Opinnäytetyö (AMK)

Tietotekniikan koulutusohjelma

Mediatekniikka

2013

Janne Arola

INTRANETIN SUUNNITTELU JA TOTEUTUS

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma | Mediatekniikka

Kevät 2013 | 37

Ohjaajat: Ins. (AMK) Keijo Leinonen, Yliopettaja FT Mika Luimula

Janne Arola

INTRANETIN SUUNNITTELU JA TOTEUTUS

Työn tarkoituksena oli suunnitella jo toteuttaa yritykselle www-pohjainen intranet. Intranetin tulisi sisältää omat osa-alueensa käyttäjien, dokumenttien ja yrityksen viikkotiedotteiden hallintaan, sekä sovelluksen, jonka avulla on mahdollista kerätä yrityksen henkilökunnan mielipide yrityksessä yhteisesti päätettäviin asioihin.

Työ on toteutettu PHP- ja MySQL -alustalle ja sen toteuttamisessa on käytetty projektia varten rakennettua omaa PHP-ohjelmistokehystä, joka pohjautuu MVC-ohjelmistoarkkitehtuuriin. Ohjelmistokehyksen suunnittelun lähtökohtana oli sen joustava sulautuminen intranetin rakenteeseen, keveys sekä tietoturva.

Lopputulos on täysin www-pohjainen intranet, joka sisältää viikkotiedotteet sovelluksen, jossa pystyy muokkaamaan, lisäämään ja poistamaan viikkotiedotteita ilman html-tietämystä tai erillisiä ohjelmia. Äänestyssovelluksen, johon järjestelmän ylläpitäjien on mahdollista luoda kysymyksiä ja joihin intranetin käyttäjät pystyvät äänestämään ja kommentoimaan. Järjestelmä sisältää myös kattavan käyttäjien hallinnan sekä henkilötietojärjestelmän, jonka kautta henkilökunnan yhteystietoja on mahdollista listata eri variaatioilla tai hakea vapaalla tekstihaulla.

Intranet vähensi yrityksen sähköpostin kautta kulkenutta sisäistä liikennettä, sekä toimii säilytyspaikkana kaikille yleisimmin käytetyille sisäisille asiakirjoille. Intranet myös mahdollisti helpomman tavan yrityksen sisäisien päätöksien käsittelyyn ja helpottaa viikkotiedotteiden kautta henkilöstölle jaetun informaation hakemisen jälkikäteen.

ASIASANAT:

PHP, MySQL, intranet, ohjelmistokehys

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Information Technology | Digital Media

2013 | 37

Keijo Leinonen, B. Eng
Mika Luimula, Ph. D., Principal Lecturer

Janne Arola

DESIGNING AND IMPLEMENTING OF AN INTRANET

The objective of this Bachelor's thesis is to design and implement a web-based intranet system for a medium sized company. The intranet in question should have sections for accessing and maintaining documents and company weekly briefings and it should also contain an application which could be used to collect employees' opinion on shared subjects.

The intranet was made with PHP and MySQL technologies and it used a custom PHP-framework that had been developed specifically for this project. The PHP-framework made for the project was based on Model-View-Controller programming architecture. The main principle in design of the framework is that it is secure, fits well to the project, and is as lightweight as possible.

The result of the project is a completely web-based intranet that has its own applications for editing and creating weekly briefings, voting application and complete user management system. The usage of the intranet does not require any html programming knowledge or separate programs and all of the information that is held in the intranet is editable through its administrator section.

As a result, the designed and implemented intranet reduced the amount of company’s in-house e-mail traffic and also works as storage for all of the mostly used documents. Because of the voting application, the intranet also provides an easier way of handling the company’s inner decision-making in the subjects where the company wants the employees' opinion.

KEYWORDS:

PHP, MySQL, Intranet, framework

SISÄLTÖ

KUVIOT

OHJELMAT

SANASTO

Ajax	Lyhenne sanoista Asynchronous JavaScript And XML. Joukko www-sovellustekniikoita, joiden avulla on mahdollista sovelluksesta vuorovaikutteisempia.
HTML	HyperText Markup Language. Kuvauskieli jota käytetään pääsääntöisesti internet-sivujen sisällön kuvaamiseen.
HTTP	Hypertext Transfer Protocol. Tiedonsiirtoprotokolla jota esimerkiksi selaimet ja www-palvelimet käyttävät.
MVC	Model-View-Controller. Ohjelmistoarkkitehtuuri, jossa ohjelma on jaettu kolmeen osa-alueeseen: Malliin, näkymään ja kontrolleriin.
POST ja GET	HTTP-kutsu jonka avulla on mahdollista lähettää käyttäjän tietokoneelta palvelimelle tietoa.

1 JOHDANTO

Opinnäytetyön tarkoituksena oli toteuttaa yritykselle www-pohjainen intranet, jonka kautta yritys voisi helpommin hoitaa sisäistä viestintäänsä. Samalla se toimisi myös yrityksen asiakirjojen säilytyspaikkana. Intranet tulisi osittain korvaamaan yrityksen nykyisen tavan välittää sisäistä tietoa sähköpostin välityksellä ja näin vähentää yrityksen sisäistä sähköpostiliikennettä.

Intranetin tulee sisältää omat osionsa viikkotiedotteiden luomiseen ja lukemiseen, äänestyssovelluksen, jonka kautta yrityksen henkilökunnan on mahdollista antaa mielipiteensä yrityksen yhteisissä päätöksenteoissa sekä kattava yhteystietokirjasto, johon on lisätty kaikki yrityksen henkilökunnan jäsenet ja heidän tietonsa sekä yrityksen tiimien tiedot. Yhteystietoja tulee olla myös mahdollista etsiä vapaan tekstihaun perusteella ja henkilökunta täytyy voida listata tiimikohtaisesti.

Kaikkia intranetin osa-alueiden sisältämää tietoa tulee myös päästä muokkaamaan intranetin kautta. Tieto pitää olla muokattavissa ilman, että siihen tarvitaan erillistä työkalua. Tämän takia intranet sisältää myös hallintaosion, jonka kautta viikkotiedotteita, äänestyksiä ja käyttäjiä on mahdollista muokata. Hallintaosioon on pääsy vain hallinnointikäyttäjillä.

Intranetin tulee toimia PHP- ja MySQL–ympäristöissä ja sen toteuttamiseen on mahdollista käyttää valmista ohjelmistokehystä tai rakentaa järjestelmä alusta saakka itse. Luvut 3.1 ja 3.2 käsittelevät tarkemmin, miksi päädyin rakentamaan oman MVC-ohjelmistokehyksen jo olemassa olevien kehyksien käyttämisen sijaan.

Luku 4 käsittelee MVC-arkkitehtuuriin liittyviä komponentteja ja niiden ominaisuuksia ja luvussa 5 on käsitelty tarkemmin muita MVC-arkkitehtuuriin liittymättömiä, mutta intranetin kannalta olennaisia järjestelmän osa-alueita.

Koska intranet sisältää paljon salassa pidettävää yrityksen sisäistä informaatiota, tietoturva on yksi olennaisimpia kriteerejä järjestelmää suunniteltaessa. Lu-

vussa 6 käsitellään tarkemmin erilaisia järjestelmän tietoturvaan liittyviä ratkaisuja ja toteutuksia.

Koska toimeksiantajayritys kehittää jatkuvasti omaa sisäistä viestintäänsä ja toimintaansa, on todennäköistä että intranetiin on tarpeellista tehdä kokonaan uusia osioita tai jo olemassa osioita pitää muokata. Luku 7 käsittelee intranetin toteutusvaiheessa vastaan tulleita jatkokehitys- tai parannusmahdollisuuksia.

2 TOIMEKSIANTO

2.1 Toimeksiantaja

Opinnäytetyön toimeksiantaja on Oy Käännös- ja Tulkkauspalvelu Mokoma Ab (myöh. Mokoma). Mokoma on viittomakielen tulkkausta ja opetusta tarjoava, koko Suomen alueella toimiva yritys. Yrityksellä on tällä hetkellä viisi toimipistettä, yksi Pohjanmaalla, Satakunnassa, Varsinais-Suomessa, Uudellamaalla sekä Keski-Suomessa. Yrityksessä työskentelee yli 50 henkilöä, joista suurin osa on ammatiltaan viittomakielen tulkkeja. [1], [2]

2.2 Toimeksianto

Työn toimeksianto oli luoda yrityksen sisäiseen viestintään tarkoitettu sisäinen tietoverkko eli intranet. Koska yrityksellä on useita toimipisteitä eri paikkakunnilla ja tulkit työskentelevät pääsääntöisesti toimistojen ulkopuolella, intranetin tulee sijaita julkisessa internetissä, eikä esimerkiksi yrityksen toimistojen sisäisessä verkossa. Intranetin pääsääntöisenä käyttötarkoituksena on toimia yrityksen omana tiedotuskanavana sekä salassa pidettävien tietojen ja asiakirjojen säilytyspaikkana. Tästä syystä tietoturva on merkittävässä roolissa intranetin rakennetta suunniteltaessa. Projektin aloitushetkellä suurin osa yrityksen sisäisestä viestinnästä tapahtui sähköpostin ja osittain myös puhelimen välityksellä. Koska sisäistä viestintää on yrityksessä paljon, nykyinen menetelmä on todettu raskaaksi käyttää [2].

