VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Taavi Halkola

VARASTO- SEKÄ PAKKAAMISHALLINTAJÄRJESTELMÄN TOTEUTUS CAKEPHP-SOVELLUSKEHYKSELLÄ

Liiketalous ja matkailu
2014

VAASAN AMMATTIKORKEAKOULU
Tietojenkäsittely

TIIVISTELMÄ

Tekijä	Taavi Halkola
Opinnäytetyön nimi	Varasto- sekä pakkaamishallintajärjestelmän toteutus CakePHP sovelluskehyksellä
Vuosi	2014
Kieli	suomi
Sivumäärä	118 + 2 liitettä
Ohjaaja	Raija Tuomaala

Tämä opinnäytetyö käsittelee suunnittelemaani ja toteuttamaani varaston- sekä pakkaamishallintajärjestelmän toteuttamista CakePHP-sovelluskehyksellä. Työn toimeksiantaja on vaasalainen mainosyritys Messunikkarit Oy.

Tavoitteena oli toteuttaa järjestelmä, joka helpottaa varaston ylläpitoa sekä mahdollistaa pakkauslistojen luomisen sekä järjestelmässä olevien että järjestelmän ulkopuolisten tuotteiden lisäämisellä.

Työssä esittelen CakePHP-sovelluskehyksen keskeisimmät ominaisuudet sekä toiminnot. Työssä on myös mainittu mielestäni hyödyllisiä lisäosia sekä funktioita. Työssä on lisäksi esitelty MVC-arkkitehtuurin alkeet sekä työssä käyttämiäni tekniikoita ja kieliä.

Työn toiminnallisessa osassa esittelen järjestelmän toteuttamista CakePHP-sovelluskehyksellä. Esittelen järjestelmän keskeisimmät ominaisuudet sekä niiden tärkeimpien osien toteutustavat.

Työn tuloksena oli yrityksen tarpeita vastaava järjestelmä. Järjestelmän käyttöönotto toteutetaan kuitenkin vasta opinnäytetyön jälkeen, minkä vuoksi ei siitä mahdollisesti saatavia hyötyjä pystytä todentamaan. Tulosten myötä on kuitenkin mahdollista todeta, että CakePHP mahdollistaa nopean kehittämisen, ja oli oikea valinta järjestelmän toteuttamiseksi.

Avainsanat CakePHP, PHP, sovelluskehys, verkkojärjestelmä, MVC

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietojenkäsittely

ABSTRACT

Author	Taavi Halkola
Title	The Development of a Warehouse and Packaging Management System with CakePHP Framework
Year	2014
Language	Finnish
Pages	118 + 2 Appendices
Name of Supervisor	Raija Tuomaala

This thesis introduces a warehouse and packaging management system planned and implemented with CakePHP framework. The system was developed for Messunikkarit Oy, an advertising company located in Vaasa.

The goal of this thesis was to implement a system which makes it easier to maintain the company's stock. Another goal was to make it possible to create packaging lists both from the items that exist in the system and also from items that do not.

This work introduces the most important qualities of CakePHP framework and also a few plugins and functions found useful or important. Also the basics of MVC architecture as well as the techniques and languages used to develop the system are introduced.

A result of the work was a system which meets the company's requirements. The implementation of the system will be carried out following this thesis. Therefore, any possible benefits the system brings to the company cannot be verified in this work. It is still possible to verify that CakePHP makes rapid application development possible and was the right choice for implementing this system.

Keywords CakePHP, Framework, MVC, PHP, Web Application

SISÄLLYS

KUVIO- JA TAULUKKOLUETTELO

KÄYTETYT TERMIT JA LYHENTEET

AJAX	**(**Asycnhronous JavaScript and XML) Kokoelma verkkotekniikoita, jotka mahdollistavat vuorovaikutteisten verkkosovelluksien luomisen.
Autentikointi	Käyttäjän tunnistamista.
Bootstrap	Opinnäytetyössä käytetty front end -sovelluskehys.
CakePHP	Opinnäytetyössä käytetty PHP-sovelluskehys.
Callback	Callback metodeiksi kutsutaan metodeja, joita kutsutaan sovelluksen tietyssä vaiheessa.
CLI	Command-line interface eli komentorivikäyttöliittymä.
Cron	Ajastettujen tehtävien ajopalvelu.
CRUD	Lyhennys kuvaa tietokannan neljää yleisintä toimintoa eli Create, Read, Update ja Delete.
CSS	**(**Cascading Style Sheets) CSS-tiedostot ovat Www-dokumenttien tyyliohje-tiedostoja.
DOM	**(**Document Object Model) Dokumenttioliomalli mahdollistaa dokumentin sisällön muokkauksen.
Front end	Ohjelmistokehityksen maailmassa front endiksi kutsutaan sovelluksen käyttäjälle näkyvää osaa. Front end on käyttäjän ja sovelluslogiikan välinen liittymä.
GIT	Versionhallintaohjelmisto.
HTML	**(**Hypertext Markup Language) Hypertekstin kuvauskieli, jota käytetään pääasiassa verkkosivustojen ulkoasujen toteuttamiseen.

JSON	**(**JavaScript Object Notation) Tiedonvälitykseen tarkoitettu tiedostomuoto.
LightHTTPD	Avoimen lähdekoodin verkkopalvelin.
MVC	**(**model-View-Controller**)** Malli-näkymä-ohjain tarkoittaa ohjelmistoarkkitehtuuria.
ORM	Object relational mapping eli olioiden relaatiokuvaus.
PHP	(PHP: Hypertext Preprocessor) PHP on dynaamisten verkkosivujen luontiin käytetty ohjelmointikieli.
PO	Kielikäännös-tiedosto.
POT	Kielikäännösten pohjatiedosto.
PuTTY	SSH-asiakasohjelma.
Regular expression	Säännöllisten merkkijonojen määrittämiseen käytetty lauseke.
Responsiivisuus	Repsonsiivisuus tarkoittaa sovelluksen tai sivuston ulkoasun skaalautumista useille näyttökooille.
RIA	**(**Rich Internet Application) Rikkaat Internet-sovellukset pyrkivät pääsemään ominaisuuksien sekä toiminnallisuuksien kannalta mahdollisimman lähelle työpöytäsovelluksia.
SSH	**(**Secure Shell) Salatun tietoliikenteen protokolla.
Validointi	Tiedon oikeellisuuden tarkistamista.
WinSCP	Tiedostonsiirto (FTP) ohjelma Windows-ympäristöön.

1 JOHDANTO

Opinnäytetyönä olen toteuttanut työn toimeksiantajalle verkkojärjestelmän CakePHP-sovelluskehyksellä. Työssä olen käyttänyt tekstin tukena paljon koodiesimerkkejä sekä kuvankaappauksia toteutuksen vaiheista. Esimerkkien tarkoituksena on helpottaa esittelemääni asiaa. Kuvat puolestaan auttavat käsittämään mitä kyseinen vaihe oikeasti tarkoittaa.

1.1 Toimeksiantaja

Työn toimeksiantaja on Messunikkarit Oy. Messunikkarit Oy on vaasalainen pienyritys, jonka päätoimenkuvana on messuosastojen pystytys. Yritys suunnittelee ja pystyttää asiakkailleen messuosastoja. Yrityksen toimialueena ovat Suomi sekä muut Euroopan maat.

1.2 Tavoitteet ja rajaukset

Työn tavoitteena on toteuttaa järjestelmä, joka toimii varastohenkilökunnan tukena pitäen kirjaa varastossa olevista tuotteista. Toisena tärkeänä tavoitteena on osastojen pakkauslistojen luominen, niin että järjestelmä käyttää hyväksi varastotietokantaan lisättyjä tuotteita. Pakkauslistoja luodessa täytyy olla mahdollista lisätä tuotteita varastonhallintajärjestelmästä sekä tuotteita, joita ei varastonhallintajärjestelmässä ole tarkoitus pitää. Tällaisia tuotteita voivat esimerkiksi olla kerran tai muutaman kerran käytettävät tuotteet, jota ei ole hyödyllistä pitää varastossa. Järjestelmän tulee vähentää pakkauslistaan lisätyt tuotteet varastonhallintajärjestelmästä, ja pitää näin varaston saldoa oikeana. Järjestelmän tulee osata lisätä saapuneet tuotteet takaisin varastoon. Lopullisena tavoitteena on pyrkiä vähentämään pakkauslistoja luotaessa syntyviä inhimillisiä virheitä sekä helpottaa varastohenkilökunnan työtä pitämällä ajantasaista listausta paikalla olevista tuotteista. Rajaan pois opinnäytetyöstä yrityksessä olevien tuotteiden lisäämisen varastonhallintajärjestelmään, järjestelmän hyväksyntätestauksen, käyttöönoton sekä ohjeiden laatimisen.

1.3 Työn rakenne

Käsittelen työssäni ensin yleisellä tasolla käyttämiäni tekniikoita ja kieliä, näistä kerron, mitä ne ovat ja mihin niitä käytetään. Tämän jälkeen kerron yleisesti sovelluskehyksistä, minkä jälkeen esittelen valitun sovelluskehyksen CakePHP:n. Kerron myös työn suunnitteluvaiheista sekä itse työn tekemisestä. Lopuksi olen tehnyt yhteenvedon, jossa kerron työn onnistumisesta, jatkokehityksestä sekä siitä, mitä jälkeenpäin ajateltuna olisin tehnyt toisin.

2 KÄYTETYT TEKNIIKAT JA KIELET

Nykyaikaiseksi verkkosivuksi ei enää riitä, että se toimii odotetusti tietokoneella. Sen täytyy toimia kaikilla laitteilla, mukaan lukien puhelimet ja tablet-tietokoneet. Sivuston täytyy olla sekä näyttävän näköinen että interaktiivinen ja responsiivinen. Siinä täytyy olla silmiä hiveleviä animaatioita ja toimintoja, mutta ennen kaikkea sen täytyy olla käyttäjäystävällinen. Kerron lyhyesti tekniikoista, kielistä sekä kirjastoista, joihin olen päätynyt järjestelmän toteuttamiseksi.

2.1 PHP ja MySQL

PHP on palvelinpuolen ohjelmointikieli, jota käytetään pääsääntöisesti dynaamisten verkkosivustojen luonnissa. PHP on alusta asti ollut ilman käyttö-, muokkaus-, ja jakelurajoituksia, nykyään tällaisia ohjelmistoja kutsutaan Open-Source, eli avoimen lähdekoodin ohjelmistoiksi. PHP on löyhästi tyypitetty ohjelmointikieli, mikä tarkoittaa sitä, että PHP:ssa muuttujat luodaan ja tuhotaan automaattisesti. Tämä lisää kielen käytännöllisyyttä ja antaa kehittäjän keskittyä itse toimivan ohjelmiston kehittämiseen. (Gilmore 2005, 6-7.)

PHP-ohjelmistoja luotaessa on tyypillistä, että ohjelmisto käyttää hyväkseen MySQL-tietokantaa. MySQL on vapaaseen lähdekoodiin perustuva tietokanta ja PHP tuskin olisikaan saanut nykyistä suosiotaan ilman mahdollisuutta luoda yhteyttä tietokantaan. PHP ja MySQL ovatkin lähes erottamattomasti linkitettyjä toisiinsa (Gilmore 2005, 617).

MySQL on ainutlaatuinen tietokanta. Se mahdollistaa tietojen hakemisen sekä tallentamisen eri lähestymistavoilla tallennusmoottoreiden avulla. Tallennusmoottorit mahdollistavat erilaisien ominaisuuksien käytön, jotka voivat vaikuttaa käyttäjäkokemukseen huomattavasti. Näitä eroavaisuuksia ovat muun muassa transaktioiden tukemiset, erilaiset taulu- ja rivi-tasojen lukitsemisstrategiat, viite-eheyden hallintamahdollisuudet sekä erilaiset indeksityypit. (Curioso, Bradford & Galbraith 2010, 87.)

2.2 JavaScript

JavaScript on verkko-ohjelmointikieli, jonka avulla verkkosivuista pyritään luomaan dynaamisempia ja vuorovaikutteisempia. JavaScriptiä käytetään yleensä yhdessä HTML:n sekä CSS:n kanssa luomaan interaktiivisempia käyttäjäkokemuksia verkkosivujen käyttäjille. Kun HTML ja CSS määräävät, miltä sivu näyttää, niin JavaScript lisää sivulle toiminnallisuutta. JavaScriptillä pystytään luomaan esimerkiksi interaktiivisia kuvaesityksiä tai vaikkapa yksinkertaisia verkkopelejä. JavaScriptin ollessa käytössä loppukäyttäjän tietokoneella, voi kieli auttaa muuttamaan staattisen sivun sisällön mukaansa tempaavaksi, interaktiiviseksi ja jopa älykkääksi käyttökokemukseksi (Goodman 2010, 3).

Jokainen nykyaikainen selain sisältää JavaScript-käsittelijän, joka mahdollista JavaScriptillä luodun koodin suorittamisen.

2.2.1 AJAX

AJAX eli ”Asynchronous JavaScript and XML” on vaihtoehtoinen tapa kommunikoida selaimen ja palvelimen välillä. AJAX mahdollistaa tiedon lataamisen palvelimelta ilman sivun uudelleenlataamista. AJAX on suuri askel eteenpäin rikkaissa Internet-sovelluksissa (RIA), jotka tarkoittavat pyrkimystä luoda sovelluksia, jotka ominaisuuksiltaan ja käytettävyydeltään vaikuttavat perinteisiltä sovelluksilta. Rikas Internet-sovellus on Internet-sovellus, joka pyrkii poistamaan käytettävyyseroja työpöytäsovellusten ja tavallisten Internet-sovellusten välillä (Eichorn 2007, 4).

2.2.2 JQuery

JQuery on JavaScript-kirjasto, joka mahdollistaa monimutkaisten JavaScript-toimintojen luomisen verkkosivuille, vaikka JavaScript-osaaminen ei olisikaan vahva. JQueryn suurin vahvuus on DOM (Document Object Model) -elementtien dynaaminen käsittely. JQueryllä voidaan muokata elementtien attribuutteja, tyyliominaisuuksia, tai luoda kokonaan uusia elementtejä täysin dynaamisesti. Kaiken lisäksi jQuery mahdollistaa näiden elementtien käsittelyn näyttävästi. Elementin voi esimerkiksi piilottaa näkyvistä kirjastoon sisäänrakennetuilla animaatioilla.

JQueryllä voidaan luoda toimintoja ja animaatioita vain muutamalla koodirivillä. Sen avulla voidaan myös helposti suorittaa AJAX-pyyntöjä kirjastoon sisäänrakennetuilla AJAX-funktioilla. JQuery on JavaScript-kirjasto, joka on suunniteltu muokkaamaan verkkosivuja lennosta (Benedetti & Cranley 2011, 5).

2.3 HTML ja HTML5

HTML:ää (Hypertext Markup Languagea) käytetään useimpien verkkosivujen luontiin. Se antaa verkkosivuille rakenteen ja toimii pohjana, jonka päälle monimutkaisempia toimintoja voidaan rakentaa

HTML5-nimitys on opinnäytetyötä kirjoittaessani vielä epämääräinen. Nykyisellään HTML5:llä tarkoitetaan uusien web-tekniikoiden niputtamista yhden iskusanan alle, eli HTML5. HTML5:een yhdistetään usein HTML:n kehitteillä oleva uusi versio ja sen ja CSS:n uudet ominaisuudet. HTML5:tä pyritään määrittelemään täsmällisesti, mutta vielä on epäselvää, mitkä määrittelyt siihen tullaan liittämään (Korpela 2011, 13). HTML5 tuo mukanaan paljon uusia ja näyttäviä ominaisuuksia, joita on jo mahdollista käyttää. Selaimet tukevat näitä uusia ominaisuuksia vaihtelevasti ja kehittäjien onkin tarkoin harkittava mitä ja miten uusia ominaisuuksia on hyödyllistä sivustolla käyttää. Selainten uusien ominaisuuksien tukemista sekä vertailua varten löytyy verkosta useita sivustoja, kuten http://html5readiness.com/.

2.4 CSS ja CSS3

CSS (Cascading Style Sheets) eli tyylitiedostot ovat tiedostoja, joissa määritellään elementtien ulkoasu eli tyyli. Kun HTML:ssä määritellään dokumentin elementit, niin CSS määrittelee, miltä nämä elementit näyttävät.

Kun puhutaan CSS3:sta, tarkoitetaan CSS:n kehitteillä olevaa uutta versiota. CSS3:n myötä CSS saa suuren joukon uusia ominaisuuksia. Kuten HTML:n uudet ominaisuudet, niin myös CSS:n uudet ominaisuudet ovat jo käytettävissä, mutta vaihtelevasti erilaisten selainten tukimahdollisuuksien vuoksi. Myös näiden ominaisuuksien selaintuen on mahdollista käydä tarkistamassa osoitteesta http://html5readiness.com/.

Bootstrap

Bootsrap on suosituin front end -sovelluskehys responsiivisten ja mobiililähtöisten projektien kehittämiseen (Bootstrap 2014, www). Bootstrap on myös GitHubin eniten tähtiä ansainnut projekti (Ks. https://github.com/search?q=stars%3a%3E1&s=stars&type=Repositories). Bootsrapin avulla responsiivisen käyttöliittymän toteuttaminen on huomattavasti nopeampaa, koska se on suunniteltu skaalautuvaksi kaiken kokoisille näytöille. Bootstrap tarjoaa valmiiksi tyyliteltyjä HTML-pohjia sekä elementtejä. Nämä HTML-pohjat ovat pääasiassa tarkoitettu projektien nopeaan aloittamiseen, joista kehittäjien on mahdollista lähteä yksilöimään ulkoasua omaan tarpeeseensa. Bootstrap tarjoaa valmisten tyylien sekä pohjien lisäksi vapaasti valittavissa olevia JavaScript-komponentteja. Bootstrapin käyttäminen mahdollistaa sivuston tai sovelluksen nopeamman luonnin yhdellä ulkoasulla, kaikille laitteille.

3 SOVELLUSKEHYS

Sovelluskehykset ovat työvälineitä, joiden avulla kehittäjät pystyvät nopeuttamaan perustoimintojen luomista huomattavasti. Sovelluskehyksessä tavanomaisimmat toiminnot ovat sisäänrakennettuja, näin kehittäjä pystyy resursoimaan enemmän aikaa oman sovelluksen toiminnallisuuksiin. Sovelluskehys tarkoittaa kokoelmaa koodia, jonka takaa löytyvä toimintomalli on osana lähes jokaista sovellusta. Nämä toiminnallisuudet on koottu eräänlaisiksi abstraktiokerroksiksi tietyn arkkitehtuurin mukaan.

Verkkosovelluskehys on kokoelma koodia, joka on organisoitu tiettyyn arkkitehtuurin, jota on mahdollista käyttää nopeaan sovelluskehitykseen. Sovelluskehyksiä voi ajatella puolivalmiiksi kehitettyinä sovelluksina, joita on mahdollista laajentaa omaa tarkoitusta vastaavaksi. (Porebski, Przystalski & Leszek 2011, 2.)

Sovelluskehyksien suurimpana haittana sekä hyötynä on koodi- sekä nimeämiskäytännöt, joita sovelluskehystä käytettäessä on seurattava (Porebski ym. 2011, 2). Seurattavat käytännöt voivat muodostua haitaksi pakollisen koodinkirjoitustavan seurauksena. Mikäli näitä käytäntöjä ei seuraa, on hyvin todennäköistä, että myöskään mikään ei toimi. Hyödyt ovat kuitenkin paljon suuremmat. Pakolliset käytännöt takaavat yhtenäisen koodinkirjoitustyylin tiimin kesken.

3.1 Sovelluskehykset ja PHP

PHP:ta alettiin pitää ammattimaisena työkaluna vasta, kun olio-ohjelmoinnin mahdollistava PHP5 julkaistiin vuonna 2004. Muutaman vuoden päästä tästä syrjäyttivät PHP-sovelluskehykset ”Ruby on Railsin” suosituimman sovelluskehyksen asemasta. (Porebski ym. 2011, 5.)

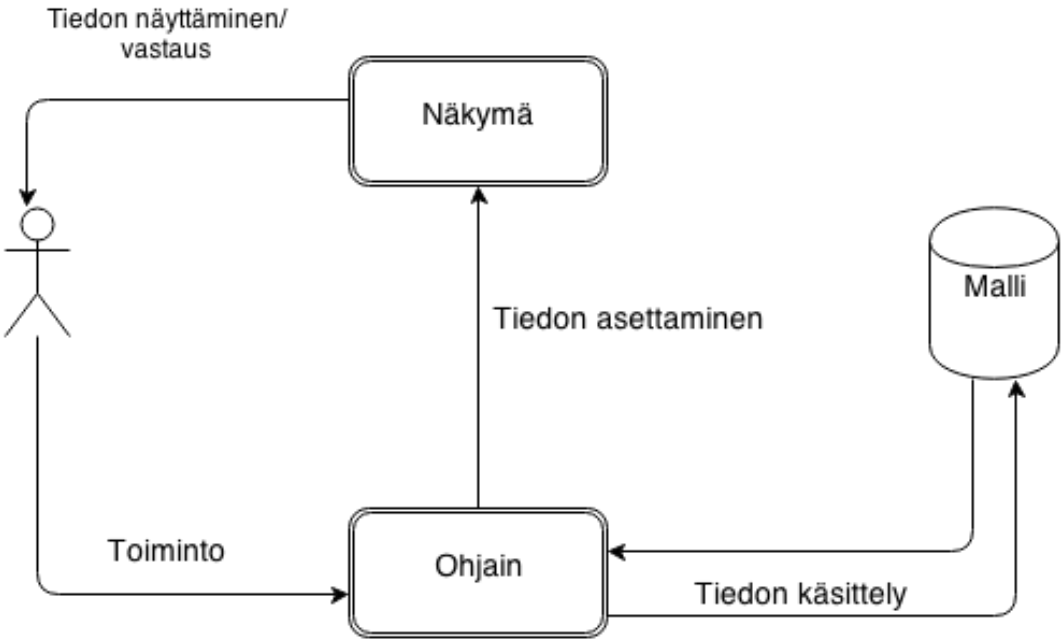
PHP-sovelluskehyksien tarkoituksena on tarjota kehittäjille yleisimpien toimintojen lisäksi rakenne, jonka päälle oma sovellus on mahdollista rakentaa. Sovelluskehystä käyttämällä varmistuu siitä, että käytetyn koodin ovat useat sovelluskehittäjät todenneet toimivaksi. PHP-sovelluskehys koostuu ajoympäristöstä sekä ko-

koelmasta koodia, kirjastoja ja luokkia, jotka auttavat kehittäjiä rakentamaan sovelluksia nopeammin. (Syam & Bari 2008, 6.)

3.2 MVC-arkkitehtuuri

MVC-arkkitehtuuri on laajasti käytetty sovellusmalli, jonka tarkoituksena on erottaa liiketoimintalogiikka sovellus- ja esityslogiikasta. Monet PHP-sovelluskehykset käyttävätkin hyväkseen MVC-arkkitehtuuria. MVC-malli koostuu kolmesta olennaisesta osasta: Model-, View- ja Controller-osista eli mallista, näkymästä ja ohjaimesta. MVC-mallissa hyödyttävät erityistesti sovelluksen jakaminen loogisiin osiin sekä koodin uudelleenkäyttö. Tämä nopeuttaa sovelluskehitystä varsinkin siinä vaiheessa, kun koodiin täytyy tehdä muutoksia. Myös virheiden etsintä helpottuu huomattavasti, kun koodi on loogisesti järjestetty. MVC-malli helpottaa yksittäisten osien muokkaamista.

PHP-pohjaisten järjestelmien tulee myös kohdata se todellisuus, että muutosten hallinta on yksi suurimmista haasteista jokaisessa sovelluksessa (Abeysinghe 2009, 34). Juuri tämän ongelman ratkaisua MVC-malli helpottaa koodien loogisella erottelulla. Kuvio 1 esittää MVC-mallin toimintaa sen yleisimmässä muodossa.



Kuvio 1. MVC-malli.

Kerron hieman jokaisen osan toiminnasta, mutta samalla täytyy pitää mielessä, että jokaisen osan yksityiskohtaisempaan toimintaan vaikuttaa käytetty sovelluskehys.

3.2.1 Ohjaimet

Ohjaimet käsittelevät HTTP-pyyntöjä ja ohjaavat ne ohjainta vastaavaan malliin. Ohjain palauttaa mallista saadun vastauksen näkymälle.

Ohjain vastaa tapahtumiin, tyypillisesti käyttäjän toimiin, ja voi tapahtuman perusteella aiheuttaa toimia mallissa tai näkymässä. PHP-pohjaisissa sovelluksissa ohjain vastaa koodista, joka käsittelee varsinaista toimintaa. Se myös yhdistyy HTTP- logiikkaan – olettaen että käyttäjä kommunikoi sovelluksen kanssa käyttäen HTTP:tä. Sekä näkymä että ohjain ovat riippuvaisia mallista, mutta malli ei ole riippuvainen näkymästä tai ohjaimesta. Tämä on yksi suurimmista hyödyistä eri osien erottamisen näkökulmasta. Erottaminen mahdollistaa mallin luomisen ja testaamisen riippumattomana visuaalisesta näyttämisestä. Verkkosovelluksissa näkymien ja ohjainten erottaminen toisistaan on tarkkaan määritelty. (Abeysinghe 2009, 32–33.)

3.2.2 Mallit

Mallit käsittelevät tietojen tallentamista, käsittelemistä, hakemista sekä sen oikeellisuutta. Usein mallit ovat ainoa osa sovellusta, josta on pääsy tietokantoihin. Yleinen käsitys malleista on se, että ne pitävät sisällään sovelluksen liiketoimintalogiikan.

Malli kuvastaa tietoa, jota sovellus käsittelee. Malli hallinnoi sovelluksen toimintaa ja tietoja. Se vastaa näkymistä saatuihin tietopyyntöihin ja ohjaimista saatuihin tietojen muokkauspyyntöihin. PHP-ympäristössä malli vastaa tietokantarakennetta. Kun PHP-sovelluksissa tarvitaan pysyvää tietoa, on tietokannan hallintajärjestelmä suuressa roolissa. (Abeysinghe 2009, 32.)

3.2.3 Näkymät

Näkymät ovat sovelluksen näkyvä osuus eli esityskerros. Näkymät eivät muokkaa tietoa vaan ne ainoastaan esittävät sitä. Lähes kaikki sivukohtainen tieto tulee näkymien kautta.

Näkymä muuttaa mallista saadun tiedon käyttäjäystävälliseen muotoon. Toisin sanoen näkymä hallitsee tiedon näyttämistä. PHP-pohjaisissa sovelluksissa näkymä vastaa yleisimmin tiedon HTML-pohjaisesta esittämisestä käyttäjälle (Abeysinghe 2009, 32).

4 CAKEPHP SOVELLUSKEHYKSENÄ

CakePHP on PHP:lle kehitetty nopean kehityksen (RAD) sovelluskehys, joka käyttää hyväkseen MVC-arkkitehtuuria. Se on ilmainen ja vapaaseen lähdekoodiin perustuva sovelluskehys. CakePHP tarjoaa myös konsolisovelluskehyksen, joka mahdollistaa konsolisovellusten luonnin sekä erittäin nopean tavan luoda valmiita malleja, ohjaimia sekä näkymiä. Tässä luvussa esittelen CakePHP:n tärkeimpiä ominaisuuksia sekä käytäntöjä. CakePHP:n tärkeimpänä tavoitteena on kehittäjäystävällisyys, kehitysnopeus sekä helppokäyttöisyys (Porebski ym. 2011, 9).

4.1 Huomioitavaa

Tätä opinnäytetyötä tehdessä on käytössä ollut CakePHP:n 2.4.9-versio. Kaikki tässä työssä esitellyt ominaisuudet sekä käytännöt pohjautuvat CakePHP:n sen hetkiseen dokumentaatioon. Sovelluskehystä käytettäessä tulisi näin ollen tarkistaa ajankohtaiset esimerkit sekä käytännöt suoraan dokumentaatiosta (Ks. CakePHP 2014, www).

Tätä työtä tehdessä ei myöskään ole otettu kantaa sovelluskehyksen heikkouksiin. Sovelluskehystä olisi mahdollista verrata muihin PHP-sovelluskehyksiin. Muita suosittuja sovelluskehyksiä ovat mm.

- CodeIgniter
- Symfony
- Zend Framework
- Yii.

En kuitenkaan vertaile näiden sovelluskehysten suorituskykyä, ominaisuuksia tai käytettävyyttä, koska muista sovelluskehyksistä ei ole käytännön kokemusta. Verkosta on kuitenkin luettavissa vertailuja niin ominaisuuksien kuin suorituskyvyn muodossa. Vertailuja löytyy esimerkiksi osoitteesta http://systemsarchitect.net/performance-benchmark-of-popular-php-frameworks/ sekä osoitteesta http://socialcompare.com/en/comparison/php-frameworks-comparison.

4.2 Nopean kehityksen malli

Nopean kehityksen mallin, eli Rapid Application Developmentin (RAD), tarkoituksena on nopeuttaa sovelluskehitystä tarjoamalla valmiita ratkaisuja yleisimpiin ongelmiin. Kuten nimikin jo kertoo, malli mahdollistaa huomattavasti nopeamman sovelluskehityksen.

Yleisimpien suunnittelumallien integrointi CakePHP:hen tarkoittaa sitä, että kehittäjien ei tarvitse hukata aikaa yrittäessään ratkaista ongelmia, jotka ovat yleisesti osana lähes jokaista verkkoprojektia. Nämä ongelmat on jo ratkaistu CakePHP:ssa. Näin kehittäjät voivat keskittyä sovelluksen ominaisiin toimintoihin. Tämän ansiosta päästään paljon nopeampaan sovelluskehitykseen. (Syam ym. 2008, 6.)

