OULUN SEUDUN
AMMATTIKORKEAKOULU

Pyry Pennanen

SÄHKÖISEN ILMOITUSTAULUN TOTEUTTAMINEN CODEIGNITER-OHJELMISTOKEHYKSELLÄ

SÄHKÖISEN ILMOITUSTAULUN TOTEUTTAMINEN CODEIGNITER-OHJELMISTOKEHYKSELLÄ

Pyry Pennanen
Opinnäytetyö
Syksy 2011
Tietojenkäsittelyn koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä: Pyry Pennanen
Opinnäytetyön nimi: Sähköisen ilmoitustaulun toteuttaminen Codeigniter -ohjelmistokehyksellä
Työn ohjaaja: Jouni Juntunen
Työn valmistumislukukausi ja -vuosi: Syksy 2011 Sivumäärä: 34

Työn tarkoituksena on tutkia web-sovelluskehityksessä yleisesti käytettäviä tekniikoita, sekä luoda näitä yhdistelemällä edullinen ja käytännöllinen sähköinen ilmoitustaulu. Sähköinen ilmoitustaulu on tietokonejärjestelmä, joka mahdollistaa käyttäjien välisen julkisten viestien lisäämisen ja katselemisen. Sovelluksen kohderyhmänä ovat organisaatiot, joilla IT-resurssit ylläpidon ja koulutuksen osalta ovat vähäiset.

Sovelluksessa yhteen sovitettavat Codeigniter -sovelluskehys, JavaScript -ohjelmointikieli ja Ajax-tekniikka muodostavat kokonaisuuden, jolla voidaan tuottaa laadukkaita selainkäyttöisiä sovelluksia web-ympäristöön. Nämä tekniikat ovat hyvin laajasti tuettuja ja tekevät sovelluksesta lähes alustariippumattoman. Valittujen tekniikoiden laadukkaan soveltamisen varmistamiseksi työssä tutkittiin niiden historiaa, perusarkkitehtuureja ja hyviä käytäntöjä. Tekniikoiden tutkimisessa painottuu niiden tuoma lisäarvo etenkin käytettävyydelle kehittäjän, ylläpitäjän tai käyttäjän näkökulmista. Tällaiset järjestelmät toimivat yleisimmillä web-selaimilla ilman ylimääräisiä liitännäisiä ja pystyvät kuitenkin tarjoamaan samanlaisen käyttökokemuksen kuin paikallisesti asennetut ohjelmat. Ne pystytään myös sijoittamaan kevyille palvelimille, joiden ylläpidon tarve on vähäinen.

Työn tuloksena muodostunut sähköinen ilmoitustaulu on kevyt ja käytännöllinen web-pohjainen sovellus, jolla on helppo toteuttaa ja ylläpitää tiedotteita. Kehittämisen yhteydessä tutkittujen tekniikoiden ja tapojen avulla on mahdollista nopeuttaa web-sovelluskehitystä kuitenkaan vaarantamatta työn laatua. Niiden toimintamallien ymmärtäminen antaa hyvän vertailukohdan myös muiden ohjelmistokehitysratkaisujen hahmottamiseen ja auttaa toivottavasti tulevaisuudessa uusien toimintatapojen omaksumisessa.

Asiasanat: verkko-ohjelmointi, Ajax-ohjelmointi, käyttäjäkokemus, käytettävyys, Codeigniter, sovelluskehykset.

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Technology

Author: Pyry Pennanen
Title of thesis: Building electronic bulletinboard using Codeigniter-framework
Supervisor: Jouni Juntunen
Term and year when the thesis was submitted: Fall 2011 Number of pages: 34

The aim of this work was to study web-development and to create advantageous and usable electronic bulletin board by combining different techniques. Electronic bulletin board is a computer system which enables users to publish and view public messages. The target group consisted of organizations which do not have large and specific IT-resources for tutoring and management.

A combination of Codeigniter web application framework, JavaScript programming and Ajax-technology comprise an environment where it is possible to create high-quality web-based software with greater focus on business logic. These techniques are widely supported and hold minimal limitations for the target platform. For ensuring implementation quality of chosen techniques this thesis also examined their origins, base architectures and best practices. The thesis mainly focused on added value especially regarding usability for the end-user, administrator and developer. Systems described with these specifications should work on common browsers without third party add-ons or modules and are still able to provide similar user experience to locally installed software. These systems can also be installed on lightweight web-servers which require very little maintenance.