2.3 Intranetiltä vaaditut ominaisuudet

2.3.1 Viikkotiedotteiden arkisto

Yritys käyttää sisäisien asioiden viestinnässä viikoittain jaettavaa tiedotetta, jatkossa nimellä viikkotiedote. Viikkotiedotteisiin on koottu aina kuluvan viikon aikana vastaan tulleet ilmoitusasiat, yrityksen sisällä tapahtuneet muutokset tai muut ilmoitusluontoiset asiat, jotka koskevat koko yritystä. Viikkotiedotteet lähetetään sähköpostitse, tiettynä viikonpäivänä, koko yrityksen henkilökunnalle. Koska viikkotiedotteet sisältävät paljon tietoja joihin henkilökunnan voi olla tarvetta palata myös viikkotiedotteen vastaanottamisen jälkeen, yritys halusi jonkin sijainnin viikkotiedotteiden arkistoimista ja säilyttämistä varten, sekä hakuominaisuuden, minkä avulla viikkotiedotteissa olevaa sisältöä olisi mahdollista hakea. Viikkotiedotteita pitäisi olla myös mahdollista muokata suoraan intranetin hallinnan kautta, ilman että uuden viikkotiedotteen lisäämiseen tai muokkaamiseen tarvittaisiin erillistä ohjelmaa tai html-ohjelmoinnin tuntemusta.

2.3.2 Äänestyssovellus

Yksi yrityksen ongelmista sisäisen viestinnän osalla on ollut henkilökunnan mielipiteen kerääminen yrityksen yhteisien asioiden päätöksenteon yhteydessä. Tämänhetkinen tapa asian hoitamiseen on ollut asioiden käsitteleminen sähköpostin välityksellä. Sähköposti ei kuitenkaan ole tehokas tapa keskustella tai äänestää yhteisistä asioista, koska tällöin yrityksessä muodostuu paljon ylimääräistä sähköpostiliikennettä. Tämän toimintamallin korvaajaksi on intranetin äänestyssovellus. [2]

Äänestyssovelluksen tarkoitus on että intranetiä hallinnoiva taho voi halutessaan luoda sovellukseen uuden kysymyksen, johon käyttäjien on mahdollista äänestää valitsemalla jonkin kysymykseen asetetuista vastausvaihtoehdoista, sekä kommentoida kysymystä. Tällä tavoin yhteen aihepiiriin liittyvän kysymyksen keskustelu pysyy helposti seurattavasti samassa paikassa, sekä kysymyk-

sien vastaukset ovat nopeasti kerättävissä sovelluksen hallinnasta. Äänestyssovellukseen on mahdollista luoda joko anonyymejä äänestyksiä, joissa ylläpidon ei ole mahdollista nähdä ketä on äänestänyt mitäkin vastausvaihtoehtoa. Äänestyssovellukseen voi tehdä myös julkisia äänestyksiä, joista on mahdollista tarkastella käyttäjäkohtaisesti vastausvalintoja.

2.3.3 Henkilöstön yhteystiedot

Toimeksiantajan toiveena oli saada intranetiin helposti ylläpidettävä sovellus, johon on mahdollista tallentaa kaikki yrityksen työntekijät sekä tiimit ja näihin liittyvät yhteystiedot. Sovellus tulisi korvaamaan yrityksen nykyisen menetelmän ylläpitää ja jakaa työntekijöiden yhteystietoja taulukkolaskentatiedoston muodossa. Yrityksen henkilökunta on jaettu alueittain tiimeihin, johon jokaiseen kuuluu 5–20 työntekijää. Koska tiimit ovat tärkeässä roolissa yrityksen sisäisessä hierarkiassa, työntekijöitä on pystyttävä listamaan sovelluksesta, ei ainoastaan nimen, vaan myös tiimin perusteella, sekä yhteystietosovelluksessa tulee olla myös vapaa tekstihaku, jonka kautta yhteystietoja on mahdollista hakea nimen perusteella.

2.3.4 Käyttäjien hallinta

Käyttäjien hallinta liittyy läheisesti henkilöstön yhteystietoihin. Yhteystiedot ja käyttäjät muun muassa käyttävät yhteistä tietokantataulua. Kaikilla intranetin käyttäjillä on oikeus muokata omia henkilökohtaisia tietojaan, lukea viikkotiedotteita ja dokumentteja sekä äänestää ja kommentoida äänestyssovelluksessa. Tavallisella käyttäjällä on myös oikeus poistaa omia kommenttejansa sekä tarkastella muiden käyttäjien julkisia yhteystietoja.

Tavallisien käyttäjien lisäksi intranetissä on hallinnointikäyttäjien ryhmä, jonka jäsenillä on täydet oikeudet intranetin eri osa-alueisiin ja niiden toimintoihin. Hallintoryhmän käyttäjät ovat muilta ominaisuuksiltaan tavallisia käyttäjiä, jotka kuuluvat tiettyyn tiimiin ja heillä on samat perustiedot kuin muillakin käyttäjillä.

3 KÄYTETYT TEKNOLOGIAT

Intranetin tuli pohjautua PHP- ja MySQL-alustaan, koska molemmat teknologiat ovat hyvin yleisiä ja niille on tarjolla myös paljon edullisia palvelinratkaisuja. Yrityksellä oli myös projektin toteutushetkellä käytettävissään näitä teknologioita tukeva palvelinratkaisu, joten tästäkin syystä valinta oli luonnollinen.

MySQL on maailman käytetyin relaatiotietokantaohjelma [4]. PHP eli Hypertext Preprocessor on laajalti käytössä oleva avoimeen lähdekoodiin perustuva ohjelmointikieli. PHP on yksi ensimmäisiä server side scripting eli palvelimella ajettavia ohjelmointikieliä ja sen syntaksi muistuttaa paljolti C, perl ja java ohjelmointikielien syntaksia. PHP:n yksi tärkeimmistä ominaisuuksista on että ohjelmointikieltä on mahdollista upottaa HTML-koodin sekaan, joka tekee PHP:sta erityisen soveltuvan ohjelmointikielen web-ohjelmointiin. [3]

Koska intranet sisältää paljon toisistaan eroavia ja toiminnaltaan erilaisia websovelluksia, järjestelmää ei ole järkevää rakentaa lineaarista ohjelmointitapaa käyttäen. Lineaarisessa ohjelmoinnissa html-sivujen sekaan on upotettu PHP-ohjelmakoodia ja toiminnallisuudet tapahtuvat joko HTML-sivuilla itsessään tai mukaan liitettävissä erillisissä PHP-tiedostoissa, ilman minkäänlaista luokkarakennetta. On jatkon kannalta parempi rakentaa järjestelmä olio-ohjelmointitapaa käyttäen. Olio-ohjelmoinnin tueksi on olemassa erilaisia pohjarakenteita eli ohjelmointiarkkitehtuureja. Yksi yleisimpiä ohjelmointiarkkitehtuureja on MVC eli Model-View-Controller -arkkitehtuuri, jossa toiminnallisuus, sekä käyttäjälle näytettävä sisältö on erotettu toisistaan [5].

Suunniteltaessa on hyvä ottaa myös huomioon valmiin ohjelmistokehyksen (framework) käyttämistä järjestelmän toteuttamisessa. Ohjelmistokehys on kolmannen osapuolen valmiiksi kehittämä perusrunko rakennettavalle sovellukselle. Ohjelmistokehys mahdollistaa ohjelmien kehittämisen nopeammin, tarjoamalla valmiita ja usein tarvittavia ratkaisuja, malleja tai funktioita, jotta ohjelmoijan ei tarvitse tehdä kaikkia sovelluksen osia alusta saakka itse. PHP-ohjelmointikielelle on olemassa useita kymmeniä erilaisia ohjelmistokehyksiä,

jotka eroavat toisistaan toimintamalliensa ja rakenteensa osalta. Suurin osa PHP-ohjelmointikielelle tehdyistä ohjelmistokehyksistä perustuvat MVC-arkkitehtuuriin [6]. [5], [7]

Vaikka PHP-ohjelmointikielelle on tarjolla runsaasti erilaisia, pitkään kehitettyjä ja hyvin kattavia ohjelmistokehyksiä, päädyin ratkaisuun oman ohjelmistokehyksen rakentamisesta intranetin toteuttamista varten. Oman ohjelmistokehyksen rakentamisessa on hyvät ja huonot puolensa, samoin kuin valmiin sovelluskehyksen käytössä.

3.1 Valmiin ohjelmistokehyksen edut ja ongelmat

Ohjelmistokehykset mahdollistavat nopean alun uusien www-sovelluksien kehittämiseen tarjoamalla ison määrän valmiita funktioita ja toiminnallisuuksia. Koska ohjelmistokehys antaa juuritason perustoiminnot valmiina pakettina, ohjelmoijan on mahdollista keskittyä varsinaisen web-sovelluksen kehittämiseen, eikä hänen tarvitse käyttää aikaa yleisien perustoiminnallisuuksien rakentamiseen. Esimerkkinä näistä ohjelmistokehyksien tarjoamista yleistoiminnallisuuksista voivat olla tietokantaan tallentamiseen ja sieltä tiedon hakemiseen, lomakkeiden käsittelyyn ja automaattiseen ohjelmakoodin lataamiseen liittyvät funktiot. [7], [8], [9]

Koska lähes kaikki olemassa olevat PHP-ohjelmistokehykset perustuvat mvc-arkkitehtuuriin, niiden käyttäminen kannustaa ohjelmistokehyksien avulla tehdyt sovellukset käyttämään kyseistä arkkitehtuuria. Tällöin ohjelmakoodi on helposti hallittavissa ja ohjelman näkymään, tietojenhallintaan ja toimintoihin liittyvät osa-alueet pysyvät erillään toisistaan. Oman rakenteensa ansioista, valmiit ohjelmistokehykset helpottavat ohjelmoijaa tekemään hyvän ohjelmointistandardin mukaista ohjelmakoodia. Toisaalta koska ohjelmistokehys tarjoaa käyttäjälle valmiin pohjan, käyttäjä joutuu sitoutumaan näihin ohjelmistokehyksen tarjoamiin suunnitteluratkaisuihin eikä näin ollen pysty vaikuttamaan ohjelmiston rakenteeseen niin paljoa kuin mitä alusta saakka tehdessä. [5], [7], [8], [9]