4.3 Arkkitehtuuri ja käytännöt

Kuten aikaisemmin mainitsin, CakePHP perustuu MVC-arkkitehtuuriin. Kappaleessa 3.2 esittelin MVC-mallin arkkitehtuuria. Tässä kappaleessa käsitellään yksityiskohtaisemmin juuri CakePHP:n soveltamaa MVC-arkkitehtuuria.

4.3.1 Mallit

CakePHP:ssa mallit edustavat tiettyä tietokantataulua. Jokaisella taululla tulisi olla vastaava malli. Kaikki PHP-koodit, jotka liittyvät taulun käsittelyyn, lisäämiseen, muokkaamiseen tai rivien poistamiseen, tulee käsitellä malleissa. Mallit pitävät sisällään myös mallien väliset yhteydet. Lisäksi mallit määrittelevät tiedon validointisäännöt tietojen lisäämiselle ja päivittämiselle. Mallit voi kuvitella sovelluksen tietotasoksi. Malleissa käsitellään jokaisen mallin liiketoimintalogiikka. Mikäli sovelluksessa on esimerkiksi malli, joka edustaa autoja, tulee kaikki toiminta siihen liittyen, kuten auton myyminen, ostaminen ja niin edelleen, määritellä mallissa. (Syam ym. 2008, 8.)

Mallien kautta kulkee sovelluksen kaikki tieto, kuten lisättävä, muokattava ja poistettava tieto. Kaikki mallit tulee lisätä sovelluskehyksen *app/model* -kansioon. Mallien nimeäminen tapahtuu siihen liittyvän tietokantataulun pohjalta, mutta ni-

meäminen tapahtuu yksikkömuodossa. Mikäli tietokantataulun nimi on *events*, tulee malli tiedosto nimetä *Event.php* ja luokan nimeksi asettaa *Event*. Mikäli taulun nimi on *event_files*, tulee mallin tiedostonimen olla *EventFile.php* ja vastaavasti luokkanimeksi asettaa *EventFile*. Kaikki mallit ovat luokan *AppModel* -aliluokkia, joka löytyy tiedostosta *app/Model/AppModel.php*. Malleissa voidaan määritellä myös tauluun tallentamista, päivittämistä sekä hakemista edeltävät sekä seuraavat toiminnot, joista kerron tarkemmin kappaleessa 4.10.

4.3.2 Ohjaimet

CakePHP:ssa ohjaimet ohjaavat sovelluksen kulkua. Jokainen verkkopyyntö on ohjattu tiettyyn ohjaimeen ja tiettyyn ohjaimen metodiin, jossa käyttäjän pyyntö käsitellään. Ohjainten metodien käytöstä ja tarkoituksesta kerron tarkemmin kappaleessa 4.15. Ohjaimen logiikka päättää, minkälainen vastaus tuotetaan. Yleensä ohjaimen logiikka pitää sisällään pyynnön malliin, jonka avulla ohjain voi hakea tietoa tai tarkistaa pääsyoikeuksia. Pyynnön lopuksi ohjain välittää vastauksen näkymälle. Ohjainta voidaan pitää sovelluksen ohjauslogiikkatasona. Kuten aikaisemmin jo todettiin, mallin tulisi pitää sisällään sovelluksen liiketoimintalogiikka. Ohjaimen tulisi ainoastaan ohjata toiminnot malliin ja olla kevyt. Tätä mallifilosofiaa kutsutaan joskus ”fat models and thin controllers”-malliksi. (Syam ym. 2008, 8.)

Ohjaimissa ei ole tarkoitus käsitellä monimutkaista koodia, vaan ohjata pyyntö oikeaan malliin ja sitä kautta asettaa näkymälle tarvittavat tiedot käytettäviksi. Yleistä on, että yksi ohjain vastaa yhden mallin logiikasta (Syam ym. 2008, 50). Ohjaimien tehtävänä on myös estää pyynnöt, joita kyseinen toiminto ei salli. Tämä tapahtuu tavallisimmin joko HTTP-pyyntömenetelmän tai käyttäjäoikeuksien perusteella.

Ohjaimien nimeäminen tapahtuu malleista eroten monikossa. Jos oletetaan tietokantataulun olevan *events*, tulee ohjaimen tiedosto nimetä *EventsController.php:ksi* ja vastaavasti luokan nimeksi asettaa *EventsController*. Ohjainten tiedostot tulevat sovelluksessa kansioon *app/Controller*, ja kaikki ohjaimet ovat luo-

kan *AppController* aliluokkia. *AppController*-luokka löytyy tiedostosta *app/Controller/AppController.php.*

4.3.3 Näkymät

Näkymät tulostavat ohjaimesta saadun vastauksen käyttäjäystävälliseen muotoon. Ne koostuvat yleensä HTML-koodista, johon on liitetty PHP:ta, mutta ne voivat olla myös muuta, kuten XML:ää, PDF-dokumentteja tai muuta vastaavaa (Syam ym. 2008, 9). Näkymien tulisi olla sovelluksen ainoa taso, joiden kautta tietoa asetetaan näytettäväksi käyttäjälle. Niissä ei kuulu käsitellä sovelluksen logiikkaa eikä tietokantatoimintoja, ainoastaan saattaa ohjaimesta saatu tieto helposti luettavaan muotoon.

Näkymien käyttäminen sovelluksessa tapahtuu seuraavasti: oletetaan, että selaimen pyyntö on ohjautunut ohjaimeen nimeltä *EventsController* ja sen metodiin nimeltä *view*. Tämän toiminnon näkymätiedosto tulee sovelluksessa olla kansiossa *app/View/Events* ja tiedostonimellä *view.ctp*. CTP-tiedostoformaatti on CakePHP:n formaatti ulkoasutiedostoille.

4.4 Kansiorakenne

CakePHP:n asennuksen kansiorakenne koostuu kolmesta tärkeimmästä kansiosta: *app*, *lib* ja *vendors*. Kansioiden nimet on onnistuneesti nimetty kuvaamaan niiden sisältöä. Tässä kappaleessa tarkastelen lyhyesti, mihin mitäkin kansiota kehitysvaiheessa käytetään.

Lib-kansio pitää sisällään kaiken CakePHP:n toiminnallisuuden eli sovelluskehyksen ytimen. Tämän kansion tiedostojen muokkaamista on hyvä välttää. Vendors-kansioon asetetaan mahdolliset kolmannen osapuolen PHP-kirjastot.

App-kansio pitää sisällään sovellustiedostot, joita kehittäjä luo ja käyttää kehityksen aikana. Seuraavaksi mainitut kansiot kuuluvat *app*-kansion alle:

- Config – asetustiedostot
- Console – konsolisovellus sekä omat konsolitehtävät

- Contoller – ohjaimet sekä komponentit
- Lib – omat kirjastot
- Locale – kielimerkkijonot
- Model – mallit, käyttäytymiset sekä tietolähteet
- Plugin – lisäosat
- Test – testaustiedostot
- tmp – väliaikaiset tiedostot, kuten lokit ja välimuistit
- Vendor – kolmannen osapuolten kirjastot
- View – näkymät, elementit, ulkoasut, virhesivut sekä avustajat ja
- webroot – kuvat, tyyli- sekä JavaScript-tiedostot. Tämä kansio toimii sovelluksen juurena.

4.5 Mallit ja tiedonkäsittely

Kuten aikaisemmin olen esittänyt, niin mallin tarkoituksena on edustaa tiettyä taulua tietokannassa. Tämä tarkoittaa luonnollisesti sitä, että mallilla täytyy olla pääsy taulun tietoihin. CakePHP:ssa tietokantakyselyt suoritetaan malleihin sisäänrakennetuilla metodeilla. Esittelen tässä kappaleessa tiedon hakemista, tallentamista sekä poistamista tietokannasta mallin kautta. On yleistä, että edellä mainitut tietokantatoiminnot tapahtuvat ohjaimessa viittaamalla mallin kyseiseen metodiin. Tässä kappaleessa esitetyt esimerkit ovat kuitenkin tehty suoraan malliin. Kappaleessa 4.6 kerron enemmän malliin viittaamiseen ohjaimesta.

Malli on paljon enemmän kuin pelkästään tietokannan abstraktiokerros. Se auttaa meitä erottamaan liiketoimintalogiikan sovellus- ja esityslogiikasta. Sen lisäksi se tarjoaa hyödyllisiä toimintoja, jotka tekevät tietokannan toiminnoista helppoja, ilman että tarvitsee kirjoittaa yhtäkään riviä SQL-kyselyitä. (Syam ym. 2008, 73.)

4.5.1 Tiedon hakeminen

Tiedon hakeminen malliin liittyvästä taulusta on todellakin tehty helpoksi. Seuraavassa esimerkissä näytän, miten yksinkertaista se parhaimmillaan voi olla. Esimerkissä on nähtävillä muutama yksinkertainen haku mallin kautta. Ensimmäisessä esimerkissä haetaan *Events*-malliin liittyvän taulun kaikki rivit tietokannas-

ta, toisessa haetaan tiettyä riviä *id*:n perusteella ja kolmannessa haetaan kaikki rivit, jotka vastaavat annettuja ehtoja.

```
//Haetaan kaikki tapahtumat
$allEvents = $this->find('all');

//Haetaan vain yksi tapahtuma, jonka id on 12
$event = $this->find('first', array(
     'conditions' => array('Event.id' => 12)
));

//Haetaan kaikki tapahtumat joiden sarakkeen arrived arvo on 0
$events = $this->find('all', array(
     'conditions' => array('Event.arrived' => 0)
));
```

Kaikissa *find*-metodin kutsuissa on annettava käytettävä hakutyyppi. Esimerkeissä on nähtävissä *all* sekä *first*. Toisessa ja kolmannessa esimerkissä on nähtävissä *find*-metodin toisena argumenttina assosiatiivinen arraytaulukko. Tätä argumenttia käytetään määrittämään haun käyttämät asetukset.

Taulukossa 1 esittelen *find*-metodin hyväksymät hakuasetukset CakePHP:n dokumentaation mukaan (Ks. CakePHP 2014, www). Huomioitavaa on, että osa asetuksista odottaa arvon olevan arraytaulukko, kun osa odottaa sen olevan int-tyyppinen. Kaikki asetukset ovat vapaaehtoisia.

Taulukko 1. CakePHP hakujen asetukset.

Asetuksen avain	Asetuksen tyyppi	Asetuksen selitys
conditions	array	Asettaa haun käyttämät haku ehdot, jotka muodostavat SQL-lauseen WHERE-sivulauseen.
recursive	int	Määrää, haetaanko samalla malliin liittyvät assosiaatiot.
fields	array	Asettaa haettavat sarakkeet.
order	array	Asettaa rivien järjestyksen eli SQL-lauseen ORDER BY-sivulauseen.
group	array	Asettaa tulosten ryhmittelyn eli GROUP BY-sivulauseen.
limit	int	Asettaa LIMIT-sivulauseen.
page	int	Sivun numero, käytetään sivutetuissa haussa.
offset	int	Asettaa LIMIT-sivulauseen offset-arvon eli kertoo, kuinka monta ensimmäistä tulosta haku ohittaa.
callbacks	true/false/ ’before’/’after’	Käytetään kertomaan mikäli callback-metodit suoritetaan hakua tehtäessä.

Taulukossa 2 on listattuna CakePHP:n tarjoamat hakutyypit sekä niiden käyttötarkoitus.

Taulukko 2. CakePHP:n hakutyypit.

Hakutyyppi	Käyttötarkoitus
all	Tätä käytetään, kun oletettavissa on useita tuloksia.
first	Tätä käytetään, kun oletettavissa on yksi tulos.
count	Tämä hakutyyppi palauttaa hakua vastaavien tulosten määrän.
list	Tämä haku palauttaa numeroidun arraytaulukon. Toimii erityisen hyvin, mikäli tarkoituksena on tarjota alasvetovalikko tuloksista.
neighbors	Haku palauttaa hakutulosta vastaavat naapurit eli rivin edellisen sekä seuraavan rivin.
threaded	Haku palauttaa sisäkkäisen listan, joka sisältää malliin liitetyt assosiaatiot, joiden viiteavaimeksi on asetettu *parent_id*-sarake.

CakePHP tarjoaa myös niin sanottuja Magic Find Types eli ”taikahakutyyppejä”. Näillä tyypeillä tarkoitetaan hakua, jonka asetukset on annettu suoraan *find*-metodin nimeen. Esitän näistä yhden esimerkin, jossa mallissa haetaan siihen liittyvän taulun riviä *id*-sarakkeen perusteella.

```
$event = $this->findById(12);
```

4.5.2 Tiedon tallentaminen

CakePHP:lla tallennettaessa tietoa on tärkeää pitää huolta tallennettavan tiedon oikeasta rakenteesta. Tämän takia suosittelen käytettävän CakePHP:n FormHelper-avustajaa lomakkeiden luonnissa. Avustajasta kerron enemmän kappaleessa 4.14.2. Seuraavassa esimerkissä on nähtävissä tallennettavan tiedon oikea rakenne sekä kutsu mallin *save*-metodiin luotaessa uutta tapahtumariviä. On otettava huomioon, että mikäli mallin *id* on asetettu *$this->id = 2* tai taulun PRIMARY KEY-sarake on mukana tallennettavassa tiedossa, suorittaa *save*-metodi SQL UPDATE-lauseen INSERT-lauseen sijasta. Mikäli halutaan varmistua siitä, että suoritettava

toiminto on INSERT, voidaan kutsua *create*-metodia mallissa. Create-metodi alustaa mallin tilan uuden tiedon lisäämiseksi (CakePHP 2014, www).

```
$this->create();

$data = array(
  'Event' => array(
    'sarake1' => 'sarakkeen 1 arvo'
    'sarake2' => 'sarakkeen 2 arvo'
  )
);

if ($this->save($data)) {
  //Tallentaminen onnistui, tehdään jotain
}
```

Kuten kappaleen alussa mainitsin, on yleistä, että myös *save*-metodia kutsutaan suoraan ohjaimesta tallentamaan tietoa, jota lomakkeesta on lähetetty. CakePHP:ssa HTTP-pyynnön mukana lähetetyt tiedot ovat ohjaimen saatavilla *request*-objektin data-muuttujassa. Seuraava esimerkki on täysin toimiva esimerkki uuden tapahtuman luonnista ohjaimen *add*-toiminnossa.

```
public function add()
{
  if ($this->request->is('post')) {
     $this->Event->create();
     if ($this->Event->save($this->request->data)) {

        $this->Session->setFlash(
          'Tapahtuma tallennettu.',
          'default',
          array('class' => 'alert alert-success')
        );

        return $this->redirect(
            array('action' => 'view', this->Event->id)
        );
      } else {
        $this->Session->setFlash(
          'Tapahtumaa ei voitu tallentaa.
          Ole hyvä ja yritä uudelleen.',
          'default',
          array('class' => 'alert alert-danger')
        );
      }
   }
}
```

Edellisessä esimerkissä tarkistetaan onko saatu pyyntömenetelmä HTTP-POST. Tämän jälkeen varmistetaan uuden rivin luominen kutsulla mallin *create*-metodiin

ja tallennetaan ohjaimen *request*-objektin *data*-muuttujassa oleva tieto kutsulla mallin *save*-metodiin. Mikäli tallentaminen onnistuu, asetetaan käyttäjälle viesti onnistuneesta tallennuksesta ja ohjataan selain juuri luodun tapahtuman *view*-toimintoon. Session-komponentista ja sen kautta viestien asettamisesta kerron enemmän kappaleessa 4.12.2. Taulukossa 3 esittelen muut CakePHP mallien tietokantaan tallentamiseen tarkoitetut metodit.

Taulukko 3. CakePHP:n tiedontallennusmetodit.

Metodi	Käyttötarkoitus
save	Metodi on tarkoitettu yhden rivin tallentamiseen.
saveField	Metodilla päivitetään yksi sarake.
updateAll	Metodilla päivitetään yksi tai useampi rivi.
saveMany	Metodi on tarkoitettu saman mallin usean rivin tallentamiseen.
saveAssociated	Tämä metodi on tarkoitettu mallin rivin ja siihen liittyvien assosiaatioiden tallentamiseen.
saveAll	Metodi antaa CakePHP:n päättää, käytetäänkö saveMany vai saveAssociated-metodia tallennettavan tiedon perusteella.

4.5.3 Tiedon poistaminen

Tietojen poistaminen CakePHP:ssa tapahtuu joko mallin *delete*- tai *deleteAll*-metodilla. *Delete*-metodi poistaa rivin *id*:n perusteella ja *deleteAll*-metodi poistaa kaikki rivit, jotka vastaavat annettuja ehtoja.

Poistettaessa on myös mahdollista asettaa poistettavaksi malliin asetetut assosiaatiot *cascade*-asetuksella. *Cascade*-asetus on automaattisesti asetettu, mikäli mallin assosiaatiossa *dependent*-arvo on asetettu. Kerron mallien assosiaatioista enemmän kappaleessa 4.8. Seuraavassa esimerkissä esittelen yksinkertaisen tiedon poistamisen käyttäen molempia tapoja.

```
//Poistetaan tapahtuma id:n perusteella
$this->delete($id);

//Poistetaan kaikki tapahtumat joiden arrived sarakkeen arvo on 1
$this->deleteAll(array('Event.arrived' => 1));
```

4.6 Tiedon esittäminen

Kappaleessa 4.3.2 mainitsin, kuinka ohjain yleisimmin vastaa yhden mallin logiikasta. CakePHP:ssa ohjaimelle liitetään siihen kuuluva malli automaattisesti nimeämiskäytäntöjen perusteella. Tämä mahdollistaa mallin metodien käytön suoraan ohjaimesta, jonka avulla ohjain saa tietoa asetettavaksi näkymälle. Näitä metodeja voivat olla sekä malleihin sisäänrakennetut metodit että käyttäjän omat metodit. Kappaleessa 4.5.2 on nähtävissä, kuinka ohjaimesta kutsutaan mallin *save*-metodia. Seuraavassa esimerkissä kutsutaan malliin sisäänrakennettua metodia *find* ohjaimen *index*-toiminnosta ja asetetaan saatu tulos näkymän käytettäväksi ohjaimen *set*-metodilla.

```
public function index()
{
  //Haetaan tietokannasta kaikki tapahtumat
  //kutsumalla Event mallin find metodia
  $events = $this->Event->find('all');

 //Asetetaan tulokset näkymän käytettäväksi ohjaimen set metodilla
  $this->set('events', $events);
}
```

Esimerkissä näkyvä *set*-metodi saa ensimmäiseksi argumentiksi merkkijonon, jonka niminen muuttuja asetetaan näkymälle käytettäväksi. Toisena argumenttina on muuttujalle asetettava tieto. Muuttuja *events* on näin käytettävissä ohjaimen *index*-toimintoa vastaavassa näkymässä.

4.7 Mallit ja validointi

Tiedon validointi on yksi tärkeimmistä osista jokaisessa verkkosovelluksessa. Validointi auttaa varmistamaan, että syötetty tieto on yhdenmukaista mallin olettaman tiedon kanssa. (Porebski ym. 2011, 132.)

On tärkeää luoda hyvin mietityt validointisäännöt varmistamaan, että kaikki syötetty tieto on sallittua tietoa. CakePHP sisältää tehokkaan validointimoottorin, jo-

ka mahdollistaa useiden sääntöjen, kuten sähköpostin, postinumeron, IPv4:n, sosiaaliturvatunnuksen, luottokorttien sekä monien muiden sääntöjen käytön (Porebski ym. 2011, 139). On kuitenkin huomioitava, että useat näistä säännöistä, kuten esimerkiksi postiosoite tai puhelinnumero, on räätälöity toimimaan tietyn maan sääntöjen mukaisesti. Opinnäytetyötä tehdessä ei näiden maiden joukosta ainakaan oletuksena Suomea löydy. Tällaisissa tapauksissa on mahdollista syöttää oma *regular expression*-sääntö. Seuraavassa esimerkissä esittelen yksinkertaiset validointisäännöt uuden käyttäjän lisäämiseksi järjestelmään.

```
public $validate = array(
    'username' => 'alphaNumeric',
    'email' => 'email'
);
```

Tässä esimerkissä säännöt on lisätty niiden yksinkertaisimmassa muodossa, eivätkä ne ole kovin hyödylliset. Seuraavassa esimerkissä näytän, miten sääntöjä on mahdollista tarkentaa ja miten voi antaa useamman säännön yhdelle sarakkeelle.

```
public $validate = array(
    'username' => array(
        'alphaNumeric' => array(
            'on'       => 'create',
            'rule'     => 'alphaNumeric',
            'required' => true,
            'message'  => 'Vain kirjaimet ja numerot sallittu'
        ),
        'between' => array(
            'rule'    => array('between', 3, 15),
            'message' => 'Käyttäjänimen tulee olla 3-15 merkkiä
                          pitkä'
        ),
        'unique' => array(
            'rule'    => 'isUnique',
            'message' => "Käyttäjänimi on jo käytössä"
        )
    ),
    'firstname' => array(
        'on'         => 'create',
        'rule'       => 'notEmpty',
        'message'    => 'Etunimi on pakollinen',
        'required'   => true,
        'last'       => true
    ),
    'email' => array(
        'required' => array(
            'on'         => 'create',
            'rule'       => 'notEmpty',
            'message'    => 'Sähköposti on pakollinen',
            'required'   => true,
            'last'       => true
        ),
        'notempty' => array(
            'rule'       => 'notEmpty',
            'message'    => 'Sähköposti on pakollinen',
            'allowEmpty' => false,
            'last'       => true
        ),
        'email' => array(
            'rule'       => 'email',
            'message'    => 'Virheellinen sähköposti',
            'last'       => true
        ),
         'unique' => array(
            'on'        => 'create',
            'rule'      => 'isUnique',
            'message'   => "Sähköposti on jo käytössä"
        )
    )
);
```

Validointisäännöt asetetaan validoitavan mallin sisälle public-määreen arraytaulukko-muuttujaan nimeltä *validate*. Yllä olevat säännöt koskevat *User*-mallin *username*-, *firstname*- ja *email*-sarakkeita. Validate-arraytaulukon ensimmäinen

avain tarkoittaa validoitavaa saraketta. Mikäli validoitava sarake tarvitsee useamman kuin yhden säännön, kuten edellisen esimerkin *username* ja *email*, on säännöt lisättävä omavalintaisen avaimen alle. Suosittelen näiden omavalintaisten avainten nimeämistä sääntöä kuvaavilla nimillä koodin lukemisen helpottamiseksi. Taulukossa 4 esittelen säännöissä käytettävissä olevat asetukset.

Taulukko 4. Validointisäännöt.

Asetus	Selitys	Mahdolliset arvot
rule	Asettaa käytetyn säännön. Tämä on pakollinen asetus.	Joko merkkijono tai arraytaulukko
required	Määrittää, mikäli validoitavan sarakkeen olemassa olo on pakollista.	true/false tai ’create’/’update’
allowEmpty	Määrittää, mikäli validoitava sarake saa olla tyhjä.	true/false
on	Määrää, minkä tyyppisessä tallennustoiminnossa kyseistä sääntöä käytetään.	’update’/’create’
message	Asettaa näytettävän virheilmoituksen, mikäli sarakkeen arvo ei läpäise kyseistä sääntöä.	Merkkijono

Kaikki käytettävissä olevat säännöt ohjeineen löytyvät CakePHP:n dokumentaatiosta osoitteesta http://book.cakephp.org/2.0/en/models/data-validation.html (tai Ks. CakePHP 2014, www).

4.8 Mallit ja assosiaatiot

Tietokantarelaatioita, kuten muitakin relaatioita, on aina hankalaa ylläpitää. Tämän relaatioiden aiheuttaman tuskan helpottamiseksi tarjoaa CakePHP kehittäjille yksinkertaisen mutta tehokkaan ominaisuuden nimeltä ’object relational mapping’ eli ORM:n. Sen avulla pyritään yksinkertaistamaan tietokannan relaatioiden käsittelyä. (Syam ym. 2008, 119.)

CakePHP:ssa tietokantataulujen relaatiot kuvataan assosiaatioilla. Kun mallien assosiaatiot on määritelty vastaamaan taulujen relaatioita, ovat assosiaatioiden

upeat toiminnot käytettävissä. CakePHP:n ORM:llä on mahdollista tallentaa, hakea sekä poistaa malliin liittyvää tietoa. (Syam ym. 2008, 119.)

CakePHP:n assosiaatiot toimivat myös ilman minkäänlaisia relaatioita tietokannassa, mutta suosittelen niitä käytettävän viite-eheyden varmistamiseksi. Yksinkertaisesti sanottuna assosiaatiot määräävät mallin suhteen toisiin malleihin.

Taulukossa 5 esittelen CakePHP:ssa käytettävissä olevat assosiaatiot ja niiden esimerkit CakePHP:n dokumentaation mukaan (Ks. CakePHP 2014, www).

Taulukko 5. CakePHP:n tarjoamat assosiaatiot.

Relaatio	Assosiaatio	Esimerkki
yhden suhde yhteen	hasOne	Käyttäjällä on yksi profiili.
yhden suhde moneen	hasMany	Käyttäjällä voi olla useita reseptejä.
monen suhde yhteen	belongsTo	Monta reseptiä kuuluu käyttäjälle.
monen suhde moneen	hasAndBelongsToMany (HABTM)	Resepteillä on monta ja ne kuuluvat moneen ainesosaan.

Mikäli luodaan assosiaatio taulujen *events* ja *stands* välille suhteella *event hasMany stands*, tulee *stands*-tauluun lisätä viiteavain viitattavan taulun yksikkö muotoisella nimellä- jota seuraa *_id*. *Stands*-tauluun lisätään tässä tapauksessa *event_id*. *Event* mallille tulee kertoa tarkoitetusta assosiaatiosta seuraavasti:

```
public $hasMany = array(
  'Stand' => array(
    'className' => 'Stand',
    'foreignKey' => 'event_id',
    'dependent' => true
  )
);
```

Tämän jälkeen *Stand*-malli on käytettävissä *Event*-mallin kautta. Esimerkissä oleva *depentent => true*–asetus kertoo mallille, että tätä mallia poistettaessa myös tämän assosiaation kautta liittyvät tulokset tulee poistaa. Oletettavasti on myös

toivottavaa, että *Event*-malli on *Stand*-mallin käytettävissä, joten myös *Stand*-mallille tulee ilmoittaa assosiaatiosta puolestaan näin:

```
public $belongsTo = array(
  'Event' => array(
    'className' => 'Event',
    'foreignKey' => 'event_id'
  )
);
```

Taulukossa 6 on muutamia yleisimpiä assosiaatioita koskevia asetuksia.

Taulukko 6. Assosiaatioiden yleisimmät asetukset ja niiden käyttötarkoitus.

Asetus	Assosiaatiot	Selitys
dependent	hasOne, hasMany	Boolean arvo, joka kuvastaa mikäli mallia poistettaessa, myös kyseiseen assosiaatioon liittyvä tieto poistetaan.
foreignKey	kaikki	Määrittää viiteavaimen.
className	kaikki	Määrittää mallin nimen.
order	kaikki	Asetta SQL-lauseen ORDER BY-sivulauseen eli määrittää palautettavien tulosten järjestyksen.
conditions	kaikki	Asettaa WHERE-sivulauseen eli määrittää haun ehdot.
fields	HABTM, belongsTo, hasOne	Määrittää haetut sarakkeet.
counterCache	belongsTo	Ylläpitää tulosten määrää viitattavassa taulussa

4.9 Mallit ja virtuaalikentät

Virtuaalikentät mahdollistavat virtuaalisten sarakkeiden luomisen SQL-lauseiden perusteella sekä niiden liittämisen mallin sarakkeiksi. Näihin sarakkeisiin ei ole

mahdollista tallentaa tietoa, mutta ne toimivat muiden kenttien tapaan tietoa haettaessa (CakePHP 2014, www). Virtuaalikentät ovat hyödyllisiä, kun on mallin toiminnan kannalta tarpeellista luoda esimerkiksi yhdistettyjä tai yhteenlaskettuja sarakkeita. Virtuaaliset kentät luodaan malli-tiedostoon seuraavan esimerkin mukaisesti.

```
public $virtualFields = array(
  'packaging_status' =>
     'round((ItemList.item_list_item_completed * 100) /
     ItemList.item_list_item_count)'
);
```

Yllä oleva esimerkki luo mallille virtuaalisen sarakkeen, joka esittää pakattujen tuotteiden prosentuaalista arvoa verrattaessa tuotteiden määrään. Mallista haettaessa tietoa, palauttaa malli *packaging_status*-sarakkeen samalla tavalla kuin minkä tahansa tietokantataulun muunkin sarakkeen.

4.10 Mallit ja callback-metodit

CakePHP:n malleissa on mahdollista käyttää callback-metodeja. CakePHP:ssa callback-metodeja suoritetaan ennen ja jälkeen mallin tiedonkäsittelymetodeja.

Käytettävissä olevia callback-metodeja ovat

- beforeFind
- afterFind
- beforeValidate
- afterValidate
- beforeSave
- afterSave
- beforeDelete
- afterDelete
- onError.

Metodin nimeä vastaavasti kyseinen callback suoritetaan ennen tai jälkeen kutsua mallin kyseiseen toimintaan. Esimerkiksi *afterSave*-metodi suoritetaan aina tietokantaan tallentamisen jälkeen.

```
public function afterSave($created, $options = array())
{
  //Jos kyseessä oli päivitys
  if (!$created) {
    if (isset($this->data['ItemListItem']['checked'])) {
      $this->ItemList->updateCompletedCount(
        $this->data['ItemListItem']['checked'],
        $this->data['ItemListItem']['item_list_id']
      );
    }
  }
}
```

Yllä oleva esimerkki tarkistaa aina tallennuksen jälkeen, mikäli tallennus loi uuden rivin ja mikäli tallennettava tieto pitää sisällään *checked*-sarakkeen. Jos ehdot täsmäävät, kutsutaan *updateCompletedCount*-metodia.

4.11 Mallit ja käyttäytymiset

Mallien käyttäytymiset on tapa organisoida mallien toiminnallisuutta. Niiden avulla voidaan erottaa ja luoda uudelleenkäytettävää logiikkaa, joka luo eräänlaisen käyttäytymisen. (CakePHP 2014, www.)

CakePHP:ssa mallien käyttäytymisistä käytetään nimitystä Behaviors ja, kuten mainittu, ne mahdollistavat tietyn logiikan uudelleenkäytön useissa malleissa. Käyttäytymisiä on myös mahdollista luoda itse. Kappaleessa 10.8.2 esittelen luomani käyttäytymisen, jonka avulla käyttäytymisten tarkoitus voi olla helpommin hahmotettavissa. Taulukko 7 esittelee CakePHP:hen sisäänrakennetut käyttäytymiset.

Taulukko 7. CakePHP käyttäytymiset.

Käyttäytyminen	Selitys
ACL	Käyttäytyminen integroi mallin pääsyoikeuksien listaukseen.
Containable	Tämä mahdollistaa hakutulosten karsimisen.
Translate	Tämä liittää käännösten hallinnan malliin.
Tree	Tämä asettaa mallin käyttäytyvän hierarkkisesti.

Käytettävät käyttäytymiset osoitetaan mallille seuraavan esimerkin mukaisesti. Mikäli käyttäytyminen vaati määrityksiä tai tarjoaa vapaaehtoisia asetuksia, osoitetaan ne käyttäytymiselle esimerkissä näkyvän *FileModel*-käyttäytymisen tavoin. *FileModel* on luomani käyttäytyminen, johon viittasin aikaisemmin tässä kappaleessa.

```
public $actsAs = array(
  'FileModel' => array(
    'prefix' => 'event_',
    'name_by_field' => 'id',
    'maxSize' => '5MB',
    'allowedExtensions' => array('doc', 'pdf')
  ),
  'Translate'
);
```

Käyttäytymisiä käytettäessä on hyvä muistaa, että mikäli kyseinen käyttäytyminen käyttää callback-metodeja, suoritetaan ne ennen mallin vastaavaa metodia. Mallien callback-metodeista kerroin tarkemmin kappaleessa 4.10. Käyttäytymisiä on myös mahdollista ottaa käyttöön sekä poistaa käytöstä ajonaikaisesti seuraavan esimerkin mukaisesti.

```
//Liitetään Containable käyttäytyminen
$this->Behaviors->load('Containable');
//Poistetaan Containable käyttäytyminen käytöstä
$this->Behaviors->unload('Containable');
```

4.12 Ohjain ja komponentit

CakePHP-komponentit ovat uudelleen käytettäviä luokkia, jotka ovat jokaisen ohjaimen käytettävissä. Määritelmän mukaan komponentit ovat käytettävissä ainoastaan ohjaimissa sekä toisissa komponenteissa. Niiden tarkoitus on koodin uudelleenkäytettävyys. (Syam ym. 2008, 68.)

Komponentit on mahdollista liittää käyttöön joko ohjainkohtaisesti tai kaikille ohjaimille. Mikäli komponentti on tarkoitus liittää jokaiseen ohjaimeen, on se kätevin tehdä suoraan *AppControlleriin*.

```
public $components = array('Paginator', 'Session');
```

Komponentteja on myös mahdollista tehdä itse, kuten kappaleessa 10.8.3 esittelemäni hyvin yksinkertainen komponentti. Taulukko 8 esittelee CakePHP-asennuksen mukana tulevat komponentit, minkä jälkeen esittelen tarkemmin työssä käyttämiäni komponentteja.

Taulukko 8. CakePHP-komponentit.

Komponentti	Selitys
Pagination	Tulosten sivuttaja
Sessions	Istunto-komponentti
Authentication	Käyttäjien tunnistautuminen
Security	Lisättyä turvallisuutta
Request Handling	Tarjoaa lisätietoa HTTP-pyynnöistä
Cookie	Evästeiden hallintaa
Access Control Lists(ACL)	Pääsyoikeuslistaus

4.12.1 Pagination

PaginatorComponent eli sivuttaja-komponentti helpottaa sivutettujen tietokantahakujen luomisessa. Komponentti toimii yhdessä näkymiin tarkoitetussa PaginatorHelperin eli sivuttaja-avustajan kanssa, joka mahdollistaa tulosten listaamisen sivulla komponentissa annetun määrän mukaan. Mikäli tämä määrä ylittyy, tarjoaa avustaja seuraavat ja edelliset linkit tulosten selaamiseen.

4.12.2 Session

SessionComponent eli istunto-komponentti tarjoaa tavan säilyttää tietoa sivupyyntöjen välillä. Se toimii PHP:n $_SESSION-päällysluokkana sekä tarjoaa tähän muutamia käyttöä helpottavia metodeja. (CakePHP 2014, www.)

CakePHP käyttää Session-komponenttia käyttäjille tarkoitettujen ilmoitusten asettamiseen. CakePHP:ssa näitä kutsutaan nimityksellä ”flash-messages”.

Sessioviesti näkymälle asetetaan seuraavan esimerkin mukaisesti.

```
$this->Session->setFlash('Tapahtuma tallennettu.');
```

4.12.3 Authentication

Lähes jokainen verkkosovellus tai sen osa, joka ei ole suurelle yleisölle tarkoitettu vaati käyttäjien hallinnan sekä kirjautumisen.

Käyttäjien tunnistaminen, todentaminen sekä käyttöoikeuksien tarkistaminen on yleistä lähes jokaisessa verkkosovelluksessa. CakePHP:ssa *AuthComponent* tarjoaa liitettävän tavan näiden suorittamiseen. (CakePHP 2014, www.)

4.12.4 Request Handling

RequestHandlerComponent eli pyynnönkäsittelijä-komponenttia käytetään CakePHP:ssa keräämään lisätietoja HTTP-pyynnöistä. Sitä voi esimerkiksi käyttää ilmoittamaan ohjaimelle AJAX-pyynnöstä. (CakePHP 2014, www.)

CakePHP:n 2.x-versioissa useat toiminnot on siirretty ohjaimen *response*- ja *request*-objektien alle, mukaan lukien AJAX-pyyntöjen tarkistaminen, minkä ansiosta tätä komponenttia ei tarvitse AJAX-pyyntöjen tarkistamiseen. Seuraavassa esimerkissä esittelen AJAX-pyynnön tarkistamisen *RequestHandler*-komponentilla sekä *request*-objektilla ja vastauksen lähettäminen JSON-muodossa. *RequestHandler*:illä esitelty tapa on poistumassa käytöstä, ja suositeltavaa onkin suorittaa HTTP-pyyntömenetelmän tarkistukset edellä mainitun *request*-objektin kautta. Esittelenkin *RequestHandler:in* tavan ainoastaan siitä syystä, että kyseinen tapa on ollut yleinen ja on käytössä edelleen useissa verkon esimerkeissä.

```
//RequestHandlerin poistumassa oleva tapa
if ($this->RequestHandler->isAjax()) {
  //Poistetaan näkymän suorittaminen
  $this->autoRender = false;
  //Asetetaan HTTP-vastauksen tyypiksi json
  $this->RequestHandler->respondAs('json');
  //Esimerkki vastauksen antamisesta json muodossa
  $Events = $this->Event->find('all');

  return json_encode($Events);
}

//Request ja response objektin suositeltu tapa
if ($this->request->is('ajax')) {
  //Poistetaan näkymän suorittaminen
  $this->autoRender = false;
  //Asetetaan HTTP-vastauksen tyypiksi json
  $this->response->type('json');
  //Esimerkki vastauksen antamisesta json muodossa
  $Events = $this->Event->find('all');

  return json_encode($Events);
}
```

On yleistä, että AJAX-pyynnöissä ei ole tarpeen palauttaa vastauksena muuta kuin tietoa. Siitä syystä yllä olevassa esimerkissä näkymän automaattinen suorittaminen on poistettu käytöstä. Yllä oleva esimerkki on tähän mielestäni yksinkertaisin, mutta vastauksen voisi myös suorittaa CakePHP:n JSON-näkymällä, johon löytyy ohjeet CakePHP:n dokumentaatiosta osoitteesta http://book.cakephp.org/2.0/en /views/json-and-xml-views.html (tai Ks. CakePHP 2014, www).

Komponentti tarjoaa kuitenkin edelleen muita hyödyllisiä toimintoja, joista mainittakoon mobiiliselainten tarkistus.

4.12.5 Cookie

CookieComponent eli eväste-komponentti on päällysluokka PHP:n setcookie-metodille (CakePHP 2014, www). Komponentin tarkoituksena on nopeuttaa ja helpottaa evästeiden luomista.

4.13 Ohjain ja dynaaminen rakenne

Dynaaminen rakenne eli ”dynamic scaffolding” tarkoittaa näkymien dynaamista luomista lennosta. Se on tarkoitettu projektin alkuvaiheessa tietokantarakenteen testaamiseen, eikä sitä tulisi käyttää tuotannossa.

4.14 Näkymät ja avustajat

CakePHP:n avustajat eli ”helpers”, ovat komponentti-tyylisiä luokkia, jotka tarjoavat logiikkaa näkymille, elementeille sekä ulkoasuille (CakePHP 2014, www). Taulukossa 9 on esitelty CakePHP:n asennuksen mukana tulevat avustajat, minkä jälkeen kerron tarkemmin työssä käyttämistäni avustajista. Avustajia on myös tarpeen tullen mahdollista luoda itse.

Taulukko 9. CakePHP:n avustajat.

Avustaja	Kuvaus
CacheHelper	Mahdollistaa sivujen asettamisen välimuistiin.
FormHelper	Avustaa lomakkeiden käsittelyssä.
HtmlHelper	Avustaa HTML-elementtien luomisessa.
JsHelper	Avustaa JavaScriptin kanssa.
NumberHelper	Tarjoaa metodeja numeroiden esittämiselle.
Paginator	Luo sivuttajalinkit, avustaja toimii yhdessä kappaleessa 4.12 esittelemässäni Pagination-komponentin kanssa.
RSS	Avustaa XML:n luonnissa.
SessionHelper	Vastaa istunto-komponentilla asetettujen tietojen näyttämisestä näkymissä.
TextHelper	Tarjoaa metodeja tekstin luomiseksi käyttäjäystävällisemmäksi.
TimeHelper	Auttaa aikatietojen käsittelemisessä.

4.14.1 CacheHelper

CacheHelper eli välimuistiavustaja avustaa näkymien sekä koko ulkoasun välimuistiin laittamisessa. Näkymien välimuistiin laittaminen CakePHP:ssa tallentaa

jäsennetyn ulkoasun yksinkertaisiksi PHP- ja HTML-tiedostoiksi. (CakePHP 2014, www.)

Välimuistiavustaja luo sivusta PHP-tiedoston sellaisena, kun sivulla ensimmäisellä kerralla käytäessä näyttää. Kun sivulla seuraavan kerran käydään, tarkistaa sovellus, mikäli kyseiselle sivulle löytyy välimuistista tiedosto. Mikäli tiedosto löytyy, ohittaa sovellus kokonaan ohjaimen toiminnon. Tämä tarkoittaa sitä, että sovellus ei suorita toimintokohtaista koodia, vaan tarjoaa edellisellä kutsulla suoritutetusta koodista tallennetun version.

Työtä tehdessäni huomasin, että välimuistiavustajaa käytettäessä on oltava varovainen niillä sivustoilla, jotka tarjoavat tietoa, joka vaatii käyttäjiltä sisäänkirjautumista. Mikäli välimuistitiedosto on luotu sellaisen käyttäjän toimesta, joka on kirjautunut sisään, on mahdollista, että tämä tiedosto tarjotaan myös käyttäjille, jotka eivät ole kirjautuneina sisään.

4.14.2 FormHelper

FormHelper eli lomakeavustaja avustaa lomakkeiden luonnissa. Lomakeavustajaa käyttämällä varmistuu siitä, että käytettävä koodi seuraa CakePHP:n nimeämiskäytäntöjä. Käyttämällä lomakeavustajaa on muokkaussivujen kenttien täyttäminen nopeaa. Lomakeavustaja täyttää lomakekentät automaattisesti, mikäli kenttää vastaava arvo löytyy ohjaimessa *request*-objektiin asetetusta *data*-muuttujasta. Data-muuttujan täytyy olla arraytaulukko seuraavan esimerkin mukaisesti:

```
//EventsController/edit
$this->request->data = array(
  'Event' => array(
    'id' => '19',
    'name' => 'Tapahtuman nimi'
  )
)
```

Seuraavassa esimerkissä esittelen, miten lomake-avustajalla luodaan yksinkertainen lomake tapahtuman muokkaus näkymään.

```
echo $this->Form->create('Event');
echo $this->Form->input('id');
echo $this->Form->input('name');
echo $this->Form->end();
```

Kun yllä olevan esimerkin mukaisesti on luotu lomake, muodostuu siitä seuraavaa esimerkkiä vastaava HTML-koodi.

```
<form
	action="/events/edit/19"
	id="EventEditForm"
	method="post"
	accept-charset="utf-8"
>
	<div style="display:none;">
		<input name="_method" value="PUT" type="hidden">
	</div>

	<input
		name="data[Event][id]"
		value="19"
		id="EventId"
		type="hidden"
	>
	<div class="input text required">
		<label for="EventName">Name</label>
		<input
			name="data[Event][name]"
			maxlength="250"
			value="Tapahtuman nimi"
			id="EventName"
			required="required"
			type="text"
		>
	</div>
</form>
```

4.14.3 HtmlHelper

HTML-avustaja avustaa HTML:n luonnissa. HTML-avustajan rooli CakePHP:ssa on tehdä HTML-pohjaisten asetusten luomista sekä muokkausta nopeammaksi ja helpommaksi (CakePHP 2014, www).

HTML-avustaja helpottaa varsinkin linkkien, kuvien, tyyli- sekä JavaScript-tiedostojen ja metatietojen luonnissa. Seuraavan esimerkin mukaisesti on mahdollista lisätä sivustoon tyylitiedostoja.

```
echo $this->Html->css(array('styles', 'bootstrap.min'));
```

HTML-avustaja muodostaa siitä seuraavaa esimerkkiä vastaavan HTML-koodin.

```
<link rel="stylesheet" type="text/css" href="/css/styles.css" />
<link
   rel="stylesheet" type="text/css" href="/css/bootstrap.min.css"
/>
```

4.15 Ohjain ja toiminnot

CakePHP:ssa jokaista ohjaimen public-määreen metodia kutsutaan toiminnoksi. Jokainen toiminto edustaa verkko-osoitetta. Kun verkko-osoitetta pyydetään selaimesta, sitä vastaavaa toimintoa kutsutaan. (Syam ym. 2008, 49.)

Mikäli selaimella kutsutaan osoitetta *http://domain.com/cakephp/events/add*, kutsutaan sillä *events*-ohjaimen metodia *add*. Ohjaimen *index*-metodia kutsutaan osoitteella *http://domain.com/cakephp/events*. CakePHP:ssa *index*-metodi tarkoittaa oletuksena ohjaimeen liittyvän mallin kaikkien tulosten listausta ja toimii ohjaimen oletustoimintona.

4.16 Yhteenveto nimeämiskäytännöistä

Parhaimmillaan CakePHP tekee sovelluskehityksen todella nopeaksi. Edellytyksenä sille, että kaikki toimii CakePHP:n tavalla, on nimeämiskäytäntöjen seuraaminen tärkeää. Mikäli tiedoston tai luokan nimeää CakePHP:n käytäntöjen vastaisesti, on mahdollista, että CakePHP ei onnistu automaattisesti luomaan tarvitsemiansa suhteita ja ilmoittaa siitä virheilmoituksella. Tässä kappaleessa kokoan yhteen edellisissä kappaleissa mainitsemani tärkeimmät nimeämiskäytännöt.

- Tietokantataulujen nimeäminen tapahtuu englanniksi ja monikossa, esimerkiksi *events*.
- Taulujen viiteavaimet nimetään muodolla *viitattava_id*, jossa viitattava on viitattavan taulun yksikkömuotoinen nimi, esimerkiksi *event_id*.
- Mallit nimetään tietokantataulun nimen yksikkömuodossa alkaen isolla kirjaimella, esimerkiksi *Event*. Tiedosto tallennetaan nimellä *Event.php*.

- Ohjaimet nimetään siihen liittyvän mallin perusteella mallin nimen monikkomuodossa lisättynä ”Controller”. Nimi alkaa isolla kirjaimella, esimerkiksi *EventsController*. Tiedosto tallennetaan nimellä *EventsController.php.*
- Ohjaimen toimintoa vastaavat näkymätiedostot tallennetaan metodin nimeä vastaavasti. Esimerkiksi *EventsControllerin add*-toimintoa vastaava näkymä tallennetaan tiedostoon *app/View/Event/add.ctp.*
- *EventsControllerin* toimintoa *add* pyydetään selaimella osoitteessa *http://domain.com/cakephp/events/add.*

4.17 Layout

Layout eli ulkoasu vastaa CakePHP:ssa HTML-teemaa. Kappaleessa 4.3.3 on esitetty kuinka jokainen näkymä vastaa jotakin ohjaimen toimintoa. Näkymän tulisi myös esittää ainoastaan kyseisen toiminnon HTML. Näin ollen se vastaa ainoastaan kyseisen sivun sisällöstä, ei koko teeman esittämisestä.

CakePHP:ssa teemat luodaan *app/View/Layouts*-kansioon. Oletuksena on käytössä default.ctp-niminen teema. Teemojen käyttö mahdollistaa sellaisen HTML:n sijoittamisen yhteen paikkaan, joka esiintyy kaikilla järjestelmän sivuilla. On myös mahdollista luoda useita teemoja, joka mahdollistaa esimerkiksi eri teeman käytön järjestelmän hallintasivuilla. Kappaleen 10.1.4 esimerkissä on nähtävillä oletusteeman vaihto järjestelmän laajuisesti. On myös mahdollista vaihtaa teemaa ohjain- tai jopa toimintokohtaisesti.

Jokaisessa normaalia käytäntöä seuraavassa teemassa kutsutaan näkymää. Tämä tarkoittaa sitä että teemassa luodaan sivuston perusrakenteet, joiden sisälle liitetään kyseistä sivua, eli ohjaimen toimintoa, vastaava näkymä. Tämä tapahtuu seuraavan esimerkin mukaisesti.

```
echo $this->fetch('content');
```

4.18 Elementit ja requestAction

Elementit ovat teemoissa sekä näkymissä haettavia koodipätkiä. Elementit pitävät yleensä sisällään ulkoasullisia osia, jotka esiintyvä useassa eri teemassa tai näkymässä Elementit mahdollistavat esimerkiksi ”Uusimmat uutiset”-tyyppisten osien näyttämisen jokaisella sivulla.

RequestAction-funktio mahdollistaa ohjaimen toiminnon kutsumista mistä tahansa. Tämä mahdollistaa kyseisen HTTP-pyynnön ulkopuolisen toiminnon kutsumisen. Mikäli *requestAction*-funktiota kutsutaan ilman välimuistia, voi se aiheuttaa suorituskyvyllisiä ongelmia (CakePHP 2014, www). Kappaleessa 4.22 esittelen miten elementti on mahdollista asettaa välimuistiin.

RequestAction toimii parhaiten elementeissä, koska ne on mahdollista asettaa välimuistiin. Aikaisemmin mainitsemani ”Uusimmat uutiset”-tyyppisen elementin on oletettavaa tarvitsevan *requestAction*ia hakeakseen uutiset esimerkiksi *news-Controller*-nimisestä ohjaimesta. Funktiota on mahdollista kutsua seuraavan esimerkin mukaisesti.

```
$news = $this->requestAction('/news/get_newest');
```

4.19 Konsolisovelluskehys ja Bake-konsolisovellus

Verkkosovelluskehyksen lisäksi CakePHP tarjoaa myös konsolisovelluskehyksen konsolisovellusten luontiin. Konsolisovellukset ovat optimaalisia taustatehtävien, kuten huoltotoimenpiteiden, suorittamiseen tai tehtävien loppuun viemiseen ilman käyttäjän vuorovaikutusta. (CakePHP 2014, www.)

CakePHP:n asennuksen mukana tulee muutamia konsolisovelluksia. CakePHP:ssa myös cron-tehtäviä on mahdollista suorittaa konsolisovelluksen kautta. Cron-tehtävät ovat ajoitettuja tehtäviä, joita järjestelmä suorittaa taustalla määritetyin aikavälein. Taulukossa 10 esittelen muutaman yleisimmän sisäänrakennetun sovelluksen. Tämän jälkeen kerron tarkemmin työssä käyttämästäni konsolisovelluksesta Bake.

Taulukko 10. CakePHP:n konsolisovellukset.

Sovellus	Kuvaus
acl	Tämä on tarkoitettu acl-tietokannan hallintaan ja tarkasteluun.
bake	Mahdollistaa mallien, ohjainten sekä näkymien luomisen.
i18n	Tarjoaa nopean tavan kielten käännöspohjien luonnille.
schema	Sovellus mahdollistaa tietokantakuvausten luonnin sekä tietokannan päivittämisen.

Bake

CakePHP tarjoaa meille helpon tavan luoda perussovelluspohja-koodit, joista varsinaisen työn voi aloittaa. Sen sijaan, että loisimme kaikki sovelluskoodit tyhjästä, voimme käyttää konsolikomentoja koodin luomiseen. Se mahdollistaa nopean tavan luoda uusia malleja, ohjaimia sekä näkymiä sovellukseemme käyttämällä konsolikomentoja. (Syam ym. 2008, 165.)

CakePHP:n konsolisovellus *Bake* tarjoaa nopean tavan saada projekti alkuun. *Bake*-sovelluksella pystyy luomaan sovelluksen peruskomponentit (CakePHP 2014, www). Tässä työssä käytän jatkossa *Bake*-sovelluksen suorittamisesta nimitystä leipominen. Leipomisella on mahdollista luoda täysin toimiva CRUD-pohjainen käyttöliittymä ja kaikki tämä, riippuen tietenkin järjestelmän koosta, on mahdollista tehdä vain muutamassa minuutissa. Esittelen konsolisovelluksen käyttämistä sekä leipomista Bake-konsolisovelluksella kappaleessa 10.2.

4.20 Kansainvälistäminen sekä lokalisoiminen

Kansainvälistäminen ja lokalisoiminen, eli Internationalization and Localization, usein lyhennettynä i18n ja l10n, tarkoittavat yksinkertaisesti sanottuna järjestelmässä tai sivustossa olevaa useamman kuin yhden kielen valmiutta. Kansainvälistäminen viittaa järjestelmän valmiuteen lokalisoimiselle, ja lokalisoiminen tarkoittaa järjestelmän sopeuttamista tietyn kielen tai kulttuurin vaatimuksiin (CakePHP 2014, www).

Opinnäytetyönä toteuttamassani varastonhallintajärjestelmässä ei ole tarvetta useaan kielen tukeen. On kuitenkin erittäin yleistä, varsinkin Suomessa, että sivustot tukevat useita kieliä. Esittelen lyhyesti, miten CakePHP:ssa tekstien kääntäminen onnistuu. On kuitenkin huomioitava, että pelkästään tekstien kääntäminen toiselle kielelle ei vielä välttämättä tarkoita täysimittaista sivuston lokalisoimista. Kappaleen lopussa mainitsen, mitä muuta CakePHP tarjoaa lokalisoimisen tueksi.

Ensimmäiseksi tulee näkymien merkkijonot ajaa CakePHP:n tekstikäännös funktion *__()* läpi. *__()*-funktio palauttaa merkkijonon käännetyn version, mikäli käännös on saatavilla, muuten se palauttaa merkkijonon sellaisenaan (CakePHP 2014, www). Seuraavassa esimerkissä on nähtävissä yksikielisen ja monikielisen järjestelmän ero, kun luodaan sivuston sisältöä.

```
<!--Käännösvalmis merkkijono-->
<h1><?php echo __('Kaikki osastot'); ?></h1>
<!--Yksikielinen merkkijono-->
<h1>Kaikki osastot</h1>
```

Seuraava tehtävä on ajaa CakePHP:n *i18n*-konsolisvellus. Sovellus etsii merkkijonoja, jotka ovat ajettu edellisessä esimerkissä esitellyn *__()*-funktion läpi. Sovellus luo löydettyjen merkkijonojen perusteella käännöspohjan, jota kehittäjä käyttää käännöstiedostojen luomiseen. Muita merkkijonojen kääntämiseen tarkoitettuja funktioita ovat *__n()*, *__d()* sekä *__dn()*. d-kirjaimella esitellyt käännösfunktiot ovat käteviä lisäosien käännöksissä. Kun *i18n*-sovelluksen ajaminen on valmis, löytyy luotu POT-tiedosto kansiosta *app/Locale/default.pot*.

Käännösten luomiseen on CakePHP:n dokumentaatiossa suositeltu käytettävän ilmaista PoEdit-työkalua, joka toimiikin todella kätevästi. Sovelluksen saa ladattua osoitteesta http://poedit.net/.

Mikäli luodaan suomenkielistä käännöstä, tallennetaan PoEditillä luotu PO-tiedosto kansioon *app/Locale/fin/LC_MESSAGES* tiedostonimellä *default.po*. Käännökset ovat valmiina, mutta CakePHP on mahdollisesti osoitettava käyttämään tarkoitettua kieltä valitun kielenhallintatavan mukaan. Suositeltu tapa on asettaa käytetty kieli sessioon. Sessioon asetettu kieli on mahdollista tarkistaa esimerkiksi *AppControllerin beforeFilter*-metodissa ja asettaa se CakePHP:n *Configure*-luokan *Config.language*-asetukseen jokaisen sivunlatauksen yhteydessä. Toinen mahdollinen tapa on asettaa kieli evästeeseen, jolloin kieli on mahdollisesti valmiina, mikäli käyttäjä tulee sivustolle uudelleen. CakePHP tarjoaa käännösten hallintaan myös *Translate*-käyttäytymisen. CakePHP:n kansainvälistämisestä sekä lokalisoimisesta on mahdollista lukea CakePHP:n dokumentaatiosta osoitteesta http://book.cakephp.org/2.0/en/core-libraries/internationalization-and-localization.html (tai Ks. CakePHP 2014, www).

CakePHP tarjoaa myös virallisen lisäosan lokalisoimisen avuksi. Lisäosan nimi Localized, ja se tarjoaa useita lokalisoituja validointi luokkia sekä käännöksiä. Opinnäytetyötä tehdessä ei lisäosa kuitenkaan tarjoa näitä suomenkielelle. Lisäosa on löydettävissä osoitteesta https://github.com/cakephp/localized.

4.21 Virheenpaikannus ja DebugKit

Virheenpaikannus eli debuggaus on väistämätön osa jokaisen järjestelmän kehityksessä. CakePHP:ssa virheenpaikannukseen on luotu muutamia helpottavia työvälineitä, jotka nopeuttavat virheenpaikannusta huomattavasti. CakePHP:n virheenpaikannustoimintoja varten tulee *Configure::debug*-asetus olla suurempi kuin 0. (CakePHP 2014, www.)

Yllä mainittu *debug*-asetus on mahdollista asettaa *app/Config/core.php*-tiedostoon seuraavan esimerkin mukaisesti.

```
Configure::write('debug', 2);
```

CakePHP tarjoaa virheenpaikannusluokan nimeltä *Debugger*. Luokka korvaa PHP:n oletusvirheenhallinnan ja se on oletuksena käytössä. Kun järjestelmässä tapahtuu virhe, esittää *Debugger*-luokka virhettä vastaavaa tietoa sekä kirjaa virheen virhelokiin (Kuvio 21).

CakePHP:n *debug*-funktio on globaalisti käytössä oleva funktio, joka tulostaa muuttujan helposti luettavassa muodossa. Funktion toiminta vastaa PHP:n *var_dump*-funktiota (CakePHP 2014, www).

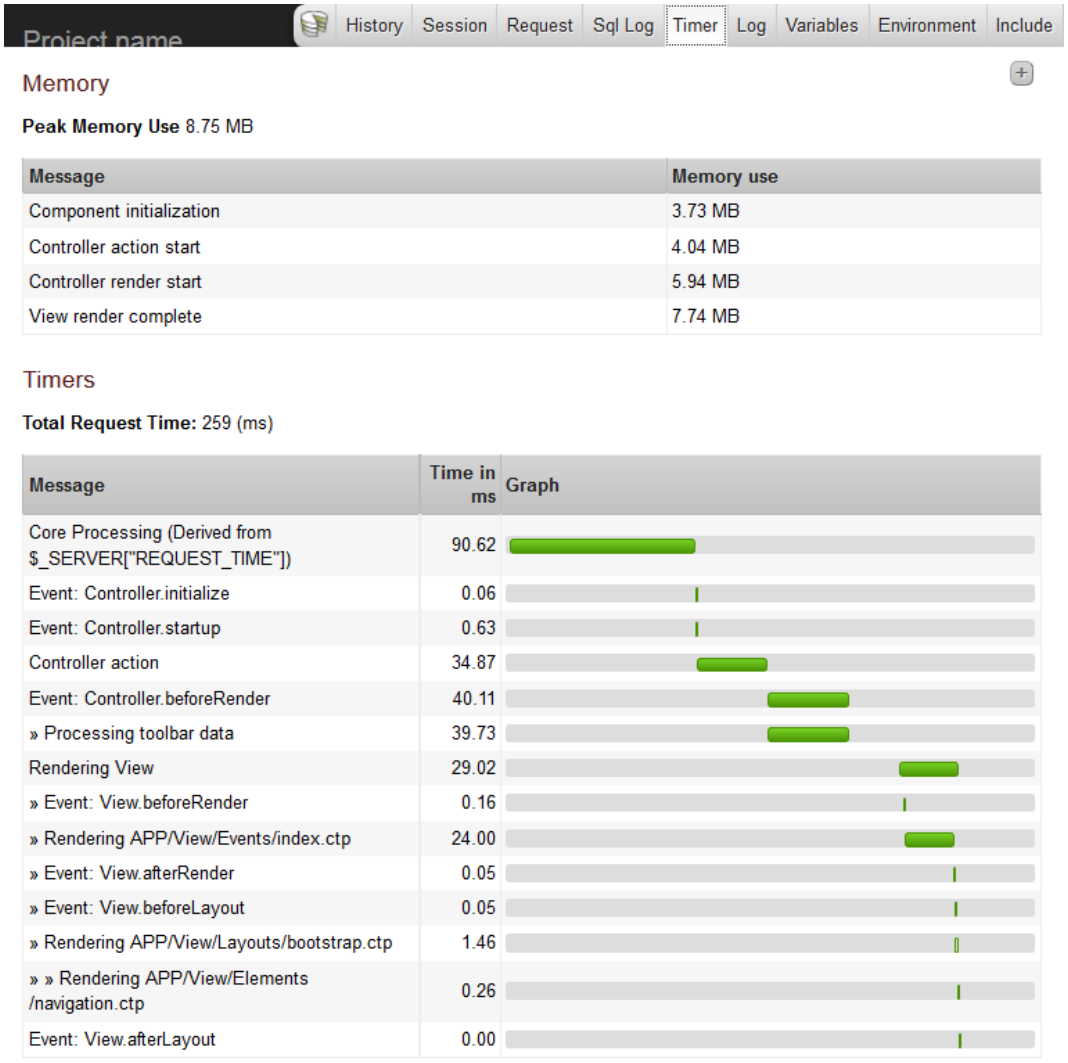
DebugKit

Virheenpaikannusta helpottamaan tarjoaa sovelluskehys myös virallisen lisäosan nimeltä *DebugKit*. Lisäosa esittää suuren määrän pyynnöstä saatavilla olevaa tietoa. Näytettävät tiedot ovat valittavissa paneelien muodossa ja lisäosa tarjoaa myös mahdollisuuden luoda omia paneeleita. Taulukossa 11 on esiteltynä lisäosan mukana tulevat paneelit.

Taulukko 11. DebugKitin oletuspaneelit.

Paneeli	Sisältö
History	Paneeli mahdollistaa pääsyn edellisiin pyyntöihin.
Session	Paneeli esittää Sessiossa olevat tiedot.
Request	Paneeli esittää tietoa nykyisestä pyynnöstä.
Sql Log	Paneeli esittää nykyisen pyynnön kaikki SQL-kyselyt.
Timer	Paneeli esittää ja erittelee pyynnön käsittelyssä käytettyjä järjestelmän osia sekä ilmoittaa pyynnön eri osissa käytettyä muistia sekä aikaa.
Variables	Paneeli esittää ohjaimessa näkymälle asetettuja muuttujia.
Environment	Paneeli esittää sovelluskohtaiset, eli itse asetetut, ja CakePHP-asennuksen vakiot sekä PHP ympäristömuuttujat.
Include	Paneeli esittää järjestelmään liitetyt tiedostot sekä polut.

Lisäosa ja sen käyttöönotto löytyy osoitteesta https://github.com/cakephp/debug_kit. Kun lisäosa on otettu käyttöön, ilmestyy sivustolle oikeaan yläkulmaan uuden sivunlatauksen yhteydessä ikoni. Tämän ikonin takaa löytyy *DebugKit*. Ikonia painettaessa ilmestyy näkyville valikko, jossa on oletuksena kaikki paneelit käytössä. Avaamalla yhden paneeleista on paneelin tarjoama tieto nähtävillä (Kuvio 2).



Kuvio 2. DebugKitin Timer-paneeli.

4.22 Välimuisti luokka

Välimuistiin asettamista käytetään vähentämään aikaa ulkoisista resursseista luettaessa tai sinne kirjoitettaessa (CakePHP 2014, www). CakePHP:n *Cache*-luokka mahdollistaa esimerkiksi SQL-kyselyiden sekä elementtien tallentamisen. Tallen-

taminen tapahtuu, mikäli sivulla käydessä ei vastaavaa välimuisti-objektia löydy. Tällöin myös kyseinen SQL-kysely tai elementin koodi suoritetaan. Mikäli välimuisti-objekti löytyy, käyttää CakePHP kyseisen objektin sisälle tallennettua tietoa ja koodin suorittaminen ohitetaan. Välimuistista lukeminen on huomattavasti nopeampaa kuin ulkoisesta resurssista lukeminen.

CakePHP:n välimuisti luokkaan on sisäänrakennettu muutamia välimuisti moottoreita ja se mahdollistaa omien välimuisti järjestelmien luonnin. Luokka toimii yhdenmukaisesti kaikille moottoreille. Luokkaan sisäänrakennetut moottorit ovat:

- FileCache
- ApcCache
- Wincache
- XcacheEngine
- MemcacheEngine
- MemcachedEngine
- RedisEngine.

Oletuksena käytössä on *FileCache*, joka kirjoittaa välimuistit tiedostoon. *FileCache* on hitain välimuisti moottori, mutta se on helpoiten käyttöönotettavissa. CakePHP:n dokumentaatiossa, osoitteessa http://book.cakephp.org/2.0/en /core-libraries/caching.html, on luettavissa lisää muista moottoreista. Sieltä löytyvät myös linkit moottoreiden käyttämiin PHP-lisäosiin.

Välimuisti luokan määrittely on hyvä tehdä *app/Config/bootstrap.php*-tiedostossa. Luokka mahdollistaa useiden eri välimuistiasetuksien luomisen. Asetuksessa on mahdollista määritellä esimerkiksi:

- Engine - käytettävä välimuisti moottori
- Duration - välimuistin kesto
- Path - välimuistin polku
- Prefix - välimuistin nimeä edeltävä tunnus.

Seuraava esimerkki esittää oman välimuisti-asetuksen luomista *app/Config /bootstrap.php*- tiedostoon sekä kyseisen asetuksen käyttämistä asetettaessa elementti välimuistiin teema-tiedostossa.

```
//app/Config/bootstrap.php
Cache::config('elements', array(
    'engine' => 'File',
    'duration' => '+1 hour',
    'path' => CACHE
));

//View/Layouts/default.php
echo $this->Element('navigation', array(), array(
  'cache' => array(
    'config' => 'elements',
    'key' => 'navigation'
  )
));
```

Yllä oleva esimerkki luo *app/tmp/cache*-kansioon *cake_element_navigation*-nimisen tiedoston sivulla käydessä, mikäli kyseistä tiedostoa ei vielä ole tai se on vanhentunut. On huomioitavaa, että elementin asettaminen välimuistiin on pääsääntöisesti tarpeellista ainoastaan, mikäli elementistä kutsutaan jotain ulkoista lähdettä tai kutsutaan jonkin ohjaimen toimintoa *requestAction*-metodilla. Mikäli elementti pitää sisällään ainoastaan HTML-koodia, ei sen välimuistiin asettamisesta ole luultavasti hyötyä.

Kokonaisten sivujen sekä näkymien asettamisesta välimuistiin olen esittänyt kappaleessa 4.14.1.

4.23 Testaaminen

CakePHP mahdollistaa kattavan testaamisen PHPUnitin sisäänrakennetulla tuella (CakePHP 14, www). PHPUnit on PHP-sovelluskehys yksikkötestaamiseen. Yksikkötestaaminen tarkoittaa pienten koodipätkien testaamista, millä varmistetaan, että koodi toimii oletetusti. Tämän lisäksi on CakePHP:hen luotu myös muita testausta helpottavia ominaisuuksia.

Yksikkötestejä on mahdollista luoda itse ja niitä tulisi ajaa mahdollisimman usein. Näin varmistutaan siitä, että uutta koodia luodessa ei mikään ole hajonnut. PHPUnitin asentamisesta sekä testien luomisesta ja ajamisesta on mahdollista lu-

kea lisää CakePHP:n dokumentaatiosta http://book.cakephp.org/2.0/en /development/testing.html (tai Ks. CakePHP 2014, www).

5 KÄYTTÖÖNOTTO

CakePHP:n asennus ja tehokas käyttäminen edellyttävät HTTP-palvelinta, esimerkiksi Apachea, mutta CakePHP:n voi määritellä toimimaan myös LightHTTPD:llä tai Microsoft IIS:llä. Mikäli käytössä on Apache, on suositeltavaa, että *mod_rewrite* on käytettävissä, mutta tämä ei ole pakollista. PHP:sta tulisi olla versio 5.2.8 tai uudempi. Tietokanta ei ole pakollinen, mutta luultavasti suurin osa sovelluskehyksenkäyttäjistä tulee käyttämään tietokantaa. CakePHP tukee seuraavia tietokantoja; MySQL 4 tai uudempi, PostgreSQL, Microsoft SQL Server sekä SQLite. (CakePHP 2014, www.)

Tämä opinnäytetyö on suoritettu Apache-ympäristössä, jossa CakePHP:n käyttöönotto on yksinkertaista.

5.1 CakePHP:n lataaminen

Uusimman version lataaminen tapahtuu menemällä selaimella osoitteeseen cakephp.org ja klikkaamalla isoa Download-linkkiä ja tallentamalla tiedoston (Kuvio 3). Tiedosto tallentuu ZIP-muotoisena. Mikäli käytössä on GIT, voi latauksen suorittaa myös komennolla *git clone git://github.com/cakephp/cakephp.git*.

Kuvio 3. CakePHP:n kotisivut.

5.2 CakePHP:n asennuksen purkaminen

Ladattu ZIP-tiedosto puretaan ja siirretään palvelimelle esimerkiksi Windows-ympäristössä WinSCP:llä tai omalla tiedostonsiirto-ohjelmalla. Siirtäminen onnistuu myös siirtämällä ZIP-tiedosto palvelimelle tiedostonsiirto-ohjelmalla ja purkamalla sen SSH-asiakasohjelmalla. Kappaleessa 5.3 esittelen CakePHP:n asennuksen purkamisen PuTTY SSH-asiakasohjelmalla.

5.3 CakePHP:n asennuksen purkaminen käyttäen PuTTY:ä

Asennuksen purkaminen SSH-asiakasohjelmalla, kuten PuTTY:llä, nopeuttaa prosessia huomattavasti. Mikäli PuTTY ei ole käytettävissä, katso kappale 10.1.5, jossa esittelen PuTTY:n lataamisen. Kun PuTTY on auki ja yhdistetty palvelimelle, tulee se navigoida kansioon, johon ladattu ZIP-tiedosto on siirretty, mikäli se

on siirretty käyttäen erillistä tiedostonsiirto-ohjelmaa. Kansioon siirtyminen tapahtuu esimerkiksi komennolla *cd public_html.*

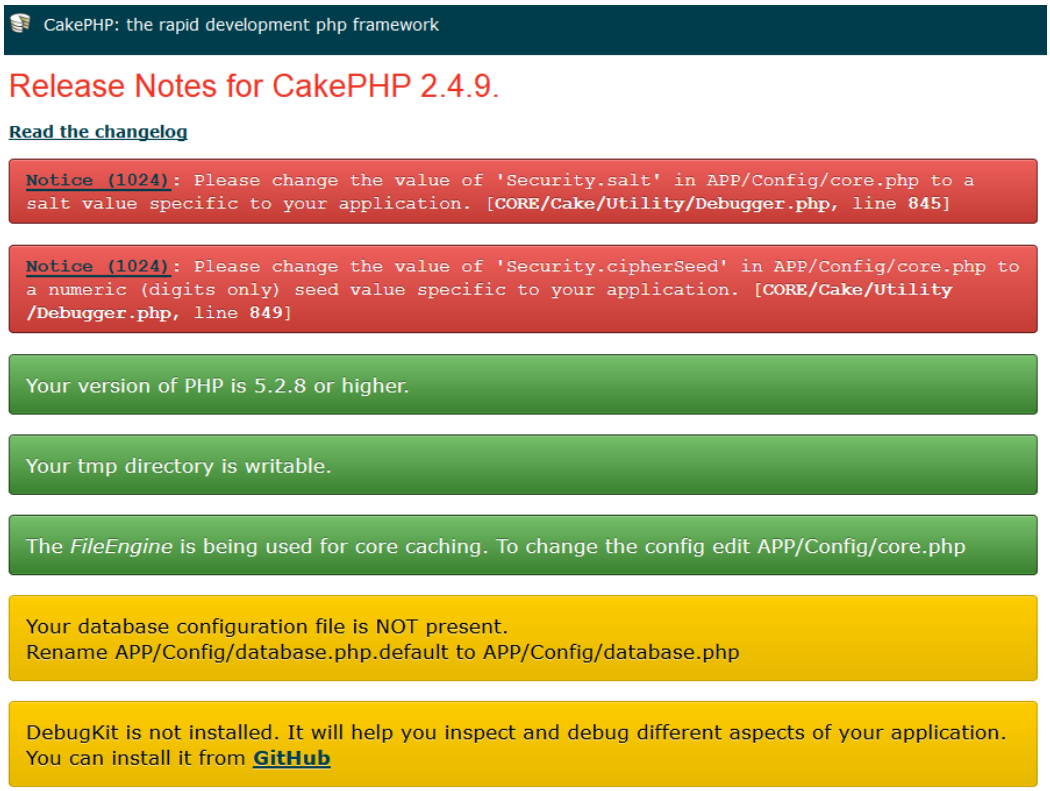
Mikäli tiedostoa ei vielä ole siirretty tai ladattu on se mahdollista tehdä esimerkiksi seuraavalla komennolla: *wget https://github.com/cakephp/cakephp/archive/2.4.9.zip*. Komento lataa kyseisen tiedoston palvelimelle ja tiedoston nimi on opinnäytetyötä tehdessä *2.4.9.zip*. Uusimman version latauslinkki löytyy osoitteesta https://github.com/cakephp/cakephp/tags.

Opinnäytetyötä tehdessä on tiedosto ladattu kappaleen 5.1 mukaisesti ja tiedoston nimi on cakephp-2.4.9.zip. Tiedoston purkaminen tapahtuu komennolla *unzip cakephp-2.4.9.zip.* Purettu kansio on vastaavasti nimetty *cakephp-2.4.9:ksi*. Nimetään tämä kansio uudestaan omaa projektia vastaavaksi tai esimerkiksi *cakephp*:ksi komennolla *mv cakephp-2.4.9 cakephp.*

Poistetaan ladattu tai siirretty ZIP-tiedosto komennolla *rm cakephp-2.4.9.zip.*

5.4 CakePHP:n asetusten määrittäminen

Kun tiedosto on siirretty ja purettu, on aika avata selain ja osoittaa se osoitteeseen *http://domain.com/cakephp*. Oletusnäkymän tulisi olla kuvion 4 mukainen.



Kuvio 4. CakePHP:n asennustilanne.

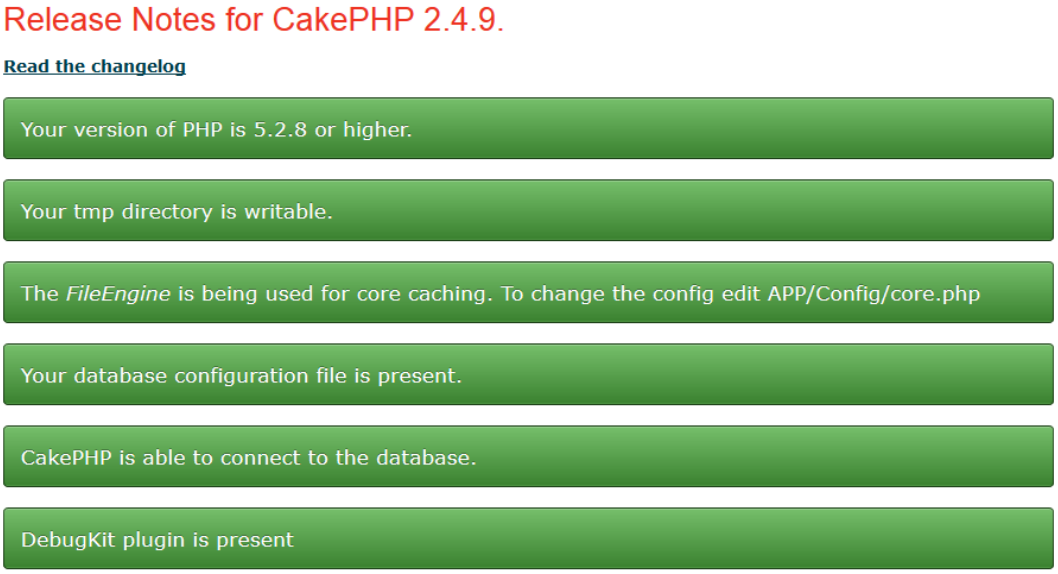
CakePHP ilmoittaa asennusta koskevista asetuksista, jotka vaativat mahdollisia toimia. Kuvankaappauksessa näkyvät ilmoitukset ovat eriteltynä seuraavassa listauksessa:

- CakePHP pyytää vaihtamaan *Security.salt*:in eli suojasuolauksen arvon *APP/Config/core.php*-tiedostossa.
- CakePHP pyytää vaihtamaan *Security.chipherSeed*-arvon *APP/Config/core.php* tiedostossa.
- PHP:n versio on tarpeeksi uusi.
- *Tmp* kansio on kirjoitettavissa.
- *FileEngine* on käytössä välimuistia varten.
- Tietokantatiedostoa ei ole olemassa.
- DebugKit ei ole asennettuna.

Seucurity.salt- sekä *Security.chipherseed*-arvoja ei ole pakko vaihtaa, mutta se on turvallisuuden kannalta suositeltavaa. DebugKitin olen esitellyt kappaleessa 4.21,

mutta myöskään se ei ole pakollinen. Tietokantatiedoston luonti tapahtuu kopioimalla tai uudelleen nimeämällä *app/Config*-kansiossa oleva *database.php.default*-tiedosto *database.php*:ksi. Tietokannan tiedot tulee antaa tänne. (CakePHP 2014, www.)

Kun tarvittavat toiminnot on suoritettu, tulisi sivun näyttää kuvion 5 mukaiselta.



Kuvio 5. CakePHP:n oletussivu onnistuneen asennuksen jälkeen.

Mikäli ilmoitukset ovat kunnossa, on asennus suoritettu onnistuneesti, ja CakePHP on valmis käytettäväksi.

6 TOTEUTETTAVAN JÄRJESTELMÄN KARTOITUS

Olen tähän mennessä esitellyt työhön käytettyjä tekniikoita sekä käytettävän sovelluskehyksen perusteita ja sen asennuksen. Tässä luvussa kerron miten yrityksessä nykyään toimitaan ja pyrin sitä kautta esittämään yleiskuvaa siitä, mitä järjestelmän oletetaan ratkaisevan.

Järjestelmän suunnittelu alkoi haastattelemalla yrityksen työntekijöitä siitä, miten asiat nyt tehdään ja miten niitä voisi parantaa. Sen jälkeen mietittiin, mitkä järjestelmän ominaisuudet ovat pakollisia toimivan kokonaisuuden toteuttamiseksi.

6.1 Nykytilan kartoitus, miten nyt toimitaan.

Messunikkarit Oy on pienyritys, jonka päätoimenkuvana on messuosastojen pystytys. Messutapahtumia on useita vuodessa, kotimaassa ja ulkomailla. Usein eri tapahtumat ajoittuvat samanaikaisesti. Messuosaston valmisteluun kuuluu uusien komponenttien rakentaminen ja jo valmiina olevien komponenttien uudelleen käyttämisen varmistaminen. Huomattava osa ajasta kuluu valmiina olevien komponenttien etsimiseen ja pakkaamiseen laatikoihin kuljetusta varten. Pakkaamisprosessi alkaa pakkauslistojen luomisella, joka nykytilanteessa tapahtuu manuaalisesti tai Excel-taulukoiden avulla. Tämä pakkauslista tulostetaan ja annetaan varastohenkilökunnan käsittelyyn. Varastohenkilökunta pakkaa tarvittavan määrän jokaista tuotetta ja merkkaa sen listaan pakatuksi. Ongelmat syntyvätkin inhimillisistä virheistä, kuten tuotteet voivat unohtua listasta, väärä tuote voidaan pakata epäselvyyksien vuoksi tai vastaavasti sama tuote voidaan pakata useaan kertaan huonon merkintätavan seurauksena. Ongelmia voi ilmaantua myös tuotteiden määrän suhteen. Varastossa olevista tuotteista ei ole olemassa määrällistä ylläpidettävää listausta. Tämä voi johtaa siihen, että varastosta etsitään tuotetta, joka onkin jo maailmalla. Suurimpana haittana on se, että erehdys huomataan mahdollisesti vasta osaston pystytyksen yhteydessä, yleensä ulkomailla, jolloin vastaavan tuotteen saaminen on hankalaa, jollei mahdotonta.

6.2 Vaatimusmäärittely

Erillistä vaatimusmäärittelyä järjestelmän toiminnasta ei tehty. Järjestelmän vaatimukset saatiin selville haastattelemalla yrityksen työntekijöitä ja kuvaamalla käyttötapaukset, jotka esittelen luvussa 7.

7 TUNNISTETUT KÄYTTÖTAPAUKSET

Käyttötapaukset kuvaavat miten järjestelmää tullaan käyttämään ja minkälaiset ulkopuoliset vaikuttajat voivat olla vuorovaikutuksessa järjestelmän kanssa (Lano 2009, 10). Käyttötapaukset kuvaavat järjestelmän toiminnallisia vaatimuksia.

Käyttötapauksissa ei ole huomioitu erilaisia käyttäjärooleja, koska kyseessä on pienyritys eikä eri rooleille ole tarvetta. Poikkeuksena kuitenkin käyttäjien hallinta, joka on sallittu ainoastaan hallintatason käyttäjille.

Käytettävyysvaatimukset ovat kaikille käyttötapauksille samat, ja ne ovat, että järjestelmä toimii odotetusti. Tässä luvussa esittelen järjestelmästä tunnistetut käyttötapaukset.

7.1 Varastoryhmät

Varastoryhmän luominen

Yleiskuvaus: Järjestelmän varastonhallintaan tulee luoda tuotekategorioita, joiden alle tuotteet kiinnitetään.

Esiehto: Käyttäjä on kirjautunut järjestelmään.

Tekstikuvaus: Käyttäjä avaa Varasto-välilehden. Avaa Uusi ryhmä-sivun. Syöttää ryhmän nimen. Syöttää ryhmän kuvauksen. Painaa Lähetä-painiketta.

Poikkeuksien kuvaukset:

1. Ryhmän nimikenttä on tyhjä. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Ryhmän nimikenttään syötetään liian pitkä arvo. Järjestelmän täytyy estää liian pitkän arvon syötön.

Lopputulos: Järjestelmässä on näkyvissä ja käytettävissä uusi varastoryhmä.

Varastoryhmän poistaminen

Yleiskuvaus: Järjestelmästä tulee pystyä poistamaan olemassa olevia ryhmiä.

Esiehto: Ryhmä on luotuna järjestelmään.

Tekstikuvaus: Käyttäjä avaa Varasto-välilehden ja painaa haluamansa ryhmän kohdalla poistamiseen tarkoitettua painiketta.

Lopputulos: Varastoryhmä on poistettu tuotteineen eikä enää näy ryhmien listauksessa.

Varastoryhmän muokkaaminen

Yleiskuvaus: Järjestelmän varastoryhmien nimeä sekä kuvausta pystyy muokkaamaan.

Esiehto: Ryhmä on luotuna järjestelmään.

Tekstikuvaus: Käyttäjä valitsee Varasto-välilehdestä haluamansa ryhmän ja siirtyy muokkaussivulle Muokkaa-linkistä. Käyttäjä voi muokata haluamaansa kenttää ja painaa Lähetä.

Poikkeuksien kuvaukset:

1. Ryhmän nimikenttä on tyhjä. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Ryhmän nimikenttään syötetään liian pitkä arvo. Järjestelmä estää liian pitkän arvon syötön.

Lopputulos: Varastoryhmän muokatut tiedot näkyvät ryhmien listauksessa.

7.2 Tuotteet

Tuotteen lisäys varastoryhmään

Yleiskuvaus: Järjestelmän varastoryhmiin täytyy pystyä kiinnittämään tuotteita.

Esiehto: Ryhmä on luotuna järjestelmään.

Tekstikuvaus: Käyttäjä avaa Varasto-välilehdestä haluamansa ryhmän painamalla ryhmän nimeä. Käyttäjä siirtyy lisäämään uutta tuotetta ryhmään painamalla

Lisää tuote ryhmään -linkkiä. Syöttää tuotteelle haluamansa tiedot ja painaa Lähetä.

Poikkeuksien kuvaukset:

1. Käyttäjä jättää pakollisen kentän tyhjäksi. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Käyttäjä syöttää virheellisen arvon kenttään. Järjestelmä ilmoittaa virheellisestä arvosta.

Lopputulos: Tuote näkyy kohderyhmän tuotelistauksessa.

Tuotteen poistaminen

Yleiskuvaus: Järjestelmän varastonhallintaan lisättyjä tuotteita tulee pystyä poistamaan.

Esiehto: Tuote on lisättynä ja kiinnitettynä jonkin varastoryhmän alle.

Tekstikuvaus: Käyttäjä painaa varastoryhmän tuotelistauksessa haluamansa tuotteen kohdalta poistamiseen tarkoitettua painiketta.

Lopputulos: Järjestelmä ilmoittaa onnistuneesta poistamisesta ja tuote poistuu järjestelmästä.

Tuotteen muokkaaminen

Yleiskuvaus: Järjestelmän varastonhallintaan lisättyjä tuotteita täytyy pystyä muokkaamaan.

Esiehto: Tuote on lisättynä ja kiinnitettynä jonkin varastoryhmän alle.

Tekstikuvaus: Käyttäjä valitsee varastoryhmän tuotelistauksessa haluamansa tuotteen painamalla tuotteen nimeä. Käyttäjä painaa Muokkaa-painiketta. Käyttäjä muokkaa haluamiansa kenttiä ja painaa Lähetä.

Poikkeuksien kuvaukset:

1. Käyttäjä jättää pakollisen kentän tyhjäksi. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Käyttäjä syöttää virheellisen arvon kenttään. Järjestelmä ilmoittaa virheellisestä arvosta.

Lopputulos: Järjestelmä ilmoittaa onnistuneesta muokkauksesta ja tuotteen muokatut tiedot näkyvät tuotteen näkymässä.

7.3 Tapahtumat

Tapahtuman luominen

Yleiskuvaus: Järjestelmään täytyy pystyä luomaan tapahtumia.

Esiehto: Käyttäjä on kirjautunut sisään.

Tekstikuvaus: Käyttäjä painaa Tapahtumat-välilehdessä Uusi tapahtuma-linkkiä. Käyttäjä täyttää haluamansa kentät.

Poikkeuksien kuvaukset:

1. Käyttäjä jättää pakollisen kentän tyhjäksi. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Käyttäjä syöttää virheellisen arvon kenttään. Järjestelmä ilmoittaa virheellisestä arvosta.

Lopputulos: Järjestelmä ilmoittaa onnistuneesta lisäyksestä ja lisätty tapahtuma tulee näkyviin tapahtumien listaussivulla.

Tapahtuman muokkaaminen

Yleiskuvaus: Järjestelmään lisättyjä tapahtumia täytyy olla mahdollista muokata.

Esiehto: Tapahtuma on lisätty järjestelmään.

Tekstikuvaus: Käyttäjä painaa haluamansa tapahtuman nimeä Tapahtumat-välilehdessä. Käyttäjä painaa Muokkaa tapahtumaa-linkkiä ja muokkaa haluamiansa kenttiä.

Poikkeuksien kuvaukset:

1. Käyttäjä jättää pakollisen kentän tyhjäksi. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Käyttäjä syöttää virheellisen arvon kenttään. Järjestelmä ilmoittaa virheellisestä arvosta.

Lopputulos: Järjestelmä ilmoittaa onnistuneesta muokkauksesta ja tapahtuman muokatut tiedot näkyvät tuotteen näkymässä.

Tapahtuman poistaminen

Yleiskuvaus: Järjestelmästä tulee pystyä poistamaan olemassa olevia tapahtumia.

Esiehto: Tapahtuma on luotu järjestelmään.

Tekstikuvaus: Käyttäjä avaa Tapahtumat-välilehden ja painaa haluamansa tapahtuman kohdalla poistamiseen tarkoitettua painiketta.

Lopputulos: Tapahtuma on poistettu osastoineen eikä enää näy tapahtumien listauksessa.

Tapahtuman merkitseminen saapuneeksi

Yleiskuvaus: Järjestelmä ei pysty tietämään, onko tapahtuman kuorma saapunut takaisin ilman, että käyttäjä siitä ilmoittaa.

Esiehto: Tapahtuma on luotu järjestelmään.

Tekstikuvaus: Käyttäjä siirtyy Tapahtumat-välilehdelle ja painaa saapumisen ilmoittamiseen tarkoitettua valintaruutua. Käyttäjä hyväksyy järjestelmän vahvistuksen.

Lopputulos: Tapahtuman Saapunut-valintaruutuun ilmestyy ok-merkintä. Tapahtuma ei enää listaudu aktiivisten tapahtumien alle sivua päivitettäessä.

7.4 Osastot

Osaston lisäys tapahtumaan

Yleiskuvaus: Järjestelmään lisättyihin tapahtumiin täytyy pystyä kiinnittämään osastoja.

Esiehto: Tapahtuma on luotuna järjestelmään.

Tekstikuvaus: Käyttäjä avaa Tapahtumat-välilehdestä haluamansa tapahtuman painamalla tapahtuman nimeä. Käyttäjä siirtyy lisäämään uutta osastoa tapahtumaan painamalla Uusi osasto-painiketta. Syöttää osastolle haluamansa tiedot ja painaa Lähetä.

Poikkeuksien kuvaukset:

1. Käyttäjä jättää pakollisen kentän tyhjäksi. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Käyttäjä syöttää virheellisen arvon kenttään. Järjestelmä ilmoittaa virheellisestä arvosta.

Lopputulos: Järjestelmä ilmoittaa onnistuneesta lisäyksestä, ja osasto näkyy tapahtuman osastolistauksessa.

Osaston poistaminen

Yleiskuvaus: Järjestelmään lisättyjen tapahtumien osastoja täytyy pystyä poistamaan.

Esiehto: Osasto on lisättynä ja kiinnitettynä jonkin tapahtuman alle.

Tekstikuvaus: Käyttäjä painaa tapahtuman osastolistauksessa haluamansa osaston kohdalta poistamiseen tarkoitettua painiketta.

Lopputulos: Järjestelmä ilmoittaa onnistuneesta poistamisesta, ja tuote poistuu järjestelmästä pakkauslistoineen.

Osaston muokkaaminen

Yleiskuvaus: Järjestelmään lisättyjen tapahtumien osastoja täytyy pystyä muokkaamaan.

Esiehto: Osasto on lisättynä ja kiinnitettynä jonkin tapahtuman alle.

Tekstikuvaus: Käyttäjä valitsee tapahtuman osastolistauksessa haluamansa osaston painamalla osaston nimeä. Käyttäjä painaa Muokkaa osastoa-painiketta. Käyttäjä muokkaa haluamiansa kenttiä ja painaa Lähetä.

Poikkeuksien kuvaukset:

1. Käyttäjä jättää pakollisen kentän tyhjäksi. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Käyttäjä syöttää virheellisen arvon kenttään. Järjestelmä ilmoittaa virheellisestä arvosta.

Lopputulos: Järjestelmä ilmoittaa onnistuneesta muokkauksesta ja osaston muokatut tiedot näkyvät osaston näkymässä.

7.5 Tapahtuman tiedostot

Tiedoston lisäys

Yleiskuvaus: Järjestelmään lisättyihin tapahtumiin täytyy pystyä kiinnittämään tiedostoja.

Esiehto: Tapahtuma on luotuna järjestelmään.

Tekstikuvaus: Käyttäjä avaa Tapahtumat-välilehdestä haluamansa tapahtuman painamalla tapahtuman nimeä. Käyttäjä siirtyy lisäämään uutta tiedostoa tapahtu-

maan painamalla Uusi tiedosto-painiketta. Käyttäjä syöttää tiedostolle haluamansa tiedot ja painaa Lähetä.

Poikkeuksien kuvaukset:

1. Käyttäjä jättää pakollisen kentän tyhjäksi. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Käyttäjä syöttää virheellisen arvon kenttään. Järjestelmä ilmoittaa virheellisestä arvosta.
3. Käyttäjä syöttää virheellisen tiedoston. Järjestelmä ilmoittaa virheellisestä tiedostosta.

Lopputulos: Järjestelmä ilmoittaa onnistuneesta lisäyksestä, ja tiedosto näkyy tapahtuman tiedostolistauksessa.

Tiedoston poistaminen

Yleiskuvaus: Järjestelmään lisättyjen tapahtumien tiedostoja täytyy pystyä poistamaan.

Esiehto: Tiedosto on lisättynä ja kiinnitettynä jonkin tapahtuman alle.

Tekstikuvaus: Käyttäjä painaa tapahtuman tiedostolistauksessa haluamansa tiedoston kohdalta poistamiseen tarkoitettua painiketta.

Lopputulos: Järjestelmä ilmoittaa onnistuneesta poistamisesta, ja tiedosto poistuu tapahtuman tiedostolistauksesta.

7.6 Osaston tiedostot

Tiedoston lisäys

Yleiskuvaus: Järjestelmään lisättyjen tapahtumien osastoihin täytyy pystyä kiinnittämään tiedostoja.

Esiehto: Osasto on lisättynä ja kiinnitettynä jonkin tapahtuman alle.

Tekstikuvaus: Käyttäjä valitsee tapahtuman osastolistauksessa haluamansa osaston painamalla osaston nimeä. Käyttäjä siirtyy lisäämään uutta tiedostoa osastoon painamalla Uusi tiedosto -painiketta. Käyttäjä syöttää tiedostolle haluamansa tiedot ja painaa Lähetä.

Poikkeuksien kuvaukset:

1. Käyttäjä jättää pakollisen kentän tyhjäksi. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Käyttäjä syöttää virheellisen arvon kenttään. Järjestelmä ilmoittaa virheellisestä arvosta.
3. Käyttäjä syöttää virheellisen tiedoston. Järjestelmä ilmoittaa virheellisestä tiedostosta.

Lopputulos: Järjestelmä ilmoittaa onnistuneesta lisäyksestä, ja tiedosto näkyy osaston tiedostolistauksessa.

Tiedoston poistaminen

Yleiskuvaus: Järjestelmään lisättyjen osastojen tiedostoja täytyy pystyä poistamaan.

Esiehto: Tiedosto on lisättynä ja kiinnitettynä jonkin osaston alle.

Tekstikuvaus: Käyttäjä painaa osaston tiedostolistauksessa haluamansa tiedoston kohdalta poistamiseen tarkoitettua painiketta.

Lopputulos: Järjestelmä ilmoittaa onnistuneesta poistamisesta, ja tiedosto poistuu osaston tiedostolistauksesta**.**

7.7 Pakkauslista

Pakkauslistan muokkaaminen

Yleiskuvaus: Osastojen alle liitettyjen pakkauslistojen deadline- sekä lisätiedot-tietoja täytyy pystyä muokkaamaan.

Esiehto: Osasto on lisättynä ja kiinnitettynä jonkin tapahtuman alle.

Tekstikuvaus: Käyttäjää siirtyy Tapahtumat-välilehdeltä haluamaansa tapahtumaan. Käyttäjää painaa Avaa pakkauslista-painiketta haluamansa osaston kohdalta tapahtuman osastolistauksessa. Käyttäjä siirtyy pakkauslistan tietojen muokkaukseen painamalla Muokkaa pakkauslistaa-linkkiä toiminnot valikosta. Käyttäjä muokkaa haluamiansa kenttiä ja painaa Lähetä-painiketta.

Poikkeuksien kuvaukset:

1. Käyttäjä jättää pakollisen kentän tyhjäksi. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Käyttäjä syöttää virheellisen arvon kenttään. Järjestelmä ilmoittaa virheellisestä arvosta.

Lopputulos: Järjestelmä ilmoittaa onnistuneesta muokkauksesta, ja muokatut tiedot näkyvät pakkauslistan näkymässä.

Tuotteen lisääminen pakkauslistaan

Yleiskuvaus: Osastojen alle liitettyihin pakkauslistoihin täytyy pystyä lisäämään tuotteita.

Esiehto: Pakkauslista on onnistuneesti luotu ja kiinnitetty osaston alle osastoa luotaessa. Järjestelmän varastonhallintaan on lisätty varastoryhmiä sekä tuotteita. Tapahtuma, jonka alle pakkauslistan osasto kuuluu, ei ole merkattu saapuneeksi.

Tekstikuvaus: Käyttäjää siirtyy Tapahtumat-välilehdeltä haluamaansa tapahtumaan. Käyttäjää painaa Avaa pakkauslista-painiketta haluamansa osaston kohdalta tapahtuman osastolistauksessa. Käyttäjä valitsee Lisää tuote -listaan lomakkeesta varastoryhmän Ryhmä-valikosta, jonka alle lisättävä tuote kuuluu. Käyttäjä valitsee Tuote-valikosta lisättävän tuotteen ja syöttää haluamansa tiedot. Käyttäjä painaa Lisää-painiketta.

Poikkeuksien kuvaukset:

1. Käyttäjä jättää pakollisen kentän tyhjäksi. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Käyttäjä syöttää virheellisen arvon kenttään. Järjestelmä ilmoittaa virheellisestä arvosta.
3. Käyttäjän lisäämää tuotetta ei ole tarpeeksi varastossa. Järjestelmä antaa varattavissa olevien tuotteiden listan. Käyttäjä valitsee varattavat tuotteet ja painaa Luo varaus-painiketta.
4. Käyttäjän lisäämää tuotetta ei ole tarpeeksi varastossa eikä varattavia tuotteita ole tarpeeksi. Järjestelmä ilmoittaa ongelmasta ja antaa varattavissa olevien tuotteiden listan. Käyttäjä valitsee varattavat tuotteet ja painaa Luo varaus painiketta.
5. Käyttäjän lisäämää tuotetta ei ole tarpeeksi varastossa eikä varattavia tuotteita ole. Järjestelmä ilmoittaa ongelmasta ja antaa käyttäjän jatkaa tuotteen lisäämistä varastosta löytyvien tuotteiden määrällä. Käyttäjä jatkaa painamalla Jatka-painiketta.

Lopputulos: Lisätty tuote ilmestyy pakkauslistaan.

Pakkauslista tuotteen muokkaaminen

Yleiskuvaus: Osastojen alle liitettyjen pakkauslistojen tuotteita täytyy pystyä muokkaamaan.

Esiehto: Tuote on lisättynä pakkauslistaan eikä tapahtuma, jonka alle pakkauslistan osasto kuuluu, ole merkattu saapuneeksi.

Tekstikuvaus: Käyttäjää siirtyy Tapahtumat-välilehdeltä haluamaansa tapahtumaan. Käyttäjää painaa Avaa pakkauslista-painiketta haluamansa osaston kohdalta tapahtuman osastolistauksessa. Käyttäjä painaa tuotteen muokkaamiseen tarkoitettua painiketta. Käyttäjä muokkaa haluamiansa tietoja. Käyttäjä painaa Muokkaa-painiketta.

Poikkeuksien kuvaukset:

1. Käyttäjä jättää pakollisen kentän tyhjäksi. Järjestelmä ilmoittaa pakollisesta kentästä.
2. Käyttäjä syöttää virheellisen arvon kenttään. Järjestelmä ilmoittaa virheellisestä arvosta.
3. Käyttäjän lisäämää tuotetta ei ole tarpeeksi varastossa. Järjestelmä antaa varattavissa olevien tuotteiden listan. Käyttäjä valitsee varattavat tuotteet ja painaa Luo varaus-painiketta.
4. Käyttäjän lisäämää tuotetta ei ole tarpeeksi varastossa eikä varattavia tuotteita ole tarpeeksi. Järjestelmä ilmoittaa ongelmasta ja antaa varattavissa olevien tuotteiden listan. Käyttäjä valitsee varattavat tuotteet ja painaa Luo varaus-painiketta.
5. Käyttäjän lisäämää tuotetta ei ole tarpeeksi varastossa eikä varattavia tuotteita ole. Järjestelmä ilmoittaa ongelmasta ja antaa käyttäjän jatkaa tuotteen lisäämistä varastosta löytyvien tuotteiden määrällä. Käyttäjä jatkaa painamalla Jatka-painiketta.

Lopputulos: Muokattu tuote päivittyy pakkauslistaan.

Pakkauslista tuotteen poistaminen

Yleiskuvaus: Osastojen alle liitettyjen pakkauslistojen tuotteita täytyy pystyä poistamaan.

Esiehto: Tuote on lisättynä pakkauslistaan ja pakkauslistan tapahtumaa ei ole merkattu saapuneeksi takaisin.

Tekstikuvaus: Käyttäjää siirtyy Tapahtumat-välilehdeltä haluamaansa tapahtumaan. Käyttäjää painaa Avaa pakkauslista-painiketta haluamansa osaston kohdalta tapahtuman osastolistauksessa. Käyttäjä painaa tuotteen poistamiseen tarkoitettua painiketta.

Lopputulos: Poistettu tuote poistuu pakkauslistasta.

Pakkauslistan tuotteen merkitseminen pakatuksi

Yleiskuvaus: Osastojen alle liitettyjen pakkauslistojen tuotteita täytyy pystyä merkkaamaan pakatuksi.

Esiehto: Tuote on lisättynä pakkauslistaan eikä tapahtuma, jonka alle pakkauslistan osasto kuuluu, ole merkattu saapuneeksi.

Tekstikuvaus: Käyttäjää siirtyy Pakkaaminen-välilehdelle ja painaa Pakkaa-painiketta haluamansa pakkauslistan kohdalla. Käyttäjä painaa pakkaamismerkkaukseen tarkoitettu valintaruutua.

Lopputulos: Valintaruutuun ilmestyy ok-merkintä.

Pakkauslistan tuotteen pakkausmerkinnän poistaminen

Yleiskuvaus: Pakkauslistojen tuotteiden Pakattu-merkintä täytyy pystyä poistamaan.

Esiehto: Tuote on lisättynä pakkauslistaan eikä tapahtuma, jonka alle pakkauslistan osasto kuuluu, ole merkattu saapuneeksi. Tuote on merkitty pakatuksi.

Tekstikuvaus: Käyttäjää siirtyy Pakkaaminen-välilehdelle ja painaa Pakkaa-painiketta haluamansa pakkauslistan kohdalla. Käyttäjä painaa pakkaamismerkkaukseen tarkoitettu valintaruutua ja hyväksyy järjestelmän varmistuksen.

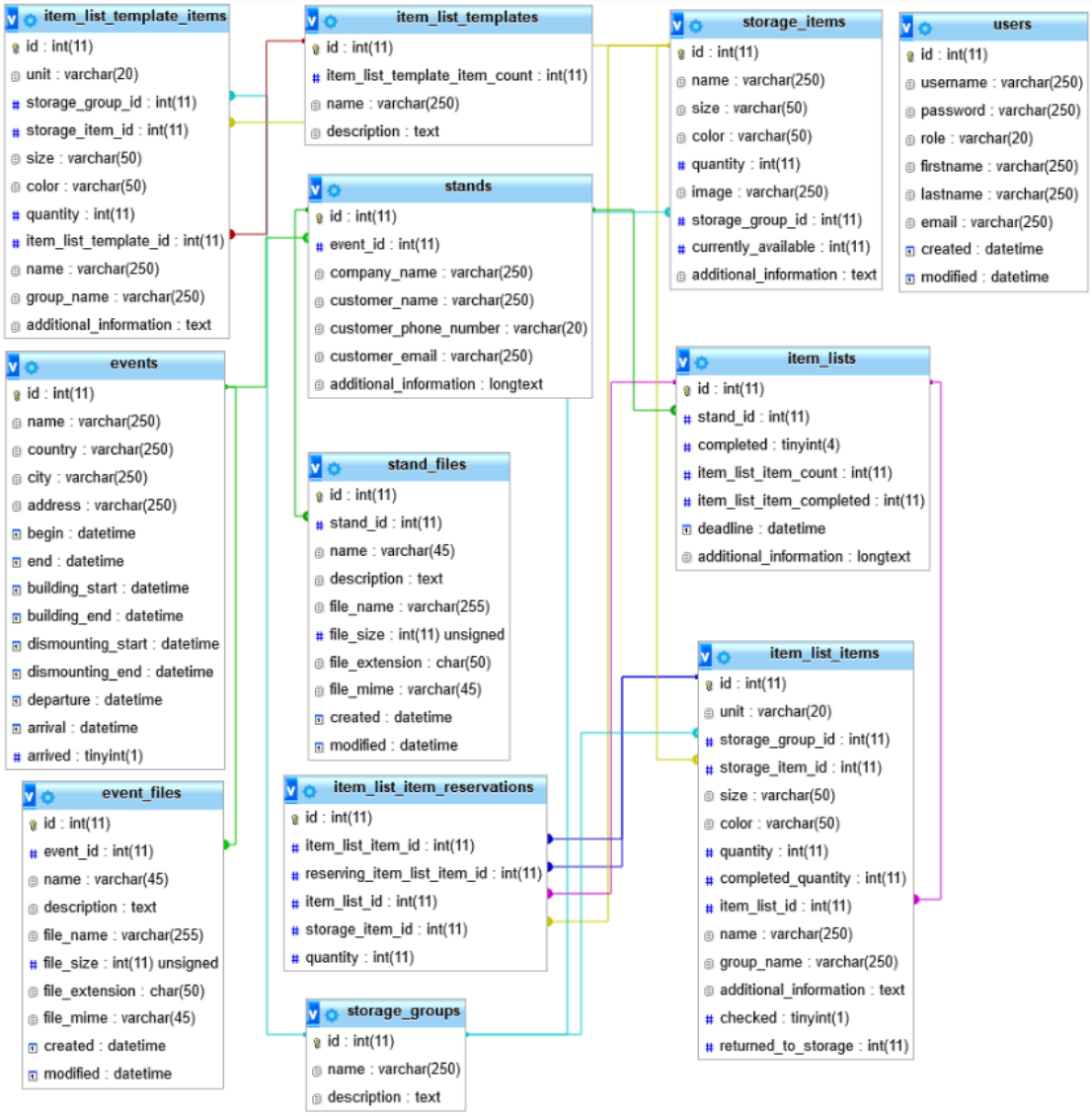
Lopputulos: Valintaruudun ok-merkintä poistuu.

8 TIETOKANTA

Opinnäytetyön tietokantana toimii MySQL-tietokannan hallintajärjestelmä ja tietokannan käsittelijänä on käytössä ollut phpMyAdmin. Tietokantamoottorina on käytössä InnoDB, joka tukee taulujenvälisiä relaatioita.

Tietokannan suunnittelussa on otettu huomioon CakePHP:n nimeämiskäytännöt. Nimeämiskäytännöistä olen luonut yhteenvedon kappaleessa 4.16. Tauluihin on luotu mallien assosiaatioita vastaavat relaatiot. Relaatioita ei kuitenkaan ole välttämätöntä luoda tietokantaan, sillä CakePHP:n mallienväliset assosiaatiot hoitavat tätä tehtävää mallikkaasti. Mallien välisistä assosiaatioista olen kertonut tarkemmin kappaleessa 4.8. Relaatiot onkin luotu viite-eheyden säilyttämiseksi sovellustason virheiden aiheuttaman haitan minimoimiseksi.

Kuviossa 6 on esiteltynä järjestelmän tietokantarakenne.



Kuvio 6. Tietokannan rakenne.

9 KÄYTETTÄVYYS JA KÄYTTÖLIITTYMÄ

Järjestelmä on toteutettu yleisimmillä verkkotekniikoilla, joten luonnollisesti sen käyttöliittymänä toimii verkkopohjainen käyttöliittymä, joka on toteutettu HTML:llä ja CSS:llä. Olen pyrkinyt toteuttamaan mahdollisimman helppokäyttöisen sekä yksinkertaisen käyttöliittymän, jonka jokainen toiminto vastaa mahdollisimman pitkälle siltä odotettua toimintoa. Olen pyrkinyt käyttöliittymän yksinkertaisella sekä selkeällä toteuttamisella luomaan mahdollisimman hyvän toteutuksen käytettävyyden kannalta.

9.1 Käytettävyys

Käytettävyydellä on kaksi puolta, joista toinen puoli on helppokäyttöisyys ja toinen puoli tyylikkyys ja selkeys. Helppokäyttöisyys vastaa fyysisestä puolesta eli siitä, että jokin toiminto tekee juuri sen, mitä siltä odotetaan. Tyylikkyys ja selkeys vastaavat psykologisista ominaisuuksista eli siitä, että jokin toiminto tekee, mitä sen odotetaan tekevän. (Reiss 2012, 18.)

Työssäni olen pyrkinyt esittämään kaikki toiminnot juuri Reissin kuvailemilla tavoilla. Painikkeet ja linkit suorittavat sen, mitä niiden sijainnin ja tekstin perusteella on oletettavaa suorittaa.

9.2 Käyttöliittymän toteuttaminen

Käyttöliittymän toteuttamisessa on otettu huomioon varsinkin yksinkertaisuus. Järjestelmä on tarkoitettu ainoastaan yrityksen sisäiseen käyttöön eikä sen näin ollen tarvitse olla näyttävä, vaan toimiva ja yhdenmukainen. Toteutuksen avainasemassa on kappaleessa 2.4 esitelty Bootstrap. Toteutuksessa on otettu myös huomioon eri toimintojen, kuten linkkien ja painikkeiden, erottuminen muusta sisällöstä. Myös mahdolliset virheilmoitukset on luotu muusta sisällöstä erottuvaksi.

Sivuston navigointi on eroteltu viiteen tärkeimpään osaan, joiden alta löytyy navigointilinkkiä vastaava sisältö. Nämä osat ovat Etusivu, Varasto, Tapahtumat, Käyttäjät sekä Pakkaaminen.

Etusivu

Etusivun alle ei opinnäytetyönä toteutetussa järjestelmässä ole vielä lisätty mitään, vaan se toimii pikemminkin aloitus sivuna. Kappaleessa 11.2 kerron miten etusivua voisi jatkossa kehittää.

Varasto

Varaston alle luodaan varastoryhmät sekä niihin kuuluvat tuotteet.

Tapahtumat

Tapahtumien alle luodaan tapahtumat sekä niihin kuuluvat osastot ja pakkauslistat. Tämä on todennäköisimmin järjestelmän tärkein osa pakkaamisen rinnalla sen jälkeen, kun varastopuolen tuotteet ovat lisättyinä järjestelmään.

Käyttäjät

Käyttäjät-osa on ainoastaan hallintatason käyttäjien käytössä. Siinä lisätään, muokataan sekä poistetaan käyttäjiä.

Pakkaaminen

Pakkaaminen-osassa suoritetaan kaikki pakkauslistojen pakkaaminen. Pakkaaminen on luultavasti järjestelmän eniten käytetty osa.

10 JÄRJESTELMÄN TEKNINEN TOTEUTTAMINEN CAKEPHP-SOVELLUSKEHYKSELLÄ

Aloitan sovelluskehityksen tarkastelun esittelemällä pohjien leipomisen *Bake*-konsolisovelluksella. *Bake*-konsolisovelluksesta olen kertonut tarkemmin kappaleessa 4.19. Pyrin työssäni esittelemään sovelluskehityksen vaiheet CakePHP:n luonnollisen kehityskaaren mukaisesti. CakePHP:ssa luonnollinen ensimmäinen vaihe on leipominen, jonka jälkeen siirrytään luomaan sovelluskohtaisia ominaisuuksia.

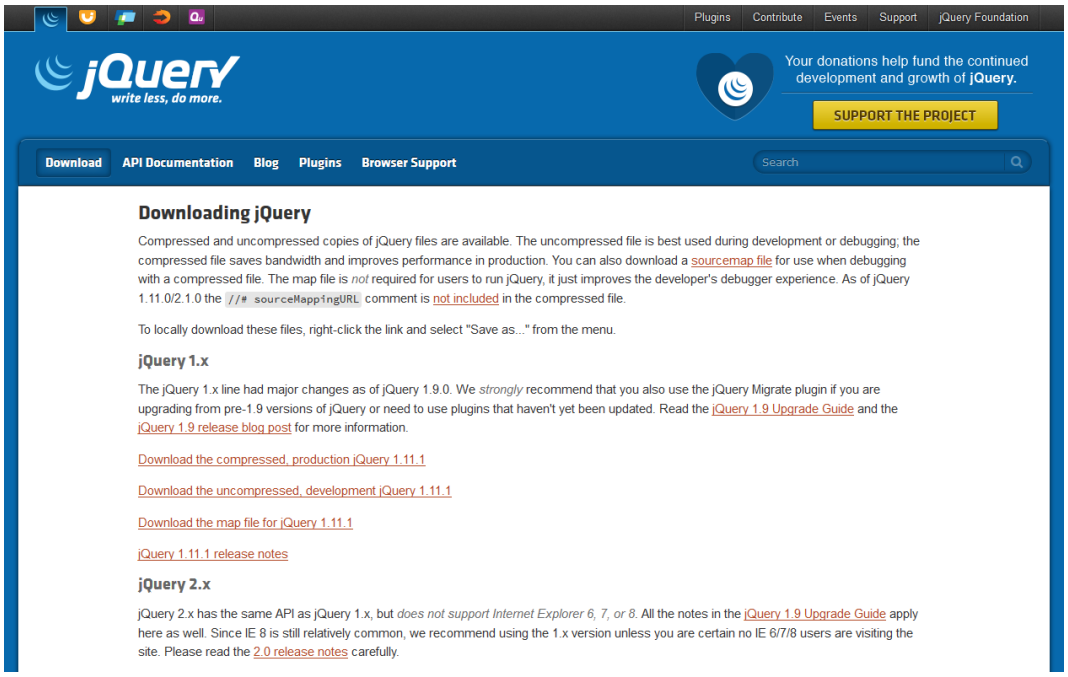
10.1 Esivalmistelu ja vaaditut ominaisuudet

Onnistuneen asennuksen jälkeen on aika liittää käyttöön tarvitut kirjastot, kuten jQuery, jQuery UI ja Bootstrap (Ks. Kuvio 5). Tarkoituksena on käyttää Bootstrappiä ulkoasun luomiseen, joten välttyäkseni oletuspohjien (niiden luomisesta lisää kohdassa 10.2) täydelliseltä muokkaamiselta, otan käyttöön myös BootsrapCake Shell Templaten. BootstrapCaken avulla Bootstrapin käyttöönotto tapahtuu helposti.

Ohjainten, mallien sekä näkymien nopeaa luomista varten CakePHP:n konsolisovelluksen avulla on käytössä oltava SSH-yhteys sekä SSH-asiakasohjelma. PuTTY sopii tähän tarkoitukseen Windows-ympäristössä.

10.1.1 JQueryn lataaminen

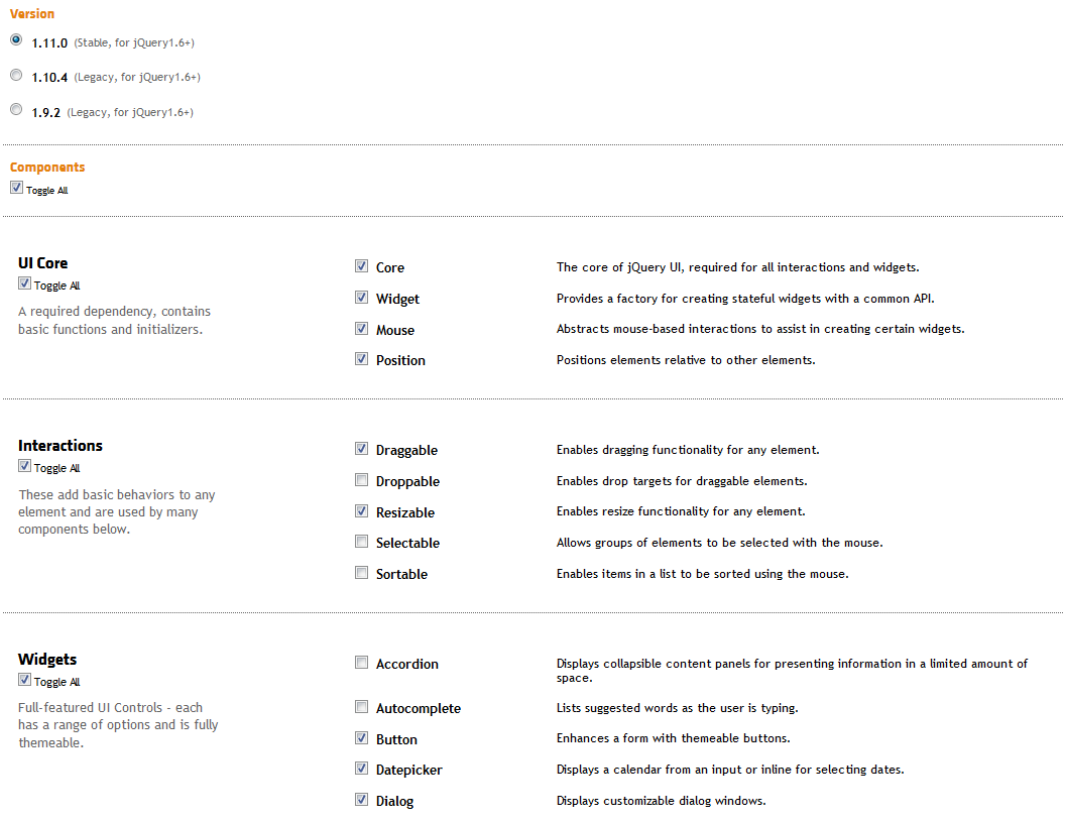
JQueryn lataaminen on helppoa. Se tapahtuu osoitteesta jquery.com/download, josta ladataan uusin 1.x-versio. Opinnäytetyötä tehdessäni uusin versio on 1.11.1. Se löytyy linkin ”Download the compressed, production jQuery 1.11.1” alta. Kuviossa 7 esitetään jQueryn lataaminen. Ladattu JS-tiedosto siirretään CakePHP:n alle kansioon *app/webroot/js*. JQuery tarjoaa myös 2.x-versiota, mutta on suositeltavaa käyttää 1.x-versiota. 2.x-versio ei tue Internet Explorerin versioita 6, 7 ja 8.



Kuvio 7. JQuery:n lataaminen.

10.1.2 JQuery UI:n lataaminen

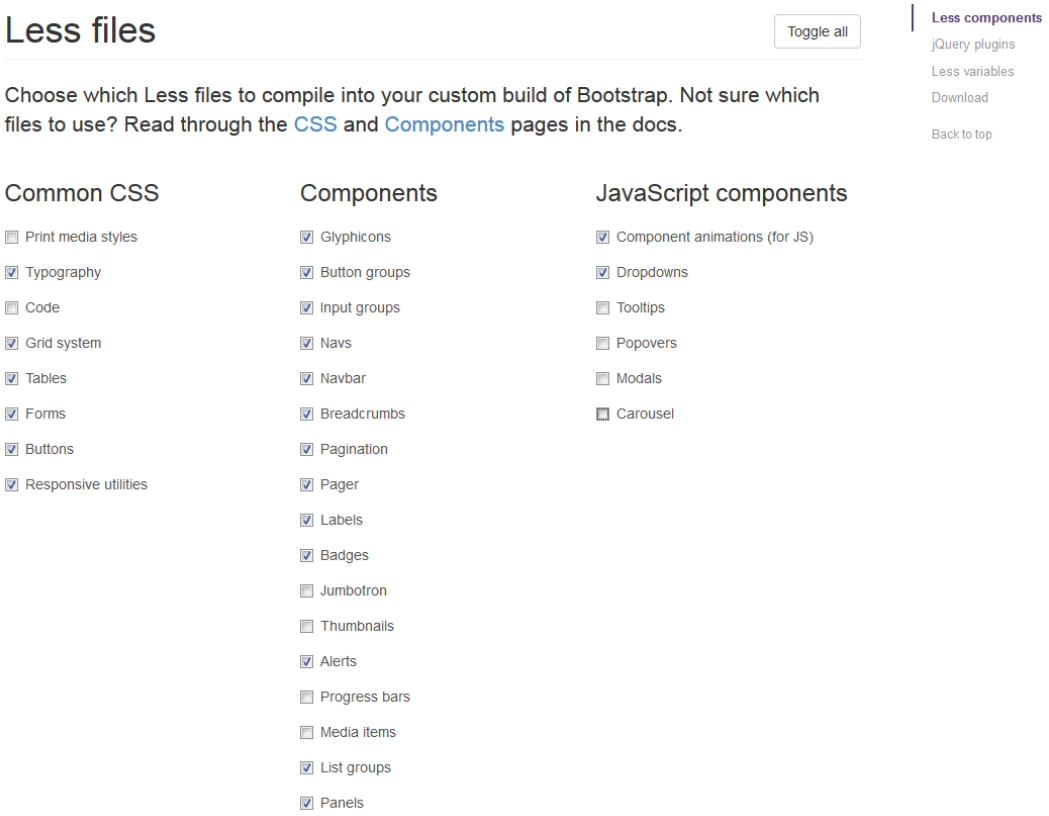
JQuery UI:n lataaminen tapahtuu osoitteesta jqueryui.com/download. JQuery UI tarjoaa ”Download builderin” eli lataustiedoston rakentajan, jonka avulla voi itse valita tarvitut osat ja näin pienentää JS-tiedoston kokoa poistamalla tarpeettomat osat. Kuvio 8 esittelee jQuery UI:n latausrakentajaa. Rakentajalla ladatun version pienennetty JS-tiedosto, *jquery-ui.min.js*, siirretään CakePHP:n kansioon *app/webroot/js* ja pienennetty CSS-tiedosto, *jquery-ui.min.css*, kansioon *app/webroot/css*.



Kuvio 8. JQuery UI:n lataaminen.

10.1.3 Bootstrapin lataaminen

Myös Bootstrap tarjoaa mahdollisuuden valita tarvittavat osat lataustiedostoon, ja sen pääsee tekemään osoitteessa http://getbootstrap.com/customize/. Kuvio 9 esittelee Bootstrap-tiedostojen rakentajaa. Ladattu *bootstrap.min.js* tiedosto siirretään jälleen kansioon *app/webroot/js* ja ladattu *bootstrap.min.css* kansioon *app/webroot/css*. Bootstrapin mukana tulee myös *fonts*-kansio, joka siirretään kansioon *app/webroot*.

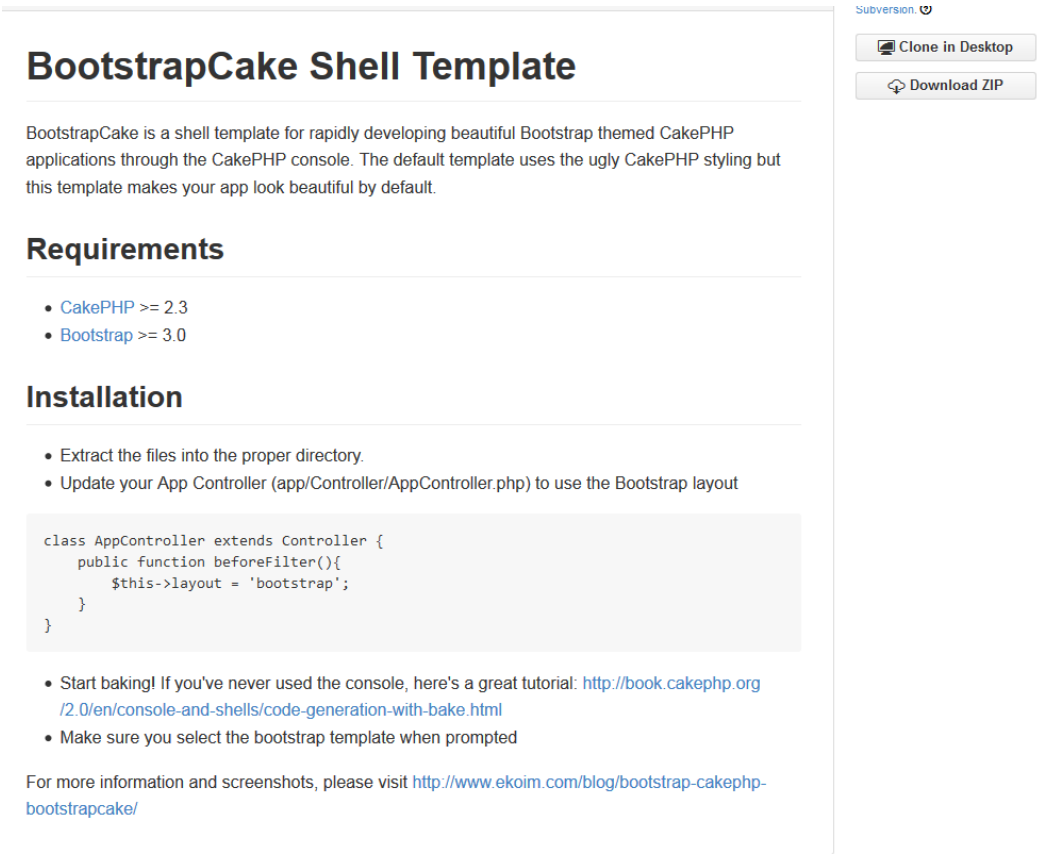


Kuvio 9. Bootstrapin lataaminen latausrakentajalla.

10.1.4 BootstrapCake:n lataaminen ja käyttöönotto

BootstrapCake löytyy osoitteesta https://github.com/EKOInternetMarketing /BootstrapCake, ladattava ZIP-versio löytyy linkin ”Download ZIP” alta, kuten Kuvio 10 on nähtävillä. Kun ZIP-tiedosto on purettu, löytyy sieltä CTP-pohjat näkymille, ohjain toiminnoille, ulkoasulle sekä navigoinnille. Ladatut tiedostot siirretään niitä vastaaviin kansioihin CakePHP:ssa. Tämän jälkeen tulee CakePHP ohjata käyttämään juuri lisättyä ulkoasua oletuksena. Tämä tapahtuu lisäämällä seuraava koodi tiedostoon *app/Controller/AppController.php*:

```
public function beforeFilter()
{
  $this->layout = 'bootstrap';
}
```



Kuvio 10. BootstrapCake:n lataaminen.

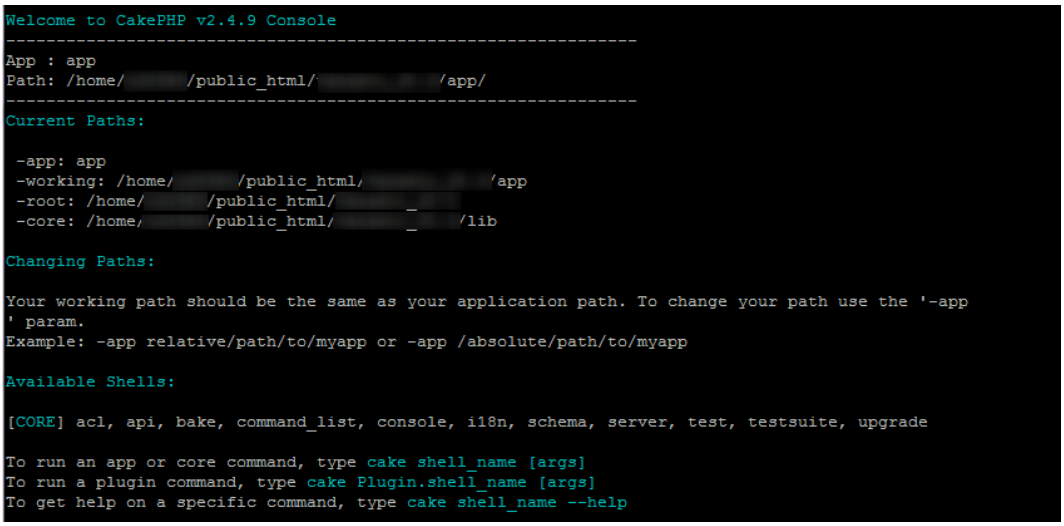
10.1.5 PuTTY:n lataaminen

PuTTY on SSH-asiakasohjelma Windows-ympäristölle ja sen saa ladattua osoitteesta http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html. PuTTY ei vaadi erillistä asennusta.

10.2 Pohjien luominen leipomalla

Kun kaikki tarvittavat kirjastot ovat paikoillaan sekä tietokanta on luotuna, on aika luoda mallit, näkymät ja ohjaimet. Se tapahtuu nopeasti käyttämällä CakePHP:n tarjoamaa Bake-konsolisovellusta. Kuten aikaisemmin mainittu, on käytettävissä oltava SSH-yhteys sekä SSH-asiakasohjelma. Opinnäytetyötä varten on käytössä PuTTY.

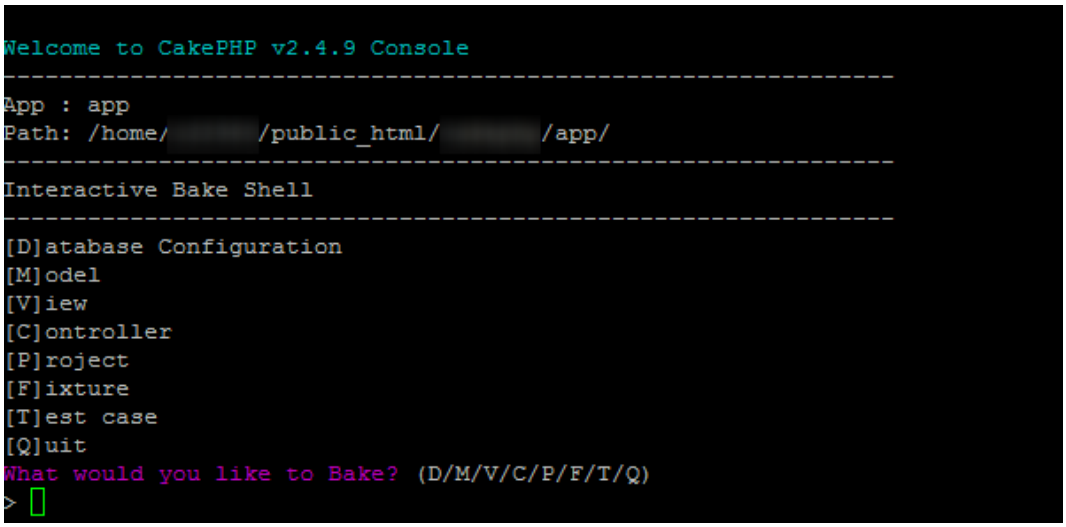
Kun PuTTY on yhdistetty palvelimeen, tulee se ohjata CakePHP:n kansioon */app* esimerkiksi komennolla *cd public_html/varasto/app*. Tämän jälkeen on hyvä varmistaa, että CakePHP:n ”cake” on suoritettavissa. Cake-konsolisovellus löytyy kansiosta *Console*. Testataan toimivuus komennolla *Console/cake.* Mikäli sovellus on käytössä ja toimii oikein, tulisi ruudulla näkyä kuviota 11 vastaava näkymä.



Kuvio 11. Konsolisovelluksen oletusilmoitus.

Mikäli suorittaminen epäonnistuu ”Permission denied”-virheen vuoksi, on tiedoston oikeudet mahdollisesti väärin. Oikeudet onnistuu muuttamaan helposti navigoimalla kansioon Console, komennolla *cd Console* ja suorittamalla *chmod 755 cake*, minkä jälkeen tiedoston pitäisi olla suoritettavissa.

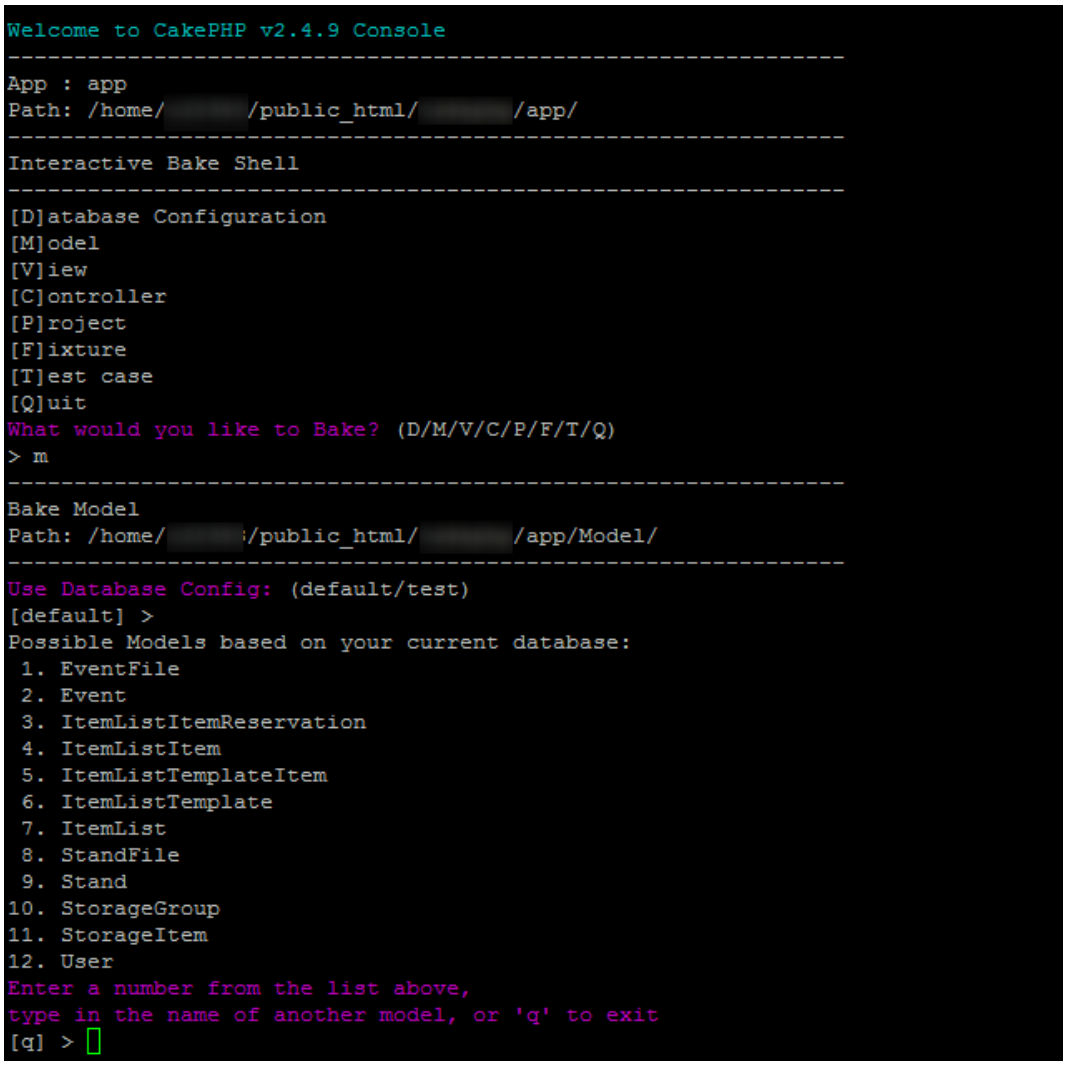
Kun konsolisovellus toimii odotetusti, on kaikki valmista leipomista varten. Tässä vaiheessa tulisi PuTTY:n olla kansiossa *app/*. Suoritetaan komento *Console/cake bake*. Mikäli kaikki toimii odotetusti, tulisi näkymän olla kuvion 12 mukainen. Seuraavissa kappaleissa esittelen mallin, näkymän sekä ohjaimen leipomisen tietokantataululle *events*, On kuitenkin luonnollista heti projektin alussa leipoa tarvittavat koodit kaikille tietokannan tauluille, jotka sitä vaativat.



Kuvio 12. Bake-oletusnäkymä.

10.2.1 Mallin leipominen

Leipominen tulee aloittaa malleista, joten valitaan kuvion 13 vastaavan vaiheen kohdalla M (Model). Sovellus kysyy, mitä tietokanta-asetusta käytetään. Vastaukseksi valitaan default (oletus) ja jatketaan painamalla *Enter*. Sovellus antaa listan kyseisen tietokannan alta löytyvistä tauluista, ja lista vastaa aikaisemmin luomaani tietokanta-rakennetta.



Kuvio 13. Mallin leipominen.

Sitten aloitetaan luomalla tapahtumat, eli valitaan 2. Sovellus kysyy, halutaanko taulun sarakkeille luoda validointisääntöjä. Valitaan Y (kyllä, oletus). Sovellus aloittaa taulun ensimmäisestä sarakkeesta, eli *id*:stä ja tarjoaa listan mahdollisista säännöistä (Kuvio 14).

```
Would you like to supply validation criteria
for the fields in your model? (y/n)
[y] >

Field: id
Type: integer
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
 1. alphaNumeric               18. maxLength
 2. between                    19. mimeType
 3. blank                      20. minLength
 4. boolean                    21. money
 5. cc                         22. multiple
 6. comparison                 23. naturalNumber
 7. custom                     24. notEmpty
 8. date                       25. numeric
 9. datetime                   26. phone
10. decimal                    27. postal
11. email                      28. range
12. equalTo                    29. ssn
13. extension                  30. time
14. fileSize                   31. uploadError
15. inList                     32. url
16. ip                         33. userDefined
17. luhn                       34. uuid

35 - Do not do any validation on this field.
---------------------------------------------------------------
... or enter in a valid regex validation string.

[35] >
```

Kuvio 14. Mallin validointisääntölista.

Jokaiselle sarakkeelle valitaan tarvittavat validointisäännöt. Näitä sääntöjä voi olla useampi kuin yksi saraketta kohden. Jätetään validointisäännöt tässä vaiheessa luomatta, sillä ne voi myöhemmin luoda suoraan tiedostoon.

Kun kaikki taulun sarakkeet on menty läpi, kysyy sovellus, halutaanko mallille luoda assosiaatioita. Mikäli taulut on luotu oikein ja kyseiseen tauluun liittyy muita tauluja, tulisi sovelluksen ehdottaa assosiaatioita kuvion 15 mukaisesti.

```
Would you like to define model associations
(hasMany, hasOne, belongsTo, etc.)? (y/n)
[y] > y
One moment while the associations are detected.
---------------------------------------------------------------
Please confirm the following associations:
---------------------------------------------------------------
Event hasMany EventFile? (y/n)
[y] >
```

Kuvio 15. HasMany-assosiaation vahvistaminen.

hasMany-assosiaatio tarkoittaa, että kyseisen mallin riviin voi liittyä monta assosiaatiomallin riviä. Kun kaikki assosiaatiot on valittu, varmistaa sovellus, että

kaikki on kunnossa ennen kuin malli luodaan (Kuvio 16). Assosiaatioita olen esitellyt kappaleessa 4.8.

```
Would you like to define model associations
(hasMany, hasOne, belongsTo, etc.)? (y/n)
[y] > y
One moment while the associations are detected.
---------------------------------------------------------------
Please confirm the following associations:
---------------------------------------------------------------
Event hasMany EventFile? (y/n)
[y] > y
Event hasMany Stand? (y/n)
[y] > y
Would you like to define some additional model associations? (y/n)
[n] > n

---------------------------------------------------------------
The following Model will be created:
---------------------------------------------------------------
Name:       Event
DB Table:   `[redacted]`.`events`
Associations:
        Event hasMany EventFile
        Event hasMany Stand
---------------------------------------------------------------
Look okay? (y/n)
[y] > y

Baking model class for Event...

Creating file /home/[redacted]/public_html/[redacted]/app/Model/Event.php
Wrote `/home/[redacted]/public_html/[redacted]/app/Model/Event.php`
PHPUnit is not installed. Do you want to bake unit test files anyway? (y/n)
[y] > n
```

Kuvio 16. Mallin luomisen hyväksyminen.

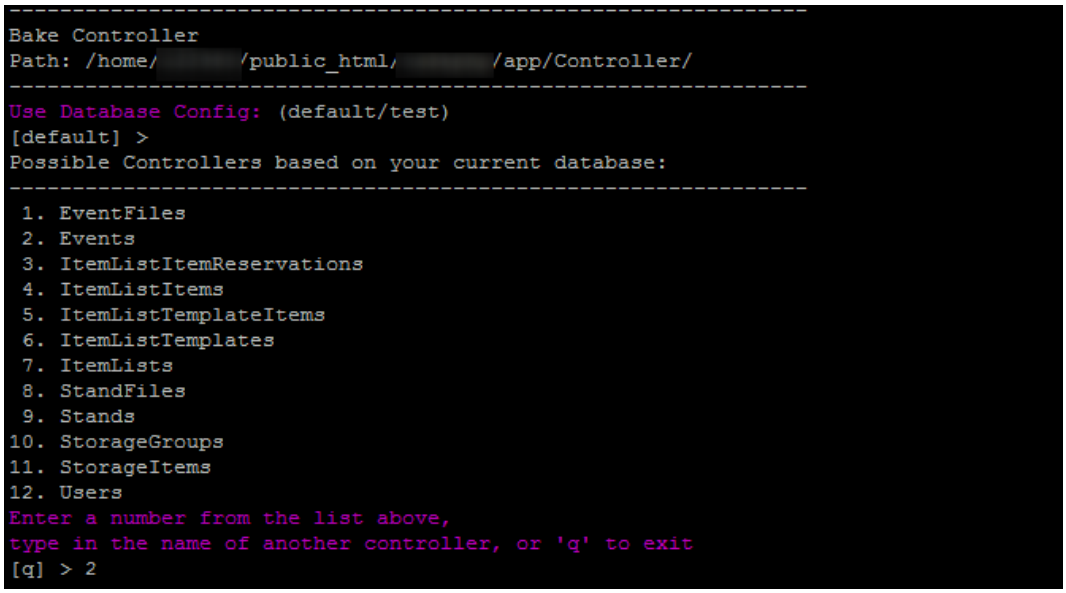
Suosittelen tässä vaiheessa toistamaan saman prosessin jokaiselle tietokannan taululle, jolloin kaikki taulut ovat valmiita ohjaimia varten.

10.2.2 Ohjainten leipominen

Kun kaikki mallit tai kyseisen taulun malli on leivottu, CakePHP-projektien luonnollinen vaihe on luoda sille ohjain. Tässä kappaleessa luodaan kappaleessa 10.2.1 luodulle mallille ohjain. Esittelen ohjaimen leipomisen kohta kohdalta. Jokaisesta kohdasta löytyy ristiviite kyseistä kohtaa selventävään kappaleeseen, mikäli se mahdollisesti sitä vaatii.

1. Suoritetaan jälleen komento *Console/cake bake* ja sovelluksen tulisi olla kuvio 12:a vastaavassa vaiheessa.

2. Valitaan C (Controller) ja jatketaan painamalla *Enter*.
3. Sovellus pyytää jälleen valitsemaan tietokanta-asetuksen, jota käytetään ohjaimen luomiseen. Valitaan default.
4. Sovellus ehdottaa luodun tietokantarakenteen pohjalta mahdollisia ohjaimia. Valitaan *Event*-mallille vastaava ohjain, eli valitaan 2 (Kuvio 17).

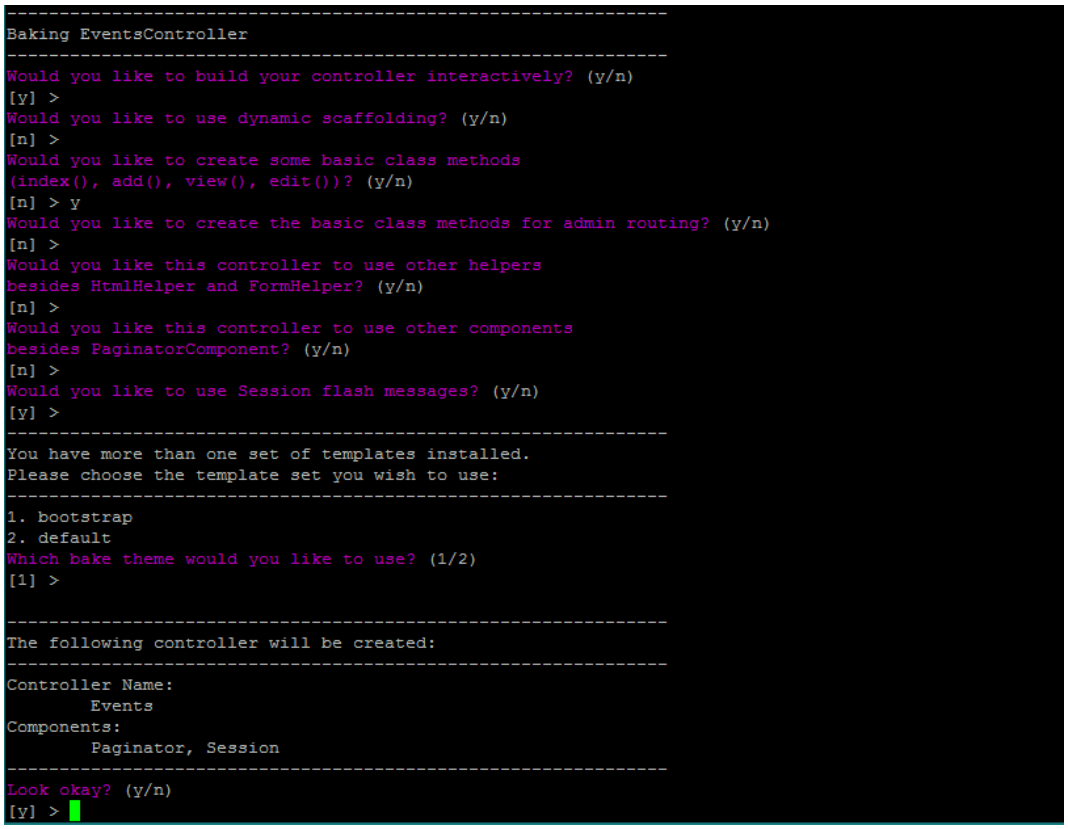


Kuvio 17. Tietokanta-asetuksen ja ohjaimen valitseminen.

5. Sovellus kysyy, mikäli ohjain halutaan luoda interaktiivisesti. Valitaan Y (kyllä).
6. Seuraavaksi sovellus kysyy, halutaanko käyttää dynaamista rakennetta. Dynaamisesta rakenteesta olen kertonut kappaleessa 4.13. Valitaan N (Ei).
7. Seuraavaksi sovellus kysyy, halutaanko luoda perusmetodit ohjaimelle. Ohjaimien metodeista olen kertonut kappaleessa 4.15. Sovellus tarjoaa metodit indeksille, lisäämiselle, näyttämiselle sekä muokkaamiselle. Valitsemalla ”kyllä” voidaan omaa toimintaa nopeuttaa, joten valitaan Y.
8. Mikäli sovelluksessa halutaan käyttää omia näkymiä hallintatason käyttäjille, valitaan seuraavassa Y.
9. Seuraavaksi valitaan ohjaimeen liittyvien näkymien tarvitsemat avustajat, joista olen kertonut tarkemmin kappaleessa 4.14. Sovellus kysyy, mikäli HtmlHelper- ja FormHelper-avustajien lisäksi halutaan käyttää muita

avustajia. Avustajia on mahdollista tarpeen tullen lisätä myöhemmin, joten valitaan N.

10. Seuraavaksi valitaan ohjaimen komponentit, joista olen kertonut kappaleessa 4.12. Oletuksena tarjolla on Paginator. Se on tässä vaiheessa riittävä ja myös komponentteja voi myöhemmin lisätä tarpeen tullen, joten valitaan N.
11. Sovellus kysyy, mikäli halutaan käyttää session-viestejä. Session-viesteistä olen kertonut tarkemmin kappaleessa 4.12.2, valitaan Y.
12. Seuraavaksi sovellus kysyy, mitä teemaa käytetään. Valitaan kappaleessa 10.1.4 lisätty teema eli valitaan 1. bootstrap.
13. Sovellus vahvistaa valitun ohjaimen ja siihen liitetyt komponentit (Kuvio 18).
14. Käytössä ei ole PHPUnitia. Valitaan N, kun sovellus kysyy, mikäli testitiedostot halutaan luoda siitä huolimatta.

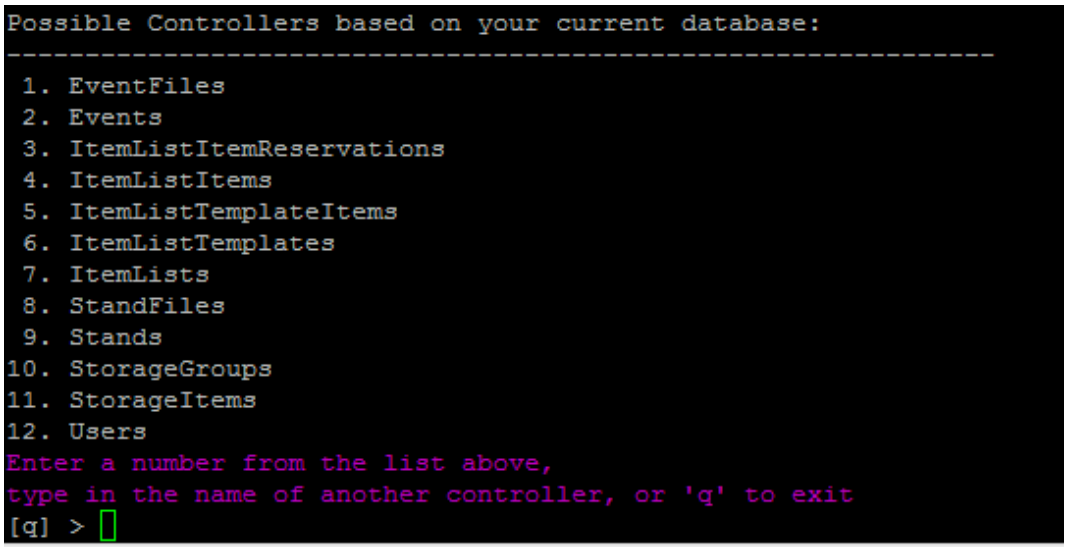


Kuvio 18. Ohjaimen valittujen asetusten vahvistaminen.

10.2.3 Näkymien leipominen

Ohjaintien luontien jälkeen, on luonnollista luoda näkymät. Tässä kappaleessa esittelen näkymien leipomisen kappaleessa 10.2.2 leivotulle ohjaimelle.

1. Ensiksi suoritetaan jälleen komento *Console/cake bake*, ja nyt sovelluksen tulisi jälleen olla bake sovelluksen oletusnäkymässä (Kuvio 12).
2. Sitten valitaan V (View) ja jatketaan painamalla *Enter*.
3. Sovellus pyytää jälleen valitsemaan tietokanta-asetuksen, jota käytetään näkymien luomiseen. Valitaan default.
4. Sovellus ehdottaa ohjainta luotujen tietokanta-taulujen perusteella (Kuvio 19). Luodaan näkymä ohjaimelle *events*, eli valitaan 2.



Kuvio 19. Ohjaimen valitseminen.

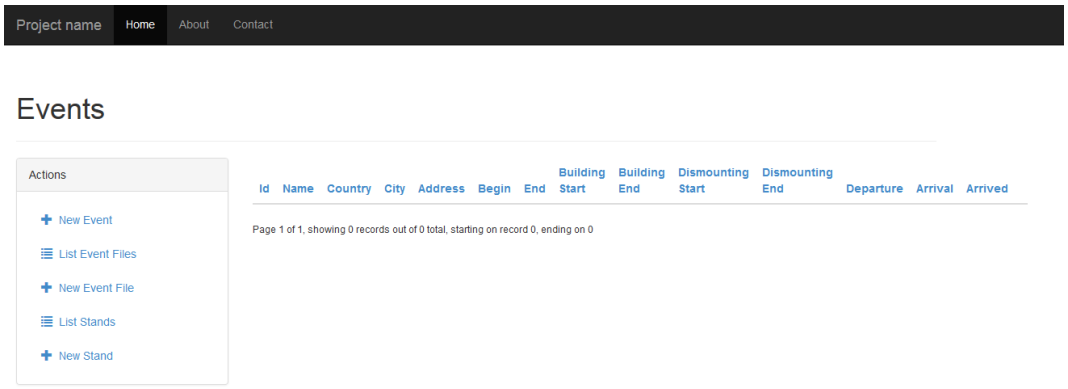
5. Sovellus kysyy, halutaanko näkymät luoda interaktiivisesti. Valitaan Y.
6. Seuraavaksi sovellus kysyy, halutaanko luoda CRUD-näkymät, valitaan Y.
7. Sovellus kysyy, halutaanko hallintakäyttäjälle luoda näkymät, valitaan N.
8. Jälleen valitaan kappaleessa 10.1.4 lisätty teema. Valitaan 1. bootstrap.

10.3 Leivottujen pohjien sekä toimintojen tarkastelu

Kun mallit, näkymät sekä ohjaimet on leivottu, on luonnollista tarkastella, miltä sivusto tällä hetkellä näyttää. Varmistetaan läpikäymäni *Events*- mallin, näkymän sekä ohjaimen toimintojen toimivuus.

10.3.1 Index-sivu

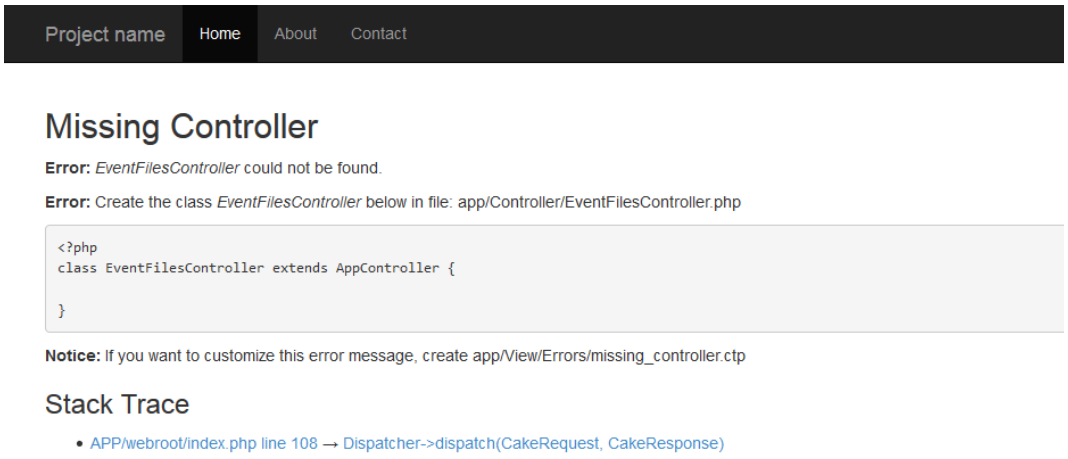
Ohjain on luotu nimellä *EventsController* ja CakePHP:n käytäntöjen mukaan se tulisi löytyä verkko-osoitteesta *http://domain.com/cakephp/events*.



Kuvio 20. Tapahtumien indeksisivu leipomisen jälkeen.

Kuvio 20:tä tarkasteltaessa on huomattavissa Bootstrapille ominainen ulkoasu sekä tietokantatauluun *events* liittyvien sarakkeiden listaus. CakePHP:n käytäntöjen mukaan verkko-osoite */events* viittaa *EventsControllerin* index-metodiin, joka puolestaan oletuksena viittaa tiedostoon *View/Event/index.ctp*. Kyseessä on siis oletuksena kaikkien *events*-tauluun liittyvien rivien listaussivu. Näkymää leivottaessa *Bake*-konsolisovelluksella on *events*-näkymälle myös luotu toimintoja assosiaatioiden perusteella, jotka vahvistettiin mallia luodessa.

Mikäli tässä vaiheessa ei vielä ole luotuna vastaavia malleja, näkymiä sekä ohjaimia *actions*-listauksesta löytyville toiminnoille, kuten *New Event File* tai *List Stands*, antaa CakePHP linkkiä seurattaessa virhettä vastaavan virheilmoituksen (Kuvio 21).

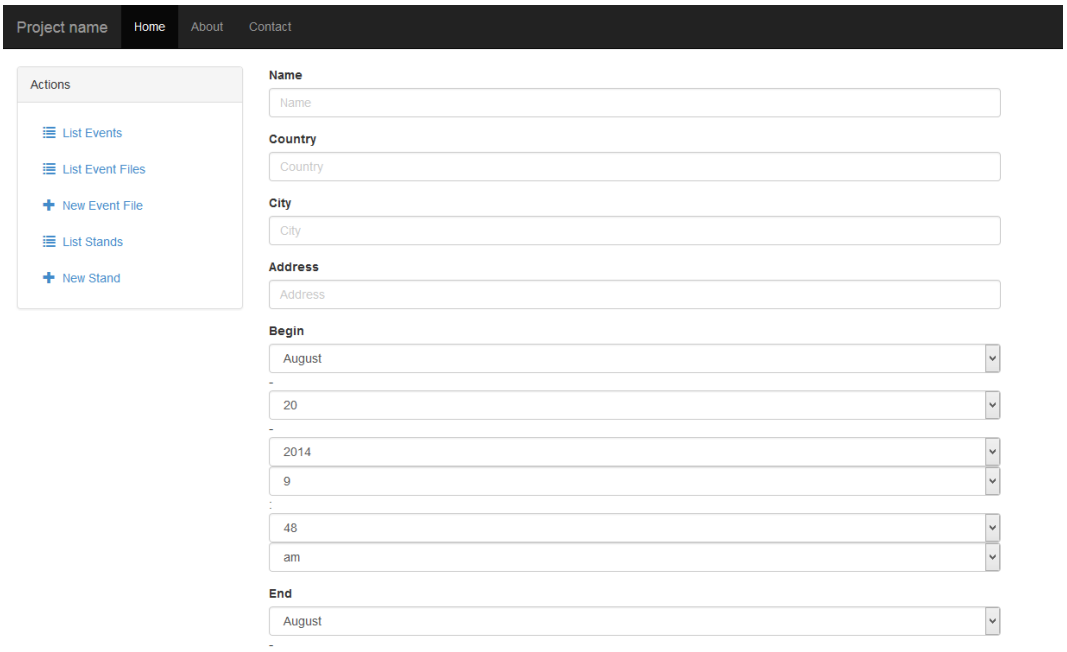


Kuvio 21. Puuttuva ohjain.

Ilmoitus on itseselitteinen ja se pyytää luomaan ohjaimen, joka vastaa pyydettyä verkko-osoitetta. Tämä tapahtuu saman kaavan mukaan kuin aikaisemmin esittelemäni *events* mallin, näkymän sekä ohjaimen leipominen.

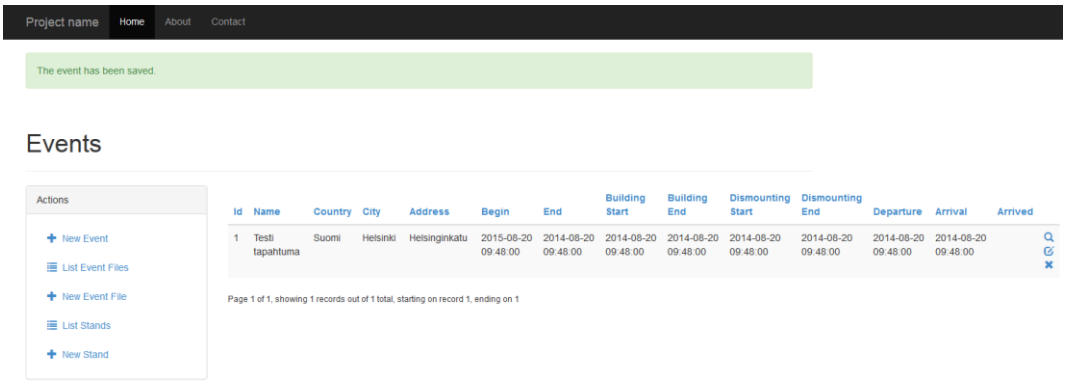
10.3.2 Add-sivu

Seurattaessa action-valikon linkkiä *New Event* (Kuvio 20) aukeaa näkyville toimiva tapahtuman lisäyssivu (Kuvio 22), jossa on lisäyskenttä jokaiselle *events*-taulusta löytyvälle sarakkeelle. Lisäyskenttiä luodessa on CakePHP ottanut myös huomioon sarakkeen tyypin. *Varchar*-tyypeille on luotu tekstityyppinen input-elementti, *datetime*-tyypeille on luotu oma select-valikko kuukaudelle, päivälle, vuodelle sekä tunnille, *tinyint(1)* tyypille on lisätty checkbox.



Kuvio 22. Tapahtuman lisäyssivu.

Kuten kuviosta 22 on huomattavissa, varsinkaan päivämääräkentät eivät ole kovin elegantisti muodostettuja, mutta ne ovat enemmän kuin hyviä testitiedon luomiseen. Testitiedon luominen onkin seuraava askel projektin etenemisessä ja kuviossa 23 on nähtävissä *events/index*-sivu onnistuneen tapahtuman lisäyksen jälkeen.

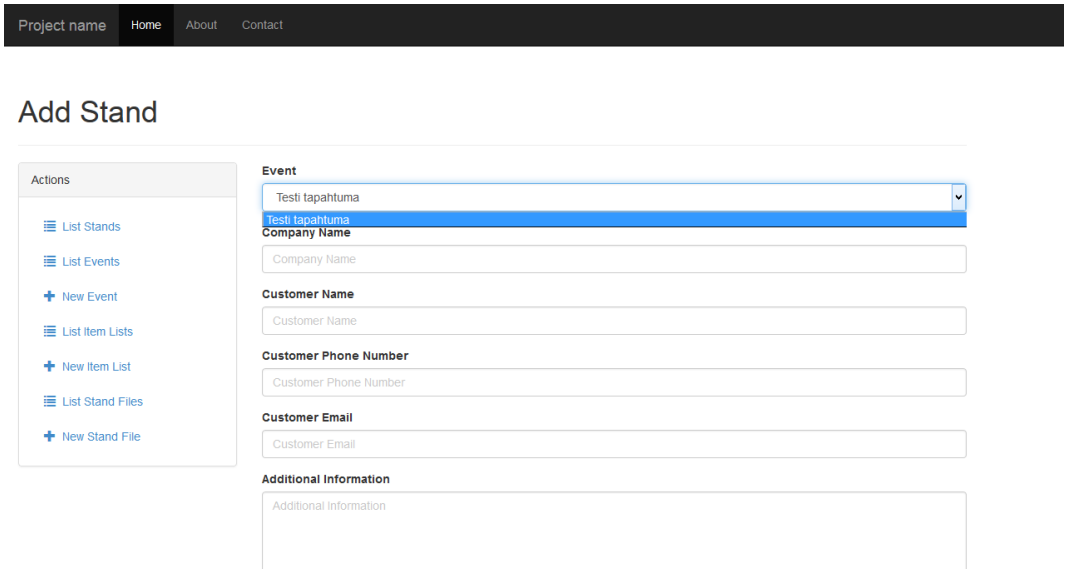


Kuvio 23. Onnistunut tapahtuman luominen.

10.3.3 Tapahtuman assosiaatiot

Kappaleessa 10.2.1 luotiin *Events*-mallille *hasMany*-assosiaatiot *Stand*- sekä *EventFile*-malleihin. Kappaleessa 4.8 olen kertonut enemmän mallien assosiaatioista. Kun ensimmäinen testitapahtumakin on jo luotu ja malli, näkymät sekä oh-

jain luotuina tietokannan *stands*-taululle, on hyvä varmistaa assosiaatioiden toiminen. *Actions*-listauksen *New Stand*-linkistä pääsee lisäämään uuden osaston. Assosiaatioiden ansiosta on CakePHP luonut osaston lisäyssivulle alasvetovalikon, joka listaa olemassa olevat tapahtumat. Listassa tässä vaiheessa on vasta aikaisemmassa kohdassa luotu tapahtuma. *Stand*-mallin assosiaatioksi on luotu *belongsTo event*, joten jokainen osasto kuuluu aina yhteen tapahtumaan.

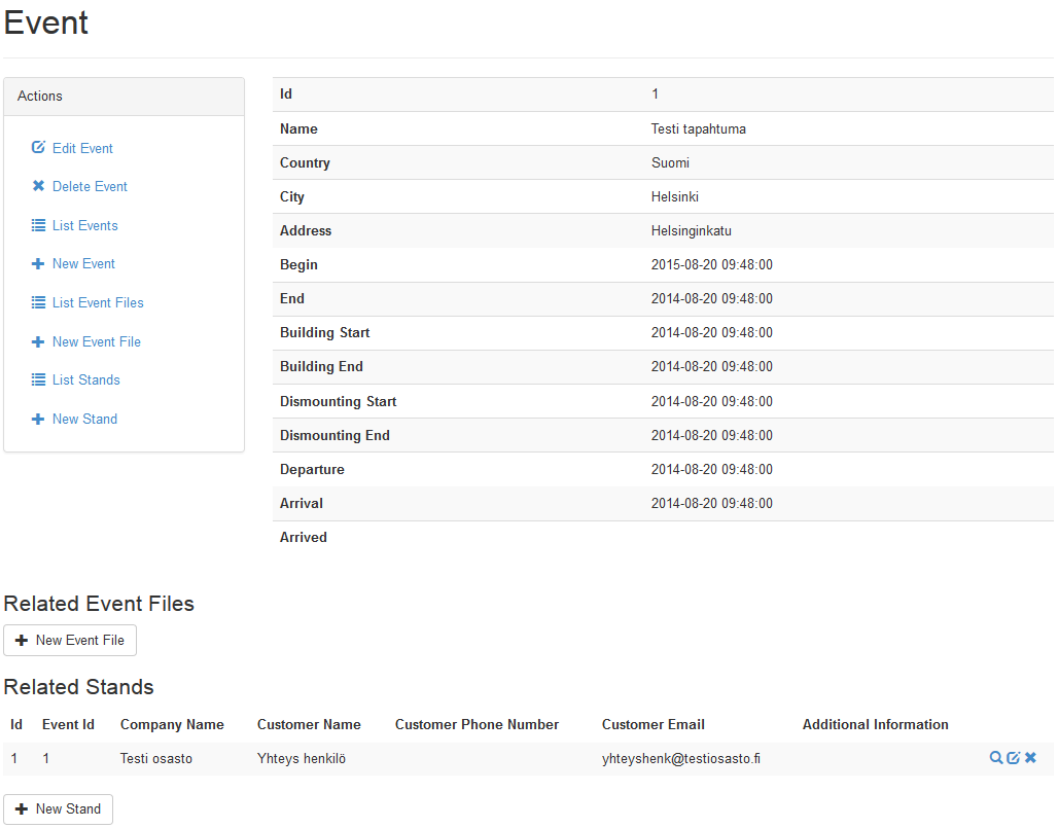


Kuvio 24. Osaston lisäys ja liittäminen olemassa olevaan tapahtumaan.

Mikäli assosiaatio on onnistuneesti lisättynä, on osaston lisäys- näkymä kuvion 24 mukainen.

10.3.4 View-sivu

View-sivu näyttää kyseisen mallin yhden rivin tiedot sekä siihen liittyvät *hasOne*- sekä *hasMany*-assosiaatiot. Kun osasto on liitetty aikaisemmin tehtyyn tapahtumaan, on se nähtävissä kyseisen tapahtuman *view*-sivulla (Kuvio 25).



Kuvio 25. Tapahtuma ja assosiaatiot.

10.3.5 Muut toiminnot

View-sivulla on havaittavissa myös *actions*-listauksessa *Delete Event*- ja *Edit Event*-linkit. Nimiänsä vastaavasti niistä pääsee muokkaamaan sekä poistamaan tapahtuman. *Edit Event*-linkkiä seuratessa avautuu tapahtuman lisäyssivua vastaava sivu, jonka kentät ovat esitäytettyinä tapahtuman tiedoilla. *Delete Event*-linkki luo HTTP-POST -pyynnön tapahtuman ohjaimen poistamisesta vastaavaan toimintoon. Nämä toiminnot ovat jo täysin toimivia.

10.3.6 Yhteenveto leivotuista pohjista ja toiminnoista

Käyttämällä Bake-konsolisovellusta on mahdollista luoda toimiva sivusto pelkästään tietokannan pohjalta. Kaikki toiminnot ovat toimivia, ja kun sovellus vielä asetetaan käyttämään lisättyä Bootstrap-pohjaa, niin kaikki myös näyttää yrityksen sisäiseen käyttöön tarpeeksi hyvältä.

10.4 Leipomisen jälkeen

Kaikki perustoiminnot ovat kunnossa, joten seuraavaksi luodaan kaikki sovelluskohtaiset ominaisuudet. Leipomisen jälkeen suurimpina tehtävinä ovat:

- Käyttäjien lisääminen, sisäänkirjautumisen mahdollistaminen sekä käyttäjä oikeuksien määrittäminen
- Dynaamisten toimintojen luominen sitä vaativille sivuille
- Tiedostojen lisääminen, poistaminen sekä lataaminen käyttäjäliittymän kautta sitä vaativille sivuille
- Varastotuotteiden pitäminen ajan tasalla
- Sivuston optimointi.

10.5 Autentikointi

Autentikointi suoritetaan autentikointi-komponentilla, jonka käyttötarkoituksesta olen kertonut kappaleessa 4.12.3. Komponentin tulee olla käytössä sivuston laajuisesti, joten on se kätevintä tehdä *AppControllerissa*. Komponentin lisääminen ja määrittäminen tapahtuu seuraavasti:

```
public $components = array(
  'Session',
  'Auth' => array(
    'loginRedirect' => array(
      'controller' => 'pages',
      'action' => 'display',
      'home'
    ),
    'flash' => array(
      'element' => 'alert',
      'key' => 'auth',
      'params' => array(
        'class' => 'alert-danger',
        'close' => true
      )
    ),
    'logoutRedirect' => array(
      'controller' => 'pages',
      'action' => 'display',
      'home'
    ),
    'authError' => 'Sinulla ei ole oikeuksia tälle sivulle!',
    'authenticate' => array(
      'Form' => array(
        'passwordHasher' => 'Blowfish'
      )
    ),
    'authorize' => array('Controller')
  ),
  'Cookie'
);
```

Tässä määritellään *Session*-, *Auth*- ja *Cookie*-komponentit käytettäväksi jokaisella ohjaimella. Seuraavassa listauksessa esittelen autentikointi-komponentille määritetyt asetukset lyhyesti:

- LoginRedirect-asetus asettaa sivun, johon uudelleen ohjataan onnistuneen kirjautumisen jälkeen.
- Flash asettaa käytettävän virheviestin käyttämän elementin sekä avaimen.
- LogoutRedirect asettaa uudelleenohjaussivun uloskirjautuessa.
- AuthError asettaa käytettävän virheviestin pyrittäessä sisälle ilman oikeuksia.
- Authorize antaa pyydettyä sivua käsittelevälle ohjaimelle oikeuden tarkistaa käyttäjäoikeuksia.
- Authenticate asettaa käytettäväksi tunnistautumistavaksi lomakekirjautumisen sekä käytettäväksi salasanan tiivistäjäksi Blowfishin.

Authorize tarkoittaa käyttäjäoikeuksien valvomista. Käyttäjä voi olla kirjautunut sisään eli autentikoitu, mutta käyttäjällä ei välttämättä ole vaadittuja oikeuksia. Esimerkiksi sivustolla voi olla sivuja, joiden käyttäminen vaatii hallintatason oikeuksia. Tällaisia sivuja ovat yleensä arkaluonteisia tietoja käsittelevät sivut sekä käyttäjien hallinta.

Käyttäjäoikeuksien valvomisen voi tehdä jokaiselle ohjaimelle luomalla *isAuthorized*-metodin *AppControlleriin*. Jokaisessa ohjaimessa pystyy näin tarpeen tullen yliajamaan *AppControllerissa* luotu sääntö. Esimerkiksi sallitaan oletuksena hallintatason käyttäjälle pääsy sivuston jokaiselle sivulle ja estetään pääsy muilta. Seuraava esimerkki on siis luotu *AppController*-luokkaan.

```
public function isAuthorized($user)
{
  if (isset($user['role']) && $user['role'] === 'admin') {
      return true;
  }

    return false;
}
```

Todellisuudessa tämä ei välttämättä ole hyvä ratkaisu, sillä metodi estää pääsyn kaikilta paitsi käyttäjärooli adminilta, mutta tämä antaa esimerkin valossa kuvan CakePHP:n tavasta hallinnoida käyttäjäoikeuksia. Mikäli kuitenkin lähdetään liikkeelle siitä, että kaikkiin ohjaimiin on pääsy kielletty kaikilta, paitsi admin-roolin käyttäjiltä, on *AppController*-luokan metodi mahdollista yliajaa ohjainkohtaisesti. On kuitenkin yksinkertaisempaa estää yksittäisiä sivuja.

CakePHP:n autentikointi on pessimistinen ja oletuksena estää pääsyn kaikkiin toimintoihin (Cakephp 2014, www). Tämä tarkoittaa sitä, että komponentin käyttöönoton jälkeen jokainen sivu vaatii sisäänkirjautumisen. Julkiset sivut täytyy erikseen määritellä. Tämä tapahtuu lisäämällä sallittavat sivut sivua vastaavan ohjaimen *beforeFilter*-metodiin. Seuraava esimerkki esittelee ohjaimen indeksitoiminnon sallimista myös sisäänkirjautumattomille käyttäjille.

```
public function beforeFilter()
{
  $this->Auth->allow('index');
}
```

10.6 Käyttäjän luominen

Kappaleessa 10.5 määriteltiin sivusto vaatimaan sisäänkirjautumista sekä käyttämään salasanan tiivisteluokkaa nimeltä Blowfish. Käyttäjän tallentaminen täytyy ohjata mallissa salasanantiivistämiseen ennen tallentamista tietokantaan:

```
public function beforeSave($options = array())
{
  if (isset($this->data[$this->alias]['password'])) {
    $passwordHasher = new BlowfishPasswordHasher();
    $this->data[$this->alias]['password'] =
        $passwordHasher->hash(
          $this->data[$this->alias]['password']
        );
  }
  return true;
}
```

Lisäksi mallille täytyy kertoa käytettävästä tiivisteluokasta:

```
App::uses('BlowfishPasswordHasher', 'Controller/Component/Auth');
```

Lisätäksemme käyttäjän, tulee ensin sallia käyttäjän lisäyssivun käyttäminen, koska oletuksena CakePHP:n autentikointi-komponentti estää kaikki sivut:

```
//Component/UserController.php
public function beforeFilter()
{
  parent::beforeFilter();
  $this->Auth->allow('add');
}
```

Tämän avulla avataan käyttäjän lisäyssivu väliaikaisesti. Kun ensimmäinen käyttäjä on lisätty, voi yllä olevan koodin poistaa ja näin estää käyttäjän lisäämisen käyttäjiltä, jotka eivät ole kirjautuneena sisään.

10.7 Dynaamisten toimintojen luominen

Sovelluksen dynaamiset toiminnot painottuvat pääasiassa pakkauslistojen luomiseen sekä pakkaamiseen. Dynaamisilla toiminnoilla tarkoitan uusien rivien lisää-

mistä ilman sivun uudelleen lataamista. Tähän tehtävään käytän jQueryn *ajax*-metodia. CakePHP:ssa AJAX toimii, kuten missä tahansa PHP-sovelluksessa. AJAXilla kutsuttavassa ohjaimen toiminnossa on mahdollista varmistaa, että kutsu on suoritettu AJAXilla. On yleistä, että tällaiset toiminnot on tarkoitettu ainoastaan kutsuttavaksi AJAXilla. Muiden pyyntöjen estäminen ja ainoastaan AJAXin salliminen tapahtuu seuraavasti:

```
if ($this->request->is('ajax')) {
  //Suoritettava koodi
} else {
  //Muuten laukaistaan ei sallittu poikkeus
  throw new MethodNotAllowedException();
}
```

Monesti AJAXilla haettaessa tietoa palvelimelta, tuodaan se JSON-muodossa ja JavaScriptissä muodostetaan saadusta vastauksesta haluttu HTML-kokonaisuus. Oletuksena CakePHP etsii suoritettavan ohjaimen toimintoa vastaavaa näkymää, joten näkymän suorittaminen on estettävä AJAX-kutsua vastaavassa toiminnossa:

```
$this->autoRender = false;
```

Palautettava tieto saadaan JSON:iksi seuraavalla:

```
return json_encode($palautettava_array);
```

Joskus on myös tarpeellista palauttaa tieto valmiiksi HTML:nä esimerkiksi näkymästä tai elementistä, jolloin JavaScriptin puolella säästytään vastauksen käsittelemiseltä. CakePHP:ssa tulee valmiiksi mukana AJAX-ulkoasu, joka palauttaa ainoastaan näkymästä tai elementistä haetun sisällön. Sovellus täytyy määrätä käyttämään AJAX-pohjaa sekä mahdollista elementtiä, jonka sisältö halutaan palauttaa. Tämä tapahtuu seuraavasti kutsua vastaavassa toiminnossa:

```
$this->layout = 'ajax';
$this->viewPath = 'Elements';
$this->render('elementin_nimi');
```

Sitä miten JavaScriptillä otetaan vastaus vastaan ja lisätään dokumenttiin, ei käsitellä tässä opinnäytetyössä. Varsinkin jQueryä käytettäessä löytyy verkosta paljon ohjeita kyseisten toimintojen suorittamiseen.

10.8 Tiedostojen sekä kuvien käsittely

Usein verkkosovelluksissa on tarpeen sallia kuvien sekä tiedostojen lisääminen. Esimerkiksi käyttäjän profiilikuva toimii hyvänä esimerkkinä. Opinnäytetyöhön täytyy sallia kuvien lisääminen varaston tuotteisiin tunnistamista varten.

Vaikka CakePHP:ssa useat toiminnot on tehty valmiiksi, niin kuvien ja tiedostojen lisäämiseen täytyy nähdä hieman vaivaa.

10.8.1 Kuvien lisääminen

Monesti toiminnot, jotka ovat mahdollisesti toistuvia, kuten se, että taulun riviin voi liittyä kuva, on hyvä tehdä uudesti käytettäväksi. CakePHP:ssa se on tehty mahdolliseksi mallien käyttäytymisillä. Kappaleessa 4.11 olen kertonut enemmän käyttäytymisten tarkoituksesta. Opinnäytetyön järjestelmään loin oman käyttäytymisluokan nimeltä *HasImageField*, joka mahdollistaa yhden kuvan liittämisen tietokantataulun riviin. Mikäli kuvia täytyy pystyä lisäämään useampia yhtä riviä kohden, tulisi lähestymistavan olla seuraavassa kappaleessa läpikäymäni käyttäytymisen tapainen. Käyttäytymisen käyttöönotto vaatii mallikohtaisten määritysten asettamisen käyttäytymistä kutsuttaessa:

```
public $actsAs = array(
  'HasImageField' => array(
    'image_input' => 'image_input',
    'image_field' => 'image',
    'dir_name' => 'itemImages',
    'max_size' => 400,
    'thumb_max_size' => 80,
    'thumb_square' => true,
    'name_by_field' => 'id'
  )
);
```

Kerron lyhyesti, mitä jokainen asetus tarkoittaa.

- Image_inputissa kerrotaan luokalle, minkä nimisestä input-kentästä kuvaa odotetaan lisättävän.

- Image_field määrittää, minkä nimiseen sarakkeeseen tiedoston nimi tallennetaan.
- Dir_name on kansion nimi, joka on CakePHP:n oletuskuvakansion alla.
- Max_size on yksinkertaisesti kuvan maksimikoko. 400 määrittää maksimikooksi esimerkiksi 400x600 tai 600x400.
- Thumb_max_size määrittää pienoiskuvan maksimikoon.
- Thumb_square määrittää, mikäli pienoiskuva leikataan neliöksi.
- Name_by_field asettaa kuvaa nimettäessä käytettävän sarakkeen, asetettaessa tämä *id*:ksi on kuva helppo jäljittää kyseiseen riviin.

Käyttäytymisen käyttöönotto vaatii yllämainittujen määritysten lisäksi tauluun lisättävän, käyttäytymisen määrityksessä määritetyn, *image_field* nimisen sarakkeen. Yllä olevan esimerkin mukaisesti olen luonut *image*-sarakkeen käyttäytymistä vaativan mallin tauluun. Lisäksi täytyy mallin lisäyslomakkeeseen, josta kuva on tarkoitus lisätä, lisätä kuvanlisäyskenttä seuraavasti:

```
<?php
  echo $this->Form->input('image_input',
    array('label' => 'Kuva', 'type' => 'file')
  );
?>
```

Kun nämä mainitut kohdat on lisättynä, tallentaa käyttäytyminen aina kuvan, kun sellainen lisättävästä tiedosta löytyy. Automaattinen lisääminen on suoritettu käyttäen CakePHP:n callback-metodia *afterSave*. Olen kertonut tarkemmin CakePHP:n callback-metodeista kappaleessa 4.10.

Käyttäytymisen kautta kuvan poistaminen tapahtuu myös automaattisesti, kun kyseinen rivi poistetaan.

10.8.2 Usean tiedoston liittäminen tietokantataulun riviin

Opinnäytetyöni tietokantataulut *events* ja *stands* vaativat mahdollisuuden liittää tiedostoja. Näitä tiedostoja voivat olla esimerkiksi tapahtuman aikataulut tai osastojen piirustukset. Myöskään tähän ei CakePHP tarjoa ”out of the box”-ratkaisua, joten loin oman käyttäytymisen nimeltä *FileModel*. Tämä käyttäytyminen eroaa

huomattavasti edellisessä kappaleessa esitellystä *HasImageField*-käyttäytymisestä. Usean tiedoston tai rivin liittäminen yhteen riviin tarkoittaa CakePHP:ssä *hasMany*-assosiaatiota. Mallien assosiaatioista olen kertonut enemmän kappaleessa 4.8.

events-tauluun liitettävä *FileModel*-käyttäytyminen vaatii uuden taulun luomista. Nimi on vapaavalintainen, mutta helposti toisiin liitettävät nimet helpottavat asioita. Valitsin nimeksi *event_files*. Seuraavassa listauksessa esittelen käyttäytymisen vaativat tietokantataulun sarakkeet.

- Id on rivin yksilöivä auto increment sarake.
- Parent_id on viitattavan taulun viiteavain ja *events*-tauluun liitettäessä se on *event_id*. Sarake määrittää *events* taulun rivin, mihin tiedosto kuuluu.
- File_name on tiedoston alkuperäinen nimi.
- File_size on tiedoston koko.
- File_mime on tiedoston tyyppi.

Näiden lisäksi loin sarakkeet *description*, *name*, *modified* ja *created*. Nämä eivät ole käyttäytymisen kannalta pakollisia sarakkeita, mutta halutessaan käyttäjä saa näin tiedostoja lisätessään antaa otsikon sekä kuvauksen tiedostolle ja myös luonti- ja muokkauspäivät tallentuvat tietokantaan.

Tiedoston lisäys näkymä vaati HTML-lomakkeen, jossa on parent_id-kenttä sekä tiedostokenttä nimi attribuutilla *file*. Seuraavassa esimerkissä on nähtävillä myös käyttäytymisen kannalta vapaaehtoiset kentät.

```
<?php echo $this->Form->create('EventFile',
       array('type' => 'file')
); ?>
      <div class="form-group">
             <?php  echo $this->Form->input('name', array(
                    'label' => 'Otsikko',
                    'class' => 'form-control',
                   'placeholder' => 'Tiedoston otsikko'
             )); ?>
      </div>

      <?php echo $this->Form->hidden('event_id'); ?>

      <div class="form-group">
               <?php echo $this->Form->input('description', array(
                    'label' => 'Kuvaus',
                   'class' => 'form-control',
                   'placeholder' => 'Tiedoston kuvaus'
             )); ?>
      </div>

      <div class="form-group">
             <?php echo $this->Form->input('file', array(
                    'label' => 'Tiedosto',
                    'type' => 'file'
             )); ?>
      </div>

      <div class="form-group">
             <?php echo $this->Form->submit('Lähetä', array(
                   'class' => 'btn btn-default'
             )); ?>
      </div>

<?php echo $this->Form->end(); ?>
```

Käyttäytyminen hoitaa tiedoston tallentamisen sekä tietojen keräämisen tiedostosta, tähän riittää, että ohjaimessa kutsutaan mallin *save*-metodia. Malliin tulee kuitenkin vielä lisätä käyttäytymisen liittäminen malliin sekä yksi pakollinen määritys.

```
public $actsAs = array(
  'FileModel' => array(
    'prefix' => 'event_'
  )
);
```

Tämän pakollisen määrityksen lisäksi, on mahdollista antaa muutamia vapaaehtoisia määrityksiä. Seuraavassa listauksessa esittelen nämä määritykset lyhyesti.

- Prefix kertoo käyttäytymiselle *parent_id*:ssä käytettävän etuliitteen eli mallin nimen lisättynä alaviivalla.
- Name_by_field kertoo käyttäytymiselle, että tiedosto tulee nimetä tässä mainitun sarakkeen mukaan. Mikäli määritystä ei aseteta, luodaan tiedosto sen alkuperäisellä nimellä.
- Dir määrittää *webroot*-kansion alla olevan kansion, mihin tiedostot tallennetaan. Oletuksena *files.*
- AllowedExtensions asetus määrittää sallitut tiedostopäätteet. Oletuksena se ei salli mitään.
- MaxSize määrittää tiedoston suurimman sallitun koon. Oletuksena se on 5MB.

Käyttäytymisen *getFileInfo*-metodi palauttaa kaikki tarvittavat tiedot tiedoston lataamiseen, josta kerrotaan enemmän seuraavassa kappaleessa. Käyttäytymisen koodi löytyy liitteistä.

10.8.3 Tiedostojen lataaminen

CakePHP:ssa tiedostojen lataaminen on tehty helpoksi. Yksinkertaisimmillaan se tapahtuu seuraavan esimerkin mukaisesti.

```
$this->response->file($filePath);
return $this->response;
```

Opinnäytteeseen on kuitenkin luotu yksinkertainen komponentti koodin uudelleenkäytettävyyden vuoksi. Komponentin nimi on Downloadable. Komponentti toimii yhdessä *FileModel*-käyttäytymisen kanssa. *Downloadable* asettaa *FileModelin getFileInfo*-metodista saadut tiedot CakePHP:n response-objektiin.

Downloadable-komponentin lataus-metodin kutsu havainnollistuu yksinkertaisimmillaan seuraavassa esimerkissä.

```
return $this->Downloadable->download(
       $this->EventFile->getFileInfo()
);
```

Download-metodi asettaa ohjaimen *response*-objektin eli vastauksen seuraavasti.

```
public function download($fileInfo)
{
  //As of 2.3
  $this->controller->response->file(
    $fileInfo['path'] . DS . $fileInfo['file'],
    array(
      'download' => true,
      'name' => $fileInfo['file_name']
    )
  );
  return $this->controller->response;
}
```

Ohjaimen response-objektiin annetaan tiedoston relatiivinen polku, pakotetaan lataus sekä asetetaan ladattavan tiedoston nimi. *FileModel*-käyttäytyminen ja *Downloadable*-komponentti mahdollistavat tiedostojen tallentamisen palvelimelle omavalintaisella tiedostonimellä ja lataamisen tiedoston alkuperäisellä nimellä.

10.9 Varastotuotteiden pitäminen ajan tasalla

Järjestelmän kannalta olennaisin osa on pitää varastossa olevien tuotteiden tiedot päivitettynä. Kun tapahtuman pakkauslistaan merkataan tuote, tulee se vähentää käytettävissä olevista tuotteista. Tapahtuman saapuessa takaisin tulee tuote merkata takaisin käytettäväksi. Ongelmalliseksi tämän tekevät useat eri vaihtoehdot, joita pakkauslistan tuotteilla voi olla. Pakkauslistan tuote voi myös olla sellainen tarvike, joka ei kuulu mihinkään varaston tuotteista. Jos se on ainoastaan kerran tai muutaman kerran käytettävä tuote, niin sitä ei ole hyödyllistä pitää varastossa.

Samalla pakkauslista tuotteiden lisäämistä sekä muokkaamista täytyy seurata pakkaustilanteen vuoksi. Tuotteille merkataan sarakkeeseen *checked* boolean arvo 1, kun tuote merkataan pakatuksi. Tämä sarake tulee aina muuttaa takaisin arvoon 0, kun tuotteen määrää muokataan pakkauslistassa. Pakkauslistassa pidetään myös kirjaa pakkauslistassa olevien tuotteiden määrästä CakePHP:n *counterCache*-asetuksella, joka on määriteltynä assosiaatio-määrittelyssä, sekä pakattujen tuotteiden määrästä manuaalisesti callback-metodeissa.

Suurin osa näistä tarkistuksista tehdään mallin callback-metodeissa, joka pitää huolen siitä, että niitä ei erikseen ole tarvetta kutsua. Seuraavassa esimerkissä ha-

vainnollistetaan, miten pakkauslistan pakattujen tuotteiden määrä pidetään ajan tasalla, kun siihen kuuluva tuote poistetaan:

```
public function afterDelete()
{
  if ($this->data['ItemListItem']['checked'] == 1) {
    $this->ItemList->updateCompletedCount(
      '-1',
      $this->data['ItemListItem']['item_list_id']
    );
  }
}
```

AfterDelete callback-metodi suoritetaan aina, kun mallissa suoritetaan poisto eikä callback-metodien suorittamista ole erikseen estetty. Tässä pakkauslistatuotteen poiston jälkeen tarkistetaan, oliko tuote merkattu pakatuksi. Jos oli, kutsutaan *ItemList*-mallin metodia *updateCompletedCount*, joka suorittaa pakkauslistan *completed_count* sarakkeen arvon pienentämisen.

Vastaavasti seuraavassa esimerkissä suurennetaan pakkauslistan *completed_count*-arvoa *afterSave*-callbackissä, mikäli tallennettavan pakkauslistatuotteen *checked*-arvo on asetettu.

```
public function afterSave($created, $options = array())
{
  //If operation was an update
  if (!$created) {
    if (isset($this->data['ItemListItem']['checked'])) {
      $this->ItemList->updateCompletedCount(
        $this->data['ItemListItem']['checked'],
        $this->data['ItemListItem']['item_list_id']
     );
    }
  }
}
```

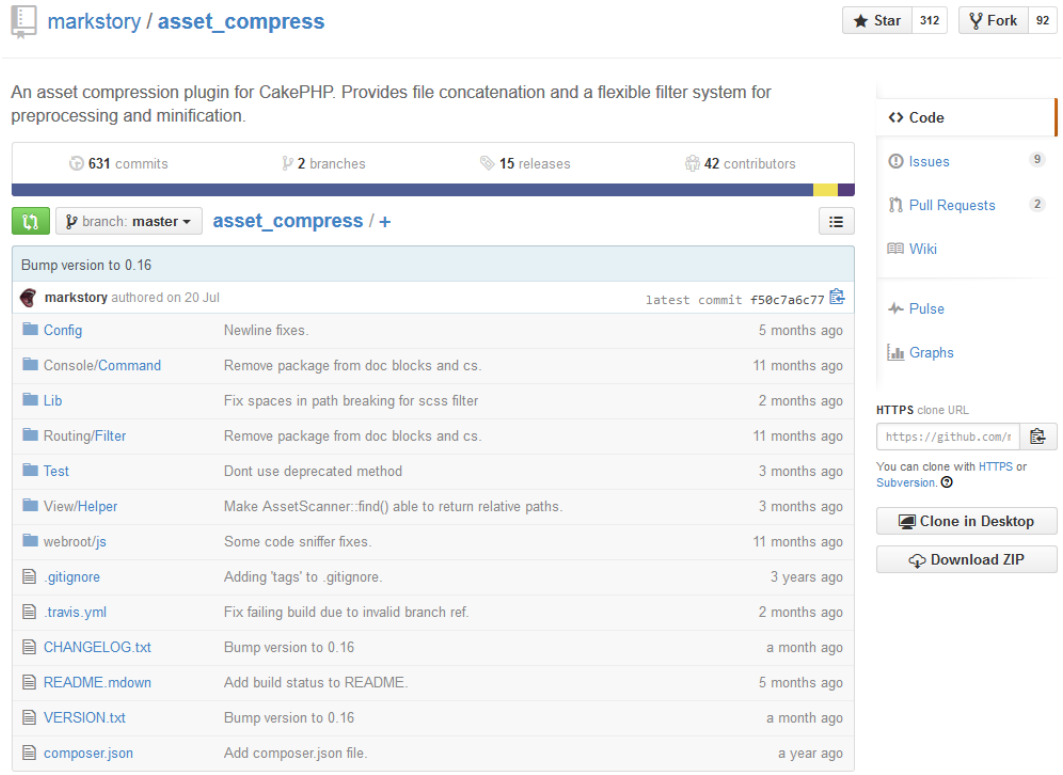
Varastotuotteiden ajan tasalla pitäminen tapahtuu *beforeSave*- ja *afterSave*-callbackeissä, mutta niiden koodit ovat esimerkiksi liian pitkät.

10.10 Sivuston optimointi

Kun toiminnot ovat valmiita ja ulkoasu toivotun mukainen, on vuorossa järjestelmän nopeuden parantaminen. Järjestelmässä, joka on tarkoitettu yrityksen sisäi-

seen käyttöön, ei optimoimisesta saa niin suurta hyötyä kuin sivustoissa, joissa käyttäjiä on jatkuvasti suuret määrät. Järjestelmään on kuitenkin tehty pientä optimointia lisäämällä käyttöön *AssetCompress*-lisäosa. Lisäosan tarkoituksena on yhdistää käytettävät tyyli- sekä JavaScript-tiedostot ja minimoida ne. Näin vähennetään HTTP-pyyntöjen määrää sekä pienennetään ladattavan tiedon kokoa ja saadaan sivuston lataaminen nopeammaksi.

AssetCompress-lisäosan käyttöönotto tapahtuu lataamalla lisäosa osoitteesta https://github.com/markstory/asset_compress esimerkiksi ZIP-tiedostona (Kuvio 26).



Kuvio 26. AssetCompress lisäosan lataaminen.

Lisäosa siirretään kansioon *app/Plugin/AssetCompress* ja *asset_compress.ini*-tiedosto siirretään kansioon *app/Config*. Edellä mainittuun INI-tiedostoon luodaan kaikki lisäosan asetukset. Tämän jälkeen on CakePHP ohjattava käyttämään lisäosaa lisäämällä *app/Config/bootstrap.php*-tiedostoon seuraava:

```
Configure::write('Dispatcher.filters', array(
  'AssetDispatcher',
));

CakePlugin::load('AssetCompress', array('bootstrap' => true));
```

Myös *app/Controller/AppController.php*-tiedostoon tulee lisätä seuraava:

```
var $helpers = array('AssetCompress.AssetCompress');
```

Sen jälkeen luodaan lisäosalle asetukset *app/Config/asset_compress.ini* tiedostoon:

```
[General]
cacheConfig = false

[js]
timestamp = true
cachePath = WEBROOT/js/
filters[] = JShrinkFilter

[css]
timestamp = true
cachePath = WEBROOT/css/
filters[] = SimpleCssMin

[script.js]
files[] = jquery-1.11.1.min.js
files[] = bootstrap.min.js
files[] = jquery-ui.min.js
files[] = custom.js

[styles.css]
files[] = custom.css
files[] = bootstrap.min.css
files[] = jquery-ui.min.css
```

Yllä olevassa esimerkissä asetukset asettaa luoduille JS- sekä CSS-tiedostoille timestampin, kertoo tallennettavien tiedostojen kohdekansion sekä käytettävän suodattimen tiedostojen pienentämistä varten. Timestampin lisääminen tiedostoon mahdollistaa selainten välimuistiin asettamien tiedostojen korvaamisen päivitetyllä versiolla. Tämän jälkeen listataan tiedostot, jotka lisäosa käsittelee. Asetusten jälkeen tulee ulkoasussa korvata käytetyt tyyli- ja JavaScript-tiedostot lisäosan luomilla tiedostoilla:

```
echo $this->AssetCompress->css('styles');
echo $this->AssetCompress->script('script');
```

Yhdistetyt ja pienennetyt tiedostot asset_compress.ini tiedoston mukaan luodaan *cake*-konsolisovelluksella komennolla *cake AssetCompress.AssetCompress build_ini*. Mikäli sovellus on onnistuneesti ajettu, on luotu tiedosto vastaavanlainen: script.v1408958431.js.

Yksi muista mahdollisista sivuston suorituskykyä parantavista tehtävistä on muun muassa CakePHP:n *CacheHelperin* käyttö, josta olen kertonut kappaleessa 4.14.1. Myös CakePHP:n *Cache*-luokka on yksi käytettävissä olevista mahdollisuuksista. Se mahdollistaa esimerkiksi tietokanta hakujen tulosten tallentamisen esimerkiksi PHP:n Memcached-lisäosalla. Sivuston suorituskyvyn parantamiseen on apuna käytetty YSlow FireFox-lisäosaa.

11 YHTEENVETO

Opinnäytetyön tavoitteena oli toteuttaa Varastonhallintajärjestelmä Messunikkarit Oy:lle, käyttäen CakePHP-sovelluskehystä. Työssäni olen käynyt läpi CakePHP:n tärkeimpiä ominaisuuksia, jotka pääsääntöisesti ovat osana jokaista CakePHP-sovellusta. Esittelemäni asiat kattavat vasta pienen osan CakePHP:n ominaisuuksista. Uskon kuitenkin, että juuri nämä ominaisuudet luovat tukevan pohjan lähes minkä tahansa projektin toteuttamiseen, johon kyseinen sovelluskehys valitaan käytettäväksi. Työssä esittelin myös toteuttamani järjestelmän tärkeimmät toiminnot ja ominaisuudet sekä lyhyesti niihin käyttämäni ratkaisut. Olen todennut kaikki työssäni esittelemät esimerkit sekä ratkaisut toimivaksi, mutta on hyvä muistaa, että CakePHP:n ominaisuudet tai niiden käyttäytyminen saattavat vaihdella uusien versioiden ilmestyessä. Tämän takia on aina hyvä tarkistaa käytössä olevan version uusimmat suositukset ja ominaisuudet suoraan CakePHP:n dokumentaatiosta eli Cookbookista (Ks. CakePHP 2014, www).

Verkkosovellukset monimutkaistuvat jatkuvasti ja CakePHP:ta käytettäessä on tullut selväksi, että sovelluskehykset ovat korvaamaton osa niiden kehityksessä. Monimutkaisten ja suurten sovellusten kehitysvaiheessa kasvaa vastaavasti tarve testaamiselle. Yksikkötestaus onkin sitä hyödyllisempää ja tärkeämpää mitä monimutkaisemmaksi sovellukset kasvavat. Työssäni olisin voinut käyttää CakePHP:n tarjoamaa testausympäristöä. Mikäli nyt aloittaisin työn tekemisen alusta, olisi yksikkötestien luominen siinä osana.

11.1 Tulokset

Onnistuin luomaan järjestelmän, joka vastaa tunnistettuja käyttötapauksia. Samalla vahvistin omaa osaamistani usealla osa-alueella, joka oli työn tekemisen yhteydessä omana henkilökohtaisena tavoitteena. Uskon myös, että järjestelmästä on yritykselle hyötyä, kunhan yrityksessä vain löytyy aikaa järjestelmän käyttöönottoon.

11.2 Kehittämisehdotuksia

Järjestelmässä on useita osa-alueita, joiden toimintaa voisi laajentaa. On myös useita ominaisuuksia, joita järjestelmään voisi lisätä. Järjestelmässä on siis jatkokehittämisen mahdollisuuksia.

Yksi näistä ominaisuuksista on ajan puutteen vuoksi pois rajattu reaaliaikainen kommentointi, joka alun perin oli tarkoituksena lisätä pakkauslistoihin, pakkauslistan tuotteisiin sekä varastotuotteisiin. Kommentointi toimisi epäselvyyksien ratkaisemisen apuna sekä kirjaamisena.

Toinen hyödyllinen lisäys olisi CakePHP:n välimuistiluokan käyttö, jonka avulla on mahdollista vähentää tietokantakutsuja tallentamalla tuloksia muistiin. Välimuistiluokka mahdollisesti nopeuttaisi järjestelmää, mutta alustavasti se lisätään vasta, jos sille esiintyy tarvetta.

Muita mietittyjä ominaisuuksia ovat etusivun pikalinkit, pakkauslistan tuotteeseen pakkaajan merkitseminen tietokantaan sekä viestien jättäminen käyttäjien kesken. Myös käyttäjien liittäminen tapahtumiin olisi mielenkiintoinen lisä. Se mahdollistaisi työntekijöiden työmatkojen seuraamisen.

LÄHDELUETTELO

CakePHP. 2014. Cookbook 2.x. Viitattu 28.9.2014. http://book.cakephp.org/2.0/_downloads/en/CakePHPCookbook.pdf.

Bootstrap. 2014. Sovelluskehyksen kotisivut. Viitattu 28.9.2014. http://getbootstrap.com/.

Abeysinghe, S. 2009. PHP Team Development : Easy and Effective Team Work Using MVC, Agile Development, Source Control, Testing, Bug Tracking, and More. Birmingham. Packt Publishing Ltd.

Benedetti, R & Cranley, R. 2011. Head first jQuery. Sebastopol. O'Reilly Media.

Curioso, A & Bradford, R & Galbraith, P. 2010. Expert PHP and MySQL. Hoboken. Wrox.

Eichorn, J. 2007. Understanding AJAX : using JavaScript to create rich Internet applications. Upper Saddle River. Prentice Hall.

Gilmore, J. 2005. PHP & MySQL: tehokas hallinta. Tehokas hallinta. Berkeley. Apress.

Goodman, D. 2010. JavaScript Bible. 7th ed. Hoboken. Wiley.

Korpela, J. 2011. HTML5: Uudet ominaisuudet. Jyväskylä. Docendo.

Lano, K. 2009. UML 2 Semantics and Applications. Hoboken. John Wiley & Sons.

Porebski, B., Przystalski, K. & Nowak, L. 2011. Building PHP Applications with Symfony, CakePHP, and Zend Framework. Hoboken. John Wiley & Sons.

Reiss, E. 2012. Usable Usability : Simple Steps for Making Stuff Better. Hoboken. John Wiley & Sons.

Syam, A & Bari, A. 2008. CakePHP Application Development: Step-by-step introduction to rapid web development using the open-source MVC CakePHP framework. Birmingham. Packt Publishing Ltd.

FileModel-käyttäytyminen

```
<?php
App::uses('ModelBehavior', 'Model');

/**
* Behavior for file management
*
* Behavior makes it easier to handle files
*
* Behavior requires certain fields in the table
* Required fields are:
* id
* file_name
* file_size
* file_mime
*/
class FileModelBehavior extends ModelBehavior
{


/**
* Method for behavior configuration
*
* Possible settings are:
* prefix    Used to prefix the file name
* name_by_field   Used to name the file by this field. If empty,
named by original file name
* dir       Relative directory to webroot where the files are
saved
* allowedExtension Sets allowed extensions, if no given, nothing
is accepted
* maxSize   Defines maximum file size
*
* @param   Model   $Model    Model into which the behavior is at-
tached into
* @param  array  $settings Settings for the behavior
* @return void
*/
```

```
public function setup(Model $Model, $settings = array())
{
    if (!isset($this->settings[$Model->alias])) {
        $this->settings[$Model->alias] = array(
            'prefix' => '',
            'dir' => 'files',
            'name_by_field' => '',
            'allowedExtensions' => array(),
            'maxSize'
        );
    }

    $this->settings[$Model->alias] = array_merge(
        $this->settings[$Model->alias], (array)$settings
    );

    foreach ($this->settings[$Model->alias] as $key => $value) {
      $this->{$key} = $value;
    }

    if (!empty($this->name_by_field) &&
    isset($Model->data[$Model->name][$this->name_by_field])) {
      $this->filename =
                  $Model->data[$Model->name][$this->name_by_field];
      }
}

/**
* Method to run after model validation
*
* Simply adds behavior specific validation rules
*
* @param Model $Model Model into which the behavior is attached
into
* @param  array  $options Model save() options
* @return boolean true
*/
public function beforeValidate(Model $Model, $options = array())
{
      $Model->validator()->add('file', array(
            'size' => array(
                  'rule' => array(
                        'fileSize', '<=', $this->maxSize
                   ),
                   'message' => 'Tiedoston suurin sallittu koko on
                   '. $this->maxSize
            ),
            'allowedExtensions' => array(
                  'rule'  => array(
                     'extension', $this->allowedExtensions
                   ),
                  'message' => 'Tiedosto tyyppiä ei ole sallittu.'
            )
      ));

      return true;
}
```

```
/**
* Method to run before every save operation to the model
*
* Adds needed fields into models data array from the uploaded file
*
* @param Model $Model Model into which the behavior is attached
into
* @param array $options Model save() options
* @return boolean true
*/
public function beforeSave(Model $Model, $options = array())
{
      $Model->data[$Model->name]['file_extension'] =
         substr(
           strrchr(
             $Model->data[$Model->name]['file']['name'],
             '.'
          ), 1);

      $Model->data[$Model->name]['file_name'] =
         $Model->data[$Model->name]['file']['name'];

      $Model->data[$Model->name]['file_size'] =
         $Model->data[$Model->name]['file']['size'];

      $Model->data[$Model->name]['file_mime'] =
          $Model->data[$Model->name]['file']['type'];

      return true;
}
```

```
/**
* Method to run after after every save operation to the model
*
* Saves the file after data is saved
*
* @param Model $Model Model into which the behavior is attached
into
* @param boolean $created Whether the save operation was an insert
or not
* @param array $options Model save() options
* @return void
*/
public function afterSave(
      Model $Model,
      $created,
      $options = array()
){
      if (!isset($this->fileName) &&
      !empty($this->name_by_field)) {
            $this->fileName =
                $Model->data[$Model->name][$this->name_by_field];
      }

      $file = isset($this->fileName) ?
          $this->prefix . $this->fileName . '.' .
          $Model->data[$Model->name]['file_extension']
      : $this->prefix . $Model->data[$Model->name]['file_name'];

      if ($Model->data[$Model->name]['file']['error'] ===
     UPLOAD_ERR_OK) {

          if (move_uploaded_file(
             $Model->data[$Model->name]['file']['tmp_name'],
             $this->dir . DS . $file)
          ) {
            return true;
          }
      }
}
```

```
/**
* Method to get model info
* @param   Model   $Model    Model into which the behavior is at-
tached into
* @param  int $id       Row id
* @return array row data
*/
public function getFileInfo(Model $Model, $id = null)
{
      if ($id) {
            $Model->id = $id;
      }

      $fileInfo = array();
      $fileInfo['path'] = $this->dir;

      if (!isset($Model->data[$Model->name]['file_extension']) ||
         !isset($Model->data[$Model->name]['file_name']) ||
         (!empty($this->name_by_field) &&
         !isset($Model->data[$Model->name][$this->name_by_field]))
      ) {
            $Model->read();
      }

      if (!empty($this->name_by_field) &&
          isset($Model->data[$Model->name][$this->name_by_field]))
      {
           $fileInfo['file'] =
             $this->prefix .
             $Model->data[$Model->name][$this->name_by_field]
             . '.' . $Model->data[$Model->name]['file_extension'];

      } else {
            $fileInfo['file'] =
               $this->prefix .
               $Model->data[$Model->name]['file_name'];
      }

      $fileInfo['file_mime'] =
            $Model->data[$Model->name]['file_mime'];

      $fileInfo['file_name'] =
            $Model->data[$Model->name]['file_name'];

      $fileInfo['file_extension'] =
            $Model->data[$Model->name]['file_extension'];

      return $fileInfo;
}
```

```
/**
* Method to run after every delete operation to the model
*
* Deletes the file
*
* @param  Model  $Model   Model into which the behavior is at-
tached into
* @param  boolean $cascade Model cascade option
* @return boolean true on success false on failure
*/
public function beforeDelete(Model $Model, $cascade = true)
{

      if (empty($this->fileName) && !empty($this->name_by_field))
{
            $Model->read();
      }

      $fileName = $this->prefix;
      if (!empty($this->name_by_field)) {

            $fileName .= empty($this->fileName) ?
               $Model->data[$Model->name][$this->name_by_field]
            : $this->fileName;

            $fileName .=
               '.' . $Model->data[$Model->name]['file_extension'];

      } else {
            $fileName .=
               $this->prefix .
               $Model->data[$Model->name]['file_name'];
      }

    if (unlink($this->dir . DS . $fileName)) {
      return true;
    }

      return false;
}
}
```

Downloadable-komponentti

```php
<?php
App::uses('Component', 'Controller');
/**
 * Component for easy downloading of a file
 *
 * This component does not do much
 * Simply returns the controllers own response where download is
forced
 *
 * Component is more of an example and a personal test
 */
class DownloadableComponent extends Component
{

      private $controller;

/**
 * Method run on initialization
 * @param controller $controller Components controller
 * @return void
 */
      public function initialize(controller $controller){
          $this->controller = $controller;
      }
/**
* Forces download by given info
*
* Needed info is:
* path
* file
* file_name
*
* @param array $fileInfo Array containing file info
* @return object of controller response
*/
      public function download($fileInfo = array()) {
            //As of v. 2.3
            $this->controller->response->file(
                  $fileInfo['path'] . DS . $fileInfo['file'],
                  array(
                      'download' => true,
                      'name' => $fileInfo['file_name']
                  )
            );

            return $this->controller->response;
      }
}
```