic with illustrative and hands-on examples. Therefore it is possible to understand the debated topic with as little beforehand knowledge as possible.

Keywords mvc, architecture, development framework, development tool, web application development

Sisällysluettelo

1 Johdanto

Opinnäytetyöni toimeksiantajalla Syrjä Interactivella on muutamissa verkkosovellusprojekteissaan käytössä CakePHP-ohjelmointikehys (development framework). Työni käsittelee kyseistä ohjelmointikehystä yleisellä tasolla, käytännönläheisesti, mutta kuitenkin siten, että työstäni olisi apua myös henkilöille, jotka ovat jo tutustuneet CakePHP-ohjelmointikehykseen käytännössä. En ole esitellyt työssäni CakePHP:n asennusta kehitysympäristöön, sillä en katsonut sen olevan työn toimeksiantajalle tarpeen.

Olen liittänyt työhöni paljon ohjelmakoodiesimerkkejä esittelemistäni toiminnoista ja käytännöistä. Olen tehnyt tämän lähinnä siksi, että havainnollistaisin lukijalle ohjelmakoodin rakennetta ja tutustuttaisin hänet CakePHP:n ohjelmointia helpottaviin toimintoihin. Olen testannut kaikki esitellyt toiminnot ohjelmakoodiesimerkkeineen WAMP-ympäristössä (Windows-Apache-MySQL-PHP).

Kirjallisen raportin ohella opinnäytetyöni käsittää myös verkkosovelluksen, jonka olen toteuttanut raportissa esiteltävien periaatteiden mukaisesti. Tämän verkkosovelluksen ohjelmakoodi on liitteenä työni lopussa.

1.1 Ohjelmointikehyksen tarkoitus verkkosovelluskehityksessä

Kysyntä tehokkaasta, luotettavasta, ylläpidettävästä ja skaalautuvasta web-sovellusten kehitystekniikasta kasvoi Internetin yleistymisen seurauksena. PHP-ohjelmointikieli on nykyisin yksi tärkeimmistä ja käytetyimmistä dynaamisten toiminnallisuuksien toteuttamiseen tarkoitetuista web-ohjelmointikielistä. Koska PHP-kehitysmalli ei ota kantaa loogisten ohjelmakokonaisuuksien pilkkomiseen ja organisoimiseen, se on tuonut mukanaan myös monia ongelmia: uudelleenkäytettävyys, ylläpidettävyys ja skaalautuvuus kärsii, kun tehdään ns. huonoa lähdekoodia, eikä toteuteta ohjelmia selkeän ohjelmarungon mukaan. (Cui, Huang, Liang & Li 2009, 947.)

Vastauksena näihin ongelmiin on kehitetty ohjelmointikehyksiä, joiden periaatteena on pilkkoa ohjelmakokonaisuus loogisiin osiin. Lisäksi ne tarjoavat sovelluskehittäjille valmiita työkaluja sovelluksen osa-alueiden toteuttamiseen. Tämä tarkoittaa käytännössä sitä, ettei sovelluskehittäjän tarvitse ”keksiä pyörää uudelleen”.

1.2 Avoimen lähdekoodin ohjelmointikehys: CakePHP

CakePHP-ohjelmointikehys on maksuton ja kenen tahansa käytettävissä, sillä se on toteutettu avoimen lähdekoodin periaattein. Lähdekoodin avoimuus tarkoittaa sitä, että ohjelma on vapaasti levitettävissä ja sen lähdekoodi on avointa ja halukkaiden saatavilla sekä muokattavissa. (The Open Source Definition 2010.)

CakePHP:lla kehitettävän verkkosovellusprojektin alkaessa, kehittäjien ei tarvitse miettiä erikseen sovellukseen toteutettavaa rakennetta, sillä ohjelmointikehyksessä on rakenne jo valmiina. Tämä helpottaa projektin suunnitteluvaihetta ja ohjelmoijat pääsevät suoraan ohjelmalogiikkaan käsiksi. (CakePHP Manual 2010, 1.1.)

Periaatteet, joiden mukaan CakePHP:lla toteutettava verkkosovellusprojekti rakentuu, ovat loppuun asti mietittyjä, testattuja ja ajankohtaisia. CakePHP:n kehitystiimi päivittää jatkuvasti sen ydintä. CakePHP:n ydin voidaankin useimmiten päivittää ilman sovelluksen lähdekoodin uudelleenmuotoilua. Toisin sanoen vaikka CakePHP.n ydin päivittyisi, itse sovellukseen ei välttämättä tarvitse tehdä muutoksia. Kuitenkin on aina muistettava testata sovelluksen toiminta CakePHP:n ytimen päivittämisen jälkeen.

Opinnäytetyöni käsittelee CakePHP:n uusinta vakaata versiota 1.2.6.

2 MVC-arkkitehtuuri

Trygve Reenskaug kehitti vuonna 1979 ohjelmointiarkkitehtuurin, jonka tarkoituksena oli tarjota ratkaisu sovelluslogiikan erottamiseen käyttöliittymästä (Reenskaug 1979). Reilun kolmenkymmenen vuoden aikana MVC-arkkitehtuuri on kehittynyt, mutta sen periaate ja tarkoitus on pysynyt samana. Tänä päivänä MVC-arkkitehtuuri on yksi keskustelluimmista ohjelmointiarkkitehtuureista.

2.1 MVC-arkkitehtuurin tarkoitus

Termillä MVC tarkoitetaan sovelluksen suunnittelumallia, joka erottelee sovelluksen kolmeen loogiseen osaan. Sen tarkoituksena on yksinkertaistaa sovelluksen kehitystä ja ylläpitoa. Lisäksi se parantaa koodin uudelleenkäytettävyyttä ja selkeyttää ohjelmistokehittäjien työnkulkua. (McArthur 2008, 201.)

MVC-arkkitehtuuri koostuu siis kolmesta loogisesta kokonaisuudesta, Mallista, Näkymästä ja Ohjaimesta, (Model-View-Controller) jotka keskustelevat keskenään, mutta ovat silti siinä mielessä itsenäisiä, että jokainen näistä osista voidaan vaihtaa tai päivittää, koskematta muihin osiin. Jos siis halutaan vaihtaa sovelluksen ulkoasu täysin, mutta säilyttää silti hyväksi havaittu ohjelmalogiikka, tarvitsee ottaa käsittelyyn ainoastaan Näkymäosio. MVC-arkkitehtuuri toimii tällöin ohjenuorana ja yhtenäistävänä voimana ohjelmistokehityksen elinkaaressa. Sen periaatteita seuraamalla voidaan parantaa ohjelmakoodia ja ryhmätyöskentelyä.

MVC-arkkitehtuurin mukainen ryhmätyöskentely on helppoa siksi, että vastuualueet ovat selkeät ja toisistaan irralliset. Kevin McArthur esittelee kirjassaan yhdentyyppisen mallin siitä, miten ohjelmistokehityksen osa-alueet voidaan jakaa MVC-arkkitehtuurin mukaiseen työskentelytapaan.

Kehitys: Kehittäjät ovat ohjelmoijia, jotka työskentelevät mallikerroksen parissa. Heillä on kattavat taidot PHP:stä, tietokantojen hallinnasta, algoritmeistä, arkkitehtuurista ja tiedon oikeellisuuden tarkistuksesta. Tämä rooli on vastuussa sovelluksen ohjelmointiyksityiskohdista ja tarjoaa ohjelmointirajapintoja sekä valvoo tiedon vuorovaikutuksen käytäntöä. (McArthur 2008, 202.)

Muotoilu: Muotoilijat hoitavat näkymää ja ovat vastuussa siitä, miltä sovellus näyttää ja tuntuu. He ovat taitavia HTML-, CSS- ja JavaScript-teknologioissa ja grafiikan käsittelyssä ja tuottamisessa. Tyypillisesti tämä rooli on vastuussa vuorovaikutuksesta, niin sisäisessä kuin ulkoisessakin kommunikaatiossa, päättääkseen realistisista toimintasäännöistä, joiden mukaan sovellusta kehitetään tai parannetaan. Muotoilu johtaa yleensä prototyyppikehitykseen eli malleihin, joista näkyy sovelluksen ihanteellinen käytännön toiminnallisuus.(McArthur 2008, 202.)

Integrointi: Yhdentäminen tapahtuu ohjainkerroksessa, joka liimaa yhteen muotoilijoiden ja kehittäjien työn. Integroijilla on tyypillisesti vähemmän kokemusta kuin kehittäjillä, ja he ovat vastuussa staattisten mallinteiden pilkkomisesta dynaamisiin alueisiin. He ovat vastuussa myös tiedon välittämisestä. He ottavat pyyntötiedot lomakkeilta, välittävät tiedon mallille, tulkitsevat tulokset ja välittävät lopulta tulkinnat näkymälle. (McArthur 2008, 202.)

McArthurin esittelemässä työnkulussa muotoilija luo ensin staattisen, itsenäisen prototyypin, joka pohjaa jo olemassa oleviin toimintavaatimuksiin ja käytösmalleihin. Muotoilijat antavat sitten muotoilusuunnitelman kehittäjille arvioitavaksi.

Kehittäjät ovat vastuussa siitä, että kaikki suunnitellut toiminnallisuudet ovat toteuttamiskelpoisia ja sopivat yrityksen turvallisuus- ja yksityisyyspolitiikkaan. Jos kaikki näyttää hyvältä, kehittäjät luovat kehityssuunnitelman, joka sisältää mallin ohjelmointirajapinnasta, ja lähettävät sen integroijille. Jos prototyypissä on ongelmia, projekti siirtyy takaisin muotoilijoille, ja prosessin kulku alkaa alusta. (McArthur 2008, 202.)

Kun muotoiluprototyyppi ja malli ohjelmointirajapinnasta ovat valmiita, yhdentäminen (integrointityö) alkaa. Tämä tarkoittaa prototyypin analysointia ja mallintamista esimerkiksi PHP-kielellä. Tämä mahdollistaa dynaamisen tiedonkäsittelyn. Sen jälkeen luodaan ohjainkerros, joka välittää kyselyitä (lomakkeita, URL parametrejä, evästeitä, jne.) verkkopalvelimen ja mallin tiedon välillä. Kun ohjain on valmis, se tarjoaa arvot kaikelle dynaamiselle sisällölle, joka kuvannetaan näkymässä. (McArthur 2008, 202.)

2.2 CakePHP ja MVC-arkkitehtuuri

CakePHP seuraa MVC-arkkitehtuurin periaatteita. CakePHP-ohjelmointikehystä hyväksikäyttämällä ohjelmistokehittäjien on vaivatonta luoda sovellusarkkitehtuuri MVC-mallin mukaan, miettimättä ennalta sen enempää MVC-arkkitehtuurin rakennetta suhteutettuna sovelluksen rakenteeseen, sillä seuraamalla CakePHP:n valmista sovellusrakennetta, ohjelmistokehittäjät seuraavat tietämättäänkin myös MVC-arkkitehtuuria.

Malli: Malli on vastuussa sovelluksen toimintalogiikasta. Mallissa sijaitsee kaikki sovelluksen tieto. Mallikerros onkin vastuussa tietokantojen toiminnasta, tiedon oikeellisuuden tarkistuksesta ja tiedon käsittelystä. (McArthur 2008, 201.)

Yksittäistä mallia vastaa tyypillisesti yksi tietokantataulu. CakePHP olettaa, että mallia vastaava taulu on nimetty samalla nimellä kuin mallikin, mutta monikkoon. User-mallia vastaa siis users-taulu. Hakemistoarkkitehtuuri CakePHP:ssä rakentuu mallien osalta niin, että User-malli sijoitettaisiin polkuun: app/models/user.php. Kaikki ohjelmistokehittäjän määrittelemät malli-luokat perivät luokan AppModel. Yksinkertaisimmillaan malli-luokan määrittely sisältää vain luokan esittelyn sekä mahdollisesti name-muuttujan (Esimerkki 1). (O'Brien 2009, 13.)

Esimerkki 1: Yksinkertainen malli-luokan määrittely

```
class User extends AppModel {
    var $name = 'User';
}
```

Tämän määrittelyn jälkeen malli-luokka saa toiminnallisuudet tietokantahakuihin. Ne periytyvät AppModel-luokan kautta, joka taas perii CakePHP:n ytimeen sisäänrakennetun Model-luokan. AppModel-luokka on oletuksena tyhjä, mutta sinne voidaan määritellä funktioita, jotka tulevat käyttöön kaikissa sovelluksen malli-luokissa. Tämä käyttäjän määrittelemä AppModel-luokka sijoitetaan app-hakemiston juureen. PHP4-versiossa on välttämätöntä määritellä malli-luokalle name-muuttuja, joka saa aina malli-luokan nimen isolla alkukirjaimella kirjoitettuna. PHP5:stä eteenpäin name-muuttuja ei ole välttämätön. (CakePHP Manual 2010, 3.7.1.)

Näkymä: Näkymäkerros käsitettäisiin tyypillisesti rakenteeksi tai mallineeksi. Se sisältää sovelluksen näkymän ja tuntuman, jota käyttäjä hallinnoi. Näkymä voidaan käsittää myös käyttöliittymäksi. Teknologiat, jotka sijoitetaan yksinomaan näkymäkerrokseen, ovat HTML, CSS ja JavaScript. (McArthur 2008, 201.)

CakePHP:ssa näkymän mallineet (view template) nimetään niitä kutsuvan funktion mukaan. Mallinetiedostot ovat periaatteessa HTML-koodia, mutta sisältävät käytännössä myös PHP-muuttujia ja -käskyjä. CakePHP:ssa näkymän mallinetiedostot ovat ctp-päätteisiä, ja niille on oma paikkansa hakemistoarkkitehtuurissa. Users-ohjaimen, Register-funktion kautta näytettävä näkymän malline sijoitetaan tiedostoon app/views/users/register.ctp. (O'Brien 2009, 14.)

Ohjain: Ohjainkerros liimaa kaiken yhteen ja sulauttaa yhteen näkymän rakennemuotoilut ja mallista kerätyn tiedon. Se on vastuussa näkymäkerrokseen syötetyn tiedon keräämisestä ja sovelluksen ohjaamisesta. Ohjain kutsuu mallin palveluita ja tulkitsee palautetun tiedon, jotta se voidaan kuvantaa näkymässä. Mallikerros käsittelee sovelluksen tiedon. Näkymäkerros näyttää tiedon käyttäjälle. Ohjain tekee kaiken muun. Ohjain käsittelee palvelimelle kohdistetut pyynnöt. Se hyväksyy käyttäjän syötteet, käsittelee käyttäjän komennot, kutsuu mallia tarvittaessa ja ohjaa näytettävän tiedon näkymälle. (McArthur 2008, 201.)

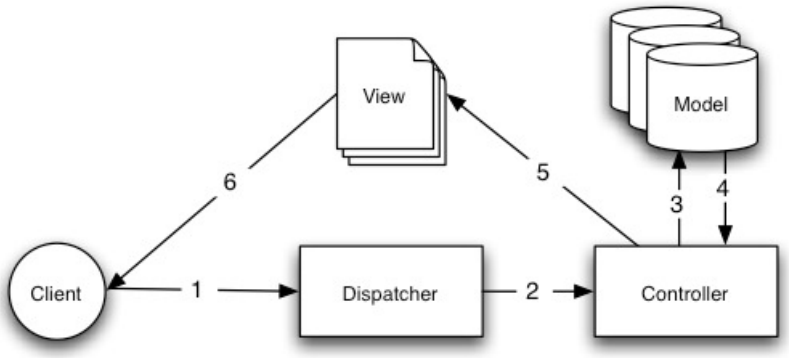
CakePHP:n periaatteen mukaan yhtä mallia vastaa yksi ohjain. Lisäksi jokaisella ohjaimella on oltava vastaava malli, vaikka ohjain ei tarvitsisikaan toiminnassaan tietokantaa. Tyypillisesti ohjaimen funktio käsittelee sovellukselle annettuja käskyjä, kutsuu mallia ja näyttää näkymän. User-mallia vastaava ohjain sijaitsee tiedostossa app/controllers/users_controller.php. Kaikki ohjelmistokehittäjän määrittelemät ohjain-luokat perivät AppController-luokan, joka taas perii CakePHP:n ytimeen sisäänrakennetun Controller-luokan. Kaikki määritykset ja funktiot, jotka ohjelmistokehittäjä määrittelee AppController-luokkaan, perityvät siis kaikkiin sovelluksen ohjain-luokkiin. Ohjelmistokehittäjän määrittelemä AppController-luokka sijoitetaan app-hakemiston juureen. (O'Brien 2009, 15.)

2.3 Tiedon virtaus

Tiedon virtaus CakePHP:ssä tapahtuu MVC-arkkitehtuuria noudattaen. Edellisessä kappaleessa selitettiin ne loogiset osat, joista MVC-malli koostuu. Seuraavaksi esitän perusmallin siitä, miten tiedon virtaus tyypillisesti tapahtuu CakePHP:n sisällä. Tämä onnistuu helpoimmin esimerkin avulla (Kuva 1).

1. Käyttäjä valitsee linkin, joka johtaa osoitteeseen http://www.esimerkki.fi/users/register, jolloin hänen selaimensa HTTP-pyyntö kohdistuu nimipalvelimen kautta esimerkki-palvelimelle (jossa siis CakePHP-ohjelmointikehyksellä rakennettu sovellus sijaitsee).
2. Palvelimella reititystarkastaja tarkistaa pyydetyn osoitteen (/users/register) ja ohjaa pyynnön oikealle ohjaimelle ja funktiolle. Tässä tapauksessa Users-ohjaimelle, register-funktiolle.
3. Ohjain tekee register-funktion määrittämät toimenpiteet. Se voisi, esimerkiksi, pyytää mallikerrokselta tietoa rekisteröidyistä käyttäjistä.
4. Malli palauttaa pyydetyt tiedot ohjaimen tulkittavaksi.
5. Ohjaimen käsiteltyä ja tulkittua tarvittavat tiedot, se antaa ne eteenpäin näkymäkerrokselle. Esimerkin mallineen polku olisi app/views/users/register.ctp.
6. Näkymäkerros purkaa tulkitun tiedon esitettävään muotoon ja lähettää esitettävän HTML-koodin käyttäjän selaimelle.

(CakePHP Manual 2010, 1.3.)



Kuva 1: Tiedon virtaus CakePHP:ssä (CakePHP Manual 2010, 1.3).

3 CakePHP yleisesti

Tässä luvussa käyn läpi tarvittavat tiedot sovelluksen kehittämisestä CakePHP:n avulla. Läpikäytävä tieto esimerkkeineen on tarpeen jokaiselle ohjelmistokehittäjälle, joka on suunnitellut tutustuvansa CakePHP-ohjelmointikehykseen. Esittelen nimeämiskäytännöt, konfiguraatiotiedostot, hakemistoarkkitehtuurin sekä tietokantaominaisuudet. Luvun lopussa esittelen muutaman hyödyllisen perustoiminnon, jotta lukija saa kuvan siitä minkälaisia toimintoja CakePHP tarjoaa verkkosovelluskehittämisen helpottamiseksi.

3.1 Nimeämiskäytännöt

CakePHP toimii siten, että se osaa itsenäisesti yhdistää mallin, ohjaimen, näkymän ja oikean tietokantataulun, kunhan ne on nimetty oikean periaatteen mukaan. Tätä periaatetta noudattamalla ohjelmistokehitys on sujuvaa ja yhteistyö kehittäjien kesken vaivattomampaa, sillä kehitettävän sovelluksen tiedostojen, luokkien ja tietokantojen nimet ovat yhdenmukaisia ja selkeitä.

CakePHP sallii kuitenkin poikkeustapauksia, mutta näistä jokainen täytyy määritellä erikseen konfiguraatiotiedostoon. Onkin paljon vaivattomampaa käyttää valmiiksi mietittyä nimeämiskäytäntöä ja luoda poikkeustapauksia vain erityistilanteissa. (CakePHP Manual 2010, 2.4.)

3.1.1 Tiedostot ja hakemistot

CakePHP:n nimeämisperiaatteen mukaan tiedostojen nimissä käytetään pieniä kirjaimia. Tyhjät välit korvataan alaviivalla (esim. Users-ohjain nimetään users_controller.php). Tiedostot nimetään tyypillisesti yksikkömuotoon. Ohjaintiedoston nimen loppuun lisätään _controller. (CakePHP Manual 2010, 2.4.)

Mallinetiedostojen nimeäminen tapahtuu hieman eri tavoin kuin ohjainten ja mallien. Ne saavat nimekseen sen ohjainfunktion, jonka kautta ohjain käskyttää mallinetta. Mallinetiedosto voi siis saada nimekseen monikkomuotoisen sanan. Lisäksi mallinetiedostojen nimessä olevat sanat kirjoitetaan aina yhteen ilman alaviivaa. Tällöin listUsers-funktion kautta käskytettävä mallinetiedosto, saa nimekseen listusers.ctp. (CakePHP Manual 2010, 2.4.1.)

Mallit ja ohjaimet sijoitetaan oman hakemistonsa juureen, mutta näkymän mallineille luodaan alihakemistot sen mukaan, mikä ohjain kyseisiä mallineita kutsuu. Users-ohjain (UsersController-luokka) kutsuu siis mallinetiedostoja kansiosta app/views/users. (CakePHP Manual 2010, 2.4.1.)

3.1.2 Luokat

Luokkien nimet kirjoitetaan isolla alkukirjaimella, ja mikäli luokan nimessä on kaksi sanaa (esim. users controller), ne kirjoitetaan yhteen siten, että molemmat sanat alkavat isolla alkukirjaimella (eli UsersController). Luokat nimetään yksikkömuotoon. (CakePHP Manual 2010, 2.4.2.)

3.1.3 Tietokantataulut ja -kentät

Jokaista mallia vastaa yksi tietokantataulu. Nämä tietokantataulut nimetään pienillä kirjaimilla monikkomuotoon. Tietokantakenttien nimet kirjoitetaan myös pienillä kirjaimilla ja tyhjä väli korvataan alaviivalla (esim. last_name). CakePHP:n vuorovaikutus tietokantojen kanssa toimii vaivattomimmin silloin, kun kannassa olevat kentät on nimetty samoin kuin niitä vastaavan lomakkeen kentät näkymässä. (CakePHP Manual 2010, 2.4.2.)

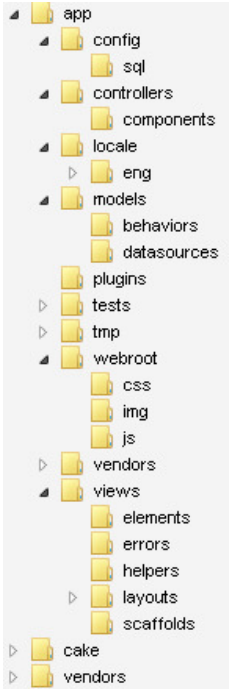
Alla (Taulukko 1) esimerkki tiedostojen ja tietokannan nimeämiskäytännöstä.

Taulukko 1: Nimeämiskäytännöt

Sovelluksen osa	**Tiedosto**	**Esimerkki**
Mallinetiedosto	app/views/<ohjain>/<funktio>.ctp	app/views/animals/get_ready.ctp
Ohjaintiedosto	app/controllers/<ohjain>_controller.php	app/controllers/animals_controller.php
Mallitiedosto	app/models/<malli>.php	app/models/animal.php
Komponenttitiedosto	app/controllers/components/<komponentti>.php	app/controllers/components/my_handy.php
Käytöstiedosto	app/models/behaviors/<behavior>.php	app/models/behaviors/please_behave.php
Avustajatiedosto	app/views/helpers/<helper>.php	app/views/helpers/help_me.php
Tietokantataulu		animals

3.2 Hakemistorakenne ja konfiguraatiotiedostot

CakePHP:n hakemistorakenteen (Kuva 2) sisäistäminen ja hallitseminen on välttämätöntä sovelluksen kehittämisessä. CakePHP:n hakemistorakenne on kokonaisuutena suunniteltu niin, että sen opettelemiseen ei kulu turhaan resursseja. Juurihakemistossa sijaitsee kolme alihakemistoa: vendors-, cake- ja app-hakemistot.



Kuva 2: CakePHP:n hakemistorakenne.

vendors-hakemisto: Sijainti kolmannen osapuolen PHP-kirjastoille, joita kehitettävä sovellus käyttää. CakePHP sisältää myös toisen vendors-hakemiston app-hakemiston sisällä. (CakePHP Manual 2010, 2.3.)

cake-hakemisto: Hakemisto, jossa sijaitsee CakePHP:n ydin. Tätä hakemistoa ja sen sisältämiä tiedostoja ei tule muokata, ellei halua muuttaa CakePHP:n toimintaa. (CakePHP Manual 2010, 2.3.)

app-hakemisto: Sovelluskehittäjien kannalta tärkein hakemisto. Tämä sisältää kaiken ohjelmakohtaisen koodin, jota kehitettävä sovellus käyttää. (CakePHP Manual 2010, 2.3.)

Tarkastelemme lähemmin app-hakemiston sisältöä. Sinne sovelluskehittäjät määrittelevät kehitettävän ohjelmiston konfiguraatiotiedot, internationalisaatiomääritykset sekä ohjelmakoodin malleille, ohjaimille ja näkymille. Hakemistorakenne on seuraavanlainen.

- **config-hakemisto:** Sisältää konfiguraatiotiedostot, joita kehitettävä sovellus käyttää (CakePHP Manual 2010, 2.3.1).

  - **bootstrap.php:** Ladataan automaattisesti kaikkialla sovelluksessa heti ytimen (CakePHP:n oman) bootstrap.php:n jälkeen. Tämän tiedoston kautta voidaan ladata funktioita, joita ei ole käytetty luokkien määrityksissä. Sisältää lisäksi käyttäjän omat määritykset mallien, ohjainten ja näkymien hakemistoille. (CakePHP Manual 2010, 3.4.7.)
  - **acl.ini.php:** Sisältää Access Control List -määritykset. Näitä määrityksiä käytetään antamaan ja kieltämään oikeuksia käyttäjille ja funktioille tiettyyn osaan sovellusta. (CakePHP Manual 2010, 5.1.)
  - **core.php:** Määritykset CakePHP:n käyttäytymiselle. Sisältää asetuksia virheenkorjaustilan (debug) määrittelyyn, käytettävään tavuesitykseen (character encoding), sessioiden hallintaan ja tietoturvan tasoon.
  - **database.php.default:** Esimerkki tietokannan konfiguraatiotiedostosta. CakePHP etsii tietokantakonfiguraatioita tiedostosta app/config/database.php. Tähän edellämainittuun tiedostoon määritellään tietokannan tiedot (mm. käyttäjätunnus, salasana, isäntäpalvelin ja tietokanta-ajuri). (CakePHP Manual 2010, 3.4.1.)
  - **inflections.php:** CakePHP:n nimeämiskäytäntöpoikkeuksien määrittelyyn tarvittava tiedosto. Tätä tiedostoa käytetään määrittelemään nimeämisen erityistapaukset, kuten monikkomuodon epäsäännölliset taivutukset. (CakePHP Manual 2010, 3.4.6.)
  - **routes.php:** Reititysmääritykset sisältävä konfiguraatiotiedosto. CakePHP käyttää reitityksessään oletuksena URL-muotoa: http://palvelin.com/ohjain/funktio/parametri1/parametri2. (CakePHP Manual 2010, 3.4.5.)

- **controllers-hakemisto:** Sisältää kehitettävän ohjelman ohjain-luokat ja ”komponentit” (nk. components, ohjain laajennokset).
- **locale-hakemisto:** Sisältää monikielisyyteen (internationalization) käytettävät merkkijonot.
- **models-hakemisto:** Sisältää kehitettävän sovelluksen malli-luokat, niiden laajennustiedostot (nk. behaviors) ja rajapinnat ylimääräisille tietolähteille (esim. twitter-rajapinta).
- **plugins-hakemisto:** Hakemisto liitännäissovelluksille, joita kehitettävä sovellus käyttää.
- **tmp-hakemisto:** CakePHP:n luomille väliaikaistiedostoille varattu hakemisto. Tyypillisesti tämä hakemisto sisältää mallien määrityksiä, logeja ja sessioinformaatiota.
- **vendors-hakemisto:** Kolmannen osapuolen PHP-kirjastot, joita kehitettävä sovellus käyttää.
- **views-hakemisto:** Näkymäkerrokseen liittyvät tiedostot sijaitsevat tässä hakemistossa. Sisältää alihakemistot, jotka ovat jaettuna näkymän osien kesken: elements, errors, helpers, layouts, scaffolds. Luotaessa mallinetta näkymälle kehittäjän on lisättävä uusi alihakemisto, joka on nimetty ohjaimen mukaan. Tänne alihakemistoon kehittäjä luo uuden .ctp-päätteisen mallinetiedoston, joka on nimetty sen funktion mukaan, jonka kautta ohjain käskyttää mallinetta.
- **webroot-hakemisto:** Tämä hakemisto toimii sovelluksen juurihakemistona. Sisältää lisäksi alihakemistot CSS-tyylitiedostoille, kuville ja JavaScriptille.

(CakePHP Manual 2010, 2.3.1.)

3.2.1 Configure-luokka

CakePHP:n käytökseen liittyvät konfiguraatiomääritykset (Taulukko 2) sijaitsevat tiedostossa app/config/core.php. Tämä on koko sovelluksen käytössä oleva luokka, johon sovelluskehittäjä voi tallentaa myös omia konfiguraatiomäärityksiään. Kaikki konfiguraatiomääritykset kirjoitetaan config-muuttujaan, joka on moniulotteinen taulukko. (CakePHP Manual 2010, 3.4.)

Taulukko 2: Oletuksena käytettävät konfiguraatiomuuttujat (core.php -tiedostossa) (CakePHP Manual 2010, 3.4.3).

Konfiguraatiomuuttuja	Kuvaus
debug	Vaihtaa tulostettavaa virheenetsintätilaa seuraavasti: 0 = Tuotantotila. Ei tulosta virheenetsintäinformaatiota. 1 = Näyttää virheet ja varoitukset. 2 = Näyttää virheet, varoitukset ja SQL-lauseet. 3 = Näyttää virheet, varoitukset, SQL-lauseet sekä täydellisen vedoksen ohjaimesta.
App.base_url	Liittyy Apachen mod_rewrite-toimintaan. Kun määritys on kommenttimerkkien sisällä Apachen mod_rewrite on käytössä (oletus).
Routing.admin	CakePHP:n admin-reititysmääritys. Oletusmäärityksenä kommenttimerkkien sisällä eli ei käytössä.
Cache.disable	Jos määrityksenä boolean-arvo true, välimuisti ei ole käytössä missään osassa sovellusta. Oletuksena kommenttimerkkien sisällä, jolloin välimuisti on käytössä.
Cache.check	Jos määrityksenä boolean-arvo true, välimuisti on käytössä myös näkymäkerroksessa. Tällöin tämä täytyy määritellä erikseen myös ohjaimeen, joka käskyttää näkymää. Oletusarvona kommenttimerkkien sisällä, jolloin välimuisti ei ole käytössä näkymäkerroksessa.
Session.save	Määrittää minkälaista sessionvarastointimenetelmää CakePHP käyttää. php = Käyttää php:n oletusvarastointimenetelmää. cache = Käyttää välimuistimoottoria, joka on määritelty erikseen ytimen Cache-luokan config-funktioon. cake = Tallentaa sessiotiedot kansioon app/tmp. database = Tallentaa sessiotiedot tietokantaan. Vaatii, että sovelluskehittäjä määrittelee tietokantataulun tiedostoon app/config/sql/sessions.sql. Oletusmäärityksenä php.
Session.table	Sen taulun nimi, johon sessiotiedot tallennetaan. Oletuksena kommenttimerkkien sisällä.
Session.database	Sen tietokannan nimi, johon sessiotiedot tallennetaan. Oletuksena kommenttimerkkien sisällä.
Session.cookie	Sen evästeen nimi, johon sessiotiedot tallennetaan. Oletuksena CAKEPHP.
Session.timeout	Session aikakatkaisu sekunteina. Tarkka arvo riippuu määrityksistä Security.level-muuttujassa. Oletuksena arvo 120.
Session.start	Kun määrityksenä boolean-arvo true, aloittaa session automaattisesti. Oletusarvona true.
Session.checkAgent	Kun määrityksenä boolean-arvo false, HTTP_USER_AGENT-määritystä ei tarkisteta sessiopyyntöjen välillä. Oletuksena true-arvo.
Security.level	Tietoturvan taso. Kertoo sessioiden aikakatkaisumuuttujan arvon tietyllä arvolla. Tästä tuloksesta saadaan todellinen aikakatkaisuarvo sekunteina. 'high' = Korkea tietoturvataso. Kertoo aikakatkaisuarvon luvulla 10. Ottaa käyttöön PHP:n session.referer_check-muuttujan tarkistuksen. Lisäksi generoi uudelleen session identifikaatioarvon jokaisen http-pyyn-

	nön välillä. 'medium' = Keskinkertainen tietoturvataso. Kertoo aikakatkaisuarvon luvulla 100. Ottaa käyttöön PHP:n session.referer_check-muuttujan tarkistuksen. 'low' = Alhainen tietoturvataso. Kertoo aikakatkaisuarvon luvulla 300. Oletusarvona merkkijono 'high'.
Security.salt	Tietoturvaan käytettävä satunnaismerkkijono. Tätä käytetään mm. salasanojen kryptaamisessa tietokantaan.
Asset.timestamp	Liittää aikaleiman käytettävän liitännäistiedoston (CSS-, JavaScript- ja kuvatiedostot) URL:n perään, silloin kun käytetään näkymäkerroksen avustajia. Tämä aikaleima sisältää tiedon kyseisen tiedoston viimeisestä muokkaamisajankohdasta. false = Boolean-arvo. Ei liitä aikaleimaa. true = Boolean-arvo. Liittää aikaleiman kun debug-määritys on suurempi kuin 0. 'force' = Merkkijono. Liittää aikaleiman huolimatta debug-määrityksestä. Oletusmäärityksenä boolean-arvo false.
Acl.database	Access Control List -toiminnon käyttämä tietokanta. Oletusmäärityksenä 'DbAcl'.
Acl.classname	Access Control List -toiminnon käyttämän luokan nimi. Oletusmäärityksenä 'default'.

CakePHP käyttää tiettyjä funktioita (Taulukko 3), joiden avulla edellä mainittuihin konfiguraatiomäärityksiin pääsee käsiksi sovelluksen ajoaikana. Nämä funktiot käsittelevät myös sovelluskehittäjän määrittelemiä konfiguraatiomäärityksiä.

Taulukko 3: Konfiguraatiomuuttujien käsittelyyn käytettävät funktiot (CakePHP Manual 2010, 3.4.3).

Funktio	Kuvaus	Esimerkki
write	Kirjoittaa tietoja konfiguraatiomuuttujaan.	Configure::write('debug', 2); Kirjoittaa debug-arvoksi kokonaisluku-muuttujan 2.
read	Lukee tietoja konfiguraatiomuuttujasta.	Configure::read('debug'); Palauttaa määritellyn debug-arvon.
delete	Poistaa tietoja konfiguraatiomuuttujasta.	Configure::delete('Cache.disable'); Poistaa määritykset Cache.disable-muuttujasta. Toisin sanoen, ottaa välimuistin käyttöön.
load	Lataa konfiguraatiomuuttujan määritellystä tiedostosta. Ladattavat konfiguraatioarvot tulee olla kirjoitettuna $config-muuttujaan. Muita tiedoston sisältämiä muuttujia ei lueta. Ladattava tiedostonimi annetaan funktiolle parametrinä ilman hakemistopolkua ja tiedostopäätettä.	Configure::load('messages'); Lataa tiedostosta app/config/messages.php löytyvän $config-muuttujan.
version	Palauttaa käytettävän CakePHP:n version.	Configure::version();

3.2.2 App-luokka

CakePHP käyttää luokkien ja kirjastojen lataamiseen sekä hakemistopolkujen hallitsemiseen sen ytimessä sijaitsevaa App-luokkaa. Tämä luokka sijaitsee tiedostossa cake/libs/configure.php. (CakePHP 1.3.x API 2010.) App-luokan avulla sovelluskehittäjät voivat siis ladata lisä-luokkia ja -kirjastoja sovelluksensa käyttöön. App-luokan funktioiden avulla tämä tapahtuu vaivattomasti, sillä se kahlaa automaattisesti läpi hakemistopolut ja varmistaa, ettei samaa luokkaa ladata moneen kertaan. (CakePHP Manual 2010, 3.4.4.)

Luokkien ja kirjastojen lataamiseen käytettävä import-funktion kutsu määritellään ennen luokkaesittelyä. Tyypillisesti tätä funktiota kutsutaan ohjainkerroksessa. Usein ladattavaa luokkaa tai kirjastoa tarvitaan kaikissa sovelluksen ohjaimissa. Tällöin kutsu kannattaa sijoittaa App-ohjaimeen. Alla lyhyt esittely App-luokan import-funktion käytöstä (Taulukko 4). Import-funktiolle voidaan määritellä useampiakin parametreja kuin taulukon esimerkeissä, mutta useimmiten tähän ei ole tarvetta. (CakePHP Manual 2010, 3.4.4.)

Taulukko 4: App-luokan funktiot (CakePHP Manual 2010, 3.4.4).

Toiminto	Funktion kutsu
Ytimestä lataaminen	
Ytimen Sanitize-kirjaston lataaminen.	`App::import('Core', 'Sanitize');`
Ohjaimen lataaminen	
Users-ohjaimen lataaminen.	`App::import('Controller', 'Users');`
Ohjaimet tarvitsee alustaa oliomuuttujaan lataamisen jälkeen.	`$Users = new UsersController;`
Jos haluamme ladata myös ohjainta vastaavan mallin suhdemäärittelyt, ohjaimen käyttämät komponentit sekä muut ohjaimessa tavallisesti ladattavat lisäosat.	`$Users->constructClasses();`
Mallien, komponenttien, käytösten ja avustajien lataaminen	`App::import('<Luokan tyyppi>', 'MyModel');`
Mallien lataaminen.	`App::import('Model', 'MyModel');`
Komponenttien lataaminen.	`App::import('Component', 'Acl');`
Käytösten lataaminen	`App::import('Behavior', 'Tree');`
Avustajien lataaminen	`App::import('Helper', 'Form');`
Liitännäisten ja vendortiedostojen lataaminen	
Liitännäisiä (plugin) ladatessa täytyy funktiokutsun parametreiksi määritellä liitännäisen nimen lisäksi hakemistopolku. Esimerkissä ladataan liitännäinen tiedostosta app/plugins/plugin_name/vendors/flickr/flickr.php.	`App::import('Vendor', 'PluginName.flickr/flickr');`
Vendortiedostojen lataaminen. Esimerkissä ladataan tiedosto app/vendors/flickr/flickr.php	`App::import('Vendor', 'flickr/flickr');`

3.2.3 Router-luokka

Ominaisuutta, joka yhdistää URL-merkkijonon osoittaman paikan tiettyyn ohjainkerroksen funktioon, kutsutaan reititykseksi (CakePHP Manual 2010, 3.4.5). Reititystä hallinnoi ytimen Router-luokka. Reitityksiä voidaan määritellä staattisiksi tai dynaamisiksi. Staattiset reitit kohdistuvat aina tietyn ohjaimen tiettyyn funktioon, tai vaihtoehtoisesti mallinetiedostoon, ja tulevat voimaan reititysmäärityksessä annettua URL-merkkijonoa käytettäessä. Dynaamiset reitit taas käsittelevät URL-merkkijonoja, jotka ovat tietyn rakenteisia. Tämän rakenteen perusteella reititin päättää mitä reititysmäärittelyä missäkin tapauksessa kutsutaan. (CakePHP 1.3.x API 2010.)

Ohjelmistokehittäjät voivat määritellä ylimääräisiä reititysohjeita sovelluksensa käyttöön app/config/routes.php-tiedostoon. Tämä tiedosto sisältää oletuksena kaksi reititysmäärittelyä (Esimerkki 2 ja Esimerkki 3). (CakePHP Manual 2010, 3.4.5.)

Esimerkki 2: Juuren reittimäärittely (Oletus)

```
Router::connect('/', array('controller' => 'pages',
                           'action' => 'display', 'home'));
```

Yllä oleva määrittely reitittää kutsun funktiolle display, joka sijaitsee Pages-ohjaimessa. Mikäli kyseistä ohjainta ei löydy, CakePHP lataa pelkästään näkymätiedoston app/views/pages/home.ctp. Tämä reititysmäärittely tulee voimaan silloin, kun kutsutaan palvelimelta CakePHP:n juurta. Kutsuttaessa palvelimen juurta, jossa CakePHP sijaitsee, URL olisi muotoa www.esimerkki.fi.

Esimerkki 3: Pages-alasivun reittimäärittely (Oletus)

```
Router::connect('/pages/*', array('controller' => 'pages',
                                  'action' => 'display'));
```

Yllä oleva määrittely kutsuu Pages-ohjaimen display-funktiota, silloin kun URL on muotoa www.esimerkki.fi/pages. Jokerimerkki (*) määrittelyssä kertoo reitittäjälle, että URL-merkkijono voi itseasiassa saada minkälaisen muodon tahansa, kunhan sen perusta on edellä mainitun muotoinen. Toisin sanoen, vaikka URL olisi muotoa www.esimerkki.fi/pages/more/complex/url, niin reititysmäärittely on silti voimassa.

Edellä esitellyt reititysmäärittelyt ovat siis oletuksena CakePHP:n käytössä, ja ovat tyypiltään staattisia reititysmäärittelyjä. Lisäksi CakePHP:n käytössä on oletuksena dynaaminen reititysmäärittely, joka ohjaa automaattisesti käskyn tietyn ohjaimen tiettyyn funktioon, riippuen siitä minkälaista URL-merkkijonoa kutsutaan (Esimerkki 4). Tämä ominaisuus on erityisen hyvin toteutettu, ja on niin kattava, että on mahdollista ettei sovelluskehittäjien tarvitse luoda lainkaan omia reititysmäärittelyjä. (CakePHP Manual 2010, 3.4.5.)

Esimerkki 4: URL-merkkijonoesimerkki kuvaa CakePHP:n dynaamista oletusreittiä.

```
http://example.com/<ohjain>/<funktio>/<param1>/<param2>/<param3>
```

Jos siis haluamme yhdistää Users-ohjaimen searchUser-funktioon parametrillä '19', annamme selaimelle seuraavanlaisen URL-merkkijonon (Esimerkki 5).

Esimerkki 5: Users-ohjaimen searchuser-funktioon yhdistäminen. Parametrinä '19'.

```
http://example.com/users/searchuser/19
```

Mikäli annettu URL-merkkijono käsittää vain ohjaimen nimen, reititin yhdistää selaimen kyseisen ohjaimen index-funktioon (CakePHP Manual 2010, 3.4.5).

3.3 Tietokanta-arkkitehtuuri

Seuraavaksi perehdymme CakePHP:n tietokanta-arkkitehtuuriin. Käytävät asiat ovat tärkeitä sovelluskehittämisen kannalta, sillä yksi verkkosovelluksen pääasiallisista toiminnoista on käyttää hyväkseen tietokantaa.

3.3.1 Tietokannan käyttöönotto

Tietokannan asetukset määritellään tiedostoon app/config/database.php. Tähän tiedostoon voidaan määritellä useita tietokanta-asetuksia eri tietokannoille. Nämä määritykset kirjoitetaan taulukko-muuttujan sisään. Tämän taulukkomuuttujan nimi vastaa tietokantamääritysten nimeä. Oletustietokannan taulukkomuuttuja on nimetty default-nimiseksi. Konfiguraatiotiedostosta löytyy asetukset mm. tietokanta-ajurille, tietokannan nimelle, isäntäpalvelimelle, käyttäjätunnukselle ja salasanalle. CakePHP:ssa on tuki monille eri tietokanta-ajureille, joista käytetyimpiä ovat MySQL, PostgreSQL, MsSQL ja Oracle. (CakePHP Manual 2010, 3.4.1.)

Kuten aikaisemmin mainitsin, CakePHP:n nimeämiskäytännön mukaan tietokanta nimetään monikkoon pienillä kirjaimilla, ja sen nimen tulee vastata sitä mallia, joka kyseistä tietokantataulua käyttää. Tietokantataulun nimessä olevat välit korvataan alaviivalla. Kenttien nimet kirjoitetaan pienellä ja välit korvataan alaviivalla.

Jokaiseen tauluun on lisäksi määriteltävä id-kenttä, joka identifioi tiedon. Tämä id-kenttä voidaan määritellä joko kokonaisluvuksi, tai UUID-tyyppiseksi merkkijono-muuttujaksi. UUID eroaa kokonaislukutyyppisestä identifikaatiosta siinä mielessä että se on

uniikki koko tietokannassa, kun taas kokonaisluku-id on uniikki vain yhden taulun sisällä. Molemmissa tapauksissa id-kenttä määritellään ensisijaikseksi avaimeksi (primary key) ja juoksevaksi (auto_increment). UUID-identifikaatiokenttä määritellään CHAR(36)- tai BINARY(36)-tyyppiseksi, kun taas kokonaislukuidentifikaatiokenttä määritellään INT(10). (CakePHP Manual 2010, 3.7.2.)

Tietokantahakujen ja syötteiden suorittamiseen CakePHP käyttää tyypillisesti moniuloitteisia taulukkomuuttujia, jotka sisältävät haetun tai syötettävän tiedon. Erityisesti syötettäessä tietoa moniin tauluihin, jotka ovat jonkintyyppisessä suhteessa toisiinsa, tulee kiinnittää tarkkaan huomioita tallennettavan taulukkomuuttujan rakenteeseen.

3.3.2 Tietokantataulujen linkitys ja mallien suhteiden määrittely

Tietokantojen käyttöön CakePHP tarjoaa monia helpottavia toimintoja, joista tietokantataulujen linkitys on kaikista tehokkain. Näitä linkitystyyppejä (suhteita) on neljä (Taulukko 5), ja ne määritellään kantaa käyttävän malli-luokan sisällä. (CakePHP Manual 2010, 3.7.6.)

Taulukko 5: Tietokantataulujen linkitystyypit (CakePHP Manual 2010, 3.7.6).

Suhde	Linkitystyypin nimi	Esimerkki
Yksi yhteen	hasOne	Käyttäjällä on yksi profiili
Yksi moneen	hasMany	Käyttäjällä on monia kuvia
Monta yhteen	belongsTo	Monta kuvaa kuuluu yhdelle käyttäjälle
Monta moneen	hasAndBelongsToMany	Kuvilla on, ja ne kuuluvat moniin etiketteihin.

Linkitystyyppi määritellään siis malli-luokan sisällä. Esimerkiksi käyttäjä-malliin määriteltäisiin, että käyttäjällä on monia kuvia. Tämä määrittely tehdään luokkamuuttujaan, joka nimetään linkitystyypin mukaisesti. Määrittelyyn voidaan lisätä monia yksityiskohtia suhteen käyttäytymisestä (esimerkiksi, miten tietokannasta haettu tieto järjestetään), jolloin määrittely tehdään moniulotteisen taulukkomuuttujan sisään. Yksinkertaisimmillaan se kuitenkin määritellään merkkijonomuuttujaan, jonka nimeksi annetaan suhteen tyyppi ja arvoksi mallin nimi, johon suhde kohdistuu. Alla esimerkki (Esimerkki 6) yksinkertaisesta linkitystyyppimäärittelystä, jossa User-mallilla on suhde yhteen Profile-malliin.

Esimerkki 6: User-mallin hasOne-määrittely

```
class User extends AppModel {
    var $name = 'User';
    var $hasOne = 'Profile';
}
```

Tietokantatauluihin tarvitsee lisäksi määritellä ulkopuolinen avain (foreign key), joka saa id-arvon toisesta taulusta. Esimerkkitapauksessa, jossa User-mallilla on yksi profiili, määritellään profiili-tauluun ulkopuolinen avain user_id. (CakePHP Manual 2010, 3.7.6.)

Kun edellämainitut määritykset on suoritettu, CakePHP osaa noutaa, tietokantahakua suoritettaessa User-mallin kautta, linkitetyt tiedot profiilitaulusta. Tallennettaessa suhteellisia tietoja, tallennukseen käytettävän taulukkomuuttujan rakenteen tulee olla täsmälleen oikeanlainen, jotta tallennus onnistuu. Tämä rakenne määräytyy sen mukaan minkälaista linkitystyyppiä mallien välillä käytetään. Alla olevan esimerkin (Esimerkki 7) mukainen taulukkomuuttuja tallentaa tietoja, jotka on määritelty yhden suhteena yhteen (hasOne). Tämä taulukkomuuttuja tallennettaisiin Users-ohjaimen kautta, jolloin CakePHP osaa automaattisesti tallentaa tiedot myös linkitettyihin tauluihin sekä lisätä oikeat id-tiedot. (CakePHP Manual 2010, 3.7.4.)

Esimerkki 7: hasOne-suhteen taulukkorakenne tallentamisessa

```
Array(
    [User] => Array(
        [username] => testaaja
        [password] => salasana
    )
    [Profile] => Array(
        [occupation] => ohjelmoija
    )
)
```

3.4 CakePHP:n perustoiminnot

Tässä luvussa esittelen kattavasti CakePHP:n tietokantaominaisuudet esimerkkien avulla. Yksistään tietokannan käytön helpottuminen on suurenmoinen apu verkkosovelluksen kehittämisen kannalta. Luvun lopussa esittelen lyhyesti muutamia perustoimintoja ja niiden käyttöä.

3.4.1 Tietokanta

CakePHP:n tietokantatoiminnot ovat pitkälle mietittyjä ja monipuolisia. Niiden toiminta saattaa aluksi vaikuttaa hankalalta, mutta rutiinin muodostuessa huomataan, että ne helpottavat ohjelmistokehittäjien työtä valtavasti. Tietokantaa käsiteltäessä CakePHP:n avulla, on tärkeintä ymmärtää moniulotteisten taulukkomuuttujien käyttöön liittyvät ongelmat. Hankalinta on muistaa minkälainen muuttujan rakenteen tulee missäkin linkitystyypissä olla. Useimmiten CakePHP osaa automaattisesti muodostaa oikeanlaisen taulukkomuuttujan, mutta käytettäessä monia linkitettyjä malleja, huomataan että tieto ei tallennukaan oikein tai ei lainkaan. Tällöin on useimmiten ongelmana vääränrakenteinen taulukkomuuttuja. CakePHP:n Html- ja Form-avustajia käytettäessä ohjaimelle saapuva taulukkomuuttuja on helpoiten muotoiltavissa oikean rakenteiseksi. Avustajia käsitellään tarkemmin luvussa 4.2.1.

Yksi tietokannan kannalta tärkeistä ominaisuuksista on tiedon validointi. Tämä tapahtuu CakePHP:ssa automaattisesti aina kun tietokantaan tallennetaan tietoa. Mallissa määritetyt validointisäännöt ovat siis käytössä ilman että ohjelmistokehittäjän täytyisi kutsua niitä erikseen. (CakePHP Manual 2010, 3.7.4.)

Tietokantatoimintoja toteutettaessa on otettu huomioon turvautuminen SQL-injektointeja vastaan. Tämä turva toimii käytännössä ilman erillistä käyttöönottoa. SQL-injektointi on tyypillinen hyökkäys, jonka avulla on tarkoituksena päästä luvatta käsiksi tietokantaan, tai saada sieltä tietoa (Kost 2004, 4). CakePHP suojaa tietokantaa SQL-injektoinnilta, mikäli käytetään sisäänrakennettuja find- ja save-funktioita ja kyselyparametrit kirjoitetaan taulukkomuuttujan sisään. Suojautuminen toimii myös käytettäessä niitä kyselyfunktioita, jotka itsessään käyttävät find- ja save- funktioita (esim. findUserById ja saveAll). (CakePHP Manual 2010, 4.2.)

Aina kun tietoa tallennetaan tietokantaan, CakePHP muistaa viimeksi tallennetun tiedon id-tunnuksen ja säilyttää sitä mallin id-muuttujassa (esimerkiksi, ohjaimesta kutsuttaessa: $this->User->id). Tämä on hyvä pitää mielessä, sillä siitä tulee varmasti hyötyä tehtäessä monia tallennuksia silmukan sisällä, tai päivitettäessä tietoa moniin tauluihin. (CakePHP Manual 2010, 3.7.4.)

3.4.1.1 Uuden tiedon tallentaminen

Tietokantaan tallentaminen tapahtuu tyypillisesti moniulotteisen taulukkomuuttujan avulla. Tämä taulukkomuuttuja sisältää kantaan tallennettavan tiedon. Uutta tietoa tallennettaessa ohjelmistokehittäjän ei tarvitse huolehtia lainkaan tietokantataulujen id-kentistä, sillä CakePHP osaa itsenäisesti lisätä ne kantaan. Tämä käytäntö toimii myös linkitettyjen taulujen kanssa. Toisin sanoen, CakePHP osaa lisätä myös oikean ulkopuolisen avaimen linkitettyyn tauluun.

Tärkeintä ohjelmistokehittäjän kannalta onkin pitää huolta siitä, että syötettävä tieto, joka on moniulotteisen taulukkomuuttujan sisällä, on oikean rakenteinen. Tämä rakenne määräytyy sen mukaan, minkätyyppisessä suhteessa tietokantataulut ovat toisiinsa nähden. Rakenne on helpoin pitää oikeanlaisena silloin, kun näkymässä käytetään Form-avustajaa HTML-lomakkeen luontiin (kappale 4.2.1).

Tiedon tallentamiseen on monia funktioita, joista tavallisimmat ovat save ja saveAll. Nämä funktiot sijaitsevat MVC-arkkitehtuurin mallikerroksessa, mutta niitä voidaan kutsua mallia käyttävän ohjaimen kautta. Alla olevassa esimerkissä (Esimerkki 8) tallennetaan käyttäjän syöttämiä tietoja Users-ohjaimen ohjelmakoodissa. Käytämme saveAll-funktiota, koska tallennamme tietoa myös profile-tauluun, joka on yksi yhteen suhteessa users-taulun kanssa.

Esimerkki 8: SaveAll-funktiolla tallentaminen.

```
$this->User->saveAll($this->data);
```

Data-muuttuja viittaa HTML-lomakkeelta kerättyyn tietoon. Tämä muuttuja on käytettävissä silloin kun käytetään Html- tai Form-avustajia (CakePHP Manual 2010, 3.7.4). Esimerkissä käytettävä data-muuttuja on rakenteeltaan seuraavanlainen (Esimerkki 9).

Esimerkki 9:Lomakkeelta palautuva data-muuttuja.

```
Array
(
    [User] => Array
        (
            [username] => teppo
            [password] => salasana
            [email] => teppo@testaajat.fi
            [first_name] => teppo
            [last_name] => testaaja
        )

    [Profile] => Array
        (
            [profile_name] => teponomaprofiili
        )

)
```

Tallentamiseen käytettävät save- ja saveAll-funktiot validoivat tallennettavan tiedon, ennen kuin itseasiassa tallentavat sen tietokantaan. Mikäli validointi ei mene läpi, tietokaan ei tallennu. Funktioiden parametreissä voidaan antaa asetuksia tiedontallentamisprosessin käyttäytymiseen liittyen. Näiden parametrien avulla voidaan, esimerkiksi tallentaa tieto ilman automaattista validointia. (CakePHP Manual 2010, 3.7.4.)

Mikäli halutaan tallentaa tietoa, joka ei tule käyttäjän syötteistä tai HTML-lomakkeelta, voidaan käyttää mallin set-funktiota. Sen avulla luodaan moniulotteinen taulukkomuuttuja, johon CakePHP osaa automaattisesti hakea id-kentät. Tämä muuttuja tallennetaan tietokantaan jättämällä tallennusfunktio täysin ilman parametreja. Alla olevassa esimerkissä (Esimerkki 10) tallennetaan tietoa User-mallin kautta, joka on yksi yhteen suhteessa Profile-mallin kanssa. Esimerkkiohjelmakoodi kirjoitettaisiin Users-ohjaimeen. (CakePHP Manual 2010, 3.7.4.)

Esimerkki 10: set-funktion käyttäminen tallentamisessa.

```
$this->User->set(array('User' => array(
                            'username' => 'testi2',
                            'email'    => 'email2'),
                       'Profile' => array(
                            'profile_name' => 'testiprofiili2')));
$this->User->saveAll();
```

3.4.1.2 Tiedon päivittäminen

CakePHP tarjoaa vaihtoehtoisia tapoja tietokannassa olevan tiedon muuttamiseen. Voidaan, esimerkiksi, käyttää edellämainittua save-funktiota, kunhan ensin määritellään mallille sen tietueen id-tunnus, jota halutaan muokata (Esimerkki 11). Tällä tavoin voidaan päivittää vain yhtä tietokantataulua kerrallaan, joten linkitettyjen taulujen päivitys täytyy tällöin hoitaa erikseen. (CakePHP Manual 2010, 3.7.4.)

Esimerkki 11: Save-funktion käyttö tiedon päivittämiseen.

```
$this->User->id = $userId;
$this->User->save($this->data);
```

Aina tietoa päivitettäessä tarvitaan päivitettävän tietueen id-tunnus. Save-funktiota käytettäessä data-muuttujan muodon on oltava seuraavanlainen (Esimerkki 12).

Esimerkki 12: data-muuttujan rakenne käytettäessä save-funktiota.

```
Array
(
    [User] => Array
        (
            [first_name] => tuomo
            [last_name] => palo
        )
)
```

Vaihtoehtoinen tapa tiedon päivittämiseen on käyttää updateAll-funktiota. Tämä eroaa kahdella tavalla muista CakePHP:n tallennusfunktioista. Ensiksikin updateAll-funktiota käytettäessä merkkijonotyyppiset muuttujat täytyy sulkea lainausmerkkien sisään käsin. Toiseksi tämä funktio hyväksyy parametrikseen vain yksiulotteisia taulukkomuuttujia, joten monissa tapauksissa tämä taulukkomuuttuja täytyy luoda käsin. (CakePHP Manual 2010, 3.7.4.)

Alla esimerkki kahden taulun päivittämisestä samaan aikaan (Esimerkki 13). Esimerkissä on lisätty lainausmerkit merkkijonomuuttujiin ja luotu updateData-taulukkomuuttuja. Esimerkin malleilla (User ja Profile) on yksi yhteen -suhde.

Esimerkki 13: Kahden taulun päivittäminen updateAll-funktiolla.

```
$updateData = array(
    'User.first_name'      => "'".$this->data['User']
['first_name']."'",
    'User.last_name'       => "'".$this->data['User']
['last_name']."'",
    'Profile.profile_name' => "'".$this->data['Profile']
['profile_name']."'");

$this->User->updateAll($updateData, array('User.id' => $userId));
```

Huomattavaa on myös se, että save-funktiolla päivitettäessä, tallennettavan mallin id-tunnus määritellään ennen tallennuskäskyä, kun taas updateAll-funktio vaatii sen toiseksi parametrikseen.

3.4.1.3 Tiedon poistaminen

Tiedon poistaminen tietokannasta on hyvin yksinkertaista. CakePHP tarjoaa tähän tehtävään kolme erilaista funktiota, joista kaksi ovat identtisiä. Identtiset remove- ja delete-funktiot ovat tarkoitettu yhden tietueen poistamiseen kerrallaan. Nämä funktiot vaativat parametrikseen poistettavan tietueen id-tunnuksen. Lisäksi funktiot käsittävät toisen parametrin, jonka avulla voidaan määrittää poistetaanko tieto myös linkitetystä tietokantataulusta. Tämä toinen parametri on boolean-tyyppinen muuttuja, ja sen oletusarvona on true, jolloin linkitetyt tietueet poistetaan. Huomioitavaa on, että haluttaessa poistaa myös linkitetyt tietueet tietokannasta edellä mainittujen funktioiden avulla, mallien suhteiden määrityksessä on oltava dependent-avain, jonka arvoksi on myös määriteltynä true. (CakePHP Manual 2010, 3.7.5.)

Alla esimerkki (Esimerkit 14.1 ja 14.2) yhden tietueen poistamisesta users-tietokantataulusta. User-mallin suhteeseen Profile-mallin kanssa on nyt määritelty dependent-avaimen arvoksi true, joten users-taulusta tietoa poistettaessa, poistetaan samalla myös siihen linkitetty tieto profile-taulusta.

Esimerkki 14.1: hasOne-määritys User-mallissa.

```
class User extends AppModel {
    var $name = 'User';
    var $hasOne = array(
            'Profile' => array(
                'className'  => 'Profile',
                'foreignKey' => 'user_id',
                'dependent'  => true)
            );
}
```

Esimerkki 14.2: Users-ohjaimen remove-funktio.

```
$this->User->remove($id);
```

Mikäli halutaan poistaa tietokannasta kaikki tieto, joka sisältyy tiettyihin ehtoihin, käytetään deleteAll-funktiota. Tämä funktio poistaa oletuksena myös linkitetyt tietueet, mutta sen toiseksi parametriksi voidaan antaa arvoksi false, jolloin linkitetyt tietueet säästetään. Ensimmäiseksi parametriksi deleteAll-funktiolle annetaan SQL-lause tai taulukkomuuttuja, joka sisältää ehdot poistettavalle tiedolle. (CakePHP Manual 2010, 3.7.5.)

Alla olevassa esimerkissä (Esimerkki 15) on poistettu users-taulusta kaikki ne tietueet, joiden first_name-kentällä on arvo 'teppo'. Funktion parametrina esimerkissä on käytetty SQL-lausetta, jolloin verrattava merkkijonotyyppinen arvo täytyy sulkea lainausmerkkien sisään.

Esimerkki 15: SQL-lauseen käyttä deleteAll-funktion parametrinä.

```
$this->User->deleteAll('User.first_name LIKE "teppo"');
```

3.4.1.4 Tietojen hakeminen

CakePHP:ssä on hyvin kattava valikoima sisäänrakennettuja funktioita tietokannan tiedon hakemiseen. Toiminnallisuutta löytyy niin yhden kentän tiedon noutamiseen yhdestä taulusta, kuin myös monimutkaisten löytöehtojen käyttämiseen noudettaessa tietoa monista tauluista. CakePHP tarjoaa myös mahdollisuuden kirjoittaa kokonaisia SQL-lauseita hakufunktion parametriksi. Esittelen näistä hakufunktioista muutamia käytännöllisimpiä.

Peruspilarina kaikille hakufunktioille voidaan pitää find-funktiota. Tällä funktiolla onnistuu kaikentyyppiset tietokantahaut, joita CakePHP tukee. Poikkeuksena mainittakoon query-funktio, jonka parametriksi kirjoitetaan kokonaisia SQL-lauseita. (CakePHP Manual 2010, 3.7.3.)

Find-funktio saa ensimmäiseksi parametrikseen tietokantahaun tyypin, joka voi olla esimerkiksi 'first', 'all' tai 'count'. Näistä hakutyypeistä find-funktio käyttää oletuksenaan first-tyyppistä hakua, joka palauttaa vain yhden tietueen tiedot. Tätä hakutyyppiä käytetään, jos haetaan, esimerkiksi, käyttäjää id-kentän perusteella. All-tyyppinen haku palauttaa kaikki ne tietueet, jotka sisältyvät hakuehtoihin. Tämä hakutyyppi soveltuu hyvin vaikkapa listattaessa käyttäjiä paikkakunnittain. Count-hakutyyppi taas palauttaa kokonaislukutyyppisen arvon, joka vastaa sitä arvoa montako tietuetta tietokannassa on, jotka sopivat hakuehtojen sisään. (CakePHP Manual 2010, 3.7.3.)

Toiseksi parametrikseen find-funktio saa moniulotteisen taulukkomuuttujan, joka sisältää tietokantakyselyn määrityksiä. Näistä määrityksistä tärkeimmät ovat conditions-, fields- ja order-määritykset. Conditions-määritykset ovat ne varsinaiset hakuehdot, joita suoritettava tietokantakysely käyttää. Esimerkiksi, jos haetaan käyttäjiä joiden ikä on 25, sisällytetään hakuehdot tämän määrityksen sisään. Fields-määritys sisältää ne kentät, jotka näkyvät kyselyn tuloksessa. Order-määritys kertoo missä järjestyksessä kyselyn tulokset esitetään. (CakePHP Manual 2010, 3.7.3.)

Alla esimerkki (Esimerkki 16) yksinkertaisesta tietokantahausta, jossa haetaan users-taulusta kaikki ne tietueet, joiden first_name-kentän arvona on 'teppo'. Hakutuloksiin sisällytetään kentät username users-taulusta ja profile_name profiles-taulusta. Nämä tulokset järjestetään aakkosjärjestykseen usernamen-kentän mukaan.

Esimerkki 16: Tietokantahaku

```
$conditions = array('conditions' => array(
                                'User.first_name' => 'teppo'),
                        'fields' => array(
                            'User.username', 'Profile.profile_name'),
                        'order' => array('User.username ASC'));
$this->User->find('all', $conditions);
```

CakePHP:ssa on yksinkertaistettu hakuehtojen määrittelemistä ja toisaalta myös koodin luettavuutta luomalla erityyppisiä funktioita yleisimpiä tietokantakyselyita ajatellen. Näin ovat syntyneet findAllBy- ja findBy-funktiot. Näiden erotuksena on yksinkertaisesti se, minkälaista hakutyyppia niiden käyttämä find-funktio käyttää. FindBy-funktiot (Esimerkki 17) käyttävät find('first') tyyppisiä funktioita kyselyitä toteuttaessaan, kun taas findAllBy-funktiot (Esimerkki 18) käyttävät find('all') tyyppisiä funktioita. Toisin sanoen, toinen näistä funktioista palauttaa monia tietueita sisältävän tuloksen, ja toinen taas keskittää kyselytuloksen vain yhteen tietueeseen. Molemmat funktioista palauttavat tiedot myös linkitetyistä tauluista. (CakePHP Manual 2010, 3.7.3.)

Esimerkki 17: Käyttäjän tietojen haku id-tunnuksen perusteella.

```
$this->User->findById($userId);
```

Esimerkki 18: Käyttäjien tietojen haku sukunimen perusteella.

```
$this->User->findAllByLastName($lastName);
```

3.4.1.5 Monimutkaisten hakuehtojen määritteleminen

Kun tarvitaan kyselyjä, joiden hakuehtoihin sisältyy paljon määrityksiä, CakePHP turvautuu jälleen käyttämään moniulotteisia taulukkomuuttujia. Nämä taulukkomuuttujat sisältävät ne hakuehdot, joita toteutettava kysely käyttää (Esimerkki 19.1). Tällä tavoin CakePHP voi toteuttaa äärimmäisen monimutkaisia hakuehtoja kyselyissään.

Esimerkki 19.1: Monimutkaisen haun toteuttaminen.

```
$conditions = array('conditions' => array(
                    'OR' => array(
                        array('User.first_name' => 'teppo'),
                        array('User.username LIKE "t%"')
                    ),
                'AND' => array('Profile.profile_name LIKE' => 't%'))
                );
$this->User->find('all', $conditions);
```

Huomioitavaa on vapaus LIKE-vertailun käytössä. Vertailuoperaattori voidaan sijoittaa joko taulukon avaimen sisään, tai sen soluun. Mikäli se sijoitetaan taulukon soluun, täytyy verrattava merkkijono sulkea lainausmerkkien sisään. Luvut kirjoitetaan sellaisenaan kummassakin tapauksessa. (CakePHP Manual 2010, 3.7.3.)

Jälleen kerran on syytä kiinnittää huomiota moniulotteisen taulukon rakenteeseen. Mikäli taulukko on rakenteeltaan vääränlainen, kysely ei tuota haluttua tulosta.

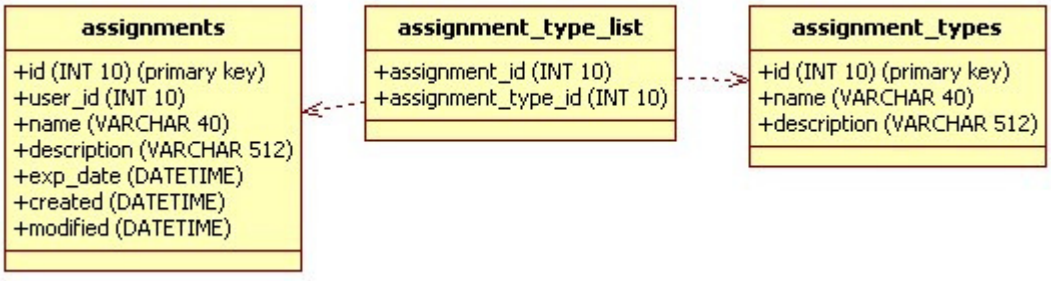
Yllä oleva esimerkki toteuttaa seuraavan SQL-lauseen (Esimerkki 19.2).

Esimerkki 19.2: Edellisen esimerkin (Esimerkki 19.1) toteuttama SQL-lause

```
SELECT `User`.`id`, `User`.`username`, `User`.`password`,
`User`.`email`, `User`.`first_name`, `User`.`last_name`,
`Profile`.`id`, `Profile`.`user_id`, `Profile`.`profile_name`
FROM `users` AS `User` LEFT JOIN `profiles` AS `Profile` ON
(`Profile`.`user_id` = `User`.`id`)
WHERE (
    (`User`.`first_name` = 'teppo')
    OR (`User`.`username` LIKE "t%"))
AND `Profile`.`profile_name` LIKE "t%"
```

3.4.1.6 Join-taulujen luonti

CakePHP:n mallien suhteissa voidaan määritellä kahden mallin suhteen olevan tyyppiä hasAndBelongsToMany (ns. HABTM-suhde). Tällaisessa suhteessa tarvitaan yhtä ylimääräistä tietokantataulua (Kuva 3), joka linkittää kaksi ulkopuolista id-kenttää toisiinsa. Kyseisessä tietokantataulussa (linkitystaulussa) ei tarvita ollenkaan ensisijaista avainta eli taulun omaa id-kenttää. (CakePHP Manual 2010, 3.7.4.2.)



Kuva 3: Esimerkki hasAndBelongsToMany-suhdemäärittelyn tietokanta-arkkitehtuurista

Olen käyttänyt opinnäytetyöni esimerkkisovelluksessa Assignment- ja AssignmentType-mallien kesken hasAndBelongsToMany-suhdetta. Kyseisen suhteen käyttö määritellään Assignment-malliin seuraavasti (Esimerkki 20).

Esimerkki 20: hasAndBelongsToMany-määrittely Assignment-mallissa.

```
var $hasAndBelongsToMany = array(
        'AssignmentType' => array(
              'joinTable'             => 'assignment_type_list'
            , 'foreignKey'            => 'assignment_id'
            , 'associationForeignKey' => 'assignment_type_id'
            , 'className'             => 'AssignmentType'
            )
        );
```

HABTM-suhdetta käyttävää tietoa esitettäessä HTML-lomakkeella, CakePHP käyttää <select>-tageilla luotavaa monivalintalaatikkoa. Tämä on helpoin ja vaivattomin tapa HABTM-suhdetta käyttävän tiedon keräämiseen, tallentamiseen ja esittämiseen, sillä CakePHP huolehtii oikeat id-tunnukset linkitystauluun, ja palauttaa suoraan tallennettavan data-muuttujan. Lisäksi se generoi monivalintalaatikon automaattisesti. Tallennettaessa uutta tietoa user_id-kenttä on lähetettävä ohjaimelta näkymälle ja liitettävä HTML-lomakkeeseen piilotettuna kenttänä (Esimerkki 21.1 ja Esimerkki 21.2).

Esimerkki 21.1: Ohjaimelta lähetettävät tiedot HTML-lomaketta varten

```
$this->set('userId', $this->Auth->user('id'));
$this->set('assignmentTypes', $this->AssignmentType->find('list'));
```

Esimerkki 21.2: Monivalintalaatikon ja piilotetun kentän luonti näkymässä.

```
echo $form->input('AssignmentType');
echo $form->hidden('Assignment.user_id', array('value' => $userId));
```

Jos haluamme, esimerkiksi, käyttää HTML-lomakkeessa checkbox-tyyppisiä input-kenttiä, on välttämätöntä muokata data-muuttujaa (Esimerkki 22) tallentamista varten. Lisäksi HTML-lomakkeessa esitettävän tiedon näyttäminen on haastavampaa. (CakePHP Manual 2010, 3.7.4.2.)

Esimerkki 22: HABTM-tallentamiseen käytettävän data-muuttujan rakenne.

```
Array
(
    [Assignment] => Array
        (
            [name] => Pese ikkunat
            [description] => Pese pohjoisen puoleiset ikkunat.
            [exp_date] => Array
                (
                    [month] => 03
                    [day] => 29
                    [year] => 2010
                )

            [user_id] => 7
        )

    [AssignmentType] => Array
        (
            [AssignmentType] => Array
                (
                    [0] => 2
                )

        )

)
```

Taulukossa AssignmentType -avaimen sisällä ovat id-tunnukset, joilla viitataan assignment_types-taulukon id-kenttiin. CakePHP poistaa oletuksena kaikki HABTM-suhteen kautta linkittyjen tietueiden id-kentät linkitystaulusta. Toisin sanoen, mikäli AssignmentType -avaimen sisällä olevat solut olisivat tyhjiä, CakePHP poistaisi kyseisellä assignment_id-tunnuksella varustetut viittaukset linkitystaulusta. Tämä on hyvä muistaa erityisesti silloin, kun tallennetaan HABTM-suhdemäärittelyn omaavaan tauluun tietoa, mutta ei haluta koskea lainkaan linkitettyyn tietoon. Esimerkiksi, jos haluan muuttaa vain exp_date-kentän sisältöä assignments-tietokantataulusta. Tällöin voidaan käyttää CakePHP:n suhdemäärittelyn väliaikaista poistoa. CakePHP:n avulla on siis mahdollista poistaa ja lisätä mallien suhdemäärittelyjä sovelluksen ajon aikana. (CakePHP Manual 2010, 3.7.6.6.)

3.4.2 Muutamia perustoimintoja lyhyesti

CakePHP sisältää suuren määrän sisäänrakennettuja toimintoja, joita se tarjoaa verkkosovelluskehittämisen helpottamiseksi. Nämä toiminnat ovat ominaisuuksiltaan kattavia ja ajankohtaisia. Esittelen seuraavaksi muutamia käytännöllisimpiä perustoimintoja lyhyesti.

3.4.2.1 Tiedon validointi (oikeellisuustarkistus)

Miltei jokainen sovellus vastaanottaa käyttäjän syötteitä. Verkkosovellus, joka on kenen tahansa käytettävissä, joutuu suosiostaan riippuen vastaanottamaan mahdollisesti tuhansia syötteitä päivittäin. Sovelluksen oikeanlaisen toiminnan kannalta on tärkeää, että käyttäjien antamat syötteet tarkistetaan ennen niiden käsittelemistä oikeana syötteenä. Aiemmin mainitsin, että CakePHP suojaa tietokantaa SQL-injektointia vastaan. Tämä ei kuitenkaan vielä riitä. Lisäksi tarvitaan tarkistusta muun muassa sille, että rekisteröityvän käyttäjän tulee syöttää vähintään kahdeksan merkkiä pitkä salasana. Tämän vuoksi CakePHP:ssa on mahdollista luoda tarkistussääntöjä käyttäjäsyötteitä varten. Tämänkaltaista tarkistusta kutsutaan tiedon validoinniksi. (Grant, 2000.)

Validointisäännöt kirjoitetaan siihen malliin, johon kyseinen tieto liittyy. Käyttäjätietoja varten kirjoitettaisiin validointisäännöt siis 'käyttäjä-malliin', validate-nimiseen luokkamuuttujaan (Esimerkki 23). CakePHP:ssa on valmiina muutamia validointisääntöjä, esimerkiksi, sähköposti- ja URL-osoitteelle. Näitä validointisääntöjä voi toki määritellä myös itse. Validointisäännöt määritellään muuttujaan, joka voi monimutkaisimmillaan olla usean säännön kattava moniulotteinen taulukko tai yksinkertaisimmillaan yhden säännön sisältävä taulukko. Kun nämä määritykset on annettu, CakePHP käyttää niitä oletuksena joka kerran kun kyseistä tietoa tallennetaan tietokantaan (ks. kappale 3.4.1.1). Lisäksi CakePHP osaa näyttää validointivirheet siinä lomakkeessa miltä käyttäjäsyötteet on kerätty. Jotta tämä toimisi tulee ladattavan näkymän olla sama, kuin se missä lomake sijaitsee. Toisin sanoen, sama sivu täytyy ladata kahteen kertaan. (CakePHP Manual 2010, 4.1.)

Esimerkki 23: User-mallin (app/models/user.php) validointimääritykset

```
var $validate = array(
            'password' => array(
                'rule' => array('between', 5, 15),
                'message' => 'Password must be between 5 to 15
characters'),
            'email' => 'email');
```

Ensimmäisen taulukon avaimet (password, email) viittaavat lomakkeen ja tietokannan samannimisiin kenttiin. Seuraavan taulukon avain 'rule' kertoo, että sen solu sisältää jonkin validointisäännön. Avain 'message' taas pitää sisällään viestin, joka näytetään käyttäjälle HTML-lomakkeessa, mikäli validointi epäonnistuu.

Tätä käyttävä lomake määritellään mallinetiedostoon (app/views/users/add.ctp) form-avustajaa hyväksi käyttäen seuraavanlaisesti (Esimerkki 24).

Esimerkki 24: Form-avustajalla luotu HTML-lomake käyttäjätietojen tallennukseen.

```
echo $form->create('User');
echo $form->input('User.username');
echo $form->input('User.password');
echo $form->input('User.email');
echo $form->input('User.first_name');
echo $form->input('User.last_name');
echo $form->input('Profile.profile_name');
echo $form->end('send');
```

Lopuksi tallennuslogiikka määritellään Users-ohjaimeen (app/controllers/users_controller.php) (Esimerkki 25).

Esimerkki 25: Käyttäjätietojen tallennuslogiikka Users-ohjaimessa

```
function add(){
        if (!empty($this->data)){
            if($this->User->saveAll($this->data)){
                $this->render('saved');
            }else{
                $this->set('info', 'Error while saving');
            }
        }
    }
```

Ohjaimen add-funktio vastaanottaa lomakkeelta saadut käyttäjäsyötteet. Se validoi ja tallentaa ne tietokantaan, jonka jälkeen tulostuu malline saved.ctp-tiedostosta. Mikäli validointi ei mene läpi, tulostetaan jälleen add-malline (add.ctp) lomakkeineen ja validointivirheilmoituksineen.

Mikäli tieto halutaan validoida ilman sen tallentamista tietokantaan, voidaan käyttää ohjaimessa set-funktiota (Esimerkki 26). Tämän jälkeen kutsutaan validates-funktiota, joka palauttaa boolean-arvon sen mukaan, validoituuko tieto. (CakePHP Manual 2010, 4.1.)

Esimerkki 26: Tiedon validointi ilman tallennusta.

```
function validateUser(){
        if(!empty($this->data)){
            $this->User->set($this->data);
            if($this->User->validates()){
                $this->set('info', 'Validates');
            }else{
                $this->set('info', 'Didn\'t validate');
            }
        }
    }
```

3.4.2.2 Internationalisointi

Verkkosovellukset, joita voidaan käyttää ympäri maailmaa, saavat huomattavasti suuremman potentiaalisen käyttäjäkunnan, mikäli sovellus voidaan esittää käyttäjän omalla kielellä. Tällaista toimenpidettä kutsutaan internationalisoinniksi. Internationalisaatiolla tarkoitetaan siis sovelluksen lokalisaatiomahdollisuuksia. CakePHP:ssa on mahdollista toteuttaa lokalisointi rajattoman monille eri kielille. (CakePHP Manual 2010, 4.8.)

Lokalisointimääritykset kirjoitetaan omaan tiedostoonsa, johon liitetään tunnus sekä lokalisoitu merkkijono. Tunnuksen tulee olla sama jokaisessa lokalisointitiedostossa. Tähän tunnukseen viitataan ohjelmakoodissa aina, kun halutaan tulostaa lokalisoitu sana tai lause. Merkkijono sisältää lokalisoidun sanan tai lauseen, joka tulostetaan käyttäjälle. Lokalisaatiotiedoston sijainti määräytyy seuraavan hakemistopolun mukaan (Esimerkki 27). (CakePHP Manual 2010, 4.8.)

Esimerkki 27: Lokalisaatiotiedoston hakemistopolku.

```
/app/locale/<kieli>/LC_MESSAGES/default.po
```

<kieli> vaihdetaan lokalisoidun maan ISO 639-2 standardin mukaiseen tunnukseen. Englannin kielellä tunnus olisi *eng*, suomella *fin.* Lokalisaatiotiedosto muodostuu seuraavanlaisesti (Esimerkki 28). (CakePHP Manual 2010, 4.8.)

Esimerkki 28: Lokalisaatiotiedoston sisältö

```
msgid "First name"
msgstr "Etunimi"
```

Msgid-määritys pitää sisällään identifikaatiotunnuksen, jota ohjelmakoodissa kutsutaan. Msgstr taas sisältää sen merkkijonon, joka palautetaan tunnusta kutsuttaessa.

Käytettävä lokalisointikieli vaihdetaan yksinkertaisesti kirjoittamalla konfiguraatiomuuttujaan avain Config.language, joka saa arvokseen maan ISO 639-2-tunnuksen (Esimerkki 29) (CakePHP Manual 2010, 4.8).

Esimerkki 29: Lokalisointikielen vaihto.

```
Configure::write('Config.language', 'fin');
```

Tämä lokalisointimääritys voidaan sijoittaa periaatteessa mihin lähdekooditiedostoon tahansa, riippuen siitä miten lokalisoinnin halutaan toimivan kehitettävässä sovelluksessa. Käytännössä on kuitenkin kaksi hyvää sijaintia, joissa tämä määrittely kannattaa tehdä. Nämä sijainnit ovat app/config/bootstrap.php ja app/app_controller.php. Mikäli määrittely sijoitetaan bootstrap.php-tiedostoon, lokalisoitava kieli on sovelluksen oletuskielenä. App-ohjaimeen sijoitettaessa kieltä on helppo vaihtaa käyttäjä- tai sivukohtaisesti. (CakePHP Manual 2010, 4.8.)

Lokalisointitunnusta kutsutaan tyypillisesti näkymäkerroksesta. Tätä kutsua varten on oma funktionsa, joka saa parametrikseen boolean-arvon sen mukaan halutaanko kyseinen lokalisaatiomerkkijono tulostaa suoraan vai vaan palauttaa merkkijonona (Esimerkki 30). (CakePHP Manual 2010, 4.8.)

Esimerkki 30: Lokalisaatiomerkkijonoa kutsuva funktio.

```
__('First name', false); // Tulostaa palautuvan merkkijonon. (oletus)
__('First name', true);  // Ei tulosta palautuvaa merkkijonoa.
```

Internationalisointia toteutettaessa on hyvä käytäntö käyttää tunnuksissa selväkielisiä englanninkielisiä sanoja, jotka ovat muotoiltuna, niin kuin ne tulostettaisi käyttäjälle. Tällöin ei tarvita erikseen englanninkielen lokalisaatiotiedostoa. Lisäksi, mikäli sovellus ei löydä tunnusta vastaavaa merkkijonoa, ja se tulostaa pelkän tunnuksen käyttäjälle, tästä tunnuksesta saatetaan kuitenkin saada selville, mitä sovelluksen olisi tarkoitus tulostaa.

3.4.2.3 Sivutus

Verkkosivuilla, joiden käyttämä tietokanta on suurikokoinen, tarvitaan usein sivutusta (pagination) tietokannasta saatujen tulosten esittämiseen helposti luettavassa muodossa. Sivutukseen CakePHP käyttää paginator-toimintoa. Tämä toiminto määritellään ohjaimessa, mutta sen käyttöä varten on olemassa näkymäkerrokselle oma paginator-avustajansa. Paginator-avustajaa ei tarvitse määritellä erikseen ohjaimessa vaan CakePHP ymmärtää sitä käytettävän kunhan paginate-muuttuja on määriteltynä. Lisäksi sivutustoiminto osaa käyttää hyväkseen Ajax-toiminnallisuutta, jolloin sivutettavaa HTML-sivua ei tarvitse ladata uudestaan toisia sivutettuja tuloksia tarkasteltaessa. (CakePHP Manual 2010, 4.9.)

Ohjaimen sivutusmäärittely tehdään paginate-nimisen luokkamuuttujan sisään (Esimerkki 31). Tämä muuttuja on moniulotteinen taulukko, johon voidaan määrittää monia eri asetuksia sivutukseen liittyen. (CakePHP Manual 2010, 4.9.)

Esimerkki 31: Sivutusmäärittely ohjaimessa.

```
var $paginate = array('User' => array(
                            'limit' => 3,
                            'order' => array(
                                'User.first_name' => 'asc')));
```

Yllä oleva määrittely rajoittaa tuloksen kolmeen tulokseen per sivu. Order-määritys lajittelee tulokset aakkosjärjestykseen etunimen mukaan.

Kun paginate-muuttuja on määritelty, ohjaimessa voidaan kutsua paginate-funktiota. Tämä funktio tekee tietokantakyselyn, joten ei ole tarvetta kutsua erikseen mallin find-funktiota. (CakePHP Manual 2010, 4.9.)

Alla olevassa esimerkissä paginate-funktion palauttamat tulokset ohjataan suoraan näkymälle users-muuttujaan (Esimerkki 32).

Esimerkki 32: Paginate funktion käyttö ohjaimessa.

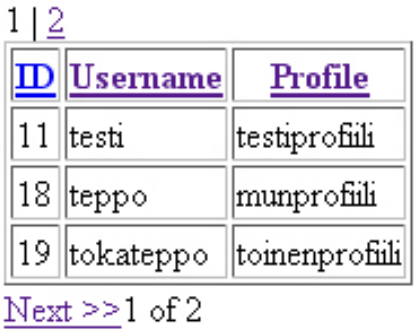
```
function listUsers(){
    $this->set('users', $this->paginate('User'));
}
```

Näkymässä kyselytuloksia voidaan näyttää monin eri tavoin, mutta yleisin tapa on sisällyttää ne HTML-taulukon sisään (Esimerkki 33). Paginator-avustajaa voidaan käyttää taulukon otsikoiden määrittämiseen. Avustaja tekee näistä otsikoista linkkejä, joita valitsemalla käyttäjä voi itse lajitella tulokset tietyn kentän mukaiseen järjestykseen. Lisäksi paginator-avustajalla on funktionsa sivunumeron näyttämiseen sekä linkkeinä, että x / y -muotoisena tekstinä. Seuraava- ja edellinen-linkeille on myös omat funktionsa. (CakePHP Manual 2010, 4.9.)

Esimerkki 33: listusers.ctp-tiedoston sisältö.

```
<?php echo $paginator->numbers(); ?>
<table border="1">
    <tr>
        <th><?php echo $paginator->sort('ID', 'id'); ?></th>
        <th><?php echo $paginator->sort('Username', 'username'); ?>
            </th>
        <th><?php echo $paginator->sort('Profile',
                                    'Profile.profile_name'); ?></th>
    </tr>
        <?php foreach($users as $user): ?>
        <tr>
            <td><?php echo $user['User']['id']; ?> </td>
            <td><?php echo $user['User']['username']; ?> </td>
            <td><?php echo $user['Profile']['profile_name']; ?> </td>
        </tr>
        <?php endforeach; ?>
</table>
<?php
    echo $paginator->prev();
    echo $paginator->next();
?>
<?php echo $paginator->counter(); ?>
```

Numbers-funktio tulostaa sivunumerot linkkeinä. Sort-funktio luo lajittelulinkit HTML-taulukon otsikoille. Funktiot prev ja next luovat edellinen- ja seuraava-linkit. Counter tulostaa x / y -tyylisen sivunumerolaskijan. Ylläoleva esimerkkikoodi tulostaa seuraavanlaisen näkymän selaimelle (Kuva 4). (CakePHP Manual 2010, 4.9.)


1 | 2

ID	Username	Profile
11	testi	testiprofiili
18	teppo	munprofiili
19	tokateppo	toinenprofiili

Next >> 1 of 2

Kuva 4: listusers.ctp-malline selaimessa

Ajax-sivutusta varten tarvitsee käytössä olevaan sommittelutiedostoon (oletuksena app/views/layouts/default.ctp) määritellä käytettävä JavaScript-kirjasto (CakePHP Manual 2010, 4.9). Lisäksi on muistettava ladata kyseinen kirjasto internetistä ja lisättävä se hakemistoon app/webroot/js/. Seuraavassa esimerkissä olen käyttänyt prototype-kirjastoa, joten olen lisännyt internetistä ladatun prototype.js-tiedoston edellä mainittuun hakemistoon. Olen lisännyt sommittelutiedoston <head>-tagien väliin ohjelmakoodin (Esimerkki 34), joka lataa JavaScript-kirjaston sovelluksen käyttöön.

Esimerkki 34: Prototype-kirjaston lataaminen (komento korostettuna) sommittelutiedostossa.

```
<head>
<title><?php echo $title_for_layout; ?></title>
<?php
    if(!empty($ajax)){
        echo $javascript->link('prototype');
    }
?>
</head>
```

Yllä olevan esimerkin if-ehtolause tarkistaa onko ajax-avustaja käytössä. Aina, kun käytetään sivutustoiminnallisuutta, ajax-avustaja otetaan automaattisesti käyttöön. Ehtolausetta ei tarvita, mikäli Ajax-avustaja on otettu käyttöön App-ohjaimessa ja on tällöin käytössä koko sovelluksessa.

CakePHP:n manuaalin mukaan Ajax-toiminnallisuutta varten on lisättävä Users- tai App-ohjaimeen esittely RequestHandler-komponentista (Esimerkki 35), joka käsittelee Ajax-kutsuja (CakePHP Manual 2010, 4.9.3). Testattaessa Ajax-sivutus toimii kuitenkin, vaikka kyseistä esittelyä ei tekisikään. Tämä saattaa johtua siitä, että Ajax-toiminnallisuutta on päivitetty uuteen CakePHP-versioon eikä korjausta manuaaliin ole vielä tehty. Suosittelen kuitenkin lisäämään esittelyn ohjaimeen, sillä se saattaa tuoda käyttöön joitakin ominaisuuksia sivutustoiminnallisuuteen liittyen, jotka ovat jääneet minulta huomaamatta.

Esimerkki 35: RequestHandler-komponentin esittely ohjaimessa.

```
var $components = array('RequestHandler');
```

Ainoa lisäys varsinaiseen sivutusmäärittelyyn tapahtuu mallineessa. Tarvitsemme määrittelyn update-asetukselle, joka kertoo mikä osa verkkosivusta päivitetään Ajax-kutsun myötä. Update-asetukselle annetaan arvoksi sen -tagin id-tunnus, johon päivitys kohdistuu. Jos lisäämme yllä olevan esimerkin mukaiseen mallineeseen (Esimerkki 33) tarvittavat määritykset, tulisi listusers.ctp-tiedostosta seuraavanlainen (Esimerkki 36). (CakePHP Manual 2010, 4.9.3.)

Esimerkki 36: Ajax-sivutus listusers.ctp-tiedostossa (lisätyt ohjelmakoodit korostettuna)

```
<?php
$paginator->options(array('update' => 'pagination'));
?>
<div id="pagination">
<?php echo $paginator->numbers(); ?>
<table border="1">
    <tr>
        <th><?php echo $paginator->sort('ID', 'id'); ?></th>
        <th><?php echo $paginator->sort('Username', 'username'); ?>
            </th>
        <th><?php echo $paginator->sort('Profile',
                                    'Profile.profile_name'); ?></th>
    </tr>
        <?php foreach($users as $user): ?>
        <tr>
            <td><?php echo $user['User']['id']; ?> </td>
            <td><?php echo $user['User']['username']; ?> </td>
            <td><?php echo $user['Profile']['profile_name']; ?> </td>
        </tr>
        <?php endforeach; ?>
</table>
<?php
    echo $paginator->prev();
    echo $paginator->next();
?>
<?php echo $paginator->counter(); ?>

</div>
```

Sivutus toimii tällöin myös siinä tapauksessa, että käyttäjällä ei ole JavaScript käytössä. Ajax-toiminnallisuus ohittuu automaattisesti ja sivut ladataan aina kokonaan.

4 Laajennusosat

CakePHP sisältää laajennusosia jokaiseen MVC-arkkitehtuurin loogiseen osa-alueeseen. Nämä ovat toimintoja, rakenteita ja näkymiä, joita tarvitaan useassa sovelluksen ohjelmakooditiedostossa. CakePHP:n ytimeen on sisäänrakennettuna muutamia hyvin tarpeellisia laajennososia, kuten form-avustaja ja auth-komponentti. Näitä laajennusosia voidaan luoda itse lisää. Esimerkiksi, HTML-navigointipalkista voidaan luoda elementti eli näkymäkerroksen laajennusosa. Luotua elementtiä voidaan sitten kutsua jokaisesta näkymäkerroksen tiedostosta. (CakePHP Manual 2010.)

Laajennusosat on luokiteltu kolmeen kokonaisuuteen. Tämä luokittelu seuraa löyhästi MVC-arkkitehtuuria eli jokaiselle osa-alueelle on oma kokonaisuutensa laajennusosia. Näkymäkerrokselle on avustajansa (helpers), mallikerrokselle käytösmääritykset (behaviors), ja ohjainkerrokselle komponentit (components). Näkymäkerroksen laajennusosiin liittyvät lisäksi elementit (elements) ja sommittelut (layouts). (CakePHP Manual 2010.)

4.1 Näkymäkerroksen laajennusosat

Kuten mainitsin, näkymäkerroksen laajennusosat käsittävät avustajat, elementit ja sommittelut. Näkymäkerroksen laajennusosia käytetään tyypillisesti mallinetiedostojen kautta. Avustajat täytyy ottaa käyttöön siinä ohjaimessa, joka käskyttää mallinetta, jossa avustajaa tarvitaan. Elementtejä kutsutaan suoraan näkymäkerroksesta. Sommittelulla luodaan rakenne mallineen ja elementtien ympärille.

4.1.1 Avustajat

Avustajat käsittelevät näkymäkerroksen toimintalogiikkaa. Avustajien avulla ohjelmistokehittäjien on vaivatonta luoda esimerkiksi HTML-lomakkeita tai sivuttaa tietokantakyselyn tuloksia (ks. Kappale 3.4.3.3 Sivutus). Nämä avustajat otetaan käyttöön mallinetiedostoa kutsuvassa ohjaimessa tai AppControllerissa, esittelemällä helpers-niminen taulukkomuuttuja, joka saa arvokseen käytettävän avustajan. Joissakin tapauksissa avustajan käyttöönottoa ei ole tarpeen määritellä ohjaimessa (ks. Kappale 3.4.3.3 Sivutus), mutta ei sen määrittelystä tällöin haittaakaan ole. Alla esimerkki Form- ja Html-avustajien käyttämisestä (Esimerkki 37.1, Esimerkki 37.2). (CakePHP Manual 2010, 3.11.)

Esimerkki 37.1:Avustajien käyttöönotto Users-ohjaimessa.

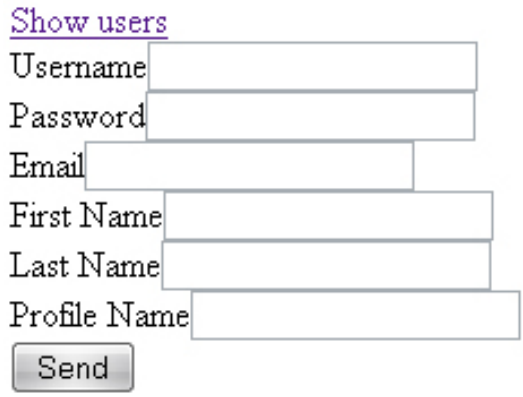
```
class UsersController extends AppController {
    var $helpers = array('Html', 'Form');
```

Yllä oleva määrittely voidaan sijoittaa myös App-ohjaimeen, jolloin kyseiset avustajat tulevat käyttöön kaikissa sovelluksen näkymäkerroksen tiedostoissa.

Esimerkki 37.2: Avustajien käyttö rekisteröintilomakkeen luontiin näkymäkerroksessa.

```
echo $html->link('Show users', array('controller' => 'users',
                                     'action'     => 'show'));
echo $form->create('User');
echo $form->input('User.username');
echo $form->input('User.password');
echo $form->input('User.email');
echo $form->input('User.first_name');
echo $form->input('User.last_name');
echo $form->input('Profile.profile_name');
echo $form->end('Send');
```

Kun ylläolevat määritykset on tehtynä, lomake on valmis. Alla kuva rekisteröintilomakkeesta selaimen kautta katsottuna (Kuva 5).



Kuva 5: Rekisteröintilomake (Esimerkki 37.2) selaimessa.

Avustajien käyttäminen on vaivatonta, mutta tarpeellista. Ne helpottavat huomattavasti kehittäjien työpanosta. Lisäksi useat avustajat ovat tarpeen CakePHP:n muita ominaisuuksia ajatellen. Esimerkiksi form-avustajaa käytettäessä, ohjaimelle palautuu HTML-lomakkeella lähetetyistä tiedoista data-muuttuja, joka voidaan halutessa tallentaa käsittelemättä tietokantaan vain yhden komennon avulla (ks. Kappale 3.4.1.1 Uuden tiedon tallentaminen).

Esittelen lyhyesti ytimeen sisäänrakennetut avustajat (Taulukko 6).

Taulukko 6: Ytimeen sisäänrakennetut avustajat (CakePHP Manual 2010, 7).

Avustaja	Kuvaus
Ajax	Käytetään yhdessä Prototype- ja Scriptaculous-JavaScript-kirjastojen kanssa. Luo Ajax-toiminnallisuutta näkymäkerrokseen.
Cache	Näkymän sisällön ja välimuistin vuorovaikutukseen käytettävä avustaja.
Form	HTML-lomakkeiden luontiin käytettävä avustaja.
Html	Helppokäyttötoimintoja sisältävä avustaja, jonka avulla voidaan luoda HTML-tageja mm. linkeille, kuville ja taulukoille.
Javascript	Käytöstä poistettava avustaja. Tämän avustajan käyttämät funktiot on siirretty Html- ja Js-avustajiin.
Js	Uusi avustaja JavaScriptin käyttöön. Tämä avustaja lataa JavaScriptkirjaston ja hallinnoi JavaScript-moottoria.
Number	Numeroiden ja valuuttojen muotoiluun käytettävä avustaja.
Paginator	Sivutukseen käytettävä avustaja. ks. Kappale 3.4.3.3 Sivutus.
Rss	Helppokäyttötoimintoja RSS-syötteen generointiin XML-tiedosta.
Session	Avustaja sessio-muuttujan käsittelyyn näkymäkerroksessa. Ei sisällä tallentamiseen käytettävää funktiota.
Text	Avustaja merkkijonotyyppisen tiedon käsittelyyn näkymäkerroksessa.
Time	Ajan esittämisen muotoiluun käytettävä avustaja.
Xml	Helppokäyttötoimintoja Xml-otsikoiden ja -elementtien luontiin.

Mikäli ytimeen sisäänrakennetut avustajat eivät palvele kehitettävän verkkosovelluksen tarpeita riittävän kattavasti, avustajia voidaan luoda itse lisää. Sovelluskehittäjän luomat avustajat sijoitetaan hakemistoon app/views/helpers/. Avustajatiedosto nimetään avustajan nimen mukaan. Link-avustajan tiedostonimeksi tulisi siis link.php. Avustajatiedostot ovat luokkia, jotka perivät AppHelper-luokan. Tähän luokkaan sijoitetaan ne funktiot, joita näkymäkerroksessa kutsutaan kyseisen avustajan kautta. Avustajat voivat lisäksi käyttää toisia avustajia hyväkseen. Nämä avustajat otetaan käyttöön luotavassa avustajaluokassa, kuten ne otettaisi käyttöön ohjaimessa (Esimerkki 38.1, Esimerkki 38.2). (CakePHP Manual 2010, 3.11.2.)

Esimerkki 38.1: Uuden avustajan luonti

```
class TestHelper extends AppHelper{
    var $helpers = array('Html');

    function createParagraphTags($paragraph){
        return $this->output($this->Html->tag('p', $paragraph));
    }
}
```

Test-avustajan createParagraphTags-funktio lisää merkkijonon ympärille <p>-tagit. Se käyttää avukseen jo olemassa olevaa Html-avustajaa. Ennen kuin voimme käyttää luotua avustajaa näkymäkerroksessa, se täytyy muistaa ottaa käyttöön mallinetta käskyttävässä ohjaimessa (ks. Esimerkki 37.1).

Esimerkki 38.2: Luodun avustajan käyttö näkymässä.

```
echo $test->createParagraphTags('testi');
```

4.1.2 Elementit

Kun tarvitaan näkymäkerroksen 'lohkoa', joka tulostetaan useasti verkkosovelluksen ajon aikana, on hyvä käyttää apuna elementtejä. Elementit sijoitetaan hakemistoon app/views/elements. Elementin tiedostonimeksi tulee elementin nimi, tiedostopäätteeksi .ctp. Elementti tulostetaan mallinetiedostossa, sommittelutiedostossa tai toisen elementin sisällä. Element-kutsun toisena parametrinä voidaan antaa moniulotteinen taulukkomuuttuja, joka lähettää määritellyn muuttujan elementille. Elementin tulostus näkymäkerroksessa tapahtuu seuraavanlaisesti (Esimerkki 39). (CakePHP Manual 2010, 3.10.3.)

Esimerkki 39: Elementin tulostaminen näkymäkerroksessa

```
echo $this->element('navigation', array('text', 'Hello world'));
```

Ylläoleva ohjelmakoodi kutsuu navigation-nimistä elementtiä. Lisäksi se lähettää kyseiselle elementille muuttujan nimeltä text. Tämä muuttuja sisältää merkkijonon 'Hello world'. (CakePHP Manual 2010, 3.10.3.)

Olen käyttänyt esimerkkisovelluksessani navigointipalkin tulostavaa elementtiä. Olen ensiksi määritellyt default-sommittelutiedostoon navigointipalkin sijainnin. Navigointi-

elementti tulostetaan default-sommittelutiedostossa edellisen esimerkin mukaisesti, mutta sille ei tässä tapauksessa tarvitse lähettää muuttujaa, jolloin kutsun toinen parametri jätetään pois. Navigointi-elementissä (Esimerkki 40.1) olen kutsunut Users-ohjaimen getNavigation-funtiota (Esimerkki 40.2), joka tarkastaa onko sovellusta käyttävä asiakas kirjautunut palveluun. Tämän mukaan getNavigation-funktio kutsuu tiettyä elementtiä (Esimerkki 40.3, Esimerkki 40.4), jossa itse navigointipalkki HTML-rakenteineen sijaitsee.

Esimerkki 40.1: app/views/elements/navigation.ctp-tiedoston ohjelmakoodi.

```
echo $this->requestAction(array('controller' => 'users'
                                , 'action'  => 'getnavigation'),
                          array('return'));
```

RequestAction-funktion kautta voidaan kutsua ohjaimen funktioita näkymäkerroksesta. Kutsun toinen parametri (array('return')) määrittelee funktiokutsun palauttavan kokonaan kuvannetun mallineen. (CakePHP Manual 2010, 3.5.5.4.6.) RequestAction-funktiota käytettäessä tulee muistaa sen rikkovan MVC-arkkitehtuurin periaatetta, jonka mukaan näkymäkerroksesta ei voida kutsua ohjainkerroksen funktioita. Tämä periaate perustuu MVC-arkkitehtuurin tiedon virtaan (ks. Kappale 2.3 Tiedon virtaus), joka kulkee näkymäkerrosta kohti. Lisäksi ongelmatilanteiden selvittäminen on hankalaa, sillä CakePHP ei anna tapansa mukaisia virheilmoituksia, mikäli virhe johtuu requestAction-funktion käytöstä. Erityisesti tulee välttää tilanteita, joissa requestAction-funktio kutsuu funktiota, joka myös käyttää requestAction-funktiota. (Geisendörfer, 2008.)

Tarkastellaan Users-ohjainta (Esimerkki 40.2).

Esimerkki 40.2: getNavigation-funktio ja Auth-komponentin määritykset.

```
function beforeFilter() {
    // Sallitaan kirjautumattomille seuraavat funktiot.
    $this->Auth->allow('add', 'getNavigation');
}
function getNavigation(){
    if($this->Auth->user('id')){
        // Tänne, jos kirjauduttu sisään.
        $this->set('username', $this->Auth->user('username'));
        $this->render('../elements/usersnavigation');
    } else {
        // Tänne, jos kirjautumatta.
        $this->render('../elements/defaultnavigation');
    }
}
```

Koska esimerkkisovellukseni käyttää Auth-komponenttia, joka rajoittaa kirjautumattomien käyttäjien oikeuksia, on Users-ohjaimessa muistettava ensin lisätä beforeFilter-funktioon määritys siitä, että getNavigation-funktioon pääsee käsiksi myös kirjautumattomat käyttäjät. GetNavigation-funktion ohjelmakoodi toimii seuraavasti. Mikäli käyttäjä on kirjautunut sisään, tulostetaan usersnavigation-elementti (Esimerkki 40.3) ja lähetetään tälle elementille kirjautuneen käyttäjän käyttäjätunnus. Jos käyttäjä ei ole kirjautunut, tulostetaan defaultnavigation-elementti (Esimerkki 40.4).

Esimerkki 40.3: usersnavigation-elementin ohjelmakoodi. Tulostuu, kun käyttäjä on kirjautunut.

```
echo '<ul><li>';
echo $html->link('Home', array('controller' => 'pages',
                               'action'     => 'index'));
echo '</li><li>';
echo $html->link('Profile', array('controller' => 'users',
                                  'action'     => 'profile'));
echo '</li><li>';
echo $html->link('List assignments',
                  array('controller' => 'assignments',
                        'action'     => 'listassignments'));
echo '</li><li>';
echo $html->link('Add assignment',
                            array('controller' => 'assignments',
                                  'action'     => 'add'));
echo '</li><li>';
echo $html->link('Log Out ('.$username.')',
                           array('controller' => 'users',
                                 'action'     => 'logout'));
echo '</li></ul>';
```

Esimerkki 40.4: defaultnavigation-elementin ohjelmakoodi. Tulostuu, kun käyttäjä ei ole kirjautunut.

```
echo '<ul><li>';
echo $html->link('Home', array('controller' => 'pages',
                               'action'     => 'index'));
echo '</li><li>';
echo $html->link('Login', array('controller' => 'users',
                                'action'     => 'login'));
echo '</li></ul>';
```

Kun nämä määritykset on tehty, navigaatiopalkki tulostuu jokaiselle verkkosovelluksen HTML-sivulle, joka käyttää default-sommittelutiedostoa.

4.1.3 Sommittelutiedostot

Sommittelutiedostot (layout) voidaan käsittää mallineen ympärille tulostettavana rakenteena. Nämä ovat ylimmät rakennemääritykset, joita verkkosovelluksessa käytetään. Näihin tiedostoihin määritellään verkkosovelluksen ulkoasun rakenne ja ne sijaitsevat hakemistossa app/views/layouts/. Tämä rakenne saa tyylimäärityksensä sommittelutiedostossa määritellystä tyylitiedostosta, joka sijoitetaan hakemistoon app/webroot/css/. Sommittelutiedostot ovat .ctp-päätteisiä ja ne sisältävät pääosin HTML-koodia. Oletuksena käytettävä sommittelutiedosto on nimeltään default.cpt. (CakePHP Manual 2010, 3.10.2.)

Sommittelutiedostossa tulostetaan verkkosivulla näytettävä sisältö. Tähän voidaan käyttää apuna näkymäkerroksen avustajia ja elementtejä. Sisältö, joka ohjautuu ohjaimen funktion kautta näkymäkerrokseen, tulee käytettäväksi sommittelutiedostossa content_for_layout-muuttujaan. Toisin sanoen, kaikki mallinetiedoston sisältö ohjautuu edellä mainittuun muuttujaan. Muita sommittelutiedoston käyttämiä muuttujia ovat title_for_layout ja scripts_for_layout. title_for_layout-muuttuja sisältää selaimessa näytettävän otsikon. Tätä otsikkoa voidaan muuttaa ohjaimessa (Esimerkki 41). scripts_for_layout-muuttujan käyttöönottoon tarvitaan erikseen Html-avustajaa. Tähän muuttujaan voidaan sisällyttää kaikki css- ja javascript-määritykset, jonka jälkeen muuttuja voidaan tulostaa sommittelutiedostossa <head>-tagien sisällä. (CakePHP Manual 2010, 3.10.2.)

Esimerkki 41: title_for_layout-muuttujan määrittely ohjaimessa.

```
$this->set('title_for_layout', 'View Active Users');
```

CakePHP:n ytimeen on sisäänrakennettu kaksi sommittelutiedostoa, joita CakePHP:n omat funktiot käyttävät: ajax- ja flash-sommittelutiedostot. Erillisiä sommittelutiedostoja tarvitaan, esimerkiksi sähköpostiviestin ja Rss-syötteen rakenteen määrittelyyn. Käytettävää sommittelutiedostoa voidaan vaihtaa sovelluksen ajon aikana. Tämä tehdään ohjaimessa seuraavan käskyn avulla (Esimerkki 42). (CakePHP Manual 2010, 3.10.2.)

Esimerkki 42: Käytettävän sommittelutiedoston vaihtaminen.

```
$this->layout = 'image';
```

4.2 Mallikerroksen laajennusosat eli käytösmääritykset

Käytösmääritykset ovat toimintoja, joita saatetaan tarvita useassa mallissa, tai ohjelmalogiikkaa, joka ei varsinaisesti kuulu mallikerrokseen mutta jota mallikerros käsittelee. Mallikerroksen laajennusosat sijoitetaan käytöstiedostoihin, app/models/behaviors/-hakemistoon. Käytöstiedostot nimetään käytösmäärityksen nimen mukaan (HierarchyBehavior-luokka saa tiedostonimekseen hierarchy.php). Käytösmääritys-luokat perivät ytimen ModelBehavior-luokan. CakePHP:ssä on sisäänrakennettuja käytöstiedostoja (Taulukko 7) ja niitä voidaan luoda itse lisää. (CakePHP Manual 2010, 3.8.)

Taulukko 7: CakePHP:n sisäänrakennetut käytösmääritykset.

Käytösmääritys	Kuvaus
ACL	Tarjoaa keinon mallin integroimiselle ACL-järjestelmään saumattomasti.
Containable	Suodattaa ja rajoittaa mallin tietokantahakuja. Käytännöllinen erityisesti silloin, kun mallilla on monia suhteita toisiin malleihin.
Translate	Internationalisointiin liittyvä käytösmääritys. Tallentaa tietokantaan lokalisointimerkkijonoja tunnuksineen.
Tree	Muuttaa mallin tietokannan käytöksen puu-hierarkkiseksi.

Käytösmääritys otetaan käyttöön mallissa esittelemällä actsAs-taulukkomuuttuja, joka saa arvokseen käytösmäärityksen nimen. Muuttujaan voidaan myös liittää määrityksiä laajennusosan toimintaan liittyen, jolloin taulukkomuuttujasta tulee moniulotteinen taulukko. Lisäksi yhteen malliin voidaan liittää useita eri käytösmäärityksiä. Alla esimerkki Tree-käytösmäärityksen käyttöönotosta (Esimerkki 43). (CakePHP Manual 2010, 3.8.)

Esimerkki 43: Käyttöönoton jälkeen Category-mallin tietokantatoiminta muuttuu puu-hierarkkiseksi.

```
class Category extends AppModel {
    var $name = 'Category';
    var $actsAs = array('Tree');
}
```

Mallia vastaavaan tietokantatauluun tarvitaan kolme lisäkenttää: parent_id, rght ja lft. Nämä kentät määritellään INT(10)-tyyppisiksi ja ne saavat oletusarvokseen NULL. Alla esimerkki categories-taulun rakenteesta (Kuva 6). (CakePHP Manual 2010, 6.4.)

Field	Type	Null	Key	Default	Extra
id	int(10)	NO	PRI	*NULL*	auto_increment
parent_id	int(10)	YES		*NULL*	
lft	int(10)	YES		*NULL*	
rght	int(10)	YES		*NULL*	
name	varchar(255)	NO			

Kuva 6: categories-taulun rakenne.

Kun ylläolevat määrityksen malliin ja tietokantaan on tehtynä, voimme siirtyä ohjaimen logiikkaan. Tree-käytösmäärityksen avulla tietokantatiedon lisääminen, poistaminen ja muokkaaminen on helppoa. Lisäksi Tree-käytösmääritys sisältää funktioita, joiden avulla saadaan tietoa puu-hierarkian haaroista ja tietueiden lapsista. Esittelen lyhyesti perustietokantatoiminnot liittyen puu-hierarkiaan. (CakePHP Manual 2010, 6.4.)

Tietoa voidaan noutaa tietokannasta perinteiseen tapaan, mutta Tree-käytösmääritys tarjoaa tiedon hakemiseen generatetreelist-funktiota. Tämä funktio luo listan, josta ilmenee puu-hierarkia. Generatetreelist-funktiolle voidaan antaa parametrina ehtoja siitä, mitä tietueita kysely koskee. Ilman ehto-parametrejä funktio palauttaa koko taulun tietueet (Esimerkki 44.1 ja Esimerkki 44.2). (CakePHP Manual 2010, 6.4.)

Esimerkki 44.1: generatetreelist-funktion käyttö ohjaimessa.

```
$treedata = $this->Category->generatetreelist(null, null, null,
'___');
```

Ylläoleva ohjelmakoodi asettaa treedata-muuttujaan categories-taulun tietueet puu-rakenteisena (Esimerkki 44.2). Lapsielementit erotetaan aikuisistaan '___'-merkkijonolla, joka määriteltiin funktiokutsun neljäntenä parametrinä. (CakePHP Manual 2010, 6.4.)

Esimerkki 44.2: treedata-muuttujan sisältö.

```
Array
(
    [1] => My Categories
    [2] => ___Fun
    [3] => _______Sport
    [4] => __________Surfing
    [5] => __________Extreme knitting
    [6] => _______Friends
    [7] => __________Gerald
    [8] => __________Gwendolyn
    [9] => ___Work
    [10] => _______Reports
    [11] => __________Annual
    [12] => __________Status
    [13] => _______Trips
    [14] => __________National
    [15] => __________International
)
```

Uuden tiedon tallentaminen tapahtuu siten, että määritämme aikuiselementin id-tunnuksen tallennettavaan tietoon (Esimerkki 45). Tree-käytösmääritys huolehtii lopusta. (CakePHP Manual 2010, 6.4.)

Esimerkki 45: Tallentaminen puu-hierarkiaan ohjaimessa.

```
$data['Category']['parent_id'] =  3;
$data['Category']['name'] =  'Skating';
$this->Category->save($data);
```

Tiedon muuttamiseen käytetään tietueen id-tunnusta (Esimerkki 46).

Esimerkki 46: Tiedon muuttaminen puu-hierarkiaan ohjaimessa.

```
$this->Category->id = 5;
$this->Category->save(array('name' =>'Fishing'));
```

4.3 Ohjainkerroksen laajennusosat eli komponentit

Komponentit ovat ohjainkerroksen laajennusosia, joita käytetään silloin, kun jotain ohjelmakoodia tarvitaan useassa eri ohjaimessa. Komponenttitiedostot sijoitetaan app/controllers/komponents/-hakemistoon. Nämä tiedostot nimetään komponentti-luokan nimen mukaan. Jos komponentti-luokan nimenä on MathComponent, se sijoitetaan tiedostoon nimeltä math.php. Komponentti-luokat perivät Object-luokan. CakePHP sisältää useita sisäänrakennettuja komponentteja (Taulukko 8). Lisäksi niitä voidaan luoda itse lisää. (CakePHP Manual 2010, 3.6.)

Taulukko 8: CakePHP:n sisäänrakennetut komponentit (CakePHP Manual 2010, 5).

Komponentin nimi	Kuvaus
Acl	Tarjoaa helppokäyttöisen rajapinnan Access Control List -toiminnon käyttöön.
Auth	Todennus-komponentti. Helpottaa huomattavasti käyttäjien kirjautumisen, rekisteröinnin ja käyttöoikeuksien hallintaa.
Cookie	Evästeiden hallintaan käytettävä komponentti.
Email	Sähköpostin lähettämiseen käytettävä komponentti.
RequestHandler	Tarjoaa lisäinformaatiota koskien asiakkaalta tulevia HTTP-pyyntöjä.
Security	Tarjoaa ”kovan luokan” tietoturvaominaisuuksia.
Session	PHP:n $_SESSION-muuttujan manipulointiin sekä sessiotiedon tallentamiseen käytettäviä funktioita.

Olen käyttänyt esimerkkisovelluksessani Auth-komponenttia, jonka avulla siis hallitaan käyttäjien kirjautumista ja käyttöoikeuksia. Komponentti otetaan käyttöön ohjaimessa esittelemällä components-taulukkomuuttuja. Tähän taulukkomuuttujaan annetaan komponentin nimi sekä mahdolliset määritykset sen toimintaan liittyen. Olen esitellyt Auth-komponentin App-ohjaimessa (Esimerkki 47), jolloin se on käytössä sovelluksen jokaisessa ohjaimessa. (CakePHP Manual 2010, 5.2.)

Esimerkki 47: Auth-komponentin käyttöönotto AppController-luokassa.

```
class AppController extends Controller {
    var $components = array('Auth');
```

Tämän jälkeen voimme lisätä Users-ohjaimeen kirjautumislogiikan, eli funktiot sisään- ja uloskirjautumiselle (Esimerkki 48). Kirjautumisfunktio ei tarvitse lainkaan ohjelmalogiikkaa sisäänsä. Uloskirjautuminen tapahtuu yksinkertaisesti Auth-komponentin logout-funktion avulla. (CakePHP Manual 2010, 5.2.)

Esimerkki 48: Kirjautumislogiikka Users-ohjaimessa.

```
function login(){
}
function logout(){
    $this->redirect($this->Auth->logout());
}
```

Kun olemme määritelleet edellämainitut määritykset ja funktiot, voimme luoda mallineen kirjautumiselle (Esimerkki 49). Sijoitamme sen tiedostoon app/views/users/login.ctp. Tähän mallineeseen luodaan Form-avustajalla lomake, johon käyttäjä syöttää kirjautumistietonsa. Oletuksena käytetään kenttiä username ja password. Lisäksi lomakkeen yläpuolelle voidaan sijoittaa $session->flash('auth')-käsky, joka tulostaa virheilmoituksen, mikäli käyttäjätunnusta tai salasanaa ei löydy tietokannasta. (CakePHP Manual 2010, 5.2.)

Esimerkki 49: app/views/users/login.ctp -tiedoston sisältö.

```
$session->flash('auth');
echo $form->create('User', array('action' => 'login'));
echo $form->input('username');
echo $form->input('password');
echo $form->end('Login');
echo $html->link('Register here', array('controller' => 'users',
                                        'action'     => 'add'));
```

Rekisteröintilogiikan rakentaminen on hieman haastavampaa, koska siihen käytämme myös mallissa annettuja validointimäärityksiä. Käytän rekisteröintiin Users-ohjaimen add-funktiota. Tähän funktioon olen sijoittanut rekisteröintiin liittyvän logiikan (Esimerkki 50). Tämän funktion lisäksi on muistettava rakentaa malline rekisteröintilomaketta varten tiedostoon app/views/users/add.ctp.

Esimerkki 50: Add-funktion tallennuslogiikka.

```
function add(){
    // Tarkistetaan, että lomakkeelta on lähetetty tietoa.
    if (!empty($this->data)){
        // Tarkistetaan että annetut salasanat ovat samat.
        if ($this->data['User']['password'] ==
                $this->Auth->password
                ($this->data['User']['password_confirm'])) {
            // Validointia varten luotava muuttuja.
            $validateData['User'] = $this->data['User'];
            // Kentän tyhjennys.
            unset($validateData['User']['password']);
            // Validointia varten annetaan käsittelemätön salasana.
            $validateData['User']['password'] =
                $this->data['User']['password_confirm'];
            $this->User->create();
            $this->User->set($validateData);
            // Tarkistetaan validoituuko syötteet.
            if($this->User->validates()){
                // Tallennetaan syötteet.
                $this->User->saveAll($this->data,
                                     array('validate' => false));
                // Kirjaudutaan sisään.
                $this->Auth->login($this->data);
                // Tulostetaan saved.ctp -malline.
                $this->render('saved');
            }
```

Kun käytetään Auth-komponenttia, rekisteröintilomakkeelta palautuva salasana on valmiiksi kryptattu. Tällöin joudumme vertaamaan password- ja password_confirm-kenttiä Auth-komponentin password-funktiota apuna käyttäen, joka palauttaa parametrinä annetun merkkijonon kryptattuna. (CakePHP Manual 2010, 5.2.5.) Validoitaessa emme voi käyttää suoraan data-muuttujaa, sillä se sisältää kryptatun salasanan, vaan on luotava uusi muuttuja, jonka rakenne on data-muuttujan kaltainen, ja joka sisältää kryptaamattoman salasanan. Jos validointi menee läpi, tallennamme data-muuttujan tiedot tietokantaan ja kirjaamme rekisteröityneen käyttäjän sisään Auth-komponentin login-funktiolla, joka saa parametrikseen HTML-lomakkeelta palautetun data-muuttujan.

Lisäksi on muistettava, että Auth-komponentti rajoittaa kirjautumattomien käyttäjien oikeuksia. Voimme asettaa add-funktion kirjautumattomien käyttäjien käytettäväksi ohjaimen beforeFilter-funktiossa (Esimerkki 51). Siellä käytämme Auth-komponentin allow-funktiota, joka saa parametrikseen niiden funtioiden nimet, joita asetukset koskevat. (CakePHP Manual 2010, 5.2.5.)

Esimerkki 51: Auth-komponentin allow-funktion käyttö ohjaimessa.

```
function beforeFilter() {
    $this->Auth->allow('add', 'getNavigation');
}
```

Auth-komponentti tarjoaa lisäksi tietoa kirjautuneesta käyttäjästä. Tämä tieto on usein hyvin tarpeellista, sillä sitä voidaan käyttää kirjautumisen voimassaolon tarkistamiseen tai tietyissä tilanteissa käyttäjän oikeuksien tarkistamiseen. (CakePHP Manual 2010, 5.2.5.) Esimerkkisovelluksessani on funktio, jonka avulla käyttäjä voi muokata luomaansa tehtävää. Tässä funktiossa tarvitaan tarkistusta siitä, onko muokattava tehtävä kirjautuneen käyttäjän omistuksessa (Esimerkki 52).

Esimerkki 52: Assignments-ohjaimen edit-funktion muokkausoikeuksien tarkistus.

```
function edit($assignmentId){
    if($this->Assignment->authorizedToEdit($assignmentId,
                                        $this->Auth->user('id'))){
```

Assignment-mallin authorizedToEdit-funktio palauttaa boolean-arvon false, mikäli käyttäjän id-tunnus ei ole sama kuin assignments-taulussa sijaitsevan tietueen (johon viitataan assignmentId-muuttujalla) user_id-kentän arvo.

5 Muutamia ominaisuuksia tarkemmin

Seuraavaksi pohdin requestAction-toiminnon käyttämistä MVC-arkkitehtuurin periaatteiden pohjalta. Lisäksi tutustutan lukijan syvemmin AJAX-toiminnallisuuteen ja sen toteuttamiseen CakePHP:n avulla. Olen esitellyt käsiteltäviä ominaisuuksia pintapuolisesti jo aiemmissa luvuissa.

5.1 RequestAction

Kuten mainitsin kappaleessa 4.1.2 Elementit, requestAction-toiminnolla voidaan kutsua näkymäkerroksesta ohjainkerroksen funktioita. Tämä sotii vastaan MVC-arkkitehtuurin periaatetta, jonka mukaan sovelluksen tieto kulkee näkymäkerrosta kohti. RequestAction-toiminnon käytössä tuleekin noudattaa varovaisuutta, sillä se hidastaa sovelluksen toimintaa, rikkoo mahdollisesti MVC-arkkitehtuuria ja vaikeuttaa virheidenetsintää. (Geisendörfer 2008.)

RequestAction-toiminnolle on muitakin mahdollisia käyttötarkoituksia, kuin kutsua ohjaimen funktioita näkymäkerroksesta. Internetiä selaamalla löysin monia tilanteita, jolloin, sovelluskehittäjän näkökulmasta, requestAction-toiminnon käyttäminen olisi houkuttelevaa. Yksi esimerkkitapauksista on sellainen, jossa tarvitaan tietoa mallista X, jota ei ole määritelty suhteeseen mallin Y kanssa, eikä sitä siis voida käyttää mallin Y ohjaimen kautta. Jos käytämme tässä tapauksessa requestAction-toimintoa, eli kutsumme X-ohjaimen funktiota Y-ohjaimen kautta, verkkosovellukseemme kohdistuu kokonaan uusi HTTP-pyyntö, jolloin verkkosivu ladataan palvelimelta uudestaan. Tämänkaltainen toiminta ruuhkauttaa verkkopalvelinta ja saa verkkosovelluksen toimimaan hitaammin. (Story 2008.)

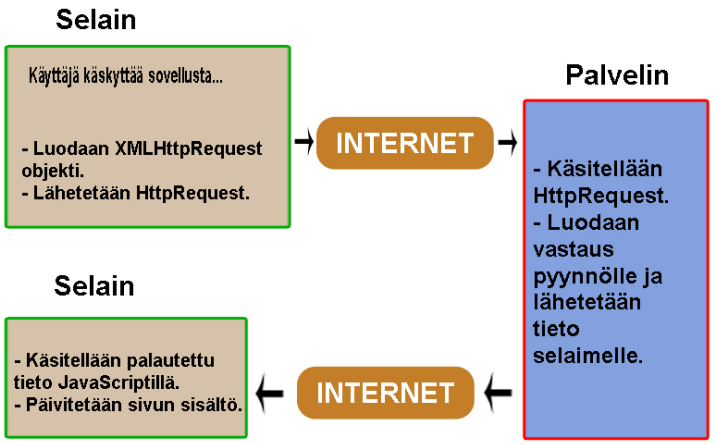
Hyvä, helppo ja käytännöllinen ratkaisu edellä esitettyyn ongelmaan olisi esitellä Y-ohjaimeen uses-muuttuja, joka saa arvokseen mallin X. Tällöin pääsemme vaivattomasta käsiksi malliin X, emmekä siis joudu turvautumaan requestAction-toimintoon. (Story 2008.)

Sovelluskehittäjän tulisi tutkia tarkoin vaihtoehtoisia toteutusmalleja, ennen requestAction-toiminnon käyttämistä. Vaikka CakePHP sisältääkin kyseisen toiminnon, se ei silti kannusta sen käyttöön (CakePHP Manual 2010, 3.5.5.4.6). Edellä mainitun esimerkin

kaltaisia tapauksia, joihin löytyy vaivattomasti requestActionia parempi ratkaisu, on monia. Suosittelen ohjelmistokehittäjiä tutkimaan ensin tarkoin CakePHP:n käyttöohjetta sekä muiden ohjelmistokehittäjien huomioita, joita löytyy Internetistä helposti vaikka Googlen kautta. Ellei ratkaisua ongelmaan löydy, saattaa requestAction-toiminnon käyttö olla perusteltua.

5.2 AJAX-toiminnallisuus

AJAX (Asynchronous JavaScript and XML) on tekniikka dynaamisen sisällön nopeaan esittämiseen verkkosovelluksessa. AJAX päivittää verkkosivustoa asiakkaan selaimeen niin, ettei koko sivua tarvitse ladata uudestaan palvelimelta. Se lisää kommunikointia palvelimen ja asiakaskoneen välillä (Kuva 7) niin, että pieniä määriä tietoa latautuu asiakaskoneen selaimeen tahdistamattomasti ja käyttäjän huomaamatta. AJAX on usean vanhemman verkkosovellustekniikan, kuten JavaScript, (X)HTML, XML, DOM ja XMLHttpRequest, yhteiskäyttöä. AJAX-toiminnallisuutta varten asiakaskoneen selaimessa täytyy olla AJAX-tuki. Tämä tuki löytyykin poikkeuksetta kaikista uusimmista selaimista. (W3Schools 2010.)



Kuva 7: AJAX-toimintaperiaate selaimen ja palvelimen välillä.

CakePHP tarjoaa kattavat toiminnot AJAX:n käyttöön. Ensisijaisesti AJAX-toiminnallisuutta käsitellään Ajax-avustajan kautta, joka käyttää Prototype- ja Scriptaculous-JavaScript-kirjastoja. Lisäksi on mahdollista ladata sovelluksen käyttöön jokin CakePHP:tukemista JavaScript-kirjastoista Js-avustajan avulla. Tuettuja JavaScript-kirjastoja ovat:

Prototype/Scriptaculous, Mootools/Mootools-more ja jQuery/jQuery UI. (CakePHP Manual 2010, 7.5.)

AJAX-toiminnallisuuteen liittyy läheisesti myös RequestHandler-komponentti, joka tarjoaa lisätietoa asiakkaan Http-pyynnöistä. Sen avulla saadaan selville minkä tyyppinen Http-pyyntö asiakaskoneelta välittyy palvelimelle. RequestHandler-komponentin isAjax-funktio palauttaa boolean-arvon true, mikäli Http-pyynnöstä löytyy X-Requested-Header. X-Requested-Header liitetään Http-pyyntöön, mikäli Http-pyyntö koskee AJAX-toiminnallisuutta. (CakePHP Manual 2010, 5.5.1.)

6 Esimerkkisovellus

Esittelen lyhyesti verkkosovelluksen, jonka olen toteuttanut osana opinnäytetyötäni. Olen käyttänyt sovelluksessa opinnäytetyöraportissani esittelemiä toimintoja ja toteuttanut sen MVC-arkkitehtuurin mukaisesti. Sovelluksen ohjelmakoodi on liitteenä opinnäytetyöraporttini lopussa (Liite 1). En koe tarpeelliseksi esitellä ohjelmakoodia sen tarkemmin, sillä kaikki toiminnot vastaavine ohjelmakoodiesimerkkeineen on jo esiteltynä opinnäytetyöraporttini aiemmissa luvuissa.

6.1 Käyttötarkoitus

Sovellus toimii muistilistan tavoin. Käyttäjä voi lisätä sovelluksen tietokantaan tehtäviä, joille annetaan erääntymispäivämäärä sekä tehtävän tyyppi. Käyttäjän kirjautuessa sisään, hänelle näytetään listaus omista tehtävistään päivämäärän mukaan lajiteltuna. Kun käyttäjä on suorittanut tehtävän, hän yksinkertaisesti poistaa sen tietokannasta.

Käyttäjä voi lisäksi muokata tallennettuja tehtäviään, selata tehtäviään tehtävätyypin mukaan ja tarkastella profiiliaan.

6.2 Suunnittelu ja toteutus

Olen ottanut jo suunnitteluvaiheessa huomioon MVC-arkkitehtuurin mukaisen tietovirran. Hankalinta suunnittelussa ja toteutuksessa on mallin ja ohjaimen eron ymmärtäminen. Ohjaimeen tulee helposti kirjoitettua sellaista ohjelmakoodia, joka MVC-arkkitehtuurin mukaan kuuluisi mallikerrokseen.

Olen havainnut hyväksi periaatteeksi nk. ”Fat models, skinny controllers” -lähestymistavan, joka nimensä mukaisesti sisällyttää mahdollisimman paljon toimintalogiikkaa mallikerrokseen (Bernat 2008). Koska malleihin pääsee helposti käsiksi ohjainkerroksesta, joko suhdemäärittelyjen tai uses-muuttujan kautta, on perusteltua suorittaa mahdollisimman paljon logiikkaa mallikerroksessa, jolloin ohjaimiin tulee vähemmän toistuvaa ohjelmakoodia.

6.3 CakePHP:n tekniikat sovelluksessa

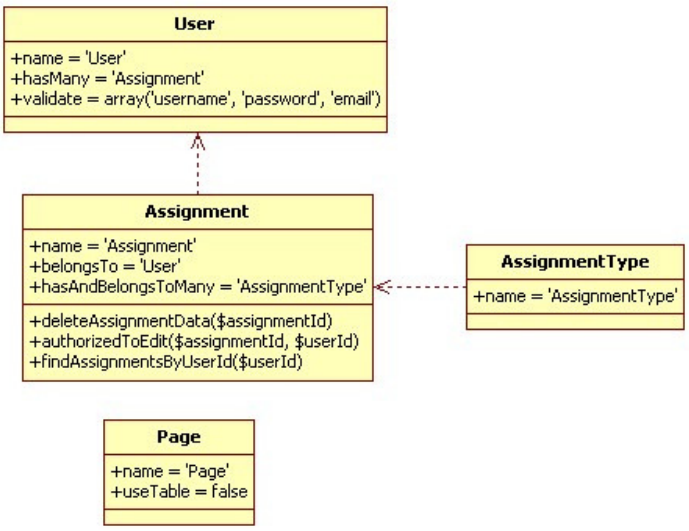
Olen käyttänyt työssäni useita näkymäkerroksen avustajia. Olen esitellyt avustajat App-Controller-luokassa, jolloin ne ovat käytössä kaikissa sovelluksen näkymätiedostoissa. Avustajien yhteiskäytössä en havainnut ongelmia, vaan ne toimivat kuten pitääkin.

Käytetyt avustajat: Html, Form, Paginator, Text, Time ja Ajax.

Lisäksi käytin navigointipalkin tulostavaa elementtiä. Kyseisestä elementistä on esittely kappaleessa 4.1.2 Elementit.

Ohjainkerroksen komponenteista sovellus käyttää vain Auth- ja RequestHandler-komponentteja. Komponentit, kuten avustajatkin, on esitelty AppController-luokassa, ja ovat täten käytettävissä koko ohjainkerroksessa.

Sovellus käyttää CakePHP:n validointitoimintoa käyttäjäprofiilin rekisteröinnin yhteydessä. Sovelluksen tietokantaa käyttävät mallit ovat määritelty suhteeseen keskenään (Kuva 8).



Kuva 8: Esimerkkisovelluksen malli-luokat

7 Johtopäätös

Verkkosovelluksia toteutetaan yhä kiihtyvällä tahdilla ja Internetin alkuaikojen sekasorrosta (selainsodat) on päästy yhteneväisempään toteutusmalliin. Standardit näyttelevät nykyään suurta osaa verkkosovelluskehityksessä. Standardien omaksumisen jälkeen on kuitenkin vielä pitkä matka ylläpidettävään, skaalautuvaan ja uudelleenkäytettävään verkkosovelluskehitykseen. Tässä vaiheessa maailmanlaajuiselle www-näyttämölle astuvat ohjelmointikehykset, jotka tarjoavat sovelluskehittäjille valmiin sovellusrakenteen ja ohjelmointikäytännön. Lisäksi ne helpottavat sovelluksen ohjelmointia niin paljon, että verkkosovelluksen ohjelmoiminen täysin omin voimin tuntuisi lähes uuvuttavalta.

Tällainen kehityssuunta tuo mieleeni uhkakuvia tulevaisuudesta. Tekevätkö ohjelmointikehykset verkkosovelluskehittämisen liiankin helpoksi? Syntyykö niiden käytön seurauksena enemmän laajoja verkkosovelluksia, joiden toteuttajilla ei ole ymmärrystä sovelluksen varsinaisesta toiminnasta kehyksen takana? Uskon, että ohjelmointikehyksiin luottaminen saattaa hyvinkin aiheuttaa välinpitämättömyyttä sitä logiikkaa kohtaan, mikä kehyksen toimintaan sisältyy. Jo tietokantahakujen ohjelmointi PHP:llä on hyvinkin haastavaa verrattuna siihen, että annamme ohjelmointikehykselle yhden komennon, jolla saamme halutun tiedon kannasta.

Pidän hyvin tärkeänä sitä, että sovelluskehittäjät eivät kannattele sovellustaan liiaksi ohjelmointikehyksen varassa. Kehykset, kuten CakePHP, tarjoavat kiistämättä hyvin toteutettuja ratkaisuja verkkosovelluskehittämisen helpottamiseksi. Tulevaisuudessa verkkosovelluskehittäjiltä vaaditaan yhä enemmän luottamusta omaan osaamiseen ja tarmoa sen kartuttamiseen. Ei tule tuudittautua siihen harhakuvitelmaan, että siitä mikä tapahtuu kulissien takana ei tarvitsisi välittää.

Olen tutustunut tähän mennessä vain CakePHP-ohjelmointikehykseen, joten en voi ottaa kantaa siihen, miten muiden ohjelmointikehysten dokumentointi on toteutettu. Antti Niemenpää, joka vertaili opinnäytetyössään eri ohjelmointikehyksiä, luokitteli CakePHP:n dokumentoinnin keskinkertaiseksi (Niemenpää 2008, 18). CakePHP:sta löytyy sen kehittäjien ylläpitämä käyttöohje (CakePHP Manual) sekä lisäksi kuvaukset kaikista luokista, niiden funktioista, tiedostoista ja paketeista (CakePHP API). Nämä ovat ainoat ”viralliset” käyttöohjeenomaiset lähteet ja ovat paikoin varsin puutteelliset. Monista toi-

minnoista on annettu hyvin yksipuolinen ja epäselvä kuvaus. Onneksi, CakePHP:n suosion myötä, on myös paljon epävirallista tietoa, joka usein on kattavampaa ja paremmin perusteltua kuin CakePHP:n tarjoama.

Yksi suurista eduista CakePHP-ohjelmointikehyksessä on sen lähdekoodin avoimuus. Verkkosovelluksen kehittäjä, joka käyttää CakePHP:tä, voi vapaasti muokata ohjelmointikehyksen toimintaa. Lisäksi ohjelmakoodin avoimuus asettaa ohjelman helpommin alttiiksi kritiikille. Parhaimpana arvostelupaikkana toimii tietenkin Internet, jonka verkkopäiväkirjat eli blogit ovat täynnä ohjelmistokehittäjien kommentteja käyttämästään ohjelmistosta. Tällainen kehitys on mielestäni hyvin tervetullutta ja avaa uusia ovia ohjelmistokehittäjien väliltä.

Opinnäytetyötä aloittaessa olin jo käyttänyt CakePHP-ohjelmointikehystä työharjoittelussa ollessani. Tästä saatu perustieto auttoi minua paljon, sillä en joutunut täysin uusien periaatteiden ja ohjelmointikäytäntöjen maailmaan. Työn lomassa opin hyvin paljon uutta, niin CakePHP:stä, kuin MVC-arkkitehtuuristakin. Uskon, että ohjelmointikehyksillä tulee olemaan tärkeä rooli tulevaisuuden työntekoani ajatellen. Monesti heräsi myös halu oman MVC-arkkitehtuurille perustuvan ohjelmointikehyksen toteuttamiseen.

Mielestäni olisi hyvä, jos tietojenkäsittelyn koulutusohjelmaan liittyisi pienimuotoinen opintokokonaisuus, joka käsittelisi ohjelmointikehyksiä. Kehykset ovat kuitenkin nykypäivää ja niitä käytetään ahkerasti. IT-ala on ollut koko uuden vuosituhannen jatkuvassa murrostilassa ja siksi on mielestäni tärkeää, että koulutus pysyy kehityksessä mukana.

Lähteet

Bernat, Mike 2008. MVC - Fat Models and Skinny Controllers. [online] [viitattu 1.4.2010] http://www.mikebernat.com/blog/MVC_-_Fat_Models_and_Skinny_Controllers_

CakePHP 1.3.x API 2010. [online] [viitattu 9.3.2010] http://api13.cakephp.org/

CakePHP Manual 2010. [online] [viitattu 25.3.2010]. http://book.cakephp.org/

Cui, Wei & Huang, Lin & Liang, LiJing & Li, Jing 2009. The Research of PHP Development Framework Based on MVC Pattern. [online] [viitattu 4.2.2010] http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5369976

Geisendörfer, Felix 2008. requestAction considered harmful. [online] [viitattu 23.3.2010] http://debuggable.com/posts/requestaction-considered-harmful:48abb514-1f9c-4443-b91c-6d0f4834cda3

Grant, Paul 2000. Data Validation - and More. [online] [viitattu 5.3.2010] http://www.questanalytical.com/Articles/data_validation.htm

Kost, Stephen 2004. An Introduction to SQL Injection Attacks for Oracle Developers. [online] [viitattu 1.3.2010] http://www.integrigy.com/security-resources/whitepapers/Integrigy_Oracle_SQL_Injection_Attacks.pdf

McArthur, Kevin 2008. Pro PHP: Patterns, Frameworks, Testing and More. [online] [viitattu 10.2.2010] http://books.google.fi/books?id=5IRcTr13SBcC&lpg=PA1&ots=OSm-zw9sjF&dq=prophp%20pt4&pg=PA1#v=onepage&q&f=false

Niemenpää, Antti 2008. MVC-ohjelmistokehys Internet-ohjelmoinnissa. Opinnäytetyö. Tampere.

O'Brien, Duane 2009. Cook up Websites fast with CakePHP, Part1. [online] [viitattu 1.2.2010] http://www.ibm.com/developerworks/opensource/tutorials/os-php-cake1/index.html

The Open Source Definition 2010. [online] [viitattu 8.3.2010] http://www.opensource.org/docs/definition.php

Reenskaug, Trygve 1979. MVC XEROX PARC 1978-79. [online] [viitattu 25.3.2010] http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

Story, Mark 2008. Reducing requestAction() use in your CakePHP sites with fat models. [online] [viitattu 30.3.2010] http://mark-story.com/posts/view/reducing-requestaction-use-in-your-cakephp-sites-with-fat-models

W3Schools 2010. AJAX Introduction. [online] [viitattu 30.3.2010] http://www.w3schools.com/Ajax/ajax_intro.asp

Liitteet

Liite 1: Esimerkkisovelluksen lähdekoodi.

Mallit

```
<?php
/**
 * Model for user.
 */
class User extends AppModel {
    var $name = 'User';
    var $hasMany = array(
            'Assignment' => array(
                'className'  => 'Assignment',
                'foreignKey' => 'user_id',
                'dependent'  => true)
            );
    var $validate = array(
            'username'    => array(
                'unique' => array(
                    'rule'    =>'isUnique',
                    'message' => 'Username already taken'),
                'notEmpty'    => array(
                    'rule'       => 'alphaNumeric',
                    'allowEmpty' => false,
                    'message'    => 'Give username')),
            'password' => array(
                'rule' => array('between', 5, 15),
                'message' => 'Password must be between 5 to 15
                              characters'),
            'email' => 'email');
}
?>
```

```
<?php
/**
 * Model for page.
 */
class Page extends AppModel {
    var $name = 'Page';
    var $useTable = false;
}
?>
```

```
<?php
/**
 * Model for Assignment type.
 */
class AssignmentType extends AppModel {
    var $name = 'AssignmentType';
}
?>
```

```
<?php
/**
 * Model for Assignments
 */
class Assignment extends AppModel {
    var $name = 'Assignment';
    var $belongsTo = array(
            'User' => array(
                'className'  => 'User')
            );
    var $hasAndBelongsToMany = array(
        'AssignmentType' => array(
              'joinTable'             => 'assignment_type_list'
            , 'foreignKey'            => 'assignment_id'
            , 'associationForeignKey' => 'assignment_type_id'
            , 'className'             => 'AssignmentType'
            , 'dependent'             => true
            )
        );
    /**
     * Checks if the logged-in user
     * has authorization to edit the assignment
     *
     * @access    private
     * @author    Toni Sissala    <toni.sissala@cs.tamk.fi>
     * @return    boolean         true if authorized.
     */
    function authorizedToEdit($assignmentId, $userId){
        $assignment = $this->findById($assignmentId);
        if($assignment['User']['id'] == $userId){
            $result = true;
        } else {
            $result = false;
        }
        return $result;
    }
    /**
     * Finds assignments by user id.
     *
     * @access    private
     * @author    Toni Sissala    <toni.sissala@cs.tamk.fi>
     * @return    array           assignment data.
     */
    function findAssignmentsByUserId($userId){
        $conditions = array(
                'conditions' =>
                    array('Assignment.user_id' => $userId),
                'order'      =>
                    array('Assignment.exp_date DESC'));
        return $this->find('all', $conditions);
    }
    /**
     * Gets assignment ids by assignment type id.
     *
     * @access    private
     * @author    Toni Sissala        <toni.sissala@cs.tamk.fi>
     * @return    Integer             assignment ids
     */
    function getAssignmentIdsByAssignmentTypes($assignmentTypeId){
        $index = 0;
        $assignmentIds = $this->AssignmentTypeList->
            findAllByAssignmentTypeId($assignmentTypeId);
```

```
        foreach($assignmentIds as $ids){
            $tmp[$index] =
                $ids['AssignmentTypeList']['assignment_id'];
            $index++;
        }
        return $tmp;
    }
}
?>
```

Ohjaimet

```
<?php
/**
 * Application wide controller.
 */
class AppController extends Controller {

    var $helpers = array('Html', 'Form', 'Time',
                         'Text', 'Ajax', 'Paginator');
    var $components = array('Auth', 'RequestHandler');
    /**
     * Executed before every page load.
     */
    function beforeFilter(){
        parent::beforeFilter();

        // Redirects after login.
        $this->Auth->loginRedirect =
            array('controller' => 'assignments',
                  'action'     => 'listAssignments');

        // Redirects after logout.
        $this->Auth->logoutRedirect =
            array('controller' => 'pages',
                  'action'     => 'index');
    }
}
?>
```

```
<?php
/**
 * Controller for Assignments.
 */
class AssignmentsController extends AppController {

    var $paginate = array('Assignment' => array(
                              'limit' => 8,
                              'order' => array(
                                  'Assignment.exp_date' => 'desc')));

    var $uses = array('Assignment'
                    , 'AssignmentType'
                    , 'AssignmentTypeList');
    /**
     * Add assignment to database or
```

```
     * print form for assignment data.
     *
     * @access    public
     * @author    Toni Sissala    <toni.sissala@cs.tamk.fi>
     */
    function add(){
        if(!empty($this->data)){
            // Saving logic.

            $this->Assignment->saveAll($this->data);
            $this->set('assignmentId', $this->Assignment->id);
            $this->render('saved');
        }else{
            // Printing the form.

            $this->set('userId', $this->Auth->user('id'));
            $this->set('assignmentTypes',
                    $this->AssignmentType->find('list'));
        }
    }
    /**
     * Shows the assignment or
     * redirects to listAssignments-function.
     *
     * @access    public
     * @param     Integer         $assignmentId       assignment id
     * @author    Toni Sissala    <toni.sissala@cs.tamk.fi>
     */
    function show($assignmentId){
        if($this->Assignment->authorizedToEdit($assignmentId,
                                        $this->Auth->user('id'))){
            $this->set('assignment', $this->Assignment->findById(
                                        $assignmentId));
        } else {
            $this->redirect(array(
                            'controller' => 'assignments',
                            'action'     => 'listassignments'));
        }
    }
    /**
     * Lists the assignments by logged-in user and assignment types.
     *
     * @access    public
     * @param     Integer / NULL     $currentTypeId      Id of the
     *                                                   currently
     *                                                   viewed
     *                                                   assignment
     *                                                   type.
     * @author    Toni Sissala       <toni.sissala@cs.tamk.fi>
     */
    function listAssignments($currentTypeId = null){
        if(!empty($this->data) || !empty($currentTypeId)){
            if(!empty($this->data)){
                $assignmentTypeId =
                $this->data['Assignment']['AssignmentTypes'];
            } else {
                $assignmentTypeId = $currentTypeId;
            }
            if($assignmentTypeId != 0){
                $assignmentIds =
                $this->Assignment->
                getAssignmentIdsByAssignmentTypes($assignmentTypeId);
```

```
                $this->set('assignments',
                    $this->paginate('Assignment', array(
                    'Assignment.user_id' => $this->Auth->user('id'),
                    'Assignment.id'      => $assignmentIds)));
                $this->set('currentTypeId', $assignmentTypeId);
            } else {
                $this->set('assignments',
                $this->paginate('Assignment',
                array('Assignment.user_id' =>
                    $this->Auth->user('id'))));
            }
        } else {
            $this->set('assignments',
                $this->paginate('Assignment',
                array('Assignment.user_id' =>
                    $this->Auth->user('id'))));
        }
        $assignmentTypes = $this->AssignmentType->find('list');
        $assignmentTypes[0] = "";
        $this->set('assignmentTypes', $assignmentTypes);
    }
    /**
     * Deletes the assignment or redirects to
     * listAssignments-function.
     *
     * @access    public
     * @param     Integer         $assignmentId       Assignment id.
     * @author    Toni Sissala    <toni.sissala@cs.tamk.fi>
     */
    function delete($assignmentId){
        if($this->Assignment->authorizedToEdit($assignmentId,
                                        $this->Auth->user('id'))){
            $this->Assignment->delete($assignmentId);
        } else {
            $this->redirect(array(
                            'controller' => 'assignments',
                            'action'     => 'listassignments'));
        }

    }
    /**
     * Edit the assignment.
     *
     * Prints the edit form or saves the edited data.
     * If user is not authorized,
     * redirects to listAssignments-function.
     *
     * @access    public
     * @param     Integer         $assignmentId       assignment id.
     * @author    Toni Sissala    <toni.sissala@cs.tamk.fi>
     */
    function edit($assignmentId){

        if($this->Assignment->authorizedToEdit($assignmentId,
                                        $this->Auth->user('id'))){
            if(empty($this->data)){
                // Prints the form.

                $this->data = $this->Assignment->findById(
                                                $assignmentId);
                $this->set('assignmentTypes',
                           $this->AssignmentType->find('list'));
```

```
            }else{
                // Saving logic.

                $this->data['Assignment']['id'] = $assignmentId;
                $this->Assignment->saveAll($this->data);
                $this->set('assignmentId', $assignmentId);
                $this->render('saved');
            }
        } else {
            $this->redirect(array(
                            'controller' => 'assignments',
                            'action'     => 'listassignments'));
        }



    }
}
?>
```

```
<?php
/**
 * Controller for pages.
 */
class PagesController extends AppController {
    /**
     * CakePHP provided beforefilter.
     *
     * @access   private
     * @author   Toni Sissala    <toni.sissala@cs.tamk.fi>
     */
    function beforeFilter(){
        $this->Auth->allow('index');
    }
    /**
     * Prints the homepage.
     *
     * @access   public
     * @author   Toni Sissala   <toni.sissala@cs.tamk.fi>
     */
    function index(){
        $this->render('home');
    }
}
?>
```

```
<?php
/**
 * Controller for users.
 */
class UsersController extends appController{
    /**
     * CakePHP provided beforefilter.
     *
     * @access   private
     * @author   Toni Sissala    <toni.sissala@cs.tamk.fi>
```

```
     */
    function beforeFilter() {
        $this->Auth->allow('add', 'getNavigation');
    }
    /**
     * Renders the correct navigation element.
     *
     * @access    public
     * @author    Toni Sissala    <toni.sissala@cs.tamk.fi>
     */
    function getNavigation(){
        if($this->Auth->user('id')){
            // If logged in.
            $this->set('username', $this->Auth->user('username'));
            $this->render('../elements/usersnavigation');
        } else {
            // if not logged in.
            $this->render('../elements/defaultnavigation');
        }
    }
    /**
     * Registers a new user.
     *
     * If passwords and validation match,
     * saves the user data to database.
     *
     * @access    public
     * @author    Toni Sissala    <toni.sissala@cs.tamk.fi>
     */
    function add(){
        if (!empty($this->data)){
            if ($this->data['User']['password'] ==
                    $this->Auth->password
                    ($this->data['User']['password_confirm'])) {
                // if confirm-password = password.

                $validateData['User'] = $this->data['User'];
                unset($validateData['User']['password']);
                $validateData['User']['password'] =
                    $this->data['User']['password_confirm'];
                $this->User->create();
                $this->User->set($validateData);
                if($this->User->validates()){
                    // if validation checks ok.
                    $this->User->saveAll($this->data,
                                    array('validate' => false));
                    $this->Auth->login($this->data);
                    $this->render('saved');
                } else {
                    unset($this->data['User']['password']);
                    unset($this->data['User']['password_confirm']);
                }

            } else {
                unset($this->data['User']['password']);
                unset($this->data['User']['password_confirm']);
                $this->set('info',
                           'The given passwords don\'t match');
            }
        }

    }
```

```
    /**
     * Login function. Empty when using auth-component.
     *
     * @access    public
     * @author    Toni Sissala    <toni.sissala@cs.tamk.fi>
     */
    function login(){
    }
    /**
     * Logout function. Functionality by auth-component.
     *
     * @access    public
     * @author    Toni Sissala    <toni.sissala@cs.tamk.fi>
     */
    function logout(){
        $this->redirect($this->Auth->logout());
    }
    /**
     * Prints the logged-in users profile.
     *
     * @access    public
     * @author    Toni Sissala    <toni.sissala@cs.tamk.fi>
     */
    function profile(){
        $this->set('userData',
                    $this->User->findById($this->Auth->user('id')));
    }
}
?>
```

Näkymät

```
<!--Default layout-file-->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fin" lang="fin">
<head>
<title><?php echo $title_for_layout; ?></title>
<?php
    if(!empty($ajax)){
        echo $javascript->link('prototype');
    }
?>
<?php echo $html->css('screen'); ?>
</head>
<body>
    <!--Div-tag for header-->
    <div class="header">
        <?php echo $html->image("header2.jpg", array(
        "alt" => "Header",
        'url' => array('controller' => 'pages',
                       'action'     => 'index'))); ?>
    </div>
    <!--Div-tag used as a wrapper-->
    <div class="content">
        <!--Div-tag for page-spesific content-->
        <div class="content-left">
            <!--Div-tag for indentation of text-->
              <div class='indentation'>
                 <?php echo $content_for_layout; ?>
              </div>
        </div>
        <!--Div-tag for left side of content-->
          <div class="content-right">
             <!--Div-tag for navigation header-->
              <div class="header-right">
                 <?php echo $html->image("navi-header.jpg",
                             array("alt" => "Header")); ?>
              </div>
             <!--Div-tag for navigation-->
              <div class="navigation">
                 <?php echo $this->element('navigation'); ?>
              </div>
        </div>
    </div>
    <!--Div-tag for footer-->
    <div class="footer">
        <?php echo $html->image("footer2.jpg", array(
            "alt" => "Header")); ?>
          <p class="footer-text">&copy; Toni Sissala 2010</p>
    </div>
</body>
</html>
```

```
<!--navigation.ctp element-file-->
<?php
echo $this->requestAction(array('controller' => 'users'
                              , 'action'     => 'getnavigation'),
                          array('return'));
?>
```

```
<!--usersnavigation.ctp element-file-->
<?php
echo '<ul><li>';
echo $html->link('Home', array('controller' => 'pages',
                               'action'     => 'index'));
echo '</li><li>';
echo $html->link('Profile', array('controller' => 'users',
                                  'action'     => 'profile'));
echo '</li><li>';
echo $html->link('List assignments',
                 array('controller' => 'assignments',
                       'action'     => 'listassignments'));
echo '</li><li>';
echo $html->link('Add assignment',
                 array('controller' => 'assignments',
                       'action'     => 'add'));
echo '</li><li>';
echo $html->link('Log Out ('.$username.')',
                 array('controller' => 'users',
                       'action'     => 'logout'));
echo '</li></ul>';
?>
```

```
<!--defaultnavigation.ctp element-file-->
<?php
echo '<ul><li>';
echo $html->link('Home', array('controller' => 'pages',
                               'action'     => 'index'));
echo '</li><li>';
echo $html->link('Login', array('controller' => 'users',
                                'action'     => 'login'));
echo '</li></ul>';
?>
```

```
<!--pages/home.ctp view-template-->
<h1>Home</h1>
```

```
<!--users/add.ctp view-template-->
<h1>Register</h1>
<div class="form">
<?php
if(!empty($info)){ echo $info; }
echo $form->create('User');
echo $form->input('User.username');
echo $form->input('User.password');
echo $form->input('User.password_confirm',
                  array('type' => 'password'));
echo $form->input('User.email');
echo $form->input('User.first_name');
echo $form->input('User.last_name');
```

```
echo $form->input('User.birth_date',
                  array('dateFormat' => 'DMY',
                        'minYear'    => date('Y')-70,
                        'maxYear'    => date('Y')));
?>
</div>
<?php echo $form->end('Register'); ?>
```

```
<!--users/login.ctp view-template-->
<h1>Login</h1>
<div class="form">
<?php
    $session->flash('auth');
    echo $form->create('User', array('action' => 'login'));
    echo $form->input('username');
    echo $form->input('password');
?>
</div>
<?php
    echo $form->end('Login');
    echo $html->link('Register here', array('controller' => 'users',
                                            'action'     => 'add'));
?>
```

```
<!--users/profile.ctp view-template-->
<h1>Profile</h1>
<?php
echo '<table border="1">
    <tr>
        <th colspan="2">'.$userData['User']['username'].'</th>
    </tr>';

echo $html->tableCells(array(
        array('First name:', $userData['User']['first_name']),
        array('Last name:', $userData['User']['last_name']),
        array('Birth date:', $time->nice($userData['User']
['birth_date'])),
        array('E-mail:', $userData['User']['email'])));
echo '</table>';
?>
```

```
<!--users/saved.ctp view-template-->
<h1>Profile saved</h1>
<?php echo $html->link(
    'Profile',
    array('controller'=>'users', 'action'=>'profile')); ?>
```

```
<!--assignments/edit.ctp view-template-->
<div class="form">
<?php
echo $form->create('Assignment');
echo $form->input('Assignment.name');
echo $form->textarea('Assignment.description',
                     array('label' => 'Description',
                           'rows'  => '8',
                           'cols'  => '38'));
echo $form->input('Assignment.exp_date',
```

```
                  array('label'      => 'Expiration date',
                        'dateFormat' => 'DMY',
                        'minYear'    => date('Y'),
                        'maxYear'    => date('Y')+70));
echo $form->input('AssignmentType');
?>
</div>
<?php echo $form->end('Save'); ?>
```

```
<!--assignments/add.ctp view-template-->
<div class="form">
<?php
echo $form->create('Assignment');
echo $form->input('Assignment.name');
echo $form->textarea('Assignment.description',
                     array('label' => 'Description',
                           'rows'  => '8',
                           'cols'  => '38'));
echo $form->input('Assignment.exp_date',
                  array('label'      => 'Expiration date',
                        'dateFormat' => 'DMY',
                        'minYear'    => date('Y'),
                        'maxYear'    => date('Y')+70));
echo $form->input('AssignmentType');
echo $form->hidden('Assignment.user_id', array('value' => $userId));
?>
</div>
<?php echo $form->end('Send'); ?>
```

```
<!--assignments/delete.ctp view-template-->
<h1>Deleted successfully</h1>
<?php
echo $html->link('List assignments',
                 array('controller' => 'assignments',
                       'action'     => 'listassignments'));
?>
```

```
<!--assignments/listassignments.ctp view-template-->
<div id="pagination">
<h1>List assignments</h1>
<?php
echo $html->link('Add assignment',
                 array('controller' => 'assignments',
                       'action'     => 'add'));

echo $form->create('Assignment',
                   array('action' => 'listassignments'));
echo $form->input('AssignmentTypes');
echo $form->end('Search');

if(!empty($currentTypeId)){
    $paginator->options(array('update' => 'pagination',
                              'url'    => $currentTypeId));
}
$paginator->options(array('update' => 'pagination'));
?>

<table border="1">
```

```
<tr>
<th><?php echo $paginator->sort('Name', 'Assignment.name'); ?></th>
<th><?php echo $paginator->sort('Description',
                                'Assignment.description'); ?></th>
<th><?php echo $paginator->sort('Expiration date',
                                'Assignment.exp_date'); ?></th>
<th>Delete</th>
</tr>

<?php
foreach($assignments as $assignment){

    $showLink = $html->link($assignment['Assignment']['name'],
                            array('controller' => 'assignments',
                                  'action'     => 'show',
                                  $assignment['Assignment']['id']));

    $deleteLink = $html->link('Delete',
                              array('controller' => 'assignments',
                                    'action'     => 'delete',
                                    $assignment['Assignment']['id']),
                              NULL,
                              'Delete Assignment?');

    $wrappedDescription =
        $text->truncate($assignment['Assignment']['description'],
                        39, '...', false);

    $tdClass = NULL;

    if($time->isToday($assignment['Assignment']['exp_date'])){

        $tdClass = array('class' => 'expiresToday');

    } elseif($time->gmt($assignment['Assignment']['exp_date'])
             < $time->gmt(date("Y-m-d"))){

        $tdClass = array('class' => 'expired');

    }

    echo $html->tableCells(array(
                            array($showLink,
                                  $wrappedDescription,
                                  array($assignment['Assignment']
                                        ['exp_date'],
                                        $tdClass),
                                  $deleteLink)
                                   ));

}
?>
</table>
</div>
```

```
<!--assignments/show.ctp view-template-->
<?php
echo '<h1>'.$assignment['Assignment']['name'].'</h1>';
echo '<table border="1">';
$editLink = $html->link($assignment['Assignment']['name'],
                        array('controller' => 'assignments',
                              'action'     => 'edit',
                              $assignment['Assignment']['id']));
echo $html->tableHeaders(array('Name',
                               'Description',
                               'Expiration date',
                               'Modified', 'Created'));
$description = $text->truncate($assignment['Assignment']
['description'],
                               39, '...', false);
echo $html->tableCells(array($editLink,
                             $description,
                             $assignment['Assignment']['exp_date'],
                             $assignment['Assignment']['modified'],
                             $assignment['Assignment']['created']));
echo '</table>';
if($assignment['AssignmentType'] != NULL){
    echo '<h1>Assignment type definition</h1>';
    echo '<table border="1">';
    echo $html->tableHeaders(array('Type name', 'Type description'));
    foreach($assignment['AssignmentType'] as $type){

        echo $html->tableCells(array($type['name'],
                                     $type['description']));
    }
    echo '</table>';
}
echo $html->link('List assignments',
                 array('controller' => 'assignments',
                       'action'     => 'listassignments'));
?>
```

```
<!--assignments/saved.ctp-->
<h1>Assignment saved</h1>
<?php
echo $html->link('See assignment',
                 array('controller' => 'assignments',
                       'action'     => 'show',
                       $assignmentId));
echo $html->link('List assignments',
                 array('controller' => 'assignments',
                       'action'     => 'listassignments'));
?>
```