Useimpia saatavilla olevia ohjelmistokehyksiä on kehitetty jo useita vuosia ja niitä kehitetään pääsääntöisesti useamman kuin yhden ihmisen voimalla. Tämän ansioista ohjelmistokehykset ovat jo varsin pitkälle kehitettyjä ja toimintavarmoja, sekä niistä on jo ehditty korjaamaan pääosa ohjelmistokehyksen kehittämisen aikana mahdollisesti tulleista ohjelmointivirheistä. Monilla ohjelmistokehyksillä on myös laaja käyttäjäkunta, jolta on mahdollista saada vertaistukea kyseisellä ohjelmistokehyksen avulla ohjelmointiin. [5], [7]

Jokainen ohjelmistokehys eroaa toisistaan ratkaisujensa ja toiminnallisuutensa puolesta. Tästä johtuen tietyllä ohjelmistokehyksellä toteutettua ohjelmakoodia voi olla ongelmallista käyttää uudelleen jossain toisessa ohjelmistokehyksessä tai ilman ohjelmistokehystä. Koska ohjelmistokehykset eroavat toisistaan, niin jokaisen uuden ohjelmistokehyksen kohdalla tulee myös vastaan kyseisen kehyksen opettelu, sekä ohjelmistokehyksen asentaminen kohdepalvelimelle [7]. Molemmat toiminnot vievät vaihtelevan määrän aikaa, mutta tietyissä tapauksissa, kuten esimerkiksi todella pienissä sovelluksissa, on syytä harkita onko nopeampaa toteuttaa sovellus ilman erillistä ohjelmistokehystä. [10]

Valmista ohjelmistokehystä käytettäessä on vaikeampaa seurata mitä tehdyn www-sovelluksen suorittamisen takana tapahtuu, koska suuri osa toiminnallisuuksista on piilossa ohjelmistokehyksen tarjoavien funktioiden takana. Pääsääntöisesti ohjelmistokehystä käytettäessä ei ole välttämätöntä tietääkään miten ohjelmistokehyksen tarjoavat funktiot toimivat, mutta mikäli vastaan tulee hankalammin selvitettävä virhetilanne, niin virheen etsiminen voi olla työläämpää. [10]

3.2 Perustelu oman MVC-ohjelmistokehyksen tekemiseen

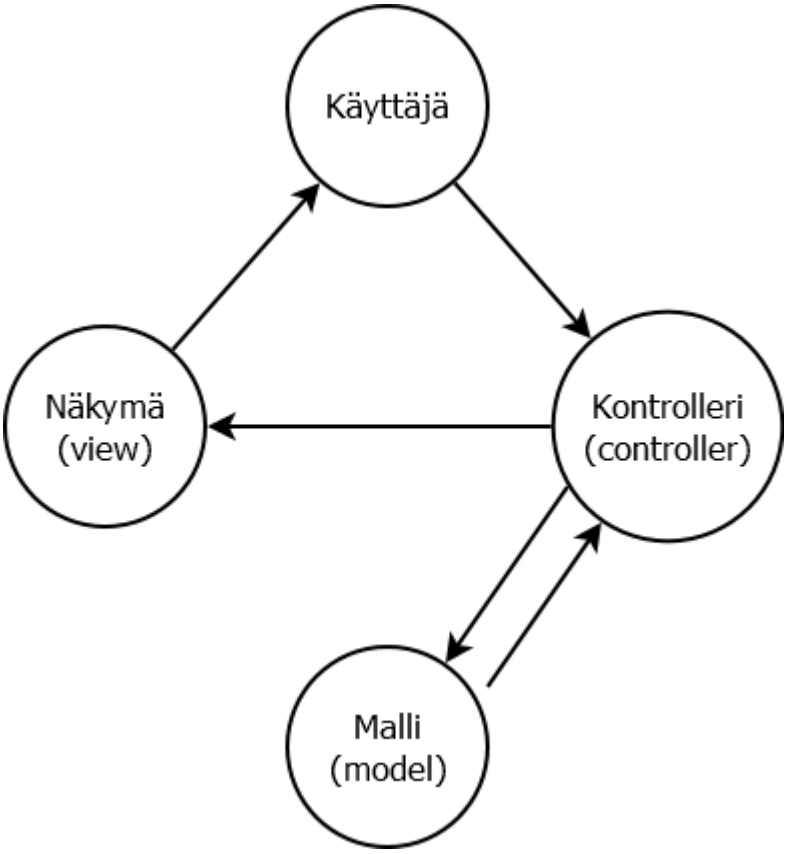
Painavin syy oman ohjelmistokehyksen kehittämiseen on useimmiten MVC eli Model-View-Controller arkkitehtuuriin tutustuminen ja sen parempi ymmärtäminen. Valmiissa ohjelmistokehyksissä, jotka perustuvat kyseiseen ohjelmistoarkkitehtuuriin, ei tarvitse erikseen miettiä miten se on rakennettu ja kuinka se toimii, koska ohjelmistokehys tarjoaa kaikki toiminnallisuudet, rakenteen ja luokat

arkkitehtuurin käyttämiseen. Omassa ohjelmistokehyksessä nämä täytyy luoda itse, jolloin niihin on myös pakollista perehtyä tarkemmin. Oman ohjelmistokehyksen luominen on myös erinomainen tapa opiskella ja tutustua olio-ohjelmointiin, suunnittelumalleihin, sekä kehittää omaa ohjelmointitietämystä. [5]

Useimmat valmiista ohjelmistokehyksistä sisältävät tuhansia rivejä ohjelmakoodia ja mahdollisesti satoja erilaisia apufunktioita ja toiminnallisuuksia, joita ei välttämättä pienikokoisen tai yksinkertaisen www-sovelluksen rakentamisessa tarvitse. Tämän vuoksi valmiit ohjelmistokehykset saattavat olla tarpeettoman raskaita kevyen www-sovelluksen kehittämiseen, kun taasen omasta ohjelmistokehyksestä on mahdollista muotoilla sellainen kuin työn alla oleva projekti vaatii. Mikäli tekeillä oleva sovellus venyy kuitenkin suunniteltua suuremmaksi, omatekoista ohjelmistokehystä on helppo laajentaa vastaamaan projektin tavoitteita. [7], [10]

4 MODEL-VIEW-CONTROLLER-RAKENNE

Model-View-Controller eli MVC on ohjelmistoarkkitehtuuri, jonka tarkoitus on yksinkertaistaa ohjelmistokehittämistä ja ylläpitoa jakamalla sovelluksen kolmeen eri osa-alueeseen, malliin, kontrolleriin ja näkymään. Malli vastaa kaikesta järjestelmän tietokantatoiminnallisuudesta ja tietojenkäsittelystä. Näkymä määrittää sovelluksen ulkoasun ja vastaa sen näyttämisestä käyttäjälle. Kontrolleri eli ohjain toimii näkymän, mallin sekä käyttäjän välisenä rajapintana, joka välittää mallilta tulevan tiedon näkymän näytettäväksi, sekä vastaanottaa käyttäjältä tulevan syötteen. [5], [11], [12]



Kuvio 1. MVC-arkkitehtuurin eri osa-alueiden toiminta.

Jokaista järjestelmän osa-aluetta eli sovellusta varten on oma luokkansa, joka periytetään järjestelmän perusluokista. Perusluokat antavat näille luokille pakolliset perusominaisuudet, joiden avulla järjestelmän on mahdollista toimia ja vas-

ta näistä perusluokista periytettyihin lapsiluokkiin rakennetaan sovelluskohtaiset toiminnot.

4.1 Malli

Mallin (model) toteutustapaan ei ole MVC-arkkitehtuurissa määritetty mitään yksittäistä oikeata tapaa. Malli hoitaa datan noutamisen ja tallentamisen sovelluksen tallennusjärjestelmästä, tässä tapauksessa MySQL-tietokannasta. Mallin ei täydy olla yksittäinen luokka, joka tekee kaikki tietokantakyselyt, vaan mallin sisällön suunnittelussa ohjelmoijalla on vapaat kädet. [12]

Intranetissä mallin pohjaluokka, josta sovelluksen eri osioiden malliluokat periytyvät, on hyvin yksinkertainen. Mallin perusluokkaan onkin lähinnä määritetty tietokantayhteyden, tietokantaobjektien ja näiden ylläpitämiseen tarvittavat ominaisuudet. Vasta mallin perusluokasta periytettyihin luokkiin on tapauskohtaisesti rakennettu funktiot dataobjektien hakemista ja tallentamista varten.

Malli ei ole yksittäinen luokka joka kuvastaa yksittäistä tietokannan taulua, vaan malli on kirjasto joka tarjoaa sovelluksen osa-aluekohtaisesti tarvittavat funktiot dataobjektien hakemista ja niiden käsittelemistä varten. Dataobjektit ovat Active Record -mallin mukaisesti toteutettuja luokkia, jonka instanssit kuvastavat yksittäistä tietokantataulun riviä [12]. Jokaiselle sovelluksen osa-alueelle on oma kontrollerinsa ja mallinsa ja yleisesti malli käsittelee vain oman osa-alueensa dataobjekteja. Esimerkiksi viikkotiedotteet sovelluksen malli-luokka käsittelee vain viikkotiedotteiden dataobjekteja. Kaikki liikenne tietokannan ja sovelluksen välillä tapahtuu edellä mainittujen dataobjektien välityksellä.

4.2 Näkymä

Näkymä (view) on MVC-arkkitehtuurin osa joka vastaa kaikesta loppukäyttäjälle näytettävästä informaatiosta. Näkymä saa näytettävän datan mallilta ja muotoilee sen näytettävään muotoon käyttäjää varten, useimmiten käyttäen apunaan jotain template-moottoria, kuten esimerkiksi Smartyä. Näkymä ei suorita tieto-

kantakyselyjä tai vastaanota käyttäjältä tulevaa dataa, vaan nämä molemmat hoidetaan mallissa tai kontrollerissa jo ennen näkymän lataamista. Näkymä voi kuitenkin sisältää joitain apufunktioita joita tarvitaan esimerkiksi tiedon näyttämiseen tai muuttamiseen näytettävään muotoon. [5], [11], [12]

Tehdyssä ohjelmistokehyksessä näkymä on toteutettu yhden yksittäisen luokan avulla ja näkymän tietojen näyttämiseen ei ole käytetty erillistä template-moottoria. Näkymä-luokan tärkein metodin on *render*-metodi, joka etsii ja lataa kyseisellä hetkellä ajettavalle kontrollerille ja näkymälle kuuluvan template-tiedoston. Template on tiedosto, johon on määritetty kaikki näkymän ulkoasuun ja parametrien näyttämiseen liittyvät tieto. Useimmiten template-tiedosto sisältää html-koodia, johon on tallennettu ulkoasun näyttämiseen tarvittavan html-koodin lisäksi, myös alueet tai lokerot, joiden tilalla näytetään järjestelmästä saatava tieto.

Jotta kontrollerilta tuleva tieto on mahdollista näyttää, *render*-metodi muuttaa kaikki näkymälle annetut tiedot sellaiseen muotoon jossa ne on mahdollista määrittää template-tiedostoon näytettäväksi. Näkymä on mahdollista asettaa piilotettuun tilaan, jolloin näkymä ei lataa template-tiedostoa tai suorita *render*-metodia. Tällainen piilotettu näkymä on kätevä esimerkiksi lomakkeita käsitteleviss̈ä näkymissä, joissa ei ole tarpeellista näyttää käyttäjälle informaatiota.

View-luokkaan on määritelty myös muutamia muita ohjelmistokehyksen käyttöä helpottavia metodeja, jotka eivät ole järjestelmän toiminnan kannalta pakollisia, mutta jotka tekevät ohjelmistokehyksen käytöstä miellyttävämpää. Yksi esimerkki tällaisista metodeista on *view*-luokan *includeJS*-metodi, jonka avulla on mahdollista ladata useita javascript-kooditiedostoja halutulle näkymälle ja näin tarvittavien javascript-koodien linkittämistä ei tarvitse tehdä template-tiedostoon, vaan järjestelmä hoitaa näiden koodien automaattisen linkittämisen.

4.3 Kontrolleri

Kontrolleri (controller) on järjestelmän ydinluokka jonka tehtävänä on koota kaikki järjestelmän osat yhteen. Kontrolleri vastaa oikean malli ja näkymä -

luokan alustamisesta, käyttäjältä tulevan datan vastaanottamisesta sekä mallin antaman datan muotoilemisesta ja välittämisestä eteenpäin näkymälle. Kontrolleri on siis eräänlainen rajapinta käyttäjän, mallin ja näkymän välillä. Kontrollerissa käsitellään myös kaikki sessioihin liittyvä data. Kontrolleriluokan alustuksen yhteydessä, luokka hakee ja alustaa toimintoa vastaavan näkymän sekä mallin ja tallentaa näiden luokkien instanssin muuttujiin myöhempää käyttöä varten. Käytön kannalta tärkeimmät luokan tapahtumat tehdään siis jo luokan alustuksen yhteydessä. [5], [11], [12]

Kontrollerin suoritettua oman osuutensa sen *destruct* eli luokan lopetusmetodissa on funktio, joka laukaisee kontrolleriluokan alustuksen yhteydessä ladatun näkymäluokan *render*-metodin. Näin jokaiseen kontrolleriluokasta periytettyyn lapsiluokassa ei tarvitse välittää näkymän renderoimiseen liittyvistä toiminnallisuuksista vaan näkymän renderöinti kutsu periytyy tältä perusluokalta.

Näiden toimintojen lisäksi kontrollerilla on muita ohjelmistokehyksen käyttöä helpottavia metodeja, jotka pääsääntöisesti liittyvät käyttäjien, sekä sessioiden hallintaan. Näiden käyttäjähallintaan liittyvien metodien (*isAdmin* ja *isLoggedIn*) avulla on helppoa määrittää näkymäkohtaisesti, että vaatiko kyseiseen näkymään pääsemisen käyttäjän sisäiänkirjautumisen tai mahdollisesti hallinnointioikeudet.

4.4 Oikean kontrollerin ja näkymän lataaminen

MVC-arkkitehtuuriin pohjautuvasta järjestelmästä ei ole hyötyä mikäli järjestelmä ei osaa ladata näitä kolmea perusluokkaa silloin kun niitä tarvitaan. Tätä varten järjestelmässä on oma *Url Dispatcher* –luokkansa, joka hoitaa kontrollereiden, mallien sekä näkymien lataamisen liittyvän motoriikan. *Url Dispatcher* -luokka ottaa url-osoitteesta ladattavan kontrollerin ja näkymän nimen sekä tarvittavat lisäparametrit.

Url dispatcher -luokka suodattaa osoitekentän mahdollisesti haitallisien tietojen varalta, jonka jälkeen se tarkistaa että osoite on syötetty oikein. Järjestelmän käyttämissä osoitteissa ei voi olla erikoismerkkejä ja osoitteen eri muuttuja tule-

vat aina olla eroteltuna kenoviivoilla. Oikein muotoillun osoitteen avulla järjestelmän on mahdollista ladata oikea kontrolleri ja näkymä. Osoitteen ensimmäinen muuttuja on aina ladattavan kontrollerin nimi ja osoitteen toinen muuttuja on ladattavan näkymän nimi. Näiden kahden muuttujan jälkeen tulevat muuttujat järjestelmä syöttää ladattavalle kontrollerille lisäparametreina. Esimerkiksi osoite ”www.domain.fi/users/edit/4/” tarkoittaa että järjestelmä lataa käyttäjät kontrollerin (users), josta kutsutaan käyttäjän editointi näkymää (edit) ja lisäparametrina on annettu numero neljä, jonka järjestelmä syöttää kontrollerille lisäparametrina. Tässä tapauksessa numero neljä tarkoittaa että muokataan käyttäjää jonka ID eli tunnistenumero on 4.

Ohjelma 1 on esimerkki oikean kontrollerin ja näkymän lataamiseen käytettävästä funktiosta.

Ohjelma 1. Esimerkki *Url Dispatcher* –luokan toimintamallista.

```
if (!isset($url)) {
                        $controller = $default['controller'];
                        $action = $default['action'];
            } else {
                        $urlArray = array();
                        $urlArray = explode("/",$url);
                        $controller = $urlArray[0];
                        array_shift($urlArray);
                        if (isset($urlArray[0]) && $urlArray[0] != NULL)
{
                                    $action = $urlArray[0];
                                    array_shift($urlArray);
                        } else {
                                    $action = $default['action'];
                        }

                        if($urlArray && $urlArray[0] != NULL){
                                    $queryString = $urlArray;
                        }
            }

            $controllerName = ucfirst($controller);
            $dispatch = new $controllerName($controller,$action);

            if ((int)method_exists($controllerName, $action)) {

            call_user_func_array(array($dispatch,$action),$queryString)
            } else {
                        trigger_error("Cannot load requested controller",
            E_USER_ERROR);
            }
```

5 INTRANETIN OMINAISUUDET

5.1 Tietokantaluokka

Tietokantaluokka hoitaa kaiken liikenteen tietokannan ja sovelluksen välillä. Kyseiseen luokkaan on määritetty kaikki yleisimmät tietokantaa käytettäessä tarvittavat funktiot, kuten tietokantaan yhdistämien, yhteyden katkaiseminen, sekä eri tietokantakyselyjen tekeminen. Tietokantaluokka toimii rajapintana active record -objektien ja tietokannan välillä. Tietokantaluokan toteutuksessa on käytetty apuna singleton-suunnittelumallia.

Suunnittelumallit ovat pohjaratkaisuja joihinkin yleisimpiin ohjelmointiin liittyviin ongelmiin. Suunnittelumallit eivät perustu pelkästään tiettyyn ohjelmointikieleen, vaan ne ovat loogisen tason ratkaisuja siitä miten jokin asia on mahdollista toteuttaa. Suunnittelumallit eivät välttämättä tarjoa valmista ratkaisua, mutta suunnittelumalleista voi olla apua ongelman ratkaisemisessa. [13]

Singleton-suunnittelumallin avulla on mahdollista luoda, ilman globaalien muuttujien käyttöä, luokka josta on aina vain yksi olemassa oleva instanssi kerrallaan [5], [12]. Koska projektissa ei ole tilanteita jossa on tarvetta useammalle kuin yhdelle tietokantaluokalle samaan aikaan, niin singleton-suunnittelumallin käyttäminen on luonteva vaihtoehto ja tarjoaa helpon tavan tietokantayhteyden hallintaan.

Singleton-suunnittelumallissa luokan rakentajametodi (*constructor*) on määritetty yksityiseksi metodiksi. Tällöin luokkaa ei ole mahdollista alustaa kyseisen luokan ulkopuolelta. Luokan kutsumiseen käytetään rakennusmetodin sijasta *getInstance*-metodia, joka tarkistaa että onko luokasta jo olemassa instanssi, jos on, metodi palauttaa kyseisen instanssin ja mikäli instanssia ei ole vielä luotu, metodi luo uuden instanssin. Tällä tavoin luokan instanssi on aina yksi ja sama riippumatta siitä, missä yhteydessä luokkaa käytetään. [5], [12]

Ohjelma 2 on lyhyt esimerkki siitä miten tietokantaluokassa *getInstance*-metodi on toteutettu.

Ohjelma 2. Tietokantaluokan Singleton-mallin toteutukseen vaadittava *getInstance*-metodi

```
public static function getIstance() {
      if (!isset(self::$instance)) {
         $c = __CLASS__;
         self::$instance = new $c;
      }
      return self::$instance;
}
```

Tietokantaluokan tietokantayhteys on toteutettu PHP-ohjelmointikeilen MySQL Improved eli MySQLi -kirjaston avulla. MySQLi-kirjasto tukee parametrisoituja eli valmisteltuja tietokantakyselyitä. Valmisteltuun tietokantakyselyyn kuuluu kaksi vaihetta. Kyselyn valmistautumisvaihe (*prepare*) ja kyselyn suorittaminen (*execute*). Valmistautumisvaiheessa tietokannalle lähetetään kyselyn rakenne, jotta tietokantapalvelin voi tarkistaa kyselyn rakenteen etukäteen ja varata kyselyn suorittamista varten resursseja. Valmistautumisen jälkeen palvelimelle syötetään suorituskomento, jonka mukana annetaan data valmisteluvaiheessa lähetettyyn kyselyyn. Kun palvelimella on etukäteen tiedossa mitä tietokantatauluja kyselyyn käytetään, se voi hakea tiedot nopeammin.[14], [15]

Parametrisoiduilla tietokantakyselyillä on toinen hyvä puoli nopeuden lisäksi. Parametrisoidut tietokantakyselyt ovat paljon perinteisiä kyselyitä turvallisempia, koska ne varmistavat sen että kyselyn mukana ei ole mahdollista syöttää palvelimelle haitallista dataa [15]. Yksi PHP ja MySQL -teknologioiden suurimpia tietoturvariskejä on tietokannan saastuttaminen syöttämällä tietokantaan sellaista dataa, jonka tietokantaohjelmisto voi tulkita ajettavaksi tietokantakyselyksi. Tällä tavoin tietokannasta voi olla mahdollista saada sellaista tietoa, jota sen ei kuuluisi näyttää tai tietokannasta voi olla mahdollista poistaa tietoja. Parametrisoiduissa kyselyissä tämä ei ole mahdollista, koska parametrisoidun kyselyn ensimmäisessä vaiheessa lähetettävä kyselyn runko määrittää kyselyn rakenteen ja tietokantaohjelmisto toimii vain tämän rakenteen mukaan, eikä sekoita kyselyn sekaan komentoina toisessa vaiheessa lähetettävää dataa.

5.2 Tietokanta-abstraktio ja Active Record -malli

Intranet on suunniteltu kokonaan oliopohjaiseksi, siksi ei ole kannattavaa tehdä tietokannan tietojen hakemista pelkkien yksittäisien tietokantakyselyjen avulla, vaan on jatkon kannalta viisasta käyttää jotain tietokantamallia. Intranetin tietojenhallinnan toteutin käyttämällä Active Record -mallia, jossa jokaista tietokannan riviä kuvastaa yksi oliomuotoinen luokka, jolla on omat metodinsa tietojen hakemiseen ja tallentamiseen [12].

Jokaisella Active Record -luokalle on määritetty yksityisinä muuttujina kaikki luokan sisältämät tiedot. Esimerkiksi käyttäjä luokan muuttujat ovat kaikki käyttäjätietoihin kuuluvat muuttujat kuten, käyttäjätunnus, etunimi, sukunimi ja sähköpostiosoite. Nämä kaikki muuttujat ovat identtisiä tietokannassa olevien sarakkeiden kanssa. Hyviä olio-ohjelmointitapoja noudattaen kaikki luokan muuttujat ovat yksityisiä muuttujia, eikä niihin viitata suoraan, vaan niille on määritetty julkiset *set* ja *get* -metodit muuttujien tietojen käsittelyä varten. Kyseisiin metodeihin on mahdollista sisältää myös tiedon oikeellisuuden tarkistaminen. Mikäli muuttujaan yritetään tallentaa siihen sopimatonta tietoa, luokan on mahdollista antaa siitä virheilmoitus.

Active record -malli rakentuu neljän tietojen käsittelyyn perustuvan perusmetodin ympärille: *Create*, *Read*, *Update* ja *Delete* (CRUD). Nämä neljä metodia ovat perusedellytykset, jotka vaaditaan kaikilta tietokantaa käsitteleviltä ohjelmilta. [12]

Save-metodi tallentaa luokan instanssin sisältämät tiedot tietokantaan. Kyseinen metodi tarkistaa onko luokan tiedot jo tietokannassa olevaa tietoa vai onko luokan instanssi kokonaan uusi tietoalkio. Mikäli kyseessä on uusi tietoalkio, metodi tallentaa instanssin tietokantaan uutena rivinä ja muussa tapauksessa metodi päivittää jo olemassa olevan tietokantarivin tietoja. Luokan *delete*-metodi poistaa luokan instanssin mukaisen alkion tietokannasta luokan tunnisteen perusteella. Edellä mainittujen metodien lisäksi luokilla on yleensä omia tietojen hakemiseen liittyviä metodeja. Esimerkkinä *loadByID*-metodi, joka nimensä mu-

kaan hakee tietokannasta metodille syötetyn tunnistenumeron perusteella oikean instanssin tiedot.

Vaikka Active Record -luokat kuvastavat tietokannan tietoja ja niiden avulla haetaan ja tallennetaan tietoa kannasta, kyseiset luokat eivät siitä huolimatta sisällä tietokannan käyttämiseen tarvittavia perustoimintoja. Tietokantakyselyjen lähettämiseen, tietokantayhteyden luomiseen ja sulkemiseen liittyvät toiminnallisuudet on jätetty erillisen tietokantaluokan hoidettavaksi.

5.3 Ladattavat tiedostot

Yleisesti kun tiedosto tallennetaan Internet-palvelimelle, tiedosto on mahdollista ladata suoralla url-osoitteella, mikäli se on tiedossa. Esimerkiksi ”http://www.esimerkki.fi/tiedostot/dokumentti.doc”-osoitteen kautta olisi mahdollista esimerkki.fi -verkkotunnuksen alla sijaitsevalta palvelimelta dokumentti.doc niminen tiedosto. Mikäli mitään lisäestoa ei ole tiedoston sisältävään kansioon tai tietoja syöttävälle Internet-palvelimelle määritetty, tuo kyseinen tiedosto on kaikkien Internet-käyttäjien saatavilla. Kaikki intranetin sisältämät tiedostot, lukuun ottamatta julkisen kansion tiedostot, ovat salassa pidettäviä. Tästä syystä tiedostojen on oltava vain niiden ladattavissa, jotka ovat kirjautuneet sisään.

Järjestelmässä on tiedostojen palvelimelle lähettämistä ja sieltä lataamista varten oma tiedostojenhallintaluokka. Tiedostojen lataamista ja tallentamista varten luokalla on oma metodinsa, joka käsittelee kaiken dokumentteihin, henkilökuviin ja ladattaviin tiedostoihin liittyvän liikenteen. Metodi tarkistaa sille annetusta tiedoston nimestä, onko kyseinen tiedosto olemassa palvelimella. Tämän jälkeen se muodostaa kyseisen tiedoston koon, tiedostoformaatin ja nimen perusteella uuden http-kutsun, jonka vastaanottajan selain tulkitsee ladattavaksi tiedostoksi. Tiedostojenhallintaluokan ei anna siis tiedoston suoraa osoitetta palvelimella, vaan se hakee kyseisen tiedoston informaation ja tarjoaa tiedoston omana http-kutsuna. Palvelimella olevia tiedostoja ei ole mahdollista ladata aiemmin puhutun suoran osoitteen kautta, koska palvelimelle määritetyt htaccess-asetukset estävät tiedostojen suorien linkkien toimivuuden.

Käyttäjän saaman tiedoston latausosoite ei ole sama osoite kuin missä tiedosto sijaitsee, vaan käyttäjä saa palvelimelta vain binäärimuotoista dataa, jonka käyttäjän Internet-selain tulkitsee ladattavaksi tiedostoksi järjestelmän antamien http-header tietojen perusteella [16]. Kaikkien sisäisien tiedostojen lataaminen tapahtuu suoraan PHP-ohjelmakoodin kautta, jolloin järjestelmään on mahdollista valvoa, onko tiedostoa lataava henkilö kirjautunut sisään ja onko kyseisellä henkilöllä valtuuksia ladata tiedostoa.

Tärkeimmät kohdat http-headerin muodostamiseen tarvittavista PHP-ohjelmakoodista. Ensimmäinen rivi määrittää annettavan tiedoston formaatin. Toinen rivi kertoo, että palvelimelta tuleva data on liitetiedosto, ja minkä niminen kyseinen liitetiedosto on. [16], [17]

Ohjelma 3. Tärkeimmät ladattavaan tiedoston http-kutsun muodostamiseen tarvittavat komennot.

```
header("Content-type: $filetype");
header("Content-Disposition: attachment;filename=$filename");
```

5.4 Äänestyssovellus, viikkotiedotteet sovellus ja puhelinluettelo

Kaikki järjestelmän eri sovellukset koostuvat useista eri näkymistä. Useimmat näkymistä ovat tietojen näyttämistä varten listamuodossa, kuten viikkotiedotteiden listaus, henkilötietojen listaamiset, äänestystuloksien listaaminen. Toinen yleinen näkymätyyppi on yksittäisen tietoalkion näyttäminen, kuten viikkotiedotteen tai käyttäjätietojen lukeminen. Tämäntyyliset näkymät ovat yksinkertaisia, ja ne käytännössä vain hakevat käyttäjältä tulevan syötteen tai ladatun näkymän lisäparametrien perusteella tietoa tietokannasta, muotoilevat tiedon halutuun muotoon ja näyttävät sen käyttäjälle. Moniin tietoa näyttäviin näkymiin liittyy pelkän tiedon näyttämisen lisäksi usein myös muita lisätoimintoja, jotka vaihtelevat ajettavasta näkymästä riippuen.

Tiedon näyttämiseen tarkoitettujen perusnäkymien lisäksi, intranetissä on useita muokkausnäkymiä, joissa muokataan tietoalkiota tai joiden kautta on mahdollista luoda uusi tietoalkio. Tällaisia muokkausnäkymiä ovat mm. käyttäjätietojen

muokkaaminen, viikkotiedotteen muokkaaminen, äänestyksen muokkaaminen tai tiimin tietojen muokkaaminen. Kaikki muokkausnäkymät kuuluvat intranetin hallintaosioon, ja siksi vain hallintakäyttäjillä on pääsy näihin näkymiin.

Kahden edellisen näkymätyypin lisäksi intranetissä on tiedon käsittelyyn ja tallentamiseen tarkoitettuja näkymiä. Näiden näkymien tehtävänä on vastaanottaa muokkausnäkymistä tai esimerkiksi kommentointilomakkeen kautta tullut tieto ja ajettavasta näkymästä riippuen, joko suodattaa ja tallentaa kyseinen tieto tietokantaan tai tiedostoon, tallentaa tietoa sessioon tai lähettää näkymälle annettu tieto sähköpostilla. Tietoa käsittelevät näkymät eivät näy käyttäjälle, vaan ne antavat tehdystä toimenpiteestä sessioiden avulla palautetta takaisin siihen näkymään, josta tieto on tullut, tai ohjaavat käyttäjän edelleen toiseen näkymään.

Näiden kaikkien näkymätyyppien lisäksi järjestelmä sisältää myös ajax-käyttöön tarkoitettuja näkymiä, joihin käyttäjä ei pääse suoraan, vaan niistä haetaan tietoa ajax-komentojen avulla. Ajax-komennot ovat a-synkronoituja javascript-pyyntöjä, joiden avulla on mahdollista hakea jo ladatulle sivulle sisältöä dynaamisesti [18].

Kaikki intranetin sovellukset sisältävät tiedon näyttämiseen, tallentamiseen ja muokkaamiseen tarkoitettuja näkymiä, jotka eroavat toisistaan lähinnä näkymän lataaman template-tiedoston ja käsiteltävien tietojen osalta. Tästä syystä en perehdy tarkemmin jokaisen sovelluksen näkymien toimintaan erikseen.

Alla on viikkotiedotteet sovelluksen kontrolleriluokan *open*-näkymän metodi, joka avaa kyseiselle toiminnolle parametrina syötetyn tunnisteen mukaisen viikkotiedotteen. Metodi tarkistaa ensin luokan yläkontrolleriluokasta periytyneen metodin avulla, onko käyttäjä kirjautunut sisään. Tämän jälkeen kontrollerin *addJavaScript*-metodin avulla määritetään *deleteconfirm.js*-niminen javascript-komentoja sisältävä tiedosto ladattavaksi näkymälle. Kyseinen metodi syöttää vain halutun javascript-tiedoston nimen eteenpäin näkymälle, minkä jälkeen näkymä osaa automaattisesti liittää kyseisen tiedoston ladattavaksi lopulliseen käyttäjälle näytettävään näkymään. Edellä mainittujen alkutoimenpiteiden jäl-

keen *open*-funktio hakee sille annetun tunnisteen perusteella mallilta viikkotiedoteobjektin ja *set*-metodilla syöttää sen näkymälle jatkotoimenpiteitä varten.

Ohjelma 4. Viikkotiedotteet sovelluksen *open*-näkymä.

```
function open($id) {
           $this->userAuth();
           $this->addJavaScript("deleteconfirm.js");
           $id = (int) $id;
           $briefingContent = $this->model->getBriefing($id);
           $this->set("briefing", $briefingContent);
   }
```

6 TIETOTURVA

6.1 Salasanat

Tietokannan yksi tärkeimmistä ja samalla myös arimmista tiedoista on käyttäjien salasanat. Salasanojen tallentaminen tietokantaan, tai johonkin muuhun tallennusformaattiin on kuitenkin pakollista ohjelman toimivuuden sekä käyttäjätilien mahdollistamisen kannalta. Käyttäjien salasanoja ei kuitenkaan koskaan tule tallentaa tietokantaan pelkkänä tekstinä, jolloin kuka tahansa, joka pääsee käsiksi tietokannan tietoihin, pystyy katsomaan käyttäjien salasanat. Salasanat täytyy aina tallentaa kryptatussa muodossa, jolloin salasanojen luettavaan muotoon on vaikeampaa ja oikealla kryptaamisalgoritmilla jopa mahdotonta. Intranetin tapauksessa salasanat tallennetaan tietokantaan kahteen kertaan kryptattuna, sekä salasanaan lisätään kryptaamisien välissä salt-sana, jolla tarkoitetaan ylimääräisiä merkkejä, joita ripotellaan salasanan sekaan sen takaisin luettavaan muotoon saamisen vaikeuttamiseksi[19]. [20]

6.2 Virheiden hallinta

Intranetin käytön yhteydessä on mahdollista että ohjelmakoodin suorituksen aikana tulee vastaan ohjelmavirheitä. Myös intranetin toteutuksessa käytetty PHP-ohjelmointikieli ilmoittaa käyttäjilleen, mikäli ohjelman ajon aikana ilmenee virhe ja nämä virheilmoitukset ovat tietoturvan kannalta huomioon otettava riski. PHP-ohjelmointikielen oletusvirheilmoitukset saattavat näyttää sellaista informaatiota, jota käyttäjien ei pitäisi nähdä, kuten sovelluksen lähdekoodia, muuttujan arvoja tai jotain muuta paljastavaa tietoa, josta mahdolliset sivustolle murtautumista yrittävät henkilöt saattavat hyötyä.

Tähän tarkoitukseen intranetissä on virheidenhallintaluokka, jonka tehtävänä on kaapata järjestelmän antavat virheilmoitukset ja ilmoittaa niistä virheen vakavuudesta riippuen sähköpostitse sivuston ylläpitäjälle, tallentaa virheilmoitukset

virhelokiin, sekä piilottaa sivustoa käyttävältä henkilöltä kaikki haitallinen tieto virheilmoitukseen liittyen. PHP-ohjelmointikielessä on useita eritasoisia virheitä, joiden avulla on mahdollista erotella virheilmoitukset toisistaan virheen vakavuuden mukaan. Kaikkein kevyin PHP:n virheilmoitus on E_NOTICE, jonka tapauksessa virheidenhallintaluokka ei lähetä sähköpostia tai tallenna virheen tietoja virhelokiin. Kaikkiin muihin virheilmoituksiin virheidenhallintaluokka reagoi tavalliseen tapaan. [3]

Virheidenhallintaluokkaan on mahdollista asettaa *debug* eli ohjelmointivirheidenetsintätila, jolloin virheidenhallintaluokka ei lähetä sähköpostia ylläpitäjälle, vaan se näyttää suoraan kaiken järjestelmän virheilmoituksen antaman tiedon ruudulla. Se että virheidenhallintaluokka lähettää sähköpostia ylläpitäjälle virhetilanteen sattuessa helpottaa mahdollisten ohjelmointivirheiden paikantamisen. Ilman kyseistä ominaisuutta vastaan tulleet virhetilanteet näkyisivät vain käyttäjän tietokoneella, eikä sivustoa ylläpitävä henkilö saisi niitä tietoon.

Virheidenhallintaluokassa toteutuksessa on käytetty PHP-ohjelmointikielen *set_error_handler* ja *set_exception_handler* -funktioita, joiden avulla on mahdollista ohjata järjestelmän antamat virheilmoitukset, sekä kaappaamattomat virheet suoraan käyttäjän määrittämälle funktiolle [3]. Tällä tavoin on mahdollista ohittaa PHP-ohjelmointikielen oletusvirheilmoituksien tulostuminen käyttäjälle ajon aikana ja määrittää jokin muu tapa käsitellä virheilmoitukset.

6.3 Istuntojen hallinta

Istunnot ovat yksi tapa välittää tietoja sivuston eri osioiden välillä. Istunnon tiedot tallentuvat sivuston www-palvelimelle, josta ne välitetään jokaisen sivulatauksen yhteydessä sivuja selaavan henkilön tietokoneeseen. Istunnot ovat aina yksilöityjä, eli kaksi käyttäjää ei voi jakaa samaa istuntoa. Istunnot erotellaan toisistaan istuntotunnisteen avulla. Jokaista istuntoa kohti palvelintietokoneen ja käyttäjän tietokoneen välistä istuntoa yhdistää satunnaisesti luotu istuntotunnus, joka säilyy samana koko istunnon ajan, mikäli järjestelmää ei pakoteta uusimaan istuntotunnusta. [3], [20]

Vaikka istunnot ovat helppo ja käytännöllinen tapa hallita käyttäjän kirjautumisia, niin istunnoiden käyttämisessäkin on omat riskinsä. Vaikka istunto on palvelimen ja asiakastietokoneen välinen, niin se on siitä huolimatta mahdollista kaapata, jos jokin kolmas osapuoli saa selville kyseisen istunnon tunnuksen. Tämän hankaloittamiseksi istuntoihin on asetettu istunnon vanhenemisaika jolloin aikaikkuna, jonka aikana kolmas osapuoli pystyy istunnon kaappaamaan, jää pienemmäksi. Jokaisen näkymän latauksen yhteydessä järjestelmä tarkistaa, onko sessioon tallennettu aika ylittänyt määritetyn session enimmäisaikarajan. Mikäli aikaraja ei ole ylittynyt järjestelmä päivittää sessioon tallennetun ajan ja muussa tapauksessa kirjaa käyttäjän pihalle. Järjestelmä myös uusii istuntotunnisteen aina käyttäjän kirjautuessa sisään ja ulos, jolloin käyttäjän ja palvelimen välinen istuntotunnistetta ei ole mahdollista kaapata ennen sisälle kirjautumista tai uloskirjautumisen jälkeen. [21], [22]

6.4 Käyttäjän syöttämien tietojen oikeellisuus

Tietojen suodattaminen on yksi tärkeä osa web-sovelluksen tietoturvassa. Nyrkkisääntönä on että kaikki tieto mikä otetaan käyttäjältä vastaan palvelimelle, on aina suodatettava tiedon tyyppiin sopivalla tavalla [22]. Tietojen suodattamisella yritetään estää se, ettei käyttäjien ole mahdollista syöttää haitallista tietoa, kuten PHP-ohjelmakoodia tai MySQL komentoja palvelimelle, ja tällä tavoin rikkoa web-sovellusta tai päästä käsiksi arkaluontoiseen tietoon, kuten käyttäjätunnuksiin. Suodatettavaa tietoa on esimerkiksi kaikkien sovelluksessa olevien lomakkeiden lähettämä tieto tai lomakkeiden kautta sivustolle ladattavat tiedostot.

Haitallisen PHP-ohjelmakoodin suodattamiseen sovelluksessa on käytetty kyseisen ohjelmointikielen tarjoamia suodatusfunktioita, jotka käytetyistä funktion lisäparametreista riippuen poistavat syötteestä kaikki erikoismerkit, lisäävät haitallisten erikoismerkkien eteen kenoviivan tai puhdistavat syötteestä kaikki merkit lukuun ottamatta numeroita. Suodatusfunktiolle annettavat lisäparametrit määritetään sen mukaan, minkä tyyppistä tietoa halutaan suodattaa. [3]

Tietokannan tietojen turvaaminen on varmistettu käyttämällä aikaisemmin mainittu parametrisoituja SQL-kyselyjä, jotka estävät haitallisen tiedon syöttämisen tietokantaan.

Toinen osa tietojen oikeellisuuden varmistamisessa tiedon suodattamisen lisäksi on tiedon oikeellisuuden tarkistaminen. Koska ei ole mahdollista varmistua siitä, että mitä tietoa ja missä muodossa käyttäjät syöttävät järjestelmään, niin kaikelle syötetylle tiedolle on syytä tehdä vähintään karkea tarkistus, että onko tieto oikean tyyppistä. Yksinkertainen esimerkki mahdollisista tiedon tarkistamisesta on sähköpostiosoite. Sähköpostiosoitteet ovat aina muotoja etuosa@verkkotunnus.pääte jossa etuosa, verkkotunnus ja pääte voi vaihdella. Tämän tyyppinen tieto on helppo tarkistaa esimerkiksi käyttämällä säännöllistä lauseketta (Regular Expression). Sen avulla pystyy vertaamaan, löytyykö syötetystä tiedosta @-merkki, onko ennen @merkkiä muita merkkejä ja löytyykö @-merkin jälkeen merkkejä, jotka päättyvät johonkin mahdolliseen päätteeseen. Sähköpostiosoitteiden lisäksi myös muullekin tekstimuotoiselle tiedolle on mahdollista tehdä oikeellisuuden tarkistaminen. Onko puhelinnumerossa mahdollista olla muita merkkejä kuin numeroita, sallitaanko käyttäjätunnukseen erikoismerkkejä, onko tiedolla jotain minimipituutta? Onko jokin tieto ylipäätänsä pakollinen lomakkeen kannalta vai onko käyttäjän mahdollista jättää tieto antamatta ja kenttä tyhjäksi?

Tietojen suodattaminen ei rajoitu pelkästään käyttäjän lomakkeiden kautta syöttämään tekstimuotoiseen tietoon. Koska Intranetissä on myös järjestelmää hallinnoivalla henkilöillä mahdollista ladata jokaiselle käyttäjälle profiilikuva, joka näkyy mm. puhelinluettelon tiedoissa, niin on myös otettava huomioon tiedostojen oikeellisuuden tarkistaminen. Tässä tapauksessa järjestelmä sallii ainoastaan .jpg-, jpeg-, .gif- sekä .png -muotoisien kuvatiedostojen lataamisen palvelimelle. Mikäli käyttäjä lataa jonkin muuta muotoa olevan tiedoston, järjestelmä antaa tästä virheilmoituksen, sekä poistaa palvelimelle ladatun virheellisen kuvatiedoston. Koska kuvatiedostoja on monen kokoista ja muotoista, kuvan lataamisen yhteydessä järjestelmä myös muuttaa kuvan leveyden ja korkeuden sopivaksi järjestelmän tietoihin sekä muuttaa tiedoston nimen sellaiseksi, mikä

on helpommin tunnistettavissa myöhemmin. Kuvatiedoston lataamisessa palvelimelle ja kuvakoon muuttamisessa on käytetty Michael Diamondin tekemää *PHP-imageuploader* -luokkaa [23].

6.5 Tietoturvan testaaminen

Järjestelmän tietoturvan kehittämisessä yksi tärkeä osa-alue on tietoturvan testaaminen. Ei ole suositeltavaa ottaa järjestelmää käyttöön ja luottaa että tehdyt tietoturvaan liittyvät ratkaisut olisivat pitäviä ilman että niitä on millään tavalla testattu. Tietoturvan testaaminen siihen tarkoitetuilla ohjelmilla ei takaa että järjestelmä olisi täysin turvallinen, mutta kyseiset ohjelmat ovat hyviä työkaluja mahdollisien aukkojen etsimisessä [24].

Tietoturvan testaamiseen käytin kolmea eri tietoturvaskanneria, jotka skannaavat sivuston rakennetta ja eri hyökkäysmenetelmiä ja metodeja käyttäen yrittävät löytää sivustosta tietoturvaongelmia. Käyttämäni tietoturvaskannerit eivät tutki varsinaista ohjelmakoodia. Skannerit testaavat sivustoa mm. tietokantainjektioiden, Cross-site Scripting -hyökkäyksien ja hakemistorakenteeseen liittyvien hyökkäyksien varalta.

Käytin testaamiseen kolmea eri skanneria. Wapiti on Python-ohjelmointikielellä toteutettu tietoturvaskanneri, joka testaa sivustoa tiedostonkäsittelyvirheiden, tietokantainjektion, Cross Site Scripting, LDAP ja CRLF -aukkojen osalta. Ohjelma antaa myös virheilmoituksen, mikäli testattava järjestelmän osa sallii tiedostojen lataamisen palvelimelle tai palauttaa käsittelemättömän virheilmoituksen palvelimelta (Palvelinvirhe koodilla 500). [25]

Uniscan on toteutettu Perl-ohjelmointikielellä ja on samantyylinen ominaisuuksiltaan sekä toiminnoiltaan Wapitin kanssa, sillä erolla että Uniscan on suunniteltu pääsääntöisesti piilotettujen tiedostojen ja hakemistojen etsimiseen [26]. W3AF on avoimen lähdekoodin www-tietoturva hyökkäys ja auditointi ohjelmistokehys, johon on mahdollista asentaa useita erilaisia lisämoduuleita [27].

Tietoturvaskannereiden antamat tulokset vastasivat pääsääntöisesti järjestelmälle asettamiani odotuksia. Wapiti ja Uniscan ohjelmilla testatessa en antanut kyseisille skannereille lähtötietoina intranetiin kirjautumiseen vaadittavia tietoja vaan asetin skannerit tutkimaan intranetin sisään kirjautumisosiota. Kumpikaan näistä skannereista ei löytänyt järjestelmästä yhtään tietoturvariskiä.

Kolmas käyttämistäni tietoturvaskannereista W3AF löysi järjestelmästä kaksi mahdollista tietoturvauhkaa, joista toinen oli intranetin antamasta palautteesta, ohjelman väärin tulkitsema uhka, mutta toinen ohjelman löytämistä aukoista on huomioonotettava tietoturvauhka. Ohjelma havaitsi että, intranetin sisään kirjautumisen käsittelevässä funktiossa ei ole estoa Cross-Site Request Forgery eli CSRF-hyökkäyksille.

CSRF-hyökkäyksessä annetaan lomakkeen tietoja käsittelevälle funktiolle käsiteltäviä arvoja jotain muuta kautta kuin varsinaisen lomakkeen läpi. Tavallisesti html-lomakkeen lähettäminen toimii siten että lomakkeeseen syötetyt tiedot lähetetään *POST* tai *GET* -muotoisena http-kutsuna eteenpäin lomakkeen tietoja käsittelevälle funktiolle joka käsittelee kyseisen informaation. Mikäli lomakkeen käsittelevälle funktiolla ei ole tarkistusta siitä onko funktion saama data tullut varmasti oikealta lomakkeelta, on funktiolle mahdollista syöttää tietoa muun lähteen kuin oikean lomakkeen kautta. [22],[28]

Yksi ratkaisu löydetyn aukon korjaamiseen on asettaa jokaiseen intranetin sisältämään lomakkeeseen satunnainen tunniste, lomakeavain, joka vaihtuu jokaisen lomakkeen lataamisen yhteydessä. Sama lomakeavain tallennetaan lomakkeen luomisen yhteydessä myös sen hetkiseen käyttäjän sessioon. Kun lomakeavain on tallennettuna sekä lomakkeen lähetettäviin tietoihin että sessioon, on lomaketta käsittelevässä funktiossa mahdollista verrata näitä kahta avainta toisiinsa. Mikäli avaimet eivät ole yhteneviä, on pääteltävissä, että lomakkeen käsittelijälle tullut data ei ole tullut oikean lomakkeen kautta. [22],[28]

7 JATKOKEHITTÄMINEN

Active record -suunnittelumalli on hyvä ratkaisu pienempiin ja yksinkertaisiin sovellustoteutuksiin sen yksinkertaisuuden ansioista. Active record -mallissa on kuitenkin myös huonot puolensa. Joissain tapauksissa active record -luokkien funktioiden määrä voi kasvaa todella suureksi [12]. Esimerkiksi viikkotiedotteet luokka sisältää muuttujien get- ja set-metodien sekä save-, loadbyid- ja delete-metodien lisäksi useita muita metodeja, joita käytetään viikkotiedotteiden hakemiseen tietokannasta jollain muilla hakuperusteilla kuin tunnuksen mukaan.

Toinen active record -mallin huono puoli on se, että yksi active record -luokka kuvastaa aina vain yhtä tietokantataulun riviä. Tämä ei ole ongelmana tapauksissa, joissa käsitellään vain yhtä tietoalkiota kerrallaan, mutta tapauksissa, joissa pitäisi muokata tai noutaa useita tietokantarivejä kerrallaan, Active record -mallin käyttäminen tietokanta-abstraktioon ei ole kaikkein tehokkain ratkaisu.

Tämän takia järjestelmän tietojenkäsittelyn toteuttamiseen olisi voinut valita jonkin Active Record -mallia monipuolisemman ja kehittyneemmän mallin, jonka avulla useiden tietoalkioiden hakeminen tietokannasta olisi tehokkaampaa. Kehittyneempiä ja monipuolisempia malleja active record -mallin tilalle ovat muun muassa Table Data gateway tai Data Mapper -malli.

Table Data Gateway -malli on perusidealtaan hyvin samantyyppinen active record -mallin kanssa. Mutta sen sijaan että malli kuvastaisi yksittäistä taulun riviä, Table data gateway -malli kuvastaa kokonaista tietokannan taulua. Tällöin isompien tietomäärien käsitteleminen on tehokkaampaa. Table Data Gateway -objektit sisältävät samat tiedon käsittelyyn tarvittavat perusfunktiot kuin active record -mallin mukaiset objektitkin eli *create, read, update* ja *delete* (CRUD). Table Data gateway -malli on kuitenkin hyvin samantyylinen Active Record -mallin kanssa, jolloin kyseisen mallin käytössä tulisi ennemmin tai myöhemmin vastaan samantyylisiä ongelmia kuin active record -mallia käytettäessä. [12]

Data mapper -luokka on erillinen kerros tietokannan sekä tietoluokkien välissä. Data mapper -luokan tehtävä on noutaa tiedot tietokannasta ja syöttää ne tietoluokkien käytettäväksi kyseisen luokan vaatimalla tasolla. Kun tietokantayhteyden ja tietoja käyttävien luokkien välissä on erillinen kerros, joka hoitaa tiedon hakemisen ja tallentamisen, on tietokantaan ja varsinaiseen ohjelmistoon muutoksien tekeminen myöhemmässä vaiheessa helpompaa, koska järjestelmän tietoluokat eivät ole suoraan sidottuina tietokannan rakenteeseen. [12]

Data Mapper -luokan käyttäminen tietokanta abstraktion luomisessa olisi todennäköisesti ollut jatkon kannalta hyvä vaihtoehto, mikäli intranetin ominaisuuksia kasvatetaan nykyisestään paljolti. Data mapper -luokka helpottaisi järjestelmän tietorakenteiden kehittämistä myöhemmässä vaiheessa. Data Mapper -luokan rakentaminen on kuitenkin paljon active record -mallin mukaisien luokkien rakentamista monimutkaisempi ja vaikeampi. Tästä syystä mikäli rakennettavasta järjestelmästä ei ole suunniteltu kasvavan kovin suurta, voi jonkin yksinkertaisemman, kuten Active Record –mallin, käyttäminen olla virtaviivaisempi tapa toteuttaa järjestelmän tietokantatoiminnallisuus.

Edelliset kaksi esille tuomaani vaihtoehtoa tietokanta abstraktion toteuttamiseksi eivät ole ainoat mahdolliset vaihtoehdot, mutta ne yksiä yleisimpiä malleja joita käytetään tietokannan kuvaamiseen [11].

Yksi myös tietoturvaa ja käytettävyyttä parantava jatkokehitettävä ominaisuus on käyttäjien salasanan automaattinen uudelleenluominen. Tällä hetkellä mikäli joku intranetin käyttäjistä unohtaa salasanansa, hänen täytyy ottaa johonkin hallintoryhmään kuuluvista käyttäjistä yhteyttä ja pyytää häntä lähettämään hänelle uudelleenluotu salasana. Ylläpitäjien ei ole mahdollista lähettää käyttäjälle hänen nykyistä salasanaansa, koska salasanat ovat tallennettuna kryptattuna tietokantaan, mutta heidän on mahdollista luoda käyttäjälle uusi salasana.

Nykyisen toimintamallin sijaan olisi kätevämpää toteuttaa salasanan uusiminen automaattisen kaavakkeen kautta, jonne käyttäjä voisi syöttää oman käyttäjätunnuksensa sekä sähköpostiosoitteensa. Tämän jälkeen mikäli käyttäjätunnus

ja sähköpostiosoite löytyvät tallennettuna järjestelmän tietoihin, se lähettäisi käyttäjälle sähköpostilla uuden automaattisesti generoidun salasanan.

8 YHTEENVETO

Työn tuloksena saatu Intranet on otettu yrityksessä säännölliseen käyttöön ja sen kehittämistä jatketaan edelleen. Intranet vastasi toiminnallisuuksiltaan ja ominaisuuksiltaan sen lähtömäärityksiä, ja se on toteutettu siten, että jatkokehittäminen on mahdollista käyttöönoton jälkeenkin.

Intranetin toteuttamiseen kehitetty oma MVC-ohjelmistokehys vastasi hyvin sille suunniteltuja määrityksiä. Suuri osa ohjelmistokehykseen tehdyistä ominaisuuksista ja muutoksista on kehitetty intranetin rakentamisen yhteydessä ja siksi intranetin ja ohjelmistokehyksen yhteistoiminta on sulavaa. Myös ohjelmistokehyksen kehittäminen jatkuu edelleen. Siihen on tarkoitus rakentaa vielä kattavampi määrä toiminnallisuuksia, ja muokata siitä vielä valmiimpi kokonaisuus, jotta sen käyttöönottaminen muihin projekteihin olisi luontevampaa.

Valmiin ohjelmistokehyksen käyttäminen projektissa olisi varmasti nopeuttanut projektin loppuun saamista, mutta ei pidä myöskään unohtaa oman ohjelmistokehyksen rakentamisen tuomaa kokemusta.

LÄHTEET

1. Oy Käännös- ja Tulkkauspalvelu Mokoma Ab, [www-dokumentti], Saatavilla: http://www.mokoma.fi (Luettu: 12.9.2012)

2. Forsen Marja, henkilökohtainen tapaaminen, 12.2.2012

3. The PHP Group, [www-dokumentti], Saatavilla: http://www.php.net (Luettu: 12.9.2012)

4. Oracle Corporation, [www-dokumentti], Saatavilla: http://www.mysql.com/why-mysql/marketshare (Luettu: 12.9.2012)

5. McArthur Kevin, Pro PHP Patterns, Frameworks, Testing and More, Amerikan yhdysvallat, Apress, 2008

6. PHPframeworks.com, [www-dokumentti], Saatavilla: http://www.phpframeworks.com/ (Luettu: 20.1.2013)

7. Merkel Dirk, Expert PHP 5 Tools, Yhdistynyt kuningaskunta, Packt Publishing, 2010

8. Sharp Josh, [www-dokumentti], Saatavilla: http://joshsharp.com.au/blog/view/why_you_should_be_using_a_framework (Luettu 12.9.2012)

9. LearnComputer, [www-dokumentti], Saatavilla: http://www.learncomputer.com/why-use-php-framework (Luettu 12.9.2012)

10. Nishizawa Michael, [www-dokumentti], Saatavilla: http://checkedexception.blogspot.fi/2010/04/why-you-should-not-use-web-framework.html

11. Fowler Martin, Patterns of Enterprise Application Architecture, Addison Wesley, 2002

12. E. Sweat Jason, PHP|Architect's Guide to PHP Design Patterns, Kanada, Marco Tabini & Associates, 2005

13. Zandstra Matt, PHP Objects, Patterns, and Practice – Second Edition, Amerikan yhdysvallat, Apress, 2008

14. The PHP Group, [www-dokumentti], Saatavilla: http://www.php.net/manual/en/mysqli.overview.php (luettu: 21.1.2012)

15. The PHP Group, [www-dokumentti], Saatavilla: http://php.net/manual/en/mysqli.quickstart.prepared-statements.php (luettu: 21.1.2012)

16. Boutell.Com, [www-dokumentti], Saatavilla: http://www.boutell.com/newfaq/creating/forcedownload.html

17. The PHP Group, [www-dokumentti], Saatavilla: http://php.net/manual/en/function.header.php (luettu: 21.1.2012)

18. Refsnes Data, [www-dokumentti], Saatavilla: http://www.w3schools.com/ajax/ajax_intro.asp (luettu: 2.2.2013)

19. The PHP Group, [www-dokumentti], Saatavilla: http://php.net/manual/en/faq.passwords.php#faq.passwords.hashing (luettu: 11.2.2013)

20. Curioso Andrew, Bradford Ronald ja Galbraith Patrick, Expert PHP and MySQL, Wiley Publishing Inc. 2010

21. PHP Security Consortium, [www-dokumentti], Saatavilla: http://phpsec.org/projects/guide/4.html (luettu: 11.2.2013)

22. MacIntyre Pete, Danchilla Brian ja Gogala Mladen, Pro PHP Programming, Amerikan yhdysvallat, Apress, 2011

23. [www-dokumentti], Saatavilla: http://www.digitalgemstones.com/code/tools/ImgUploader.php, 2013

24. The Web Application Security Consortium, [www-dokumentti], Saatavilla: http://projects.webappsec.org/Web-Application-Security-Scanner-Evaluation-Criteria (Luettu: 11.2.2013)

25. Wapiti – Web Application vulnerability scanner, [www-dokumentti], Saatavilla: http://wapiti.sourceforge.net/ (Luettu: 11.2.2013)

26. [www-dokumentti], Saatavilla: http://uniscan.sourceforge.net/ (Luettu: 21.1.2013)

27. W3AF, [www-dokumentti], Saatavilla: http://w3af.org/ (luettu: 21.1.2013)

28. The Open Web Application Security Project, [www-dokumentti], Saatavilla: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF