Ilkka Veteläsuo

Ohjelmistokehysten hyödyntäminen verkkopalvelun kehitystyössä

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Mediatekniikan koulutusohjelma
Insinöörityö
5.5.2011

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Ilkka Veteläsuo Ohjelmistokehysten hyödyntäminen verkkopalvelun kehitystyössä 53 sivua 5.5.2011
Tutkinto	insinööri (AMK)
Koulutusohjelma	mediatekniikka
Suuntautumisvaihtoehto	digitaalinen media
Ohjaajat	sähköisen markkinoinnin assistentti Sari Muona yliopettaja Kari Aaltonen

Insinöörityössä perehdyttiin ohjelmistokehysten hyödyntämiseen pienimuotoisen verkkopalvelun kehityksessä. Kehyksen käytöllä pyrittiin tehostamaan kehitettävän palvelun toteuttamista.

Ohjelmistokehykset tarjoavat sovellusta varten rungon, ja niiden pääasiallisena tarkoituksena on tehostaa ohjelmistokehitystä. Kehykset eivät itsessään ole toimivia sovelluksia, vaan sellainen saadaan aikaiseksi täydentämällä kehysten tarjoamaa runkoa. Kehysten arkkitehtuuria noudattamalla saavutetaan rakenteeltaan modulaarinen, helposti laajennettava sovellus. Toteutukseltaan kehykset hyödyntävät laajalti ohjelmistotekniikan suunnittelumalleja, jotka ovat yleisiä kuvauksia jonkin tietyn ongelman ratkaisemiseksi.

Työn perusteella saatiin luotua kattava kuva ohjelmistokehysten toiminnasta ja niiden tarjoamista ominaisuuksista. Näiden ohella perehdyttiin myös kehysten toimintaan liittyviin suunnittelumalleihin. Tietojen pohjalta pystyttiin tekemään valinta käytettävästä kehyksestä ja hyödyntämään sitä tehokkaasti.

Työn ohessa toteutetun verkkopalvelun luominen käytetyn kehyksen avulla sujui sille asetettujen tavoitteiden mukaisesti. Kehyksen käytöllä saavutettiin selvä hyöty verrattuna siihen, että sovellus olisi rakennettu tyhjän päälle. Toteutus olisi kuitenkin ollut mahdollista myös monella muulla kehysratkaisulla. Työn tekeminen tarjosi myös hyvät edellytykset muiden kehysten käyttämiselle, sillä suurin osa niistä perustuu samoihin toimintamalleihin.

Asiakasyrityksen ja sen asiakkaiden puolelta palaute verkkopalvelusta on ollut positiivista, ja yrityksen kannalta se on toiminut myös tärkeänä maineenhallinnan työkaluna.

Avainsanat	ohjelmistokehys, sovellus, PHP

Metropolia

Abstract

Author Title Number of Pages Date	Ilkka Veteläsuo Application frameworks as a tool to develop an online service 53 5.5.2011
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	Sari Muona, E-Business assistant Kari Aaltonen, Principal Lecturer

This study concentrates on using application frameworks as a tool to develop a small scale online service. The main purpose of using a framework was to make the development more fluent and efficient.

Application frameworks provide a skeleton for the application and their main purpose is to make development more efficient. On their own frameworks are not fully functioning applications, but such can be achieved by complementing them with application specific code. By using architecture provided by the framework, we get a modular and easily extendable application. Frameworks rely widely on design patterns, which describe a way to solve some commonly occurring problems in software engineering.

This study provided a comprehensive view on operation of the frameworks and what kind of features they can provide. In addition, design patterns were part of the study because those are widely used by various frameworks. The gathered information helped on making a decision to choose the right framework and to how use it efficiently.

The created online service is a portal for people who are interested in sports related commemorative coins. Mainly it offers products and info related to them. The use of a framework gave an evident boost on the development process and was helpful in many tasks. This study also gave good qualifications to use other similar frameworks, because usually they share the same patterns.

Feedback for the portal from the customers and the company has been positive. The portal has also been a good tool for reputation management and a good medium to present athletes the company cooperates with.

Keywords	application framework, application, PHP

Lyhenteet

PHP	(PHP Hypertext Preprocessor). Skriptikieli, joka on saavuttanut suuren suosion verkkopohjaisten sovellusten kehityksessä.
HTTP	(Hypertext Transfer Protocol). Protokolla, jota WWW-palvelimet käyttävät tiedonsiirtoon.
MVC	(Model-View-Controller). Ohjelmistotekniikan suunnittelumalli, joka jakaa sovelluksen kolmeen erilliseen osaan.
ORM	(Object-Relational Mapping). Tekniikka, jonka avulla toteutetaan tiedon muunnos oliopohjaisen ja skalaaristen arvojen välillä.
SQL	(Structured Query Language). Kyselykieli, jota käytetään tiedon hallintaan tietokantojen yhteydessä.
CSRF	(Cross-site request forgery). Tietoturvahaavoittuvuus, jossa käyttäjän selain saadaan tekemään pyyntö ulkopuoliselle palvelimelle

Sisällys

Liitteet

1 Johdanto

Insinöörityön tarkoituksena on perehtyä PHP-ohjelmointikielellä toteutettuihin ohjelmistokehyksiin ja niiden perustana olevaan MVC-malliin sekä tutkia niiden tarjoamia etuja pienimuotoisen verkkopalvelun kehityksessä.

Osana työtä toteutetaan CakePHP-kehyksen avulla asiakasportaali Suomen Monetalle ja tutustutaan tarkemmin kehyksen toimintaan ja ominaisuuksiin. Aiheen rajaamiseksi eri kehyksiin perehdytään vain yleisellä tasolla ja sisältö painottuu asiakastyössä käytettyyn kehykseen. Portaalin pääasiallisena tavoitteena yrityksen kannalta on tarjota urheiluaiheisista keräilytuotteista kiinnostuneille erillinen kanava niiden tilaamiseen ja sitä kautta lisätä myös niiden myyntiä.

Työn asiakasyritys, Suomen Moneta (Oy Nordic Moneta Ab), myy ja markkinoi keräilyrahoja ja mitaleita Suomessa. Yritys on osa vuonna 2001 perustettua Samlerhuset-konsernia, jolla on toimintaa yhdessätoista Euroopan maassa sekä Kiinassa. Suomessa yrityksen palveluksessa toimii noin 70 henkilöä. Tele- ja suoramarkkinoinnin lisäksi yritys myy ja markkinoi tuotteitaan laajasti myös internetin kautta, jossa markkinointi on toteutettu oman verkkokaupan ja erilaisten kampanjoiden avulla.

Yrityksellä on aikaisemmin ollut eri markkinointi- ja mainostoimistojen toteuttamia lyhytaikaisempia kampanjasivustoja. Kehitettävän portaalin tarkoituksena on tarjota yrityksen asiakkaille pidempiaikaiseen käyttöön sivusto, jossa on tarjolla urheiluun liittyviä keräilytuotteita ja niihin liittyvää sisältöä.

2 Ohjelmistokehykset

2.1 Yleisten ongelmien ratkaiseminen ja ominaisuuksien toteuttaminen

Laajat sovellukset voivat kasvaa arkkitehtuuriltaan ja rakenteeltaan hyvinkin monimutkaisiksi, ja siten myös niiden laajennettavuus ja ylläpito tulevat entistä työläämmiksi. Ohjelmistokehykset pyrkivät tehostamaan sovelluskehitystä hyödyntäen laajamittaista komponenttien ja arkkitehtuurin uudelleenkäyttöä sekä kehyksen tarjoamia perusominaisuuksia. [1, s. 187–189.]

Kehykset itsessään eivät ole varsinaisia sovelluksia, vaan tarjoavat rungon, jota täydentämällä saadaan aikaiseksi toimiva sovellus. Kehykset sisältävät laajennuskohtia, joita voidaan täyttää sovelluskohtaisella koodilla, muuttamatta itse kehyksen arkkitehtuuria. Laajennuskohdat ja niiden sovelluskohtaiselle koodille asettamat vaatimukset muodostavat yhdessä kehyksen erikoistamisrajapinnan. Erikoistamisrajapinnan yhteydessä käytetty erikoistamismekanismi määrittää, kuinka kehyskohtainen ja sovelluskohtainen koodi on sidottu toisiinsa. Erikoistamismekanismi ei ole yksiselitteinen, vaan kehys saattaa hyödyntää useita eri mekanismeja. Yleisesti kehykset voidaan luokitella niiden pääasiallisen erikoistamismekanismin perusteella abstrakteihin, muunneltaviin ja koottaviin kehyksiin. [1, s. 189–190.]

Yksinkertaisimpia toteutukseltaan ovat abstraktit kehykset, jotka sisältävät ainoastaan rajapintoja, joiden avulla määritetään sovelluksen luokkien palvelut. Abstrakti kehys ei juuri rajoita sovelluksen kehitystä, koska se vaatii ainoastaan määritettyjen rajapintojen noudattamista. Abstrakteja kehyksiä käytetään harvoin itsenäisesti, ja yleensä ne toimivat osana suurempaa kehyskokonaisuutta. [1, s. 195–196.]

Muunneltavien kehysten (white-box framework) toiminta perustuu kehyksen tarjoamiin luokkiin, jotka toimivat kantaluokkina sovelluskohtaisille luokille. Muunneltavat kehykset ovat tehokkaita työkaluja ohjelmistokehityksessä, mutta niiden tehokas käyttö vaati kehittäjältä tietämystä kehyksen sisäisestä toiminnasta. Erikoistamisrajapinnan dokumentaatio on erittäin tärkeässä osassa, koska toiminnan selvittäminen pelkän

koodin perusteella voi olla hyvinkin työlästä. [1, s. 196–198; 2.]

Koottavia kehyksiä (black-box framework) voidaan pitää kehittyneinä malleina muunneltavista kehyksistä, jolloin kantaluokat on saatu kehitettyä niin pitkälle, ettei niitä tarvitse enää periyttää. Periyttämisen sijaan kehyksen kantaluokista luodaan ilmentymiä, joiden käytöstä muunnellaan niille annettavilla parametreillä. Koottavien kehysten etuna on käytön helppous, mutta toisaalta niiden muokattavuus on rajatumpi kuin muunneltavissa kehyksissä. [1, s. 199–202; 2.]

Ohjelmistokehykset on yleensä suunnattu tiettyä sovellusaluetta, kuten esimerkiksi verkossa toimivia sovelluksia varten. Vaikka eri sovellusalueen kehykset hyödyntävätkin samoja toimintaperiaatteita, määräytyvät kehysten tarkemmat toimintaperiaatteet ja ominaisuudet sovellusalueen tarpeiden mukaan.

Saman osa-alueen sovellukset sisältävät tiettyjä usein esiintyviä ongelmia ja ominaisuuksia, joiden suorittamista kehykset pyrkivät helpottamaan. Yleisten ongelmien ratkaisemisessa kehykset hyödyntävät ohjelmistotekniikan suunnittelumalleja, joilla tarkoitetaan yleistä tapaa ratkaista jokin usein esiintyvä ongelma. Suunnittelumallit eivät ole valmiita ratkaisuja ongelmiin, vaan ainoastaan kuvaavat, kuinka ne tulisi ratkaista [3, s. 12]. Sama ongelma on siis mahdollista ratkaista usealla eri toteutustavalla, hyödyntäen yhtä ja samaa suunnittelumallia. [4, s. 6; 5, s. 121–127.]

Ohjelmistokehyksen käytöllä on myös negatiiviset vaikutuksensa sovelluksen suorituskykyyn, koska niiden tarjoamista ominaisuuksista on pyritty tekemään yleispäteviä eikä niitä ole optimoitu kovin tarkasti tiettyyn käyttötarkoitukseen. Suorituskyvyssä voi kuitenkin olla suuriakin eroja kehyskohtaisesti, riippuen kehyksen toteutuksesta ja käyttötarkoituksesta. Uuden kehyksen käytöllä ei myöskään heti saavuteta suurta hyötyä sovelluksen kehitysprosessissa, koska kehysten oppimiskynnys on yleensä korkea. Suurin hyöty saavutetaan, kun kehystä on käytetty useassa projektissa ja sen käyttö on jo ennestään tuttua. Yhden kehyksen tunteminen ei myöskään takaa, että käyttäjä osaisi suoraan käyttää muita vastaavia kehyksiä, koska

niiden toiminta ja käytännöt voivat erota suurestikin toisistaan. [1, s. 188.]

2.2 PHP-ohjelmointikielellä toteutetut kehykset

PHP (PHP Hypertext Preprocessor) on monikäyttöinen skriptikieli, joka on saavuttanut suuren suosion verkkopohjaisten sovellusten kehityksessä. Skriptikielellä tarkoitetaan ohjelmointikieltä, jonka lähdekoodi käännetään tietokoneen ymmärtämään muotoon sitä ajettaessa. Kieltä käytetään pääasiassa web-palvelinympäristössä, jossa palvelimelle asennettu kääntäjä hoitaa lähdekoodin kääntämisen ja luo sen perusteella käyttäjän selaimelle esitettävän verkkosivun. [6; 7, s. 2–6.]

Tanskalainen Rasmus Lerdorf kehitti kielen alun perin työkaluksi omien kotisivujensa päivittämiseen ja julkaisi sen ensimmäinen julkisen version vuonna 1995. Vuonna 1997 israelilaiset Zaev Suraski ja Andi Gutmans kirjoittivat kielen perustan uusiksi, ja siihen myös nykyinen versio kielestä pohjautuu. [6.]

Kielen lisenssi pohjautuu avoimeen lähdekoodiin, eli se on vapaasti kaikkien levitettävissä ja muokattavissa. Avoimen lähdekoodin ansiosta käyttö on myös ilmaista. Ohjelmoijan näkökulmasta kieli on nopeasti opittavissa, mutta se tarjoaa kuitenkin laajat ominaisuudet ja kirjastot myös vaativampaan ohjelmointiin. [7, s. 2–6; 8; 4 s. 7.]

Käyttöympäristönsä mukaisesti kielellä toetutetut ohjelmistokehykset on tarkoitettu web-pohjaisten sovellusten kehittämiseen, ja suurin osa niistä perustuu MVC-suunnittelumalliin, joka jakaa sovelluksen rakenteeltaan kolmeen loogiseen osaan. Kehykset tarjoavat vaihtelevan määrän niiden käyttöalueella yleisesti käytettyjä ominaisuuksia, joista tärkeimpiin kuulu tiedonhallinta. Tiedonhallinnalla tarkoitetaan tässä yhteydessä tietokannan tai muun tietolähteen hyödyntämistä sovelluksessa käytetyn tiedon hallintaan. Muita kehysten yleisesti tarjoamia ominaisuuksia ovat istuntojen ja käyttäjien hallinta sekä erilaiset tietoturvaa parantavat ominaisuudet. Seuraavissa alaluvuissa esitellään pääpirteisesti kolme erityyppistä kehystä ja niiden ominaisuuksia.

2.3 CakePHP – helppokäyttöisyys ohjatun rakenteen avulla

CakePHP on vuonna 2005 julkaistu, MVC-suunnittelumalliin perustuva ohjelmistokehys, jonka kehityksestä vastaa ei-kaupallinen Cake Software Foundation [9]. Kehys tukee nopean kehityksen mallia, joka on ohjelmistokehityksessä käytetty menetelmä, jolla pyritään lyhentämään ohjelmiston suunnitteluun kuluvaa aikaa prototyyppien ja lyhyisiin iteraatioihin perustuvan kehitysmallin avulla. Mallissa pyritään luomaan ohjelmistosta nopeasti toimiva prototyyppi, jonka pohjalta sovellusta rakennetaan lyhyissä jaksoissa eteenpäin. Mallin hyödyntäminen helpottaa erityisesti sovelluksen ylläpitoa ja mahdollistaa nopeiden muutosten tekemisen siihen. [10, s. 2.]

Nopean kehityksen lisäksi kehyksen perusperiaatteita on helppokäyttöisyys. Suurin osa siitä perustuu tiukkaan määrättyihin käytäntöihin ja rakenteeseen, joiden avulla saadaan myös pidettyä sovelluksen rakenne yhtenäisenä [4, s. 7–8]. Kehys noudattaa ajatusmallia, joka pyrkii vähentämään asetusten määrittämistä ja painottaa valmiiksi määrättyjen käytäntöjen noudattamista [4, s. 10]. Tämän ajatusmallin toteutuksen ansiosta asentaminen suurimmalle osalle palvelinalustoista on pystytty tekemään todella helpoksi [10, s. 1]. Kehittäjän ei myöskään tarvitse syvällisesti tuntea kehyksen sisäistä toimintaa voidakseen käyttää sitä [11, s. 12].

Kehys hyödyntää Active Record -suunnittelumalliin perustuvaa ORM-menetelmää (Object-relational Mapping), jonka tarkoituksena on helpottaa tietokantojen kanssa työskentelyä. Menetelmä mahdollistaa tietokannan taulujen kuvaamisen olioina ja helpottaa niiden välisten suhteiden hallintaa. [4, s. 73.]

Useiden suunnittelumallien ohella kehys mahdollistaa komentorivin kautta ajettavien skriptien käytön, joiden avulla pyritään helpottamaan toistuvien tehtävien suorittamista ja sitä kautta nopeuttamaan sovelluksen kehittämistä. Näiden skriptien käytöstä on erityisesti hyötyä sovelluksen alkuvaiheessa, koska niiden avulla pystytään luomaan valmiita runkoja sovelluksen eri osille. [4, s. 10, 165.]

Kehykselle on tarjolla virallinen dokumentaatio sen käytöstä ja ohjelmistorajapinnasta, ja dokumentaation lisäksi kehyksestä on kirjoitettu myös useita kirjoja. Dokumentaation ohella kehykselle on aktiivinen yhteisö ja portaali, jossa käyttäjät voivat jakaa oppaita ja kehykselle tekemiään laajennuksia. Vaikka kehys ei itsessään ole kaupallinen sovellus, on sille tarjolla myös kaupallista tukea, konsultointia, sertifikointia ja koulutusta. Kaupallisista palveluista vastaa yhdysvaltalainen Cake Development Corporation, joka on kehyksen kaupallinen kumppani ja työllistää myös 75 % kehyksen pääkehittäjistä. [12; 13, s. 290–291.]

2.4 CodeIgniter – suorituskykyinen kehys ilman ylimääräisiä rajoituksia

CodeIgniter on yhdysvaltalaisen ohjelmistoyrityksen, EllisLabin kehittämä ohjelmistokehys, jonka pääpiirteisiin kuuluvat korkea suorituskyky ja yhteensopivuus [13, s. 289]. Kehyksen ydin koostuu vain muutamasta kirjastosta, ja lisäkirjastoja ladataan vain tarvittaessa, mikä tekee kehyksestä erittäin suorituskykyisen ja skaalautuvan. Suorituskyvyn ohella on kehys pyritty tekemään mahdollisimman yhteensopivaksi eri palvelinympäristöissä, mukaan lukien jaetut palvelimet. [14.]

Kehys ei vaadi mitään tiettyjä nimeämiskäytäntöjä eikä myöskään pakota seuraamaan ennalta määrättyjä sääntöjä koodia kirjoitettaessa. Tiukkojen käytäntöjen puuttuminen antaa kehittäjille enemmän vapauksia, mutta toisaalta tekee sovelluksen rakenteesta vähemmän yhtenäisen. Hyötyä vapaasta nimeämiskäytännöstä on erityisesti tilanteissa, joissa jo olemassa oleva sovellus halutaan toteuttaa kehyksen avulla. [11, s. 11–12.]

Koska kehyksestä on tehty kevyt, se sisältää myös rajoitetummin ominaisuuksia kuin muut esitellyt kehykset [15, s. 17]. Kehys sisältää muunnellun version Active Record -suunnittelumallista, jonka avulla voidaan helposti kirjoittaa tietokantakyselyitä, mutta ei varsinaista ORM-toteutusta [11, s. 80, 82; 15 s. 53–54; 16]. Mukana ei myöskään ole rajapintaa komentokehotteen kautta ajettaville skripteille.

Kehyksen kotisivuilla on kehittäjille tarjolla käyttäjäopas ja wiki-sivusto.

Ohjelmistorajapinnasta ei ole erillistä dokumentaatiota, vaan käyttäjä joutuu perehtymään kehyksen lähdekoodiin, joka tosin on kattavasti kommentoitu. Myös kehykseen liittyvä kirjallisuus rajoittuu ainoastaan muutamaan teokseen. Yhteisönä toimii kehyksen sivustolta löytyvä keskustelupalsta, jossa käyttäjät voivat jakaa ajatuksiaan kehyksestä ja kaikkea muuta siihen liittyvää.

2.5 Zend Framework – itsenäisiin komponentteihin pohjautuva kehys

Zend Technologies on israelilaisten Zeev Suraskin ja Andi Gutmansin vuonna 1999 perustama yritys, joka on erikoistunut erilaisiin ohjelmistotuotteisiin. Tuotteiden ohella yritys tarjoaa myös niihin liittyviä palveluita, kuten konsultointia, koulutusta ja sertifikointia. Kaupallisten tuotteiden ja palveluiden lisäksi yritys tarjoaa myös avoimeen lähdekoodiin perustuvia ratkaisuja, kuten Zend Framework -ohjelmistokehyksen. [17.]

Ensimmäinen versio Zend Frameworkista julkaistiin vuonna 2005. Kehys on täysin PHP5-pohjainen, eikä se ole taaksepäin yhteensopiva kielen vanhempien versioden kanssa [18, s. 4]. Tämän ansiosta kehys pystyy hyödyntämään kielen uudelleenkirjoitettua oliomallia, joka tuo mukanaan tehokkaamman olioiden käsittelyn ja muita olioihin liittyviä ominaisuuksia, kuten metodien ketjuttamisen [7, s. 6].

Kehys koostuu useista komponenteista, joista suurinta osaa voi käyttää itsenäisesti. Komponenttipohjaisen rakenteensa ansiosta kehys mukautuu hyvin erilaisiin sovelluksiin. Kehys ei ole tiukasti MVC-arkkitehtuuriin perustuva, vaan sitä voidaan käyttää myös monella muulla tapaa. Kehys sisältää kuitenkin MVC-toteutuksen, jonka avulla sovellus voidaan rakentaa sen arkkitehtuuria käyttäen. [18, s. 2.]

Monista muista ohjelmistokehyksistä poiketen Zend Framework ei pohjaudu vahvasti valmiiksi määriteltyihin käytäntöihin, vaan pyrkii tarjoamaan valmiita oletusasetuksia, joita voi muokata sovellukseen sopiviksi. Kehys ei kuitenkaan vaadi juuri ollenkaan

erillisten asetustiedostojen käyttämistä, vaan suurin osa asetuksista voidaan määrittää käyttämällä kehyksen tarjoamia komentoja. [18, s. 4.]

Active Recordin sijaan kehys sisältää toteutuksen Table Data Gateway -suunnittelumallista, joka on hyvin samankaltainen kuin Active Record. Tarvittaessa kehyksen yhteydessä voidaan käyttää ulkoista koodikirjastoa, joka sisältää ORM-toteutuksen. [20.]

Kehys tarjoaa esitellyistä laajimman dokumentaation, niin sähköisessä kuin painetussakin muodossa. Dokumentaation lisäksi kehyksen sivusto tarjoaa myös täydellisen ohjelmointivirheiden seuranta- ja tikettijärjestelmän [18, s. 5]. Zend Technologiesin tarjoamien palveluiden ja tuotteiden ansiosta Zend Framework on saavuttanut suuren suosion erityisesti laajojen yritysprojektien alustana.

2.6 Yhteenveto kehysten ominaisuuksista

Taulukkoon 1 on kuvattu edellä esiteltyjen kehysten pääominaisuuksia.

Taulukko 1. Kooste kehysten ominaisuuksista.

Kehys	**PHP4**	**PHP5**	**ORM**	**Testauskehys**	**Rajapinta skripteille**
CakePHP	x	-	x	SimpleTest	x
CodeIgniter	x	x	-	Kehyksen oma	-
Zend Framework	-	x	x	Kehyksen oma	x

Kehyksistä CakePHP on ainoa, joka toimii myös vanhemmalla PHP4-versiolla. Muut kehykset taas hyötyvät kielen uudemman version tuomista ominaisuuksista. Kaikkien kehysten avulla on mahdollista suorittaa ohjelmistotestausta, joko erillisen tai sisäänrakennetun testauskehyksen avulla. Muut kehykset CodeIgniteriä lukuun ottamatta sisältävät rajapinnan komentokehotteen kautta ajettaville skripteille.

2.7 Tärkeimmät kriteerit kehysvalintaa tehtäessä

Tärkein kriteeri ohjelmistokehystä valittaessa on löytää kehys, joka vastaa sovelluksen vaatimuksia ja tarjoaa tarvittavat ominaisuudet ja laajennettavuuden sovelluksen luomiseen. Kehyksen käyttämiin käytäntöihin ja niiden mukanaan tuomiin rajoituksiin tulee kiinnitää erityisesti huomiota, koska ne saattavat olla rajoittava tekijä sovelluksen kehityksessä. Laajennettavuuden kannalta tulee myös selvittää, mitä ulkoisia kirjastoja kehyksen yhteydessä on mahdollista käyttää.

Kehyksen käytön kannalta on hyvin tärkeää, että sille löytyy kattava ja ajan tasalla oleva dokumentaatio. Dokumentaation tulee olla selkeä, ja sen tulisi kattaa kaikki kehyksen käytöstä aina erikoistamisrajapintaan. Myös kehykseen liittyvään kirjallisuuteen kannattaa kiinnittää huomiota, koska kirjat ovat hyvä lähde virallisten dokumentaatioiden ohella. [21, s. 204.]

Suurin osa ohjelmistokehyksistä perustuu avoimeen lähdekoodiin, eikä niille ole välttämättä saatavilla virallisten tahojen kautta tukea. Virallisen tuen puuttuessa on turvauduttava kolmansien osapuolien tarjoamaan tukeen, jota yleensä löytyy yhteisöjen ja postituslistojen kautta. Kolmansien osapuolien tarjoama tuki ei välttämättä ole yhtä luotettavaa ja ajantasaista kuin kaupallinen tuki, eikä sitä myöskään saa välttämättä nopeasti. Kaupallinen tuki kuitenkin maksaa, eikä sille ole läheskään jokaisen sovelluksen tapauksessa tarvetta, joten ennen kehyksen valitsemista tulee perehtyä sille tarjolla olevaan tukeen ja sen tarpeeseen. [21, s. 205–206.]

3 Selkeä dokumentaatio ja helppokäyttöisyys kehysvalinnan perustana

3.1 Valitun kehyksen ominaisuudet ja vaatimukset

Luvussa 2 esitellyistä kehyksistä valittiin asiakasyrityksen sovelluksen toteuttamiseen käytettäväksi CakePHP. Sovelluksen toteutus olisi ollut mahdollista kaikilla esitellyistä kehyksistä, mutta valittu kehys tuntui helpoimmin lähestyttävältä selkeästi toteutetun dokumentaation ja myös siitä kirjoitetun kirjallisuuden perusteella. Myös kehyksen helppokäyttöisyys ja sen tarjoama kattava ORM-toteutus olivät merkitseviä kriteereitä valintaa tehtäessä. Muiltakin osin kehys tarjoaa kaikki tarvittavat ominaisuudet ja laajennettavuuden toteutettavan sovelluksen luomiseen.

Kehyksen toiminnan kannalta tärkein vaatimus on HTTP (Hypertext Transfer Protocol) -protokollaa tukeva palvelin, kuten Apache tai Microsoftin IIS. Palvelimelle tulee olla myös asennettuna kehyksen minimivaatimukset täyttävä versio sen käyttämästä ohjelmointikielestä. Työn kirjoitushetkellä uusin versio kehyksestä vaatii palvelimelta vähintään PHP4-tuen, mutta tulevaisuudessa tuki vanhalle versiolle hylätään ja kehyksen toiminta toteutetaan kielen uudemmalla versiolla [22]. Siirryttäessä uudempaan kielikantaan saadaan käyttöön sen mukanaan tuomat lisäominaisuudet, joista merkittävin on uudistettu oliomalli. [13, s. 9.]

Tietokannan käyttö kehyksen yhteydessä ei ole pakollista, mutta valtaosa verkossa toimivista sovelluksista on rakennettu tietokantaa hyödyntäen. Tuki löytyy yleisimmille tietokantamoottoreille, kuten MySQL, PostgreSQL, Oracle ja Microsoft SQL Server. Kun palvelin täyttää kehyksen vaatimukset, on asennus erittäin helppoa. Kehys on periaatteessa käyttövalmis heti, kun sovelluksen käyttämä tietokanta ja sen käyttäjätiedot on määritetty asetustiedostoon. Tietokannan yhteydessä kehys tarjoaa ORM-toteutuksen, johon perehdytään tarkemmin erillisessä sitä käsittelevässä luvussa. [13, s. 9, 16.]

Asetusten määrittäminen on saatu minimoitua tarkkaan määrättyjen nimeämiskäytäntöjen ja tiedostorakenteen avulla. Nimeämiskäytäntöjä noudatetaan useassa yhteydessä, ja ne perustuvat englannin kielen yksikkö- ja monikkomuotoihin [10, s. 6–7; 4, s. 10]. Esimerkiksi tietokannan users-taulua kuvaavan luokan tulee olla nimeltään ”User”, jolloin kehys osaa automaattisesti yhdistää luokan oikeaan tauluun. Vastaavan ohjaimen tulee olla nimeltään ”UsersController”, jolloin kehys osaa ladata käyttöön users-taulua kuvaavan User-luokan. Nimeämiskäytännöt ovat tiukat, ja niihin totuttelu vie oman aikansa, mutta ne on tarvittaessa mahdollista myös ohittaa. [13, s. 31–35.]

3.2 MVC-suunnittelumallin historia ja kehitysvaiheet

Alun perin MVC-suunnittelumalli kehitettiin SmallTalk-ohjelmointikielelle Xerox PARC:n tutkijoiden toimesta 1970-luvun loppupuolella. Alkuperäisessä kuvauksessa malli oli keskeisimmässä osassa ja sisälsi suurimman osan toiminnallisuudesta, eikä se ollut lainkaan tietoinen ohjaimesta ja näkymästä, jotka toimivat sen apuna. Tässä yhteydessä puhuttiin ”paksusta” mallista ja ”laihasta” ohjaimesta, koska malli sisälsi suurimman osan toiminnoista. Kuvauksessa näkymän tehtävänä oli tarkkailla mallin muutoksia, kun taas ohjaimen rooli oli hyvin minimaalinen ja se toimi pääasiassa vain viestinvälittäjänä mallille. [11, s. 7–9.]

Seuraava merkittävä askel kehityksessä oli NeXT-nimisen yrityksen käyttöjärjestelmä ja sen sovellukset. Yritys oli Bill Gatesin perustama, ja se vaikutti 1980- ja 1990-luvuilla, kunnes se myytiin Applelle. Yritys oli onnistunut kehittämään entistä tehokkaammat mallit ja ohjaimet ja se oli muuttanut ohjaimen entistä keskeisempään rooliin ja siirtänyt suurimman osan toiminnoista sille. Muutoksen myötä oltiin siirtymässä ”laihoista” ohjaimista ”paksuihin” ohjaimiin, koska suurin osa toiminnasta siirtyi ohjaimen vastuulle. Myöhemmin Sun julkaisi Model2-nimisen lähestymistavan, jossa ohjaimet olivat entistäkin keskeisemmässä roolissa ja hoitivat pyyntöjen ohjaamisen toiminnoille. [11, s. 7–9.]

Nykyään MVC-mallia käytetään suoraan tai muunneltuna useilla eri ohjelmointikielillä

toteutetuissa ohjelmistokehyksissä. Yksi tunnetuimmista on Ruby-ohjelmointikielellä toteutettu Ruby on Rails, jolta monet kehykset ovat lainanneet toiminnallisuutta. [13, s. 2.]

3.3 MVC-suunnittelumallin rakenne ja toiminta

MVC (Model-View-Controller) on ohjelmistotekniikan suunnittelumalli, jonka avulla sovelluksen koodi voidaan jakaa kolmeen erilliseen osaan: malliin, näkymään ja ohjaimeen [11, s. 5]. Jokainen osa vastaa tietystä sovelluksen osa-alueesta, mutta niiden tarkempi rooli voi vaihdella riippuen suunnittelumallin toteutuksesta [14, s. 8].

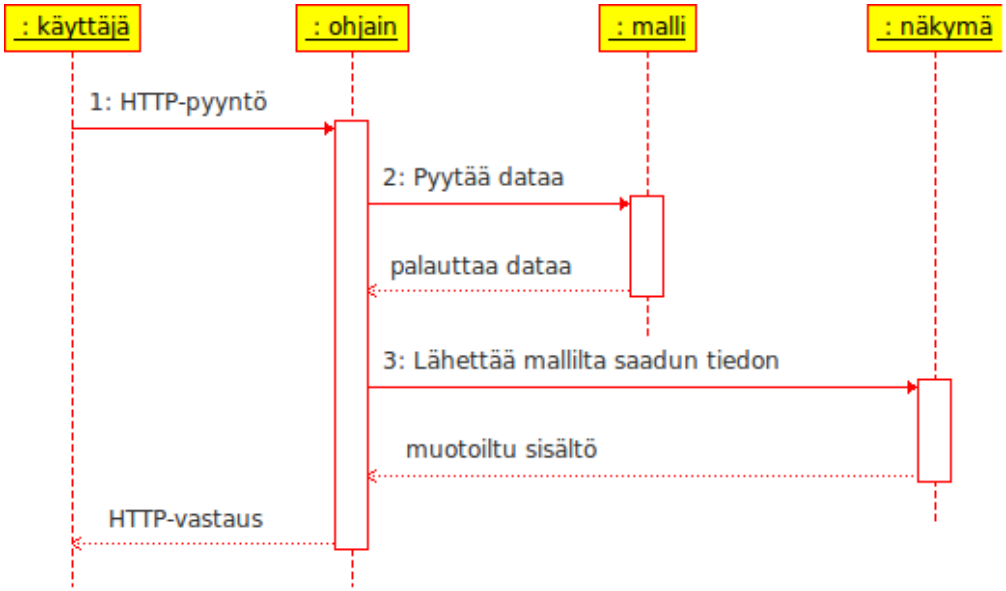
Ohjain on nimensä mukaisesti vastuussa sovellukselle tulevien pyyntöjen käsittelystä ja yleisesti sovelluksen työnkulusta. Kaikki sovellukselle tulleet pyynnöt kulkevat aina ohjaimen kautta, joka päättää jatkotoimenpiteistä. Kun käyttäjä tekee käyttöliittymän kautta pyynnön sovellukselle, kutsuttu ohjain määrää mallin tekemään tarvittavat toimenpiteet ja päättää, mikä näkymä käyttäjälle esitetään. On suositeltavaa pitää ohjaimet madollisimman ”laihoina” ja mallit ”paksuina”, jolloin ohjaimen koodi pidetään mahdollisimman yksinkertaisena ja tiedon hakuun liittyvä logiikka sijoitetaan mahdollisimman pitkälti mallin hoidettavaksi. [23, s. 284–285; 11, s. 5–6, 21, s. 203; 24.]

Malli vastaa sovelluksessa tiedon hakemisesta ja manipuloinnista. Ohjain ja näkymä ovat molemmat riippuvaisia mallista, koska ne voivat molemmat pyytää tietoa siltä. Mallin ei tule olla missään tekemisissä sovellukselle tulevien pyyntöjen tai käyttäjälle esitettävän sisällön kanssa, vaan se vastaa ainoastaan tiedon hausta ja siihen liittyvästä logiikasta. [23, s. 284–285, 11, s. 5–6.]

Näkymä sisältää sovelluksen ulkoasun ja käyttöliittymän sekä vastaa sovelluksen tiedon esittämisestä käyttäjälle. Yleensä näkymän yhteydessä käytetään kehyksen tarjoamaa tai erillistä mallinnejärjestelmää, jonka avulla pystytään erottamaan ohjelmakoodi esitettävästä sisällöstä. Esitettävän tiedon muokkaamiseen liittyvää logiikkaa tulisi sisällyttää näkymään mahdollisimman vähän jättämällä se pääasiallisesti ohjaimen

hoidettavaksi [21, s. 203–204].

Kuvan 1 sekvenssikaavio esittää MVC-mallin mukaisen työnkulun käyttäjän tekemästä pyynnöstä selaimelle palautettuun sivuun asti.



Kuva 1: Sekvenssikaavio MVC-mallin mukaisesta työkulusta (11, s. 5–6).

Ensin ohjain vastaanottaa käyttäjän tekemän pyynnön ja käsittelee sen. Pyynnön perusteella ohjain ilmoittaa mallille tarvittavien toimintojen suorittamisesta, minkä jälkeen ohjain lähettää mallin hakemat tiedot näkymälle. Tietojen perusteella näkymän sisältöä päivitetään tai luodaan kokonaan uusiksi. Ohjain jää odottamaan uutta pyyntöä käyttäjältä, jolloin sama prosessi käynnistyy taas alusta. [13, s. 17–18.]

Sovelluksen jakaminen kolmeen erilliseen osaan tarjoaa monia etuja niin rakenteen kuin laajennettavuudenkin kannalta. Mallin tarjoama rakenne pitää sovelluksen yhtenäisenä ja parantaa koodin luettavuutta, ja sen ansiosta sovellus pystytään rakentamaan modulaariseksi, mikä helpottaa laajentamista ja muutoksien tekemistä [4, s. 9]. Osajakoa pystytään hyödyntämään myös tiimityöskentelyssä, jolloin ohjelmoinnista vastaavat henkilöt voivat keskittyä työskentelemään ohjainten ja mallien kanssa, eikä heidän tarvitse miettiä, kuinka sovelluksen ulkoasu tulee rakentumaan. Sovelluksen ulkoasusta vastaavat henkilöt voivat rakentaa ulkoasun suoraan näkymiin, eikä heidän

tarvitse juurikaan tietää, kuinka sovelluksen logiikka toimii. [22, s. 201–203.]

3.4 Muokattavuutta laajennusten avulla

CakePHP:n toteutus sisältää muutamia laajennuksia alkuperäiseen MVC-malliin nähden. Laajennukset ovat luokkia, joiden avulla saadaan tuotua lisää laajennettavuutta ja lisättyä koodin uudelleenkäyttöä. Mallien, näkymien ja ohjainten ohelle kehys tarjoaa avustaja- ja komponentti-luokat. [25.]

Kehyksen ohjaimet periytyvät AppController-luokasta, johon voidaan sisällyttää kaikille ohjaimille yhteisiä muuttujia ja metodeja. AppController-luokka taas periytyy Controller-luokasta, joka kuuluu kehyksen ydinluokkiin [10, s. 18]. Ohjainten apuna voidaan käyttää komponentteja, jotka ovat luokkia, joiden tarkoituksena on jakaa ohjainten kesken yhteisiä toimintoja. [25; 10, s. 19; 13, s. 187.]

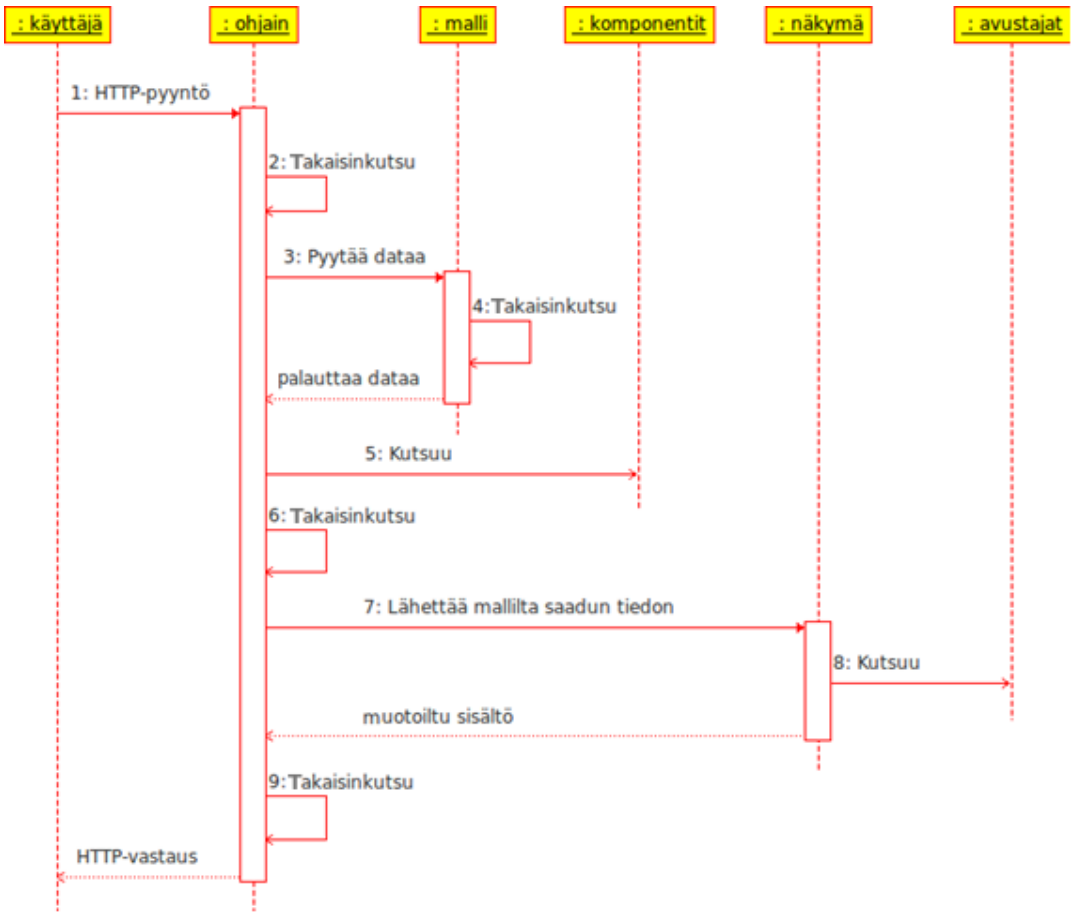
Komponentit eivät ole sidottuja tiettyyn ohjaimeen, vaan niiden tarkoituksena on tarjota yleishyödyllisiä metodeja, joita voidaan jakaa usean eri ohjaimen kesken [13, s. 187]. Käytettävät komponentit määritetään ohjaimeen, mutta tarvittaessa ne voidaan määrittää kantaluokkaan, jolloin ne ovat kaikkien ohjainten käytettävissä. Kaikkia ohjaimia ei kuitenkaan kannata ladata kantaluokan kautta, koska ne vaikuttavat sovelluksen muistinkäyttöön. Ohjainten ohessa voidaan käyttää myös takaisinkutsu-funktiota, jonka avulla voidaan suorittaa tiettyjä toimintoja kehyksen omien toimintojen välissä [13, s. 92; 25]. Esimerkiksi ennen jokaista ohjaimen sisältävää toimintoa voidaan suorittaa käyttäjän kirjautumistietojen tarkistus.

Avustajat ovat luokkia, joiden tarkoituksena on helpottaa näkymissä tarvittavien osien luomista [10, s. 21]. Määrittäminen tehdään komponenttien tapaan ohjaimeen tai niiden kantaluokkaan. Avustajien lisäksi näkymien avuksi on tarjolla myös sivupohjia ja elementtejä. Sivupohjat ovat eräänlaisia sisältökehyksiä, joiden sisään näkymien sisältö sijoitetaan. Sovellukseen voidaan luoda useita sivupohjia, mutta niiden pääasiallisena tarkoituksena on toimia ulkoasun runkona. Elementit ovat näkymien tapaisia sisällön osia, jotka halutaan esittää useassa paikassa. Kun elementti on kertaalleen luotu, se

voidaan helposti sijoittaa osaksi useaa sivupohjaa tai näkymää. Tarvittavat muutokset voidaan tehdä itse elementtiin, jolloin muutokset tulevat voimaan kaikkialla, minne elementti on sijoitettu. [4, s. 152.]

Mallien pohjana toimii AppModel-luokka, johon voidaan ohjainten tapaan sisällyttää niille kaikille yhteisiä metodeja ja muuttujia. Mallien ohessa voidaan tietokannan lisäksi käyttää myös muita tietolähteitä, kuten xml- tai csv-tiedostoja. [13, s. 243.]

Kuvan 2 kaavio kuvaa yksinkertaistettuna kehyksellä toteutetun sovelluksen kulkua, josta selviää laajennusten toiminta.



Kuva 2: CakePHP:n MVC-mallin mukainen työnkulku (26).

Aluksi ohjain käsittelee käyttäjän tekemän pyynnön ja ohjaa sen oikealle ohjaimen toiminnolle. Ennen toiminnon ajamista voidaan käyttää takaisinkutsu-funktioita esimerkiksi käyttäjän kirjautumistietojen tarkistamiseen. Seuraavaksi ohjain pyytää mallia suorittamaan tarvittavat toimenpiteet, joiden yhteydessä malli voi hyödyntää käyttäytymisiä ja tietolähteitä. Ennen mallin toimintoja ja niiden jälkeen voidaan jälleen käyttää takaisinkutsuja. Mallin palautettua tiedot ohjaimelle ohjain lähettää ne eteenpäin näkymälle, joka muodostaa niiden perusteella käyttäjälle esitettävän sivun. Lopuksi ohjain palauttaa valmiin sivun käyttäjän selaimelle. [25.]

3.5 Tietoturvan tiukentaminen laajennusten avulla

Tietoturvaa varten on tarjolla joukko asetuksia, jotka voidaan määrittää niitä kuvaavan luokan avulla. Tämän lisäksi tarjolla on erilliset komponentit, joiden avulla sovelluksen tietoturvaa voidaan määrittää tarkemmin ja tiukentaa entisestään.

Asetuksia kuvaavan luokan avulla määritetään sovellukseen liittyvät yleiset asetukset, mukaan lukien muutamia tietoturvan kannalta oleellisia asetuksia. Tärkeä osa koko sovelluksen tietoturvaa on salt-arvo, joka on merkkijono, jota käytetään salausmenetelmien avulla luotavien hajautusarvojen luomisessa. Oletuksena kehys käyttää SHA1-salausmenetelmää hajatusarvojen luomiseen, mutta haluttaessa se on mahdollista myös muuttaa [4, s. 236]. Hajautusarvoja käytetään monessa yhteydessä, kuten istuntojen ja salasanojen salaamisessa. Luokan kautta voidaan määrittää myös käyttäjän istunnon pituus ja turvallisuustaso. Nämä kaksi asetusta yhdessä määrittävät istunnon kokonaispituuden, joka kertoo, kuinka kauan tunnistautuneen käyttäjän istunto on voimassa. [4, s. 13–14, 196.]

Käyttämällä erillistä tietoturvakomponenttia voidaan sovelluksen tietoturvaa tiukentaa ja määrittää tarkemmin. Ottamalla komponentti käyttöön saadaan automaattisesti suoja CSRF (Cross-site request forgery) -hyökkäyksiä vastaan. Komponentti lisää kaikkiin html-avustajan avulla luotuihin lomakkeisiin ylimääräisen piilotetun kentän, jonka avulla varmistetaan, ettei lomakkeen kenttiä ole muokattu. Komponentin avulla

voidaan myös määrittää ohjaimen toiminnot, jotka vaativat käyttäjän tunnistautumisen, tai pakottaa tietyt toiminnot hyväksymään ainoastaan SSL-suojattuja pyyntöjä [13, s. 198].

Kehys sisältää myös erillisen komponentin, joka huolehtii sovelluksen käyttäjien autentikoinnista. Kirjautumistoimenpiteiden lisäksi komponentin avulla voidaan määrittää sovelluksen toiminnot, jotka vaativat kirjautumisen. Jos sovelluksella on tarvetta entistä laajempaan käyttäjien hallintaan, voidaan hyödyntää ACL (Access Control List) -komponenttia, joka mahdollistaa laajempien käyttäryhmien ja -oikeuksien hallinnan. [13, s. 189.]

Käyttäjien syöttämään tietoon ei tule koskaan sokeasti luottaa, ja sen takia tiedon validointi on tärkeä osa sovelluksen tietoturvaa. Kehyksessä validointisäännöt määritetään mallien yhteyteen. Ennen mallille syötetyn tiedon tallentamista malli vertaa tallennettavaa tietoa määritettyihin validointisääntöihin. Jokaiselle mallin kuvaamalle tietokannan sarakkeelle voidaan määrittää yksi tai useampia sääntöjä. Kehys sisältää useita valmiiksi määriteltyjä sääntöjä, ja niiden ohella voidaan käyttää myös säännöllisiä lausekkeita omien sääntöjen luomiseen. Validointi on myös mahdollista rajata toimimaan vain tiettyjen toimintojen yhteydessä. [10, s. 39–40; 10, s. 13–14.]

Validoinnin ohella kehys tarjoaa myös erillisen luokan, jota voidaan käyttää käyttäjien syöttämän tiedon suodattamiseen. Tarkoituksena on estää käyttäjää syöttämästä sovellukselle tietoa, joka ei vastaa sille määritettyihin rajoituksiin. Suodatuksen avulla voidaan estää sovelluksen XSS (Cross-site scripting) haavoittuvuudet poistamalla käyttäjien syötteistä ei-haluttu sisältö. [13, s. 204.]

4 Relaatiotietokannan tiedon kuvaaminen olioina

4.1 Siirrettävyyttä ja helppokäyttöisyyttä tietokantojen käsittelyyn

ORM (Object-Relational-Mapping) -menetelmä on ohjelmistotekniikassa käytetty tekniikka, jonka avulla pyritään ratkaisemaan relaatiotietokannan tiedon kuvaaminen olioina. Ongelmana oliopohjaisen tiedon tallentamisessa relaatiotietokantaan ovat erot olioiden ja tietokannan tietorakenteissa. Suurin osa käytetyimmistä tietokantojen hallintajärjestelmistä on relaatiopohjaisia, ja ne pystyvät säilömään ja käsittelemään ainoastaan skalaarisia arvoja. Olio-ohjelmoinnissa tietojen tallennus toteutetaan yleensä muokkaamalla olioita, jotka taas harvoin sisältävät ainoastaan skalaarisia arvoja. [26.]

Ongelma voidaan ratkaista joko käyttämällä vain tietokannan ymmärtämiä, skalaarisia arvoja tai muokkaamalla oliot tietojen tallennusta ja hakua varten yksinkertaisempaan muotoon. Toinen vaihtoehto on ottaa käyttöön oliopohjainen tietokannan hallintajärjestelmä, jonka avulla tietokantaan pystytään tallentamaan myös olioita. Oliopohjaiset hallintajärjestelmät eivät kuitenkaan ole yhtä laajasti käytettyjä kuin relaatiopohjaiset. Yhtenä syynä voidaan pitää relaatiopohjaisten hallintajärjestelmien tuomaa varmuutta verrattuna oliopohjaisiin. Suurten yritysten tuki ja ajan myötä karttunut osaaminen sekä relaatiopohjaisista järjestelmien tuntemus on taannut niille vahvan aseman. [27, s. 33; 37.]

ORM-toteutus piilottaa tiedonhakuun liittyvän logiikan ja vastaa SQL-kyselyiden luomisesta, jolloin sovelluksen kehittäjä säästyy kyselyiden kirjoittamiselta. Samalla koodin luettavuus paranee, koska sen sekaan ei tarvitse enää kirjoittaa pitkiä kyselyitä. Myös virheet vähenevät, koska kyselyt luodaan metodeille syötettyjen parametrien perusteella. [28, s. 26–29.]

Tiedonhakuun liittyvän logiikan piilottaminen lisää myös sovelluksen siirrettävyyttä helpottamalla sovelluksen siirtoa eri tietokantamoottorien välillä. Kunkin moottorin käyttämä SQL-kyselykieli poikkeaa yleensä yleisestä SQL-standardista, jolloin siirtyminen toiseen tietokantajärjestelmään saattaisi edellyttää monien kyselyiden

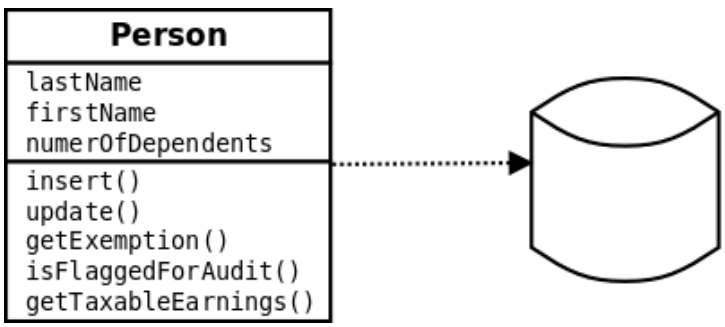
manuaalista muokkaamista. Yleensä ORM-toteutus sisältää tuen usealle eri tietokantamoottorille, jolloin kyselyiden luominen hoituu automaattisesti määritetyn moottorin mukaiseksi. [28, s. 29–30.]

Koska muunnoksen toteuttaminen vaatii ylimääräisiä luokkia ja niistä luotavia olioita, sillä on vaikutuksensa myös sovelluksen suorituskykyyn. Oliopohjaisen tietokantamoottorin käyttö olisi suorituskyvyllisesti tehokkaampi ratkaisu, koska tietotyyppien välillä joudutaan suorittamaan ylimääräinen muunnos. Suorituskyky-menetykset eivät kuitenkaan ole kriittisen suuria, ja saavutettavat hyödyt ovat merkittävästi haittoja suuremmat. [28, s. 23.]

4.2 Suunnittelumallit toteutusten perustana

Active Record

ORM-toteutuksen ohessa hyödynnetään ohjelmistotekniikan suunnittelumalleja. Active Record on yksi yksinkertaisimmista ratkaisuista tietokannan yhteyden abstraktointiin [23, s. 247]. Mallin mukaan olio sisältää sekä tiedon että sen hakuun liittyvän logiikan ja yksi ilmentymä kuvaa aina yhtä tietokannan riviä. Toteutus on yksinkertainen, mutta se sitoo koodin ja tietokantarakenteen tiukasti toisiinsa, koska luokan tulee tarkasti vastata sen pohjana toimivan tietokannan taulun rakennetta. [23, s. 227–228, 244.] Kuvan 3 kaavio esittää esimerkin avulla Active Record -mallin toiminnan.

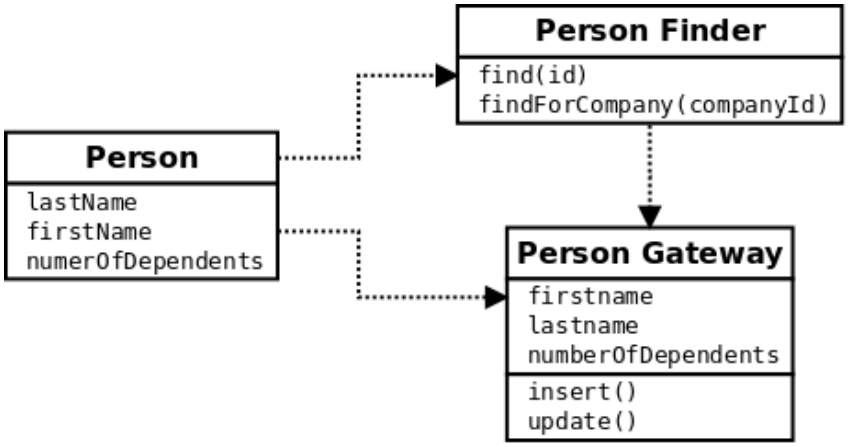


Kuva 3: Active Record -suunnittelumalli (28, s. 160).

Person-luokka sisältää tietokannan kenttiä kuvaavat attribuutit ja niiden lisäksi myös tietokanta- ja liiketoimintalogiikan. Yksi olion ilmentymä vastaa aina yhtä tietokannan riviä, ja se osaa suoraan toimia tietokannan kanssa. [27, s. 35, 160–161.]

Row Data Gateway

Row Data Gateway on hyvin samankaltainen kuin Active Record, joten usein niitä on vaikea erottaa toisistaan [27, s. 160]. Ero on huomattavissa liiketoimintalogiikan perusteella, jota Gateway-olion ei tule sisältää. Active Recordin tapaan Row Data Gateway -mallissa olio kuvaa yhtä tietokannan riviä. Tietokantaan liittyvää logiikkaa varten on hyvä ottaa käyttöön erillinen Finder-olio, joka vastaa hakujen suorittamisesta. [27, s. 152.] Kuvan 4 kaavio kuvaa Row Data Gateway -mallin toimintaa.



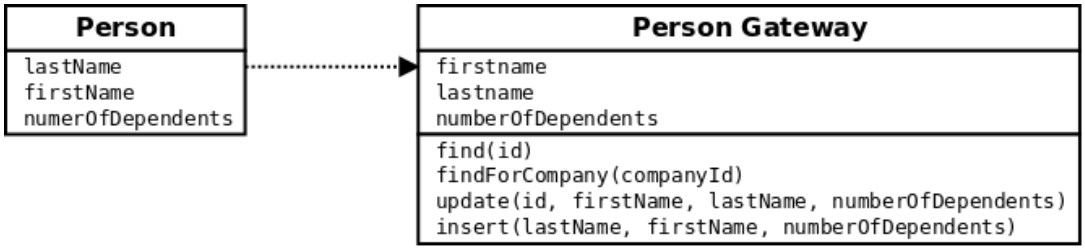
Kuva 4: Row Data Gateway -suunnittelumalli (28, s. 152).

Person-luokka sisältää ainoastaan tietokannan kenttiä kuvaavat attribuutit. Person-olio luo Finder-luokasta ilmentymän, joka suorittaa haut tietokantaan. Hakujen perusteella Finder-olio luo tietokannan rivejä kuvaavat Gateway-oliot. [27, s. 152–153.]

Table Data Gateway

Table Data Gatewayn etuna verrattuna edellä esiteltyihin malleihin on se, että mallissa olio kuvaa yhden rivin sijaan kokonaista tietokannan taulua. Olio toimii yhdyskäytänä käytettävään tietokantaan ja sisältää tietokantalogiikan lisäksi myös

liiketoimintalogiikan. Table Data Gateway hoitaa kaikki SQL-kyselyitä luovat metodit, joita kutsutaan. Yleensä jokaista tietokannan taulua varten on oma Gatewaynsa, tosin yksinkertaisissa tapauksissa voidaan käyttää kaikille tauluille yhteistä Gatewayta. [27, s. 144]. Kuvan 5 kaavio kuvaa Table Data Gateway -mallin toimintaa.

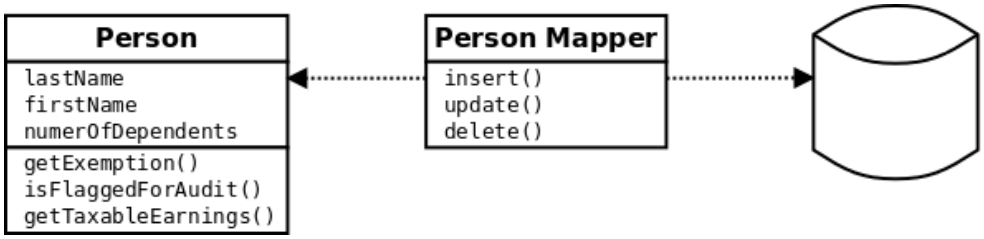


Kuva 5: Table Row Gateway -suunnittelumalli (28, s. 144).

Gateway-luokka sisältää tietokannan kenttiä kuvaavat attribuutit ja tietokanta- ja liiketoimintalogiikan. Person-luokan kautta kutsutaan Gateway-luokkaa suorittamaan tarvittavat haut. [27, s. 144.]

Data Mapper

Data Mapper -mallin toiminta perustuu Mapper-suunnittelumalliin, joka yhdistää tietokannan ja olion toisiinsa ja mahdollistaa myös tiedonsiirron niiden kesken. Toimintaperiaatteena on yhdistää kaksi erillistä resurssia toisiinsa mutta pitää ne kuitenkin itsenäisinä [5, s. 278–279]. Mapper toimii eräänlaisena muuntajana olion ja tietokannan välillä: se muuntaa olion attribuutit ja metodit tietokannan sarakkeiksi ja hoitaa muunnoksen myös toiseen suuntaan [23, s. 262]. Toteutukseltaan Data Mapper on monimutkaisin esitellyistä suunnnittelumalleista, mutta se myös eristää liiketoimintalogiikan ja tietokannan täysin toisistaan [27, s. 36]. Tämän ansiosta olio ja tietokanta eivät ole tiukasti sidoksissa toisiinsa, jolloin muutosten tekeminen jompaan kumpaan ei aiheuta muutostoimenpiteitä toiseen. [23, s. 281]. Kuvan 6 kaavio kuvaa Data Mapper -mallin toimintaa.



Kuva 6: Data Mapper -suunnittelumalli (28, s. 165).

Esimerkkinä toimivan Person-luokan ja tietokannan välissä toimii Person Mapper -luokka, joka vastaa tiedon muuntamisesta edellä mainittujen välillä. Mapper sisältää yleiset metodit tietokannan kanssa työskentelyyn, mutta niihin liittyvä logiikka on Person-luokassa. [27, s. 165–166.]

4.3 CakePHP ja mallien välisten suhteiden hallinta

Monet ohjelmistokehykset sisältävät jonkinlaisen ORM-toteutuksen, samoin myös CakePHP. Kehysten tarjoamien ratkaisujen lisäksi on myös erillisiä kirjastoja saman asian toteuttamiseen. CakePHP:n sisältämä toteutus perustuu Active Record -suunnittelumalliin, ja sen avulla pystytään helposti hallinnoimaan tietokannan taulujen välisiä suhteita ja helpotetaan SQL-kyselyiden luomista.

Jokainen Model-luokasta periytyvä luokka kuvaa yhtä tietokannan taulua, ja kaikki tauluihin tehtävät muutokset tehdään niitä vastaavan luokan kautta. Model-luokka sisältää joukon valmiita metodeja tiedon manipulointia varten. Valmiit metodit tarjoavat laajalti parametrejä, joiden avulla niiden luomia hakuja voidaan kustomoida, joten erillisten SQL-hakujen kirjoitus on tarpeen vain hyvin harvoissa tapauksissa.

Mallien väliset suhteet ja hauissa käytetty rekursio määritetään niitä kuvaaviin luokkiin. Mallien välille voidaan määrittää yksi-yhteen-, yksi-moneen- ja monta-moneen-tyyppisiä yhteyksiä. Sovelluksen ohessa toimivan tietokannan tulee sisältää määritettyjä yhteyksiä vastaava rakenne. [4, s. 119–122.]

Rekursio määrittää, kuinka syvälle yhteydessä oleviin malleihin suoritettu haku ulottuu.

Haun syvyyttä muokkaamalla pystytään yleisellä tasolla rajaamaan pois yhteydessä olevien mallien tieto, jota ei haluta sisällyttää hakuun [13, s. 93]. Rekursion avulla ei kuitenkaan pystytä tarkasti rajaamaan, mitä tiettyjä malleja hakuun sisällytetään. Tarkempien rajausten tekeminen onnistuu käyttämällä erillistä laajennusta mallin yhteydessä. Laajennuksen toiminta perustuu mallien välisiin yhteyksiin tehtäviin väliaikaisiin tai pysyviin muutoksiin, ja sen avulla pystytään tarkkaan määrittämään, mitkä mallit hakuun sisällytetään. Sen avulla pystytään myös rajaamaan mallikohtaisesti tietokannasta haettavat kentät. Hakujen rajaamisella voidaan parantaa sovelluksen suorituskykyä, erityisesti jos käytössä on laaja tietokantarakenne, joka sisältää monimutkaisia suhteita taulujen välillä. [4, s. 141–143; 13, s. 265–267.]

Kun mallien väliset suhteet on määritetty, kehys osaa automaattisesti ladata kaikki muut yhteydessä olevat mallit kyseisen luokan käytettäväksi. Tarvittaessa on myös mahdollista erikseen ladata ei-yhteydessä olevia malleja käytettäväksi.

Kuvan 7 yksinkertaistettu koodiesimerkki esittelee ORM-toteutuksen käyttöä ja havainnollistaa sen tuomia etuja kooditasolla. Esimerkkinä on tietokannan orders-taulua kuvaava Order-luokka, joka on yhteydessä Customer- ja Orderline-luokkiin.

```php
<?php

class Order extends AppModel
{
    // belongsTo- ja hasMany-muuttujat määrittävät suhteet Customer- ja Orderline-luokkiin (malleihin)
    var $belongsTo = array('Customer');
    var $hasMany = array('Orderline');


    // Metodi joka ei hyödynnä ORM:a.
    // Mallien välisiä suhteita ei voida hyödyntää, vaan taulujen liitokset luodaan SQL-kyselyyn
    function queryWithoutOrm() {
        $query = "SELECT".
                 "Order.id,".
                 "Order.created,".
                 "Order.customer_id,".
                 "Customer.id,".
                 "Customer.firstname,".
                 "Customer.lastname,".
                 "Orderline.id,".
                 "Orderline.order_id".
                 "FROM orders AS Order".
                 "LEFT JOIN customers AS Customer".
                 "ON (Order.customer_id = Customer.id)".
                 "LEFT JOIN orderlines AS Orderline".
                 "ON (Orderline.order_id = Order.id)";
        return $this->query($query);
    }

    // Saman haun toteuttava metodi, joka on hyödyntää ORM:a
    function queryWithOrm() {
        return $this->find('all');
    }
}
```

Kuva 7: Koodiesimerkki CakePHP:n ORM-toteutuksen käytöstä.

Luokka sisältää kaksi metodia, jotka molemmat luovat saman SQL-kyselyn, joka hakee kaikki tilaukset ja niihin liittyvät tilausrivit ja -asiakkaat. Ensimmäinen metodi on toteutettu kirjoittamalla suoraan SQL-kysely, jälkimmäinen taas on luotu hyödyntämällä ORM-toteutusta. Esimerkin perusteella voidaan havaita, kuinka monimutkainen kysely voidaan suorittaa hyvin lyhyellä määrällä koodia.

4.4 Menetelmän tarjoamat hyödyt

ORM-menetelmä helpottaa suuresti erityisesti monimutkaisten suhteiden hallitsemista ja niiden välisen tiedon hakua. Active Record -totetuksen tarjoamat metodit helpottavat tiedon hakua, koska suurin osa hauista voidaan suorittaa suoraan niiden avulla, jolloin säästytään erillisten SQL-lauseiden kirjoittamiselta. Metodien syntaksi on myös paljon luettavampaa kuin monimutkaiset SQL-lauseet, ja ne tarjoavat parametrien avulla erittäin laajan muokattavuuden. [4, s. 79–81.]

Kun käytettään Model-luokan tarjoamia metodeja ja noudatetaan niiden käyttöä, kehys luo tietokantakyselyt automaattisesti tarvittavan syntaksin mukaisiksi. Jos kyselyitä ei luoda oikean syntaksin mukaisesti, voi sovelluksen käyttäjä onnistua suorittamaan SQL-injektion, jolla tarkoitetaan käyttäjän SQL-kyselyyn syöttämää ylimääräistä sisältöä. Kehys noudattaa kaikessa tietokantahauista palauttamassaan tiedossa myös tiettyä tietorakennetta. Kehyksen tarjoamia metodeja käyttämällä varmistetaan, että hakujen kautta saatu tieto on aina kyseisen rakenteen mukaisessa muodossa.

ORM-toteutus on erittäin hyödyllinen laajemmassa sovelluksessa, joka hyödyntää tietokantaa. Toteutus ei rajoita kyselyiden luomista, koska sen käyttämät hakumetodit ovat laajalti muokattavissa. Käyttö on myös helppoa, kun syntaksin on kertaalleen oppinut, joten käyttöönotto suppeankin sovelluksen ohessa on helppoutensa vuoksi suositeltavaa.

5 Sovelluksen laadun parantaminen ohjelmistotestauksen avulla

5.1 Testitapausten suunnittelu

Ohjelmistotestaus on käsitteenä todella laaja, ja tässä työssä keskitytään sen osalta käytetyn kehyksen sisältämiin ominaisuuksiin eli yksikkö- ja web-testaukseen sekä niihin liittyvään teoriaan. Ohjelmistotestauksen tarkoituksena on pyrkiä löytämään sovelluksessa esiintyviä virheitä erilaisten testien avulla. Testaus voidaan karkeasti jakaa lähestymistavan perusteella kahteen eri suuntaukseen, white- ja black-box-testaukseen. [29, s. 9.]

Black-box testauksessa testaaja tarkastelee sovellusta eräänlaisena mustana laatikkona, eli ei tiedä mitään sen sisäisestä toiminnasta. Tämä testausmenetelmä perustuu pitkälti sovelluksen määrityksiin, ja tavoitteena on löytää ohjelmasta kohdat, jotka eivät toimi niiden mukaisesti. Koska ohjelman sisäisestä toiminnasta ei ole tietoa, testidata luodaan määritysten perusteella. [29, s. 9–11.]

Toisessa pääsuuntauksessa, white-box-testauksessa, testaajalla on tiedossa sovelluksen siäinen rakenne ja lähdekoodi. Näiden tietojen perusteella voidaan luoda tarkemmin sovelluksen toimintaa tutkivia testejä, mutta tämä suuntaus vaatii testaajalta tarkempaa tuntemusta sovelluksesta. [29, s. 11–14.]

Ohjelmistotestauksessa testitapaukset määrittelevät, mitä testataan ja millä menetelmillä. Testitapausten huolellinen suunnittelu on erittäin tärkeä osa testausprosessia, koska tehokas ja toimiva testaus perustuu hyvin suunniteltuihin testitapauksiin. Optimaalista olisi testata kaikki mahdolliset yhdistelmät, joita sovellus voi sisältää. Käytännössä erilaisia yhdistelmiä ja tapauksia on kuitenkin niin paljon, ettei niiden testaaminen ole ajallisesti eikä myöskään taloudellisesti järkevää. Tämän takia testitapauksia suunniteltaessa pyritään valitsemaan se osajoukko testausmenetelmistä, jonka avulla pystytään löytämään eniten virheitä. [29, s. 5, 43.]

5.2 Testien kohdentaminen sovelluksen pienempiin osiin

Yksikkötestauksessa sovellus jaetaan pienimpiin mahdollisiin osiin, ja niitä kutsutaan yksiköiksi. Testitapausten luomiseen tarvitaan kaksi asiaa, yksikön lähdekoodi ja sen määrittely. Määrittely sisältää yleensä kuvauksen yksikölle syötettävistä parametreistä ja palautusarvoista. Yksikkötestauksessa tulee ottaa huomioon kaksi asiaa: tehokkaiden testitapausten luominen ja se, kuinka yksittäiset moduulit muodostavat kokonaisen sovelluksen. [10, s. 213–214.]

Yksikkötestaukseen on vahvasti sidoksissa integraatiotestaus, jolla tutkitaan komponenttien ja niiden välisten rajapintojen toimintaa. Yksikkötestaus helpottaa sovelluksessa esiintyvien virheiden paikannusta, koska testeissä käsitellään vain pientä osaa sovellusta, jolloin virheen sijainti on helppo rajata tiettyyn osaan. On myös mahdollista testata useaa yksikköä samanaikaisesti. [29, s. 91.]

5.3 Käyttäjän toimien simulointi web-testauksen avulla

Web-testauksella voidaan laajempana terminä tarkoittaa kaikkea web-sovelluksen testaukseen liittyvää, kuten sovelluksen toimivuutta eri selainversioilla. Tässä yhteydessä termiä käytetään käyttäjän suorittamien toimintojen simuloimana toimintana. [10, s. 232.]

Monet web-sovellukset sisältävät erilaisia käyttäjän suorittamia toimintoja, kuten navigointia linkkien avulla tai erilaisten lomakkeiden täyttämistä. Näiden toimintojen toistuva manuaalinen testaaminen sovelluksen kehityksen aikana on hidasta, ja sen vuoksi prosessi kannattaa automatisoida mahdollisimman pitkälle. Kun yksikkötestaus keskittyy testaamaan pieniä osia sovelluksesta, pyritään web-testauksella sovelluksen toiminnan laajempimittaiseen testaukseen simuloiden käyttäjän tekemiä toimintoja. [10, s. 224.]

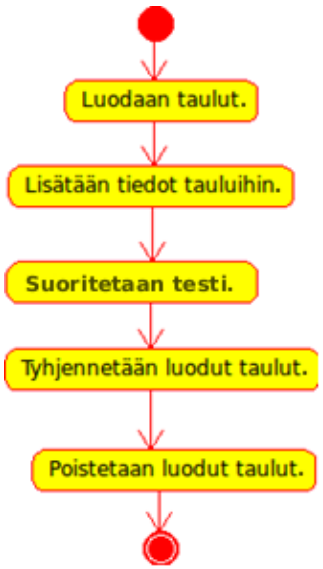
Kun halutaan testata esimerkiksi sovelluksessa olevan lomakkeen toimivuutta, sitä varten voidaan luoda testitapaus, joka syöttää halutut tiedot lomakkeeseen. Testi voidaan ajaa tarvittaessa helposti uudestaan, eikä lomaketta tarvitse enää manuaalisesti käydä täyttämässä joka kerta. Testitapauksen ei tarvitse rajoittua vain lomakkeen täyttämiseen, vaan siihen voidaan lisätä myös muita käyttäjän toimintaa mukailevia toimintoja.

5.4 Ohjelmistotestaus erillisen testauskehyksen avulla

Versiosta 1.2 lähtien CakePHP sisältää tuen ohjelmistotestausta varten. Se on toteutettu avoimen lähdekoodin SimpleTest-testauskehyksen avulla [10, s. 222]. Itse testauskehys ei sisälly kehyksen asennuspakettiin, vaan se tulee asentaa erikseen. Asennus on kuitenkin tehty erittäin helpoksi: se vaatii ainoastaan yhden arkiston purkamisen, minkä jälkeen kehys on käyttövalmis. Kehys mahdollistaa sekä yksikkö- että web-testien ajamisen.

Testejä varten voidaan määrittää erillinen tietokantayhteys, jolloin ne voidaan suorittaa

käyttäen erillistä tietokantaa ja testidata saadaan pidettyä erillään sovelluksen datasta. Kehyksen yhteydessä on mahdollista hyödyntää ominaisuutta, jonka avulla voidaan luoda väliaikaisia tauluja tietokantaan testidatan tallentamista varten. Testidata voidaan kopioida varsinaisesta tietokannasta, tai sitä varten voidaan luoda erillistä testidataa. [10, s. 226–227] Kuvan 8 kaavio kuvaa tätä ominaisuutta hyödyntävän testin etenemistä alusta loppuun.



Kuva 8: Testidatan kopioiminen sovelluksen varsinaisesta tietokannasta.

Ensimmäiseksi testitietokantaan luodaan testiin määritetyt tietokannan mukaiset taulut ja tallennetaan määritetty data niihin joko varsinaisen tietokannan tiedoista tai määritetyn testidatan perusteella. Seuraavaksi ajetaan itse testi, ja lopuksi tyhjennetään testidataa sisältävät taulut ja poistetaan ne testitietokannasta.

Käytännössä testit toteutetaan luomalla testattavaa osiota varten oma luokka, johon lisätään kaikki siihen liittyvät testitapaukset metodeina. Testien suorittaminen tapahtuu selaimella tai kehyksen tarjoaman käyttöliittymän tai vaihtoehtoisesti komentokehotteen kautta. Testejä voidaan ajaa joko yksittäin tai niistä voidaan luoda ryhmiä, joihin voidaan sisällyttää useita testejä.

Tulevaisuudessa kehyksen testaus hyödyntää PHPUnit-testauskehystä ja SimpleTestin

käytöstä luovutaan. Hyötynä on laajempi käyttäjäkunta ja tuntemus, sillä PHPUnit on kehittynyt oman alansa standardiksi käyttöalueellaan. Myös suuri osa käytetyimmistä samankaltaisista kehyksistä hyödyntää PHPUnitia.

6 Olympiaklubi-asiakasportaali

6.1 Portaali urheiluaiheisista keräilytuotteista kiinnostuneille

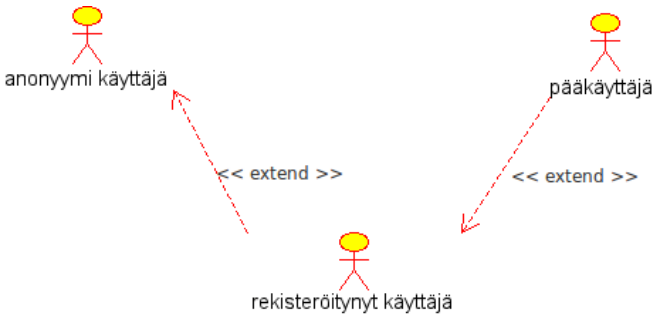
Insinöörityön osana luotiin asiakasyrityksenä toimineelle Suomen Monetalle asiakasportaali, joka päädyttiin toteuttamaan CakePHP-ohjelmistokehyksellä. Vaihtoehtona olisi ollut toteuttaa portaali jonkin sisällönhallintajärjestelmän, kuten Drupalin tai Joomlan, avulla. Valmiit sisällönhallintajärjestelmät eivät kuitenkaan tarjoa yhtä laajaa muokattavuutta, ja ne pääsevät oikeuksiinsa vasta laajemmissa toteutuksissa.

Olympiaklubi-portaalin tarkoituksena on tarjota portaali, joka on kohdennettu urheiluaiheisista keräilytuotteista kiinnostuneille asiakkaille. Tarkoituksena on myydä tuoteryhmään kuuluvia tuotteita ja tarjota niihin liittyvää sisältöä. Portaali on kohdennettu pääasiassa yrityksen jo olemassa oleville asiakkaille, mutta asiakashankintaa hoidetaan myös digitaalisen suoramarkkinoinnin kautta.

Asiakasyritys tekee yhteistyötä usean suomalaisen urheilijan kanssa, ja myös tietoa heistä ja heidän kuulumisistaan on tarjolla portaalin kautta. Käyttäjille on myös järjestetty eri urheilutapahtumien yhteydessä niihin liittyviä kilpailuja.

6.2 Portaalin käyttötapausmalli ja käyttäjien roolit

Käyttötapausmalli kuvaa portaalin käyttäjäryhmiä ja niiden käyttötapauksia, jotka ovat toimenpiteitä, joita käyttäjät voivat järjestelmän kautta suorittaa. Käyttäjät voidaan jakaa kuvan 9 mallin mukaisesti kolmeen eri ryhmään: anonyymeihin käyttäjiin, tunnistautuneisiin käyttäjiin ja pääkäyttäjiin.



Kuva 9: Portaalin käyttäjäryhmiä ja niiden välisiä suhteita kuvaava käyttötapausmalli.

Anonyymit käyttäjät eivät ole tunnistautuneet portaaliin, ja toiminnot ovat heidän osaltaan hyvin rajatut, kunnes he tunnistautuvat ja heistä tulee rekisteröityneitä käyttäjiä. Kaikkien käyttäjien yläpuolella on pääkäyttäjä, jolla on kaikki muiden toiminnot ja joukko vain heille määritettyjä toimintoja. Kuvan 10 käyttötapausmalli kuvaa tarkemmin toimintoja, joita kunkin käyttäjäroolin on mahdollista suorittaa.

Kuva 10: Portaalin toimintoja kuvaava käyttötapausmalli.

Anonyymit käyttäjät eivät ole tunnistautuneet palveluun, mikä rajaa käyttötapaukset julkisen sisällön selaamiseen ja käyttäjätilin luomiseen. Käyttäjätilin luotuaan voi anonyymi käyttäjä kirjautua palveluun ja rooli laajenee tunnistautuneeksi käyttäjäksi.

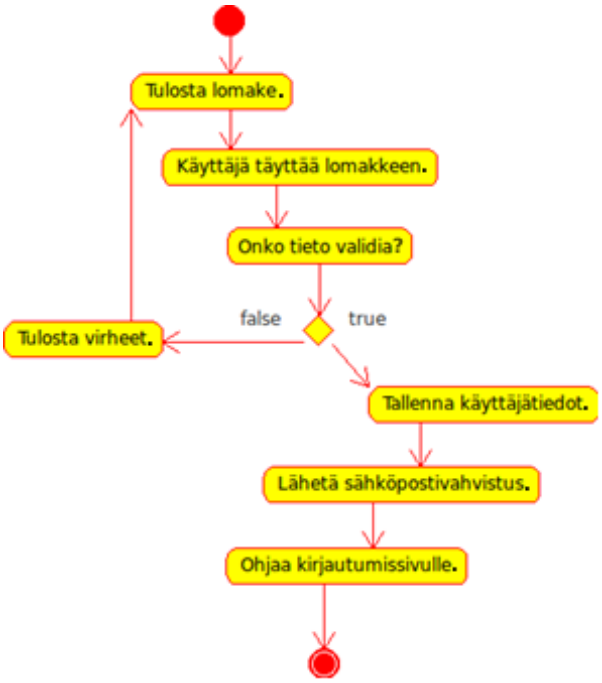
Roolin myötä on mahdollista listata palveluun lisättyjä tuotteita sekä ja niistä tilauksia. Kun käyttäjä haluaa poistua palvelusta, on mahdollista kirjautua ulos, jolloin rooli muuttuu jälleen anonyymiksi käyttäjäksi.

Kaikkien käyttäjäroolien ylintä tasoa kuvaa pääkäyttäjä, joka laajentaa sekä anonyymin että tunnistautuneen käyttäjän toimintoja. Pääkäyttäjänä on mahdollista hallita palvelun sisältöä, tuotteita ja tilauksia. Pääkäyttäjän kirjautuessa ulos rooli palautuu anonyymiksi käyttäjäksi.

6.3 Tarkennettu kuvaus portaalin toiminnoista

Rekisteröityminen

Anonyymin käyttäjän on rekisteröidyttävä palveluun voidakseen käyttää sitä. Rekisteröityminen tehdään täyttämällä vaaditut tiedot etusivulla olevaan rekisteröitymislomakkeeseen. Rekisteröitymisprosessi etenee kuvan 11 aktiviteettikaavion mukaisesti.

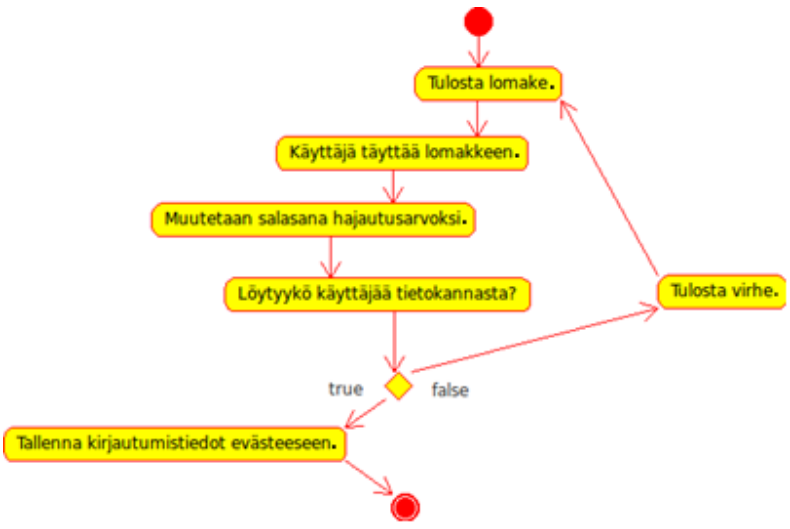


Kuva 11: Rekisteröitymisprosessia kuvaava aktiviteettikaavio.

Kun käyttäjä on syöttänyt tietonsa, ne validoidaan määritettyjen sääntöjen mukaisesti. Jos tiedot eivät läpäise validointia, esitetään rekisteröitymislomake uudelleen ja ilmoitetaan virheellisistä kentistä. Kun validointi onnistuu, muutetaan käyttäjän valitsema salasana hajautusarvoksi ja tallennetaan se muiden tietojen ohella tietokantaan. Salasana muutetaan tietoturvasyistä hajautusarvoksi, jotta se ei olisi selkokielisenä luettavissa tietokannasta. Onnistuneen rekisteröitymisen päätteeksi tulostetaan käyttäjälle kirjautumistiedot ja lähetetään samat tiedot sähköpostitse vahvistuksena. Saatuaan tunnukset käyttäjä voi siirtyä kirjautumaan sisään palveluun.

Käyttäjien istuntojen hallinta evästeiden avulla

Käyttäjän kirjautuminen pohjautuu evästeisiin, jotka ovat käyttäjän koneelle luotavia tekstitiedostoja, joihin voidaan tallentaa tietoa. Kaikki evästeisiin tallennettava tieto salataan hajautusarvojen avulla, joten niitä ei pääse selkokielisenä lukemaan. Salaus on tärkeää myös sen vuoksi, ettei käyttäjä itse pääse muokkaamaan evästeiden sisältöä. Itse kirjautumisprosessi etenee kuvan 12 kaavion mukaisesti.



Kuva 12: Kirjautumisprosessia kuvaava aktiviteettikaavio.

Käyttäjälle tulostetaan lomake, johon syötetään rekisteröidyttäessä luotu käyttäjänimi ja

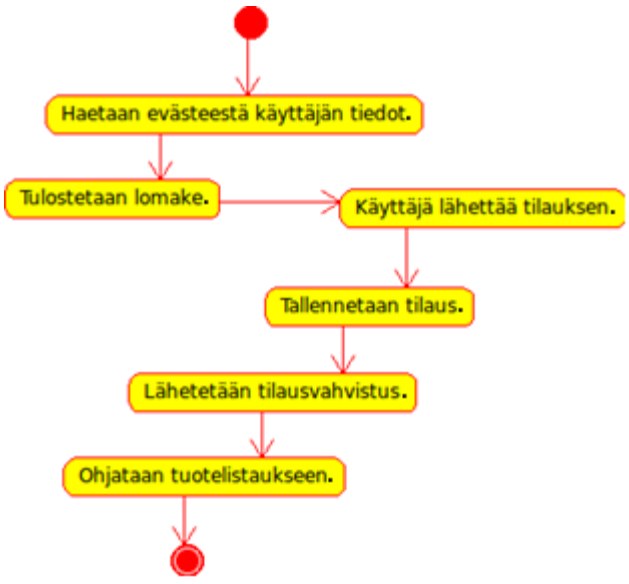
salasana, joista jälkimmäinen muutetaan hajautusarvoksi. Näiden tietojen perusteella tehdään tietokantaan haku, ja kun se tuottaa tuloksen, käyttäjä tunnistetaan ja istunnon tiedot tallennetaan evästeeseen.

Käyttäjän istunto on aktiivinen, kunnes käyttäjä kirjautuu ulos tai istunnolle määritetty enimmäisaika täyttyy. Enimmäisaika on määritetty, jotta kirjautuminen ei jäisi aktiiviseksi esimimerkiksi julkista tietokonetta käytettäessä. Molemmissa tapauksessa istunto päätetään tuhoamalla siihen liittyvä eväste.

Tuotteiden listaus ja tilauksen tekeminen

Tuotelistauksessa tunnistautunut käyttäjä näkee listan kaikista aktiiviseksi asetetuista tuotteista ja tiivistetyt tiedot niistä (liite 1). Tuotekuvista käytetään useita erikokoisia versioita, riippuen käyttötarkoituksesta. Kuvien esittäminen on hoidettu dynaamisen skaalauksen avulla, jolla tarkoitetaan erikokoisten kuvien luomista yhden kuvan perusteella. Skaalauksen ansiosta tuotteista ei tarvitse olla palvelimella kuin yksi kuva, josta luodaan kaikki tarvittavat kuvakoot.

Tuotelistauksen kautta käyttäjä voi siirtyä erilliselle tuotesivulle, jossa esitetään tuotteen tarkemmat tiedot. Tältä sivulta käyttäjä voi tehdä tuotetilauksen kuvan 13 kaavion mukaisesti.



Kuva 13: Tilausprosessia kuvaava aktiviteetikaavio.

Tuotesivulla esitetään tarkat tuotetiedot ja tilauksen mahdollistava lomake. Lomakkeeseen haetaan automaattisesti tunnistautuneen käyttäjän tiedot istunnon perusteella, jolloin helpotetaan tilauksen tekemistä käyttäjän kannalta. Kun käyttäjä lähettää lomakkeen sisällön, tallennetaan tilaukseen liittyvät tiedot tietokantaan ja lähetetään tilauksesta vahvistus sähköpostitse. Lopuksi käyttäjä ohjataan takaisin tuotelistaussivulle.

Sisällön hallinta

Sivuston sisältö on jaettu luokkiin ja niihin sisältyviin kirjoituksiin. Pääkäyttäjän on mahdollista luoda, muokata ja poistaa kumpiakin edellä mainituista. Muokkaaminen ja luominen tehdään niitä varten luotujen lomakkeiden avulla, joihin tarvittavat tiedot syötetään. Lomakkeissa hyödynnetään WYSIWYG-pohjaista TinyMCE-editoria. WYSIWYG (What You See Is What You Get) tarkoittaa sisällön muokkaamista siinä muodossa, kuin se tullaan esittämään. Tämä mahdollistaa sisällön muotoilun samaan tapaan kuin tekstinkäsittelyohjelmissa [10, s. 113]. Editori on toteutettu JavaScript-ohjelmointikielellä, ja kun käyttäjä muokkaa tekstiä, se luo taustalla tarvittavan merkkauksen tekstin muotoilun esittämiseksi.

Tuotteiden hallinta

Tuotteiden hallinta käsittää niiden luomisen, muokkaamisen, poistamisen ja näkyvyyden hallinnan (liite 2). Luominen ja muokkaaminen suoritetaan lomakkeella, joka sisältää tuotetietoja kuvaavat kentät (liite 3). Tuotekuvauksen muokkaaminen on sisällön tapaan toteutettu TinyMCE-editorin avulla.

Tuotetietojen oheen on mahdollista liittää tuotteen kuva. Kuvan palvelimelle lataamista varten luotiin erillinen avustaja-luokka, jonka avulla lataamiseen ja siihen liittyvään validointiin tarvittava logiikka saadaan erotettua ohjaimesta erilliseksi luokaksi.

Tuotteiden näkyvyyttä voidaan hallita muuttamalla ne inaktiivisiksi, jolloin ne ovat

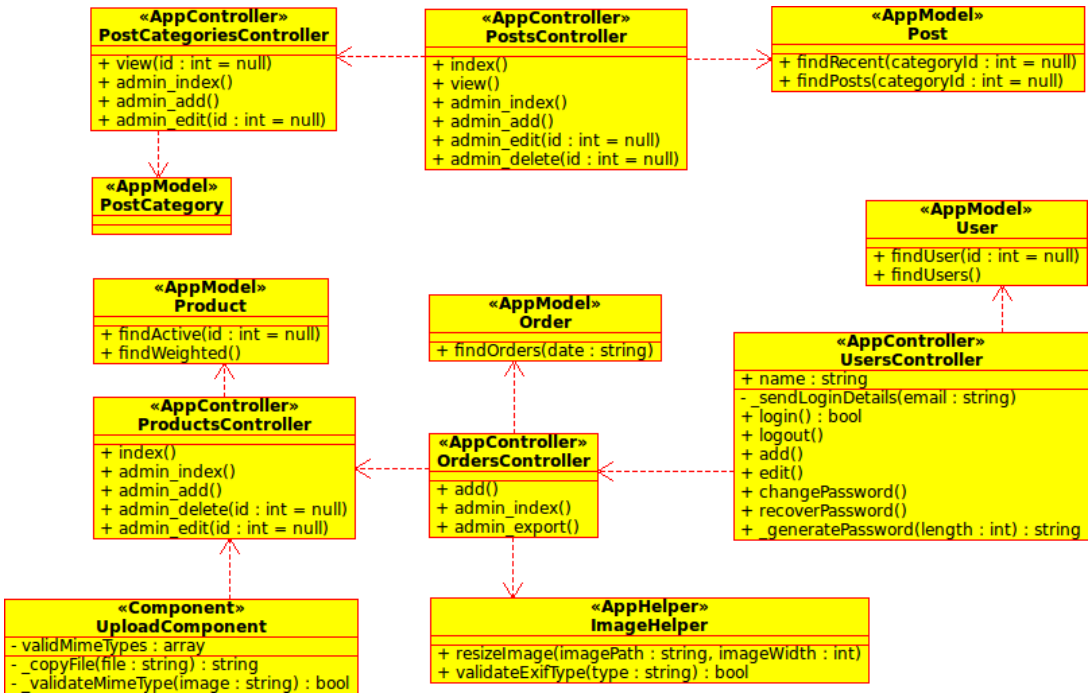
ainoastaan pääkäyttäjän nähtävissä. Tällä tavalla voidaan tarvittaessa osa tuotteista ottaa pois näkyviltä, ilman että ne kokonaan poistettaisiin. Tarvittaessa tuote on myös mahdollista poistaa kokonaan järjestelmästä.

Tilausten hallinta

Tilausten hallinta sisältää tilaustietojen viemisen Excel-laskentataulukkoon. Tiedot haetaan tietokannasta, ja niiden muuntaminen Excel-muotoon hoidetaan erillisen avustaja-luokan ja ja Excel-kirjaston avulla. Kirjasto mahdollistaa Excel-tiedostoformaatin kirjoittamisen ja lukemisen, jossa tilaustiedot viedään asiakasyrityksen rekisteriin.

6.4 Sovelluksen rakenteellinen toteutus

Olympiaklubi-portaali rakennettiin MVC-mallia hyödyntäen. Kuvan 14 luokkaakavio esittää sovelluksen luokkarakenteen ja luokkien väliset yhteydet.



Kuva 14: Sovelluksen luokkarakennetta kuvaava luokkakaavio.

MVC-mallia mukaillen kaikki ohjaintyyppiset luokat ovat riippuvaisia niitä vastaavasta, tietokannan taulua kuvaavasta mallista.

Ohjaimet ja komponentit

Ohjain-luokkien pääasiallisena tarkoituksena on vastata käyttäjän sovellukselle tekemiin pyyntöihin ja ohjata sovelluksen työnkulkua [4, s. 8]. Ohjainten tehtävänä on myös vastata sovelluksen toimintalogiikasta, ja yksi ohjain on yleensä vastuussa sitä vastaavan mallin toiminnoista.

Komponentit ovat ohjainten yhteydessä käytettäviä luokkia, jotka toteuttavat jonkin usealle ohjaimelle tarpeellisen toimenpiteen. Niiden avulla voidaan jakaa toiminnallisuutta usean ohjaimen kesken ja saadaan erotettua toiminnallisuus erilliseen luokkaan ja pidettyä ohjaimen koodi yksinkertaisempana. [13, s. 187.]

Sovelluksen kannalta tärkeimmät ja käytetyimmät komponentit liittyvät istuntojen hallintaan ja tietoturvaan. Niiden ohella käytettiin myös erillistä komponenttia sähköpostien lähetykseen ja luotiin yksi lisäkomponentti kuvatiedostojen lataamiseksi palvelimelle.

Istuntojen hallinta hoidettiin sitä varten tarkoitettua komponenttia käyttäen. Komponentin avulla pystyttiin helposti hallitsemaan käyttäjän istuntoa ohjaimesta käsin [13, s. 194]. Komponentin tukena käytettiin myös istuntojen hallintaan tarkoitettua avustajaa, joka pystyy toteuttamaan lähes samat toimenpiteet, mutta näkymän kautta [13, s. 160]. Suurin ero näiden kahden välillä on, ettei avustaja pysty tallentamaan uutta tietoa evästeisiin. Suurin osa istuntoon liittyvästä työstä hoidetaan ohjaimesta käsin, mutta joitain asioita, kuten käyttäjätiedon lukeminen lomakkeisiin, hoidettiin näkymän kautta. Kirjautumisen kannalta kehys tarjoaa kirjautumisten hallintaan käytäntöjä noudattamalla lähes valmiin ratkaisun, joka ei vaadi juuri muuta kuin tarkempien asetusten määrittämisen [13, s. 189].

Sovellukseen luotiin myös yksi lisäkomponentti, jolla toteutettiin kuvatiedostojen

lataaminen palvelimelle, jossa sovellus on. Komponentti tarkistaa ladattavan tiedoston tyypin oikeellisuuden ja kopioi tiedoston palvelimelle. Komponenttia ei ollut tarvetta hyödyntää usean ohjaimen kesken, mutta sen avulla saatiin siirrettyä sen toiminta erilliseen luokkaan, mikä mahdollistaa sen käytön muissakin tarkoituksissa.

Käyttäjätasojen hallinta etuliitteisiin perustuvalla uudelleenohjauksella

Käyttätasojen määrityksessä on hyödynnetty kehyksen tarjoamaa etuliitteisiin perustuvaa uudelleenohjausta (Prefix Routing). Tämä tekniikka mahdollistaa tietyllä etuliitteellä alkavien metodien määrittämisen tietyn osoitepolun taakse. Asetuksiin määritetään haluttu etuliite (tässä tapauksessa ”admin”), joka halutaan ottaa käyttöön. Kun etuliite on määritetty, ohjautuvat kaikki admin-etuliitteiset metodit kuvan 15 kaavan mukaisesti. [13, s. 178–179.]

Kuva 15: Etuliitteisiin perustuvaa uudelleenohjausta havainnollistava esimerkki.

Kuvassa oleva osoite /admin/products/add kutsuu ProductsController-luokan metodia admin_add.

Ennen minkä tahansa admin-etuliitteisen metodien suorittamista tarkistetaan, onko käyttäjällä pääkäyttäjän oikeudet. Jos tarvittavat oikeudet löytyvät, jatketaan metodin suorittamista, muuten ilmoitetaan, ettei sivua löydy. Näin pystytään piilottamaan hallintapuolen olemassaolo niiltä, joilla ei sinne ole oikeuksia, ja saadaan myös kaikki hallintapuolen toiminnot admin-etuliitteisen osoitepolun taakse.

Kehys mahdollistaa myös laajojen käyttäjäoikeuksien hallinnan erillisen ACL (Access Control List) -komponentin avulla [13, s. 198]. Toteutetussa sovelluksessa ei kuitenkaan ollut tarvetta kuin kahdelle käyttäjätasolle, joten ne päätettiin toteuttaa etuliitteisiin perustuvan uudelleenohjauksen avulla.

Sivupohjien ja elementtien hyödyntäminen näkymien apuna

Näkymien yhteydessä hyödynnetään sivupohjia ja elementtejä, jotka helpottavat esitettävän sisällön luomisessa. Sivupohjien avulla on luotu pohja portaalin ulkoasulle, jonka sisään elementit ja näkymien sisältö on ladattu. Näin toteutettuna jokainen portaalin sivu jakaa yhtenäisen rakenteen ja muutoksen tekeminen sivupohjaan vaikuttaa jokaiselle sivulle. Elementtien avulla saadaan helposti sijoitettua toistuvat elementit, kuten navigointivalikot paikoilleen. [4, s. 146, 152.]

Näkymien ohessa käytettiin useaa kehyksen mukana tulevaa avustaja-luokkaa, ja niiden ohelle luotiin yksi uusi. Avustajien tarkoituksena on helpottaa näkymien ja niiden sisällön luomista [13, s. 5–6]. Eniten apua oli Html-avustajasta, jonka avulla luotiin kaikki sovelluksen lomakkeet, linkit ja muita html-elementtejä, kuten kuvat.

Kuvien eri koossa esittämistä varten luotiin erillinen ImageHelper-avustaja, joka mahdollistaa palvelimelle ladattujen kuvien skaalaamisen eri kokoihin.

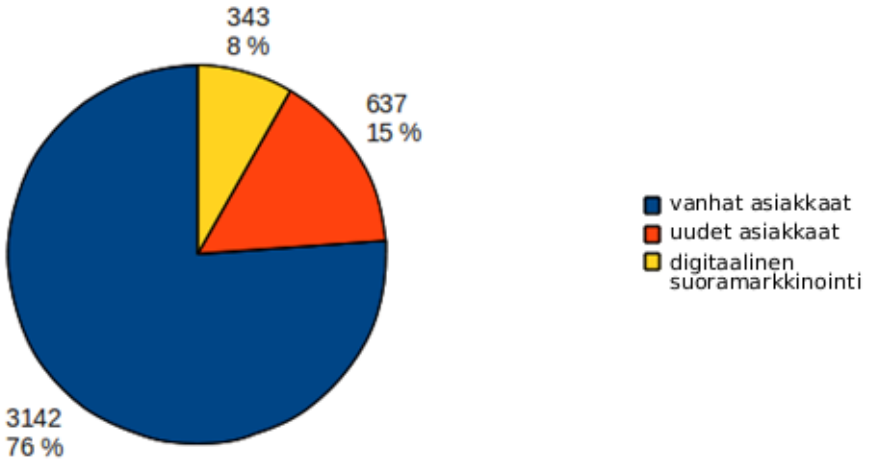
Mallit ja niiden yhteydessä käytettävän tietokannan toteutus

Sovelluksen mallit ja niihin liittyvät tietokannan taulut luotiin kehyksen nimeämisperiaatteita noudattaen, jolloin saatiin täysi hyöty kehyksen ORM-menetelmän tarjoamasta automaatiosta. Nimeämiskäytäntöjen ja määritysten avulla saadaan kätevästi luotua mallien väliset suhteet. Tämän ansiosta pystytään helposti hakemaan tietoa monesta eri tietokannan taulusta, ilman että jouduttaisiin määrittämään taulujen väliset yhteydet haun yhteydessä. Hyötynä on myös se, ettei sovellukseen tarvinnut kirjoittaa yhtäkään erillistä SQL-kyselyä, vaan ne pystyttiin kaikki luomaan kehyksen tarjoamien metodien avulla.

Käyttäjän syöttämän tiedon, kuten käyttäjä- ja tuotetietojen validointi, hoidetaan mallien yhteydessä ennen tallentamista. Varmistamalla käyttäjältä saatu syöte varmistetaan, ettei tietokantaan tai sovellukseen voida syöttää sinne kuulumatonta tietoa.

6.5 Saavutetut tulokset asiakasyrityksen kannalta

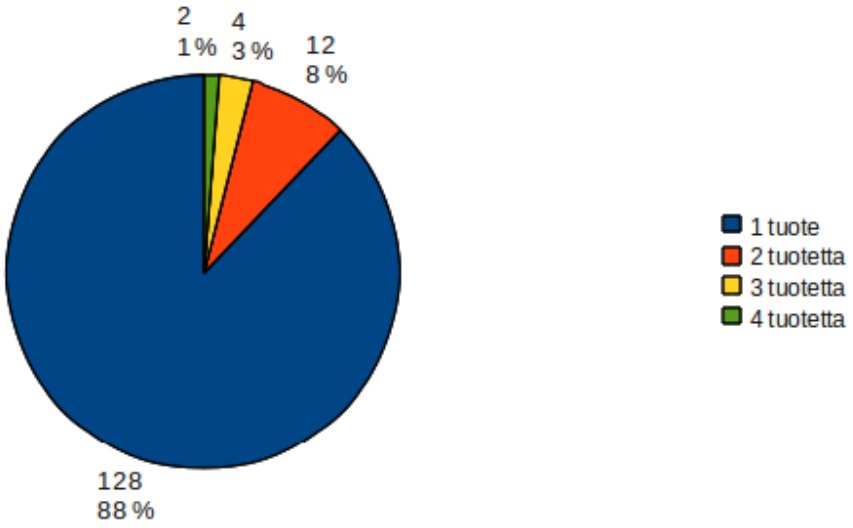
Asiakasyritykselle luotu portaali on ollut käytössä huhtikuusta 2010 lähtien. Kahdeksan kuukauden aikana portaaliin on liittynyt 4 112 käyttäjää, joista 76 % (3 142 kpl) on yrityksen vanhoja asiakkaita eli he ovat aikaisemmin tilanneet sen myymiä tuotteita jonkin kanavan kautta. Loput 15 % (637 kpl) ovat uusia asiakkaita, jotka eivät ole aikasemmin olleet tekemisissä yrityksen kanssa. Kuvan 16 kaavio havainnollistaa liittyneiden käyttäjien lähteitä.



Kuva 16: Portaaliin rekisteröityneiden käyttäjien lähteet.

Suurin osa käyttäjistä on hankittu hyödyntämällä yrityksen olemassa olevaa asiakasrekisteriä, ja siksi suurin osa on vanhoja asiakkaita. Lisäksi asiakashankinnassa hyödynnettiin myös ulkoisen yrityksen tarjoamia digitaalisia suoramarkkinointipalveluita, joiden kautta saatiin noin puolet uusia asiakkaita vastaavasta määrästä.

Portaalin kautta myytäviä tuotteita on kirjoitushetkellä myyty 173, mikä käyttäjien määrään suhteutettuna tuottaa konversioprosentiksi 3,5. Kuva 17 esittää, kuinka tilausmäärät ovat jakautuneet suhteutettuna asiakkaisiin.



Kuva 17: Tuotteiden tilausmäärien jakautuminen asiakkaiden kesken.

Tilaajista vain 12 % (18 kpl) on tilannut useamman kuin yhden tuotteen, mikä oli odotettavissa, koska myytävät tuotteet ovat luonteeltaan sellaisia, että niitä harvoin ostetaan suurta määrää kerralla. Yksittäisen henkilön tekemät tilaukset eivät myöskään sijoitu samalle kerralle vaan jakaantuvat pidemmälle aikavälille.

Kävijämäärät painottuvat vahvasti tiettyihin ajanjaksoihin. Näinä ajanjaksoina on yleensä julkaistu jokin uusi tuote tai portaalissa on ollut jotain aktiviteettiä, jolla käyttäjiä on saatu aktivoitua. Haasteena on ollut kävijöiden osalta se, ettei urheiluaiheisia keräilytuotteita julkaista kovin usein, ja tämän vuoksi saattaa mennä melko pitkiäkin aikoja, ilman että tuotteita päivitetään.

7 Yhteenveto

Insinöörityön tarkoituksena oli perehtyä PHP-ohjelmointikielellä toteutettuihin ohjelmistokehyksiin ja niiden perustana olevaan MVC-suunnittelumalliin. MVC-malli jakaa sovelluksen kolmeen erilliseen osaan: malliin, näkymään ja ohjaimeen. Jokainen osa vastaa tietystä sovelluksen osa-alueesta. Tämän osajaon ansiosta kehyksellä toteutettava sovellus pysyy rakenteellisesti yhtenäisenä ja helposti laajennettavana.

Kehysten ominaisuuksia ja erityispiirteitä vertailtiin kolmen erityyppisen kehyksen kesken. Selkeän dokumentaation ja helppokäyttöisyyden perusteella valittiin kehysten joukosta CakePHP, jonka avulla toteutettiin asiakastyönä pienimuotoinen verkkosovellus. Sovelluksen ideana oli toteuttaa asiakasportaali, jonka kautta voidaan myydä urheiluaiheisia keräilytuotteita ja tarjota niihin liittyvää sisältöä.

Sovelluksen tekninen toteutus onnistui mutkattomasti käytetyllä kehyksellä, mutta se olisi voitu tehdä myös monella muullakin tavalla. Kehysten joukosta on vaikeaa valita yhtä, joka olisi selvästi ylitse muiden, koska kaikilla niillä on omat vahvat puolensa ja erityispiirteensä. Kehysvalinnassa kannattaa kuitenkin erityisesti ottaa huomioon kehitettävän sovelluksen luonne ja se, kuinka paljon kehyksen tulee mukautua sen tarpeisiin.

Käytetyn kehyksen tapauksessa vahvoiksi puoliksi nousivat sen noudattamat käytännöt ja säännöt sekä niiden mukanaan tuoma automatiikka, joka nopeutti huomattavasti sovelluksen kehitystä. Toisaalta jos joutuu toteuttamaan jotakin näiden sääntöjen vastaisesti, siitä voi aiheutua jonkin verran ylimääräistä työtä. Niin kauan kuin pystyy toimimaan kehyksen sääntöjen mukaisesti, kehitystyö on erittäin sujuvaa. Sovelluksen tapauksessa pystyttiin helposti toimimaan näiden sääntöjen mukaisesti.

Kehykset ovat yleisesti monimutkaisia toiminnaltaan, ja niiden laajamittainen opettelu ja esittely ulottuvat helposti tämän työn rajojen ulkopuolelle. Eri kehyksiin ja niiden perustana oleviin tekniikoihin tutustuminen on kuitenkin antanut hyvän kuvan niiden toiminnasta yleisellä tasolla. Näiden tietojen pohjalta myös uusien kehysten

omaksuminen on huomattavasti sujuvampaa, koska niiden pääasialliset toimintaperiaatteet ja mahdollisuudet ovat hyvin tiedossa.

Asiakaspalaute toteutetusta portaalista on ollut myönteistä, ja samaa sanotaan myös asiakasyrityksen puolesta. Vuoden aikana portaali on kerännyt reilut 40 000 käyntiä, joista yksittäisiä kävijöitä on 17 000. Tilauskonversio on 0,45 %, mikä voisi olla parempikin, mutta strategiana on ollut pehmeämpi myyntitapa käyttäjien sitouttamiseksi. Portaali on toiminut yritykselle tärkeänä maineenhallinnan työkaluna ja sopivana mediana sen tukemien urheilijoiden esittelyyn.

Lähteet

1. Koskimies, Kai; Mikkonen, Tommi. Ohjelmistoarkkitehtuurit. Jyväskylä: Talentum Media, 2005.

2. Wake, William C. Growing Frameworks in Java: White box and black box. (www-dokumentti) <http://xp123.com/wwake/fw/ch12-bb.htm>. 2009. Luettu 10.6.2010.

3. Gamma, Erich; Helm, Richard; Johnson, Ralph; Vissides, John M. Design Patterns: Elements of Reusable Object-Oriented Software. Reading: Addison-Wesley, 1995.

4. Bari, Ahsanul; Syam, Anupom. CakePHP Application Development. Birmingham: Packt Publishing, 2008.

5. Zandstra, Matt. PHP Objects, Patterns and Practice 3rd ed. Berkeley: Apress, 2008.

6. PHP: History of PHP. (www-dokumentti). PHP. <http://www.PHP.net/manual/en/history.PHP.PHP>. 17.9.2010. Luettu 12.6.2010.

7. Welling, Luke; Thomson, Laura. PHP and MySQL Web Development 4th ed. Pearson Education, 2009.

8. PHP: What is PHP?. (www-dokumentti). PHP. <http://fi2.PHP.net/manual/en/intro-whatis.PHP> 8.6.2010. Luettu 17.6.2010.

9. Cake Software Foundation, About. (www-dokumentti). Cake Software Foundation. <http://cakefoundation.org/pages/about> 14.1.2009. Luettu 17.6.2010.

10. Chan, Kai; Omokore, John; Miller, Richard K. Practical CakePHP Projects. Berkeley: Apress, 2009.

11. Myer, Thomas. Professional CodeIgniter. Indianapolis: Wiley Publishing, 2008.

12. Cake Development Corporation: Services. (www-dokumentti). Cake Development Corporation. <http://cakedc.com/eng/services>. Luettu 10.6.2010.

13. Golding, David. Beginning CakePHP From Novice to Professional. Berkley: Apress, 2008.

14. CodeIgniter at a Glance: CodeIgniter User Guide. (www-dokumentti). Ellis

Labs. <http://codeigniter.com/user_guide/overview/at_a_glance.html>. 2009. Luettu 20.6.2010.

15. Upton, David. CodeIgniter for Rapid PHP Application Development. Birmingham: Packt Publishing, 2008.

16. Active Record: CodeIgniter User Guide. (www-dokumentti). Ellis Labs. <http://codeigniter.com/user_guide/database/active_record.html>. 2009. Luettu 20.6.2010.

17. Zend Solutions: Zend and PHP: An Overview. (www-dokumentti). Zend Technologies. <http://www.zend.com/topics/Zend-Solution-Brief-0909-WEB.pdf>. 2009. Luettu 20.6.2010.

18. Evans, Cal. PHP|architect's Guide to Programming with Zend Framework. Toronto: Marco Tabini & Associates, 2008.

19. Allen, Rob. Lo, Nick. Zend Framework in Action. Greenwick: Manning Pulications, 2009.

20. Zend Framework: Documentation: Zend_Db_Table. (www-dokumentti). Zend Technologies. <http://framework.zend.com/manual/en/zend.db.table.html>. Luettu 30.6.2010.

21. McArthur, Kevin. Pro PHP: Patterns, Frameworks, Testing and more. Berkley: Apress, 2008.

22. The Bakery: Clearing Up Some Confusion on the Release Versions of CakePHP. (www-dokumentti). Joel Perras. <http://bakery.cakePHP.org/articles/view/clearing-up-some-confusion-on-the-release-versions-of-cakePHP>. 31.8.2009. Luettu 30.6.2010.

23. Sweat, Jason E. PHP|architect's Guide to PHP Design Patterns. Toronto: Marco Tabini & Associates, 2005.

24. Brady, Pádraic. Zend Framework Book: Surviving the Deep End - Chapter 3. The Model. (www-dokumentti). <http://www.survivethedeepend.com/zendframeworkbook/en/1.0/the.model>. 2009. Luettu 21.9.2010.

25. CakePHP 1.3 Documentation: Basic Principles of CakePHP. (www-dokumentti). <http://book.cakePHP.org/complete/892/Basic-Principles-of-CakePHP>. 1.11.2008. Luettu 12.7.2010.

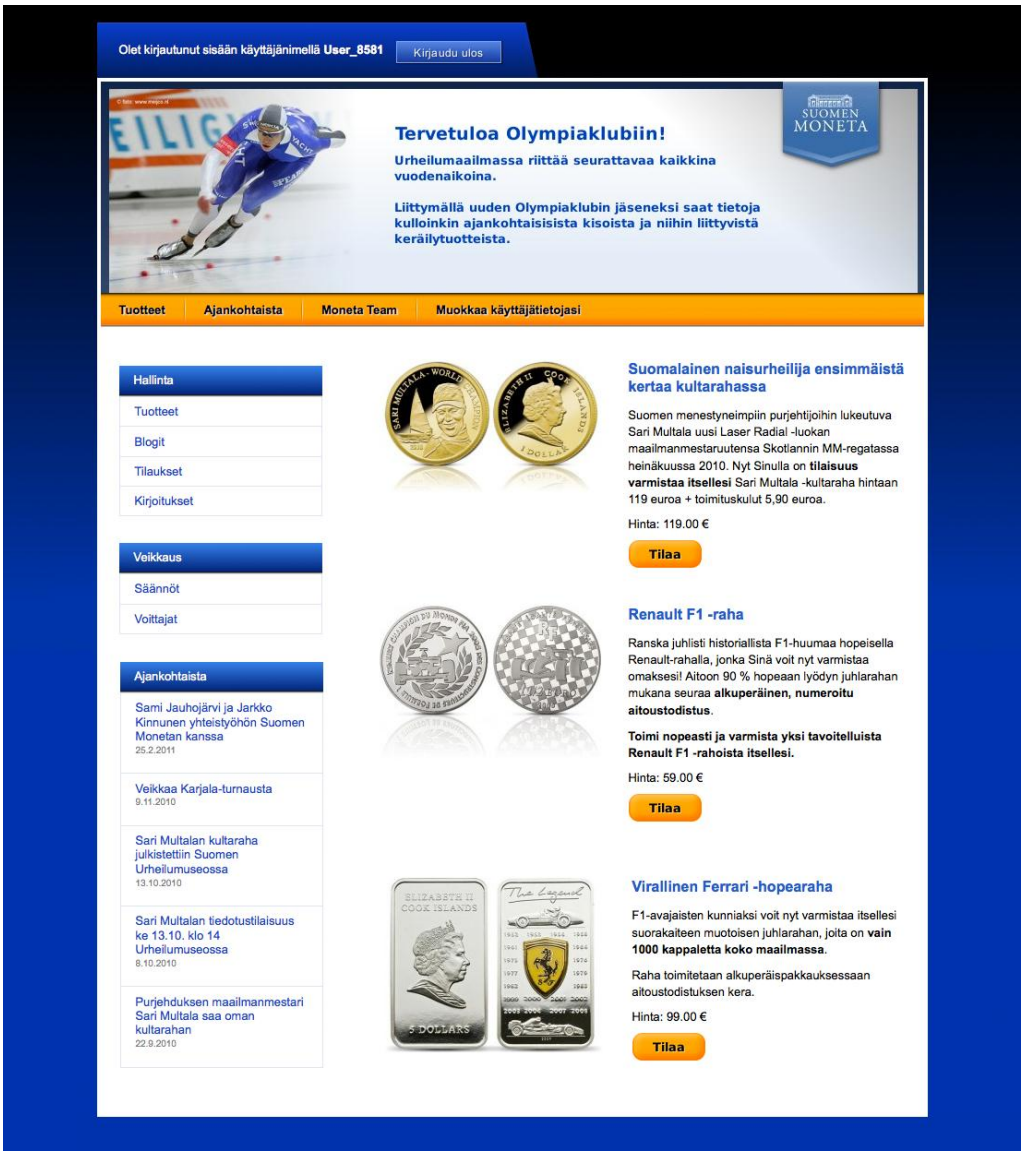
26. Barry, Douglas K. Lack of impedance mismatch. (www-dokumentti). Barry Douglas K. <http://www.service-architecture.com/object-oriented-databases/articles/lack_of_impedance_mismatch.html> Luettu 9.10.2010.

27. Fowler, Martin. Patterns of Enterprise Application Architecture. Boston: Pearson Education, 2003.

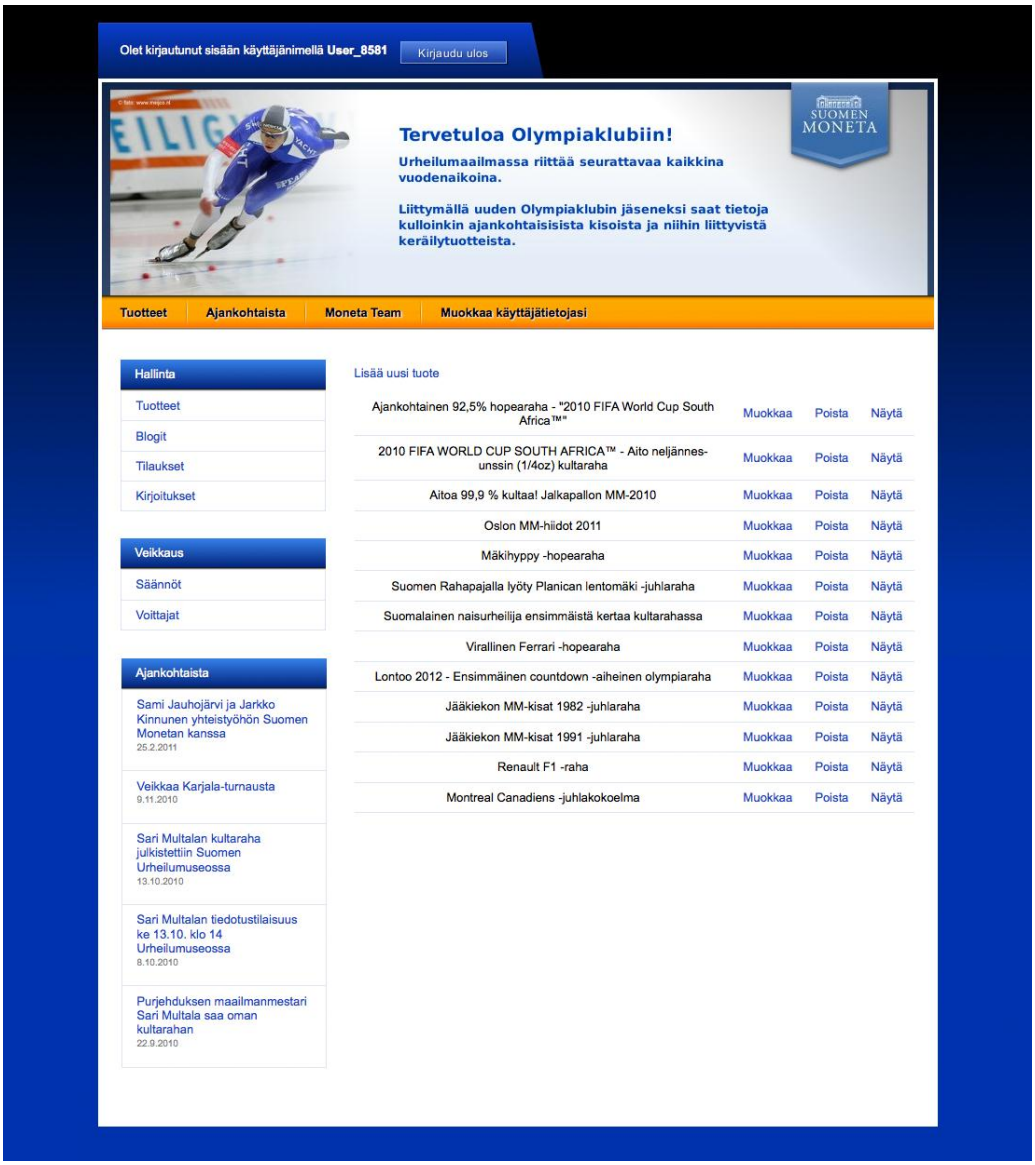
28. Bauer, Christian; King, Gavin. Hibernate in Action. Greenwich: Manning Publications, 2005.

29. Myers, Glenford J. The Art of Software Testing. New Jersey: John Wiley & Sons, 2004.

Liite 1: Portaalin tuotesivu



Liite 2: Pääkäyttäjän hallintaosion tuotelistaus



Liite 3: Tuotteen muokkausnäkymä