As the result, in the practical part of the thesis, the electronic bulletin board system built is lightweight and practical web-based software which can easily establish public exchange of notifications. With the help of studied techniques and patterns, creation of such systems can be speeded up without jeopardizing quality. Furthermore, understanding the usage of them provides concrete comparison points for evaluating other similar tools and thus will hopefully help adopting to new environments.

Keywords: Ajax programming, web programming, user experience, usability, application framework.

SISÄLLYS

1 KEHITYSPROJEKTIN LÄHTÖKOHDAT

Suorittaessani työharjoittelua Kaakkurin yhtenäisperuskoulussa tuli vastaan Tamperelaisen Avack yhtiön info-sovellus, jonka tarkoitus ihmetytti katsellessani pimeinä roikkuvia tv-vastaanottimia. Selvitettyäni asiaa löysinkin sähköpääkeskuksesta pölyä keräävän vanhentuneen työpöytäkoneen, johon sovellus oli asennettu. Ohjekirjojen ja sovelluksen hankintaan osallistuneiden henkilöiden haastatteluiden perusteella järjestelmä vaikutti aivan liian hankalalta käyttää siitä saataviin potentiaalisiin hyötyihin verrattuna. Sovellus oli ilmeisesti ollut vielä tuhansien eurojen arvoinen, mikä oli surullista, koska sitä ei ollut käytetty päivääkään. Koulun valmiiksi rakennettu, mutta käyttämätön näyttöverkko houkutteli testaamaan niitä passiivisesti toimivina web-sovellusnäyttöinä. Harjoittelun aikana kehitin koululle paremmin tiedotustarvetta palvelevan prototyypin nykyaikaisesta sähköisestä ilmoitustaulusta PHP- ja JavaScript-tekniikoilla. Kehitetty järjestelmä sisälsi selkeän ja pelkistetyn ilmoitustaulun, joka esitti muutamia tekstimuotoisia ilmoituksia kerrallaan niitä automaattisesti iteroiden.

Prototyyppiä rakentaessa osoittautui se olevan oivallinen alusta tutkia erilaisia web-ohjelmoinnin tekniikoita ja apukeinoja. Tässä opinnäytetyössä on rakennettu teknisesti parempi ja helpommin käytettävä versio prototyypin pohjalta. Erityinen painoarvo työssä oli oppia tekijälle aiemmin tuntemattomien web-kehityselementtien, kuten MVC-sovelluskehyksen ja Ajax-tekniikan käyttöä. Itse sovelluksessa näitä lisäarvoa tuottavia elementtejä käytetään ja sovelletaan niiltä osin kuin se on mahdollista.

Sovelluksen kohderyhmänä olivat ICT-alaan kuulumattomat keskisuuret organisaatiot, kuten esimerkiksi koulut. Tällaisilla organisaatioilla ICT-toiminnot ovat usein löyhästi organisoituja ja mahdolliset harvat nimetyt vastuuhenkilöt hoitavat niitä yleensä sivutoimisesti. Erityisesti tällaisissa organisaatioissa voi olla hyötyä hyvin tarkasti kohdennetuista ja pelkistetyistä IT-ratkaisuista, sillä oikein suunniteltuna niiden varaamat hallinnolliset ja käyttöä tukevat resurssit on mahdollista pitää hyvin alhaisina.

1.1 Sähköinen ilmoitustaulu

Yleensä sähköiset ilmoitustaulut toimivat osana laajempaa tietojärjestelmää, jolloin vältytään erilliseltä hallinnoinnilta. Perinteisesti ne ovat olleet organisaation hallinnon ylläpitämiä tai vapaita kaikkien jäsenten käyttöön tarkoitettuja viestintä järjestelmiä. Sähköisillä sovelluksilla on monia etuja liittyen tiedon ja metatiedon tekniseen hallintaan, mutta perinteisen fyysisen ilmoitustaulun edut käytettävyydessä ja saatavuudessa ovat selkeät.

Tässä työssä toteutettu web-tekniikkaan perustuva sähköinen ilmoitustaulu pyrkii tuottamaan mahdollisimman selkeän ja helppokäyttöisen selaimen välityksellä toimivan sovelluksen. Web-tekniikka mahdollistaa käytännössä laajimman mahdollisen sovellusalustan, eli sovellusta voi käyttää miltä selaimen sisältävältä laitteelta tahansa. Web-tekniikoista ohjelmointiin käytetään JavaScript- ja PHP-kieliä, koska niillä saavutetaan riittävä toiminnallisuuden ja käytettävyyden taso kuitenkaan vaarantamatta saatavuutta. Vaihtoehtona olevat web-tekniikat vaativat yleensä lisäosia selaimeen tai vaativamman palvelinympäristön.

Saatavuus ja käytettävyys ovat ilmoitustaulun tärkeimpiä ominaisuuksia toiminnoille ja ne pyrittiin ottamaan huomioon työssä kauttaaltaan ympäristöstä ulkoasuun. Siksi ydintoimintoja painotetaan selkeästi, eli sovelluksella esitetään ja hallitaan tiedotteita tietoverkossa ja ominaisin tapa toimia tällaiselle sovellukselle on käyttämällä selain ja palvelin web-ympäristöä.

1.2 Toiminnalliset vaatimukset

Tiedon hallinnointia varten toteutetaan luku-, lisäys-, muokkaus- ja poista-toiminnot (eng. lyh. CRUD). Yksittäinen tiedote koostuu otsikosta ja tekstistä. Lisäksi siihen voidaan liittää metatietoa, kuten ilmoituksen esiintymisajan aloitus- ja lopetuspäivät. Tiedon hallinnointi toimintojen tulee olla helposti käytettävissä, ilman että käyttöliittymää kuormitetaan turhilla elementeillä. Esittämiseen sovelluksen tulee näyttää ne järjestelmällisesti korkeintaan kolme tiedotetta kerrallaan. Tiedotteen esittämiseen vaadittavan ja käytössä olevan näytön pinta-alan huomioiden kerrallaan voidaan niitä esittää rajallinen määrä. Tiedotteisiin tapahtuvat muutokset tulee välittyä esityksiin automaattisesti ilman keskeytystä. Nämä tiedotteisiin kohdistuvat käsittelyt sovelluksessa tulee toteuttaa Ajax-toiminnoilla käytettävyyden parantamiseksi.

Tiedotteet koostuvat erilaisista elementeistä, joilla merkitys ohjelmassa vaihtelee hieman esimerkiksi muotoilun tai käsittely tarpeen mukaan. Elementtien käsittelyerot ovat kuitenkin vähäiset ja yhdistettynä CRUD-toimintoihin kertoutuu niistä monipuolinen, mutta hallittavissa oleva kokonaisuus. Tällainen vaihteleva, mutta rajattu toimintojen määrä vaatii tutkimisessa usean näkökulman huomioon ottamista ilman, että vaihtoehtoisten ratkaisujen tutkiminen ja testaaminen aiheuttaisi kohtuuttomasti työtä.

Näiden ydintoimintojen lisäksi sovelluksessa ilmenee muutamia tukitoimintoja, kuten kirjautuminen ja navigaatio. Toiminnot rajattiin käytettävyyden suhteen toissijaisiksi, eikä niihin tämän vuoksi sovellettu kehittyneempiä selain-tekniikoita. Toiminnallisuuden kannalta ne kuitenkin ovat sovellukselle tärkeitä ja tulee toteuttaa omina moduuleina.

1.3 Ei toiminnalliset vaatimukset

Sovelluksen käytettävyyttä tulee huomioida myös hallinnollisesta näkökulmasta. Kaikenlaisia riippuvaisuuksia tekniikoihin tulee välttää, jos ne vaativat kaupallisen lisenssin alaisia liitännäisiä tai eivät ole sulautettavissa sovellukseen kiinteästi. Tällä pyritään helpottamaan sovelluksen käyttöönottoa erilaisilla selainpohjaisilla alustoilla. Teknisten ratkaisujen lisäksi riippuvaisuudet tulee lisenssien osalta rajoittua avoimeen lähdekoodiin. Toteuttamalla sovelluksen palvelintoiminnot PHP-kielellä ilman riippuvaisuuksia erillisiin järjestelmiin, kuten tietokanta-ohjelmistoihin on se mahdollista asentaa yleisimpiin web-palvelimiin. Samoin se pystytään siirtämään tiedostonhallinnan avulla sovelluskohtaiset tiedot säilyttäen ilman erillistä asennusohjelmaa.

Sovelluksenkehityksessä käytetään avoimen lähdekoodin työkaluja vapaamman käytön mahdollistamiseksi. Sovelluksen liiketoimintalogiikka tulee toteuttaa ohjelmistokehyksen sisällä, jotta kehitystoiminnassa pystytään keskittymään ydintoimintoihin ja tukitoimintoihin hyödyntäen paljolti yleisesti saatavilla olevia valmiita kirjastoja. Ohjelmistokoodin kirjoittamisessa tulee pyrkiä noudattamaan yleisiä hyviä ohjelmointitapoja ja -malleja sen myöhemmän tulkinnan ja muokkaamisen mahdollistamiseksi.

2 WEB-SOVELLUSKEHITYS

Web-sovellus on asiakas-palvelinsovellus, joka yleensä käyttää selainta asiakassovelluksen alustana. Selaimet lähettävät pyyntöjä palvelimille ja palvelimet muodostavat vastauksia, jotka palautetaan selaimille. Web ja selain -tekniikkojen merkitys universaalina sovellusalustana on suuri ja kasvaa edelleen niiden laajan ja syvenevän sulautuneisuuden myötä. (Rosen & Shklar 2003, 202.)

2.1 Web-tekniikoiden historiaa

HTML syntyi osana World Wide Webbiä 1980–1990 –lukujen vaihteessa Euroopan hiukkasfysiikan tutkimuslaitoksessa (CERN). Tutkimuslaitoksessa työskentelevä Tim Berners-Lee ehdotti, että kehitettäisiin internetiä hyväksikäyttävä hyperteksti-järjestelmä. Järjestelmän tarkoituksena oli helpottaa tiedon hallintaa laitoksessa, jossa tutkijoita oli tuhansia aina muutamia vuosia kerrallaan. Berners-Leen johtajan, Mike Sendallin mielestä tämä "Epämääräinen mutta innostava" ehdotus oli laajemman tutkimisen arvoinen ja sen kehitystyö sai luvan jatkua. Alkuperäinen ehdotus tiedon käsittelystä sisälsi tutkimuslaitoksen sisäisiä kriteereitä, kuten muun muassa ehdon käytetyn järjestelmän riippumattomuuden suhteen, hajautetun systeemin mallin sekä Ted Nelsonin määrityksen mukaisen lukukelpoisen tiedon rajoittamattoman keskinäisen linkittämisen. (Berners-Lee 1989, hakupäivä 2.10.2011.)

Ensimmäinen WWW-sivun määritelmä oli kuvaus HTML:stä ja sen elementeistä, joilla selaimet muodostavat tulkinnan dokumentista luettavassa muodossa. Kuvauksessa kerrotaan kyseessä olevan malli, joka sisältää pääasiassa asiakirjan muotoiluun tarvittavat elementit. Dokumentti oli aluksi lineaarinen (suoraviivainen) ennalta määritettyjen solmujen (elementti puujärjestelmässä) sarja, eikä sallinut nykyisin standardeja rakenteellisia sisäistettyjä solmuja. (Berners-Lee 1991, hakupäivä 2.10.2011.)

Internetin käyttö keskittyi aluksi yksisuuntaiseen tiedon lukemiseen. Lukeminen toimi yhden suhde yhteen kardinaalisuuden periaatteella, jolloin yhtä pyyntöä kohden saatiin käsiteltäväksi yksi tiedosto. Pyynnön perusteella palvelin etsi kyseisen tiedoston ja lähetti tämän selaimelle,

joka muodosti staattisen tulkinnan tiedoston sisällöstä. Teknologian kehittyessä tuli mahdolliseksi kahteen suuntaan toimiva tiedon lähetys ja sen prosessointi. Käytännössä se tarkoitti aluksi lomakkeiden keräämien tietojen lähettämistä ja käsittelyä. (Darlington 2005, 186.) Lomakkeita pystyttiin liittämään web-sivuihin ensimmäisen kerran Mosaic-selaimen 2.0alpha1 versiossa, joka julkaistiin 31.1.1994 (NCSA 1994, hakupäivä 11.12.2011).

1993 luodun CGI:n (Common Gateway Interface) tarkoituksena oli luoda dynaamisuutta internetiin sallimalla käyttäjän pyyntöjen perusteella suoritettavat ohjelmat palvelimilla. Niiden avulla pystyttiin tarjoamaan muun muassa yhteys tietokantoihin ja erilaiset lomakkeiden käsittelyt. CGI altistaa palvelimet kuitenkin mahdollisille suorille hyökkäyksille ei-toivottujen ohjelmakutsujen kautta. (Asleson & Schutta 2006, 4.)

CGI rajapinnan kautta web-palvelimet pystyvät kutsumaan niille avoimia kolmannen osapuolen ohjelmia (Coar & Robinson 2004). Käännettyjen ohjelmien ja käyttöjärjestelmä skriptien lisäksi alettiin käyttää tulkattavia ohjelmia, joiden kehittäminen oli nopeampaa. Eräs Linux-järjestelmävalvojien siihen aikaan suosima tulkattava kieli oli Perl, joka saattoi vaikuttaa sen suosioon CGI-ohjelmointikielenä (Summers 2008, hakupäivä 30.11.2011). Aluksi pienenä kävijälaskuri-ohjelmana syntynyt PHP oli perinteinen käännetty CGI-ohjelma, mutta myöhemmin se liitettiin suoraan web-palvelimen moduuliksi. Sisäänrakennetun moduulin etu erillisiin ohjelmiin ja skripteihin (tulkattava ohjelma) oli se, että sitä ei tarvinnut erikseen käynnistää, käsitellä erillisellä virtuaalikoneella tai muutenkaan varata ylimääräisiä resursseja. PHP-kielen ympäristönä toimivalle Zend-moottorille tuli nopeasti kilpailevia kaupallisia ohjelmistopalvelimia kuten nykyään Adoben omistama Coldfusion, Microsoftin alun perin IIS-lisämoduulina toiminut ASP ja Sun Microsystems yhtiön JSP-tekniikka. JSP-tekniikan lanseerauksen yhteydessä Sun Microsystems julkaisi sille määrityksiä ja käyttötekniikoita, jotka voidaan katsoa käsittelevän ensimmäisen kerran MVC-mallin arkkitehtuurista sovellutusta web-ympäristöön (Bedell & Turner 2002, 20–21).

Palvelimen ja asiakkaan muodostamassa järjestelmässä yhteys keskitettyyn tietovarastoon on melkein aina palvelinsovelluksen tehtävä. Palvelinsovellus tekee yhteydestä turvallisemman, sillä sitä voidaan hallita huomattavasti selainta tarkemmin. Palvelinsovellus on hallitumpien rajapintojensa vuoksi myös soveltuvampi huolehtimaan käyttäjäntunnistamisesta ja yhteyden turvallisuudesta, sekä monista hajautetun verkon palvelemiseen tarvittavasta toiminnasta, kuten asiakassovelluksen ympäristöön perustuva tiedonmuotoilu. Vaikka palvelin-sovellukset ovat tehokkaita, turvallisia ja kykeneviä interaktiiviseen toimintaan, eivät ne pysty kuitenkaan

toteuttamaan korkeampaa interaktiotasoa tai graafisten elementtien dynamiikkaa. (Powell 2003, 406–408.)

Macromedia on julkaissut ensimmäisenä rikkaan internet sovelluksen (RIA) termin ja määritykset, jossa esitetään rikkaan selaintekniikan pystyvän "tarjoamaan taloudellisen ja tehokkaan ympäristön suoritettavalla koodille, sisällölle ja tiedonsiirrolle". Tekniikan tuli myös sisällyttää niiden rajapinnat yleisesti saataville sekä muun muassa tarjota voimakkaita ja laajennettavia oliomalleja interaktiivisuuden luomiseksi. (Allaire 2002, hakupäivä 6.11.2011.) Selainohjelmistot toteuttavat tärkeitä toimintoja, kuten tarkistavat lomakkeita vähentäen palvelimien rasitusta, sekä muokkaavat käyttöliittymää graafisen tai laskennallisen työn esittämiseksi. Haasteellisesta ympäristöstä huolimatta selaimella toimivia komponentteja oikein yhdistämällä on mahdollista luoda merkittävää lisäarvoa ja käytännöllisyyttä. (Powell 2003, 471–473.)

Palvelimien varassa ollut web-sovellusten dynamiikka ei ollut mitenkään käyttökokemukseltaan verrattavissa paikallisesti työasemilla ajettaviin ohjelmiin. Erityisen ongelman aiheutti työskentelyn katkonaisuus, jonka sovelluksessa etenemiseen käytetty sivujen lataaminen aiheutti. Käyttökokemus vaikuttaa muun muassa suorituksen aikaiseen keskittymiskykyyn (Forlizzi & Ford 2000, hakupäivä 11.12.2011).

JavaScript-ohjelmointikieli julkaistiin 1995 ja sillä pystyttiin liittämään toiminnallisuutta staattisiin web-sivuihin (Crockford 2007). Kielen tuoma ominaisuus vaikuttaa web-sivuun interaktiivisesti suoraan käyttäjän selaimella mahdollisti osittain keskeytymättömän käyttökokemuksen. Tarve edelleen lähestyä paikallisten työasemasovellusten käyttökokemusta kuitenkin säilyi, sillä edelleen jokainen palvelimelle lähetetty pyyntö keskeytti työskentelyn. Jokaista pyyntöä seurasi myös esteettisesti häiritsevä näytön päivitys, jonka ratkaisemiseen kehitettiin monia tekniikoita. Viimeisimpänä nykyään yleisesti tuettuna standardina julkaisi Microsoft 1999 XMLHttpRequest-teknologian (Asleson & Schutta 2006, 6–14.)

Selaimella tapahtuvaan laskentaan kehitettiin myös muita tekniikoita, kuten Shockwave, joka keskittyi multimedian käsittelyyn sekä Java Virtual Machine-alustan tuki, jolla pystyttiin ajamaan useiden eri kielien varaan rakennettuja ohjelmia (Elia 1996, hakupäivä 2.11.2011; Engel 1999, 2). Myös Microsoft kehitti oman ActiveX-tekniikkansa, jolla pystyttiin sisällyttämään toiminnallisuutta web-sivuihin. ActiveX-komponentteja voidaan liittää web-sivujen mukaan käyttöjärjestelmään asennettaviksi ohjelmiksi ja niitä voidaan tehdä useilla ohjelmointikielillä, kuten C++-kielellä. (Cluts 1996. hakupäivä 2.11.2011.) Nämä tekniikat rakentuvat kuitenkin selaimiin liitettävien lisäosien varaan, eivätkä ne siten ole oletusarvoisesti "saatavilla".

2.2 Sovelluskehykset

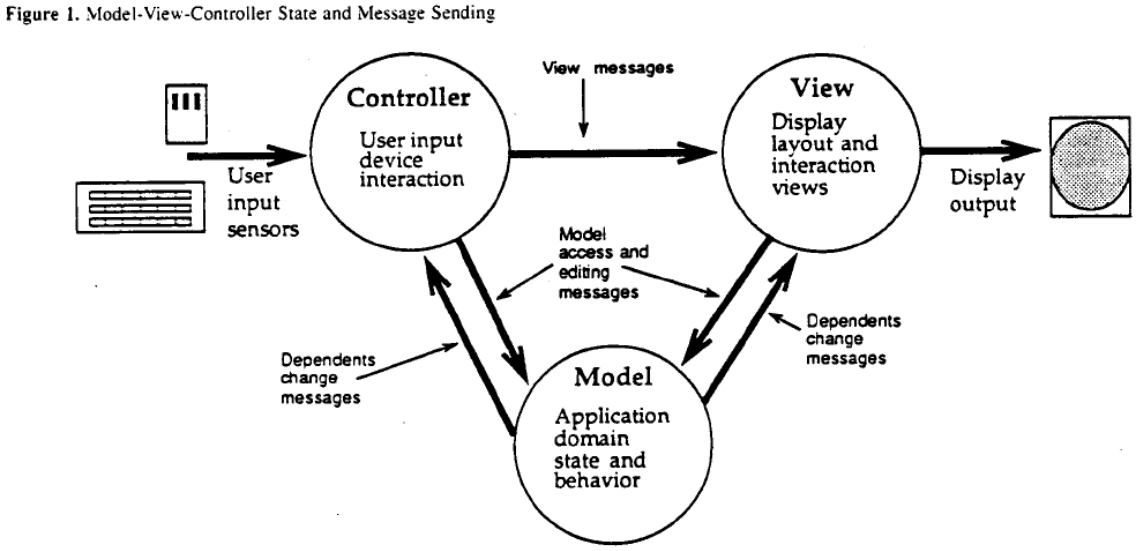
Sovelluskehykset ovat valmiita osia sisältäviä paketteja käytettäväksi nopeassa ohjelmistokehityksessä. Sovelluskehykset eroavat kirjastoista niiden ollessa valmiiksi olemassa oleva osa sovellusta, joka vaikuttaa kehittäjän rakentamiin moduuleihin. Kirjastot tarjoavat enemmänkin liitettäviä moduuleita kehitettävän sovelluksen päälle. MVC-malli on sovelluskehyksissä hyvin laajasti sovellettu arkkitehtuuri, tosin kehykset sitoutuvat noudattamiinsa arkkitehtuureihin eri vahvuuksilla. Zend ja CodeIgniter muun muassa ovat löyhästi sitoutuneita kehysrakenteita. Käytännössä se tarkoittaa niiden tuoman kehitysnopeuden olevan alhaisempi, mutta joustavuuden parempi. (Porębski, Przystalski & Nowak 2011, 2–3.)

MVC-malli syntyi Xerox PARC:lla vuonna 1978 yleisenä vastauksena ongelmaan, jonka muodostivat käyttäjät sekä suuret ja monimutkaiset tietokannat. Sen tarkoituksena alun perin oli tukea käyttäjän muodostamaa mielikuvaa oleellisesta tiedosta sekä mahdollistaa tämän tiedon tarkastelu ja muokkaaminen hänelle. (Reenskaug 2007.) MVC-malli on hyvin looginen ja toimiva modulaarinen ryhmittely, joka auttaa selkeyttämään ohjelmien toimintaa rajaamalla selkeimmin itsenäiset tehtävät. Moduulien keskinäisissä suhteissa on vaihtelua riippuen sovelletusta teknologiasta.

Daniel Suthers (2001) esittää kontrollerin olevan ainoastaan väline, jolla käyttäjä muokkaa mallin tilaa, ja näkymä on mallin tilan muutoksen luoma tuloste. Qian, Fu ja Tao näkevät MVC:stä olevan useita versioita ja sen kuuluvan Interaction-Oriented Software Architectures pääluokkaan. MVC-I on yksinkertaistettu versio, jossa kontrolleri ja näkymä ovat yksi hallinnoinnin superolio ja malli toteuttaa kaiken tiedon käsittelemiseen liittyvän toiminnallisuuden. MVC-II erittelee kaikki kolme oliota taas omiksi yksiköikseen, jotka pystyvät kommunikoimaan toisilleen suoraan. Näiden lisäksi on olemassa vielä kolmas aliluokka PAC (Presentation Abstraction Control), jossa kaikki toiminnot ja liittymät ovat agentteja jotka sisältävät lähes MVC-mallin mukaiset komponentit. (Qian, Fu & Tao 2009, 199–214.) Passiivisen näkymänmalli on Martin Fowlerin MVC-variaatio, jossa poistetaan riippuvuudet mallin ja näkymän väliltä. Näkymä ei tunne mallin olemassa oloa, eikä siten myöskään reagoi siellä tapahtuneisiin muutoksiin. Hän esittää tämän mallin etujen keskittyvän helpompaan testaamiseen, koska siinä näkymä ei suorita minkäänlaista synkronisaatiota. (Fowler 2006a, hakupäivä 11.11.2011.) RIA ja muihinkin web-sovelluksiin on esitetty myös omat lisämoduulit, kuten Presentation-model ja Front Controller, joiden tarkoitus on

tukea hajautettujen järjestelmien rakentamista (Fowler 2006b, hakupäivä 4.12.2011; Greer 2007, hakupäivä 4.12.2011).

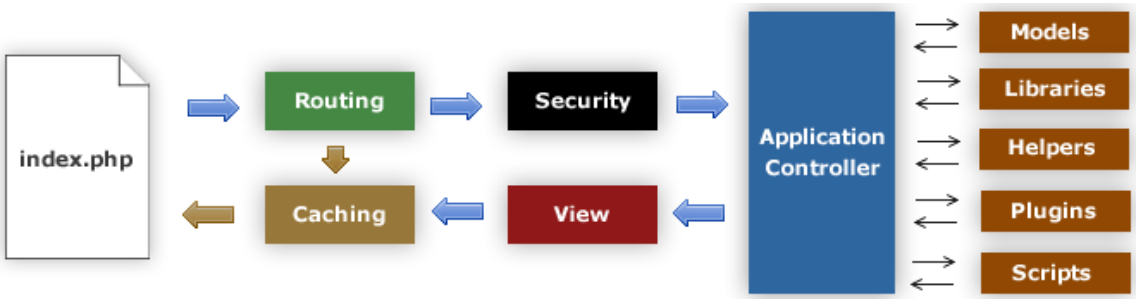
Xerox PARC:lta eriytyneen ParkPlace Systems, Inc yhtiön Smalltalk toteutti MVC-ohjelmointi mallia ja metodologiaa ohjelmointikielenä. Kieli korosti mallin tuomaa etua koodin kierrätettävyyttä ja modulaarisuuden luomaa liitettävyyttä valtavana etuna. (Krasner & Pope 1988) Kuvassa Smalltalk-80 -ohjelmointiympäristön mukainen moduulien välisiä viestejä tulkitseva kaavio (Kuvio 1.)



KUVIO 1. MVC-viestintä Smalltalk-80 -ohjelmointiympäristössä

Mallista johdettujen versioiden lisäksi sen painotuksessa on vaihtelua. Cunningham & Cunningham esittää parhaan tavan jakaa laskentavastuuta olevan kuvattu määrittelyllä "We need SMART Models, THIN Controllers, and DUMB Views". Heidän mielestään mallin suuri haaste on estää kontrolleria kehittymästä kaikenkattavaksi jumala-moduuliksi. (Cunningham & Cunnigham 2011, hakupäivä 26.10.2011.) Steve Jobsin perustaman Next:n MVC tutkijat keskittyivät pienentämään kontrollereiden taakkaa siirtämällä näkymille vastuuta käyttäjän syötteiden seuraamisesta. Näin kehittyi keino seurata hiiren liikkeitä ja napinpainalluksia. (Myer 2008, 9.)

Tässä työssä on käytetty avoimen lähdekoodin CodeIgniter PHP sovelluskehystä, joka perustuu MVC-malliin. Codeigniter pakottaa sovelluksen kontrolleri ja näkymä toteutukseen url-reitityksen kautta, mutta mallin käyttäminen ei ole välttämätöntä. (ExpressionEngine Dev Team 2011, hakupäivä 22.9.2011.) Codeigniter noudattaa passiivisen näkymän-mallia, jolloin kontrolleriin sisältyy kaikki toiminnallisuus mallin kanssa. Kuviossa 2 on kuvattu CodeIgniter-sovelluskehyksen implementaatio MVC-mallista, jossa tukitoiminnot esitetään samalla tasolla MVC-moduulien kanssa.



KUVIO 2. Codeigniterin tietovuokaavio

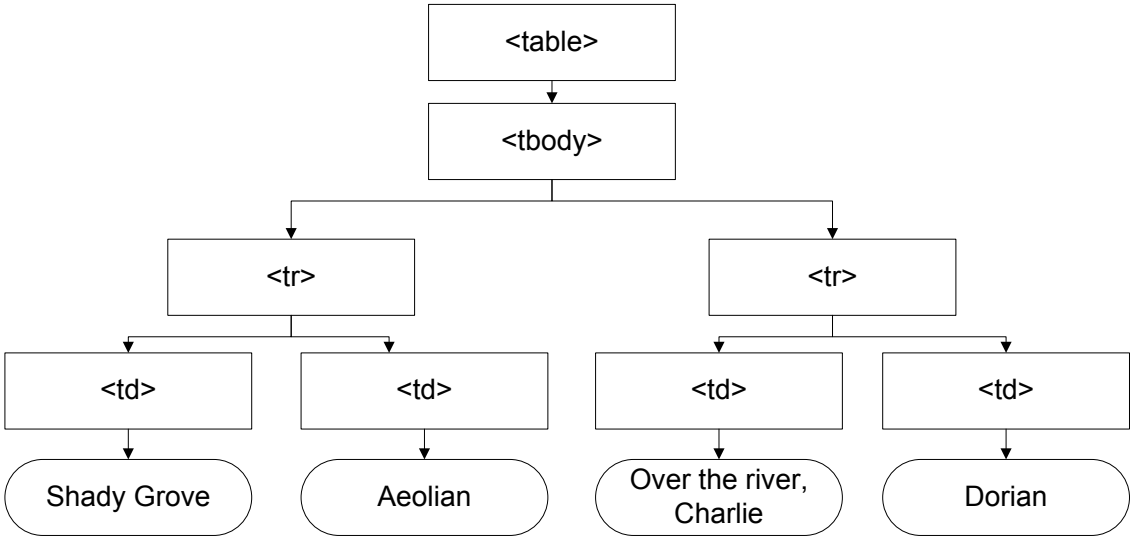
PHP-ohjelmointikielen kehittäjä Rasmus Lerdorf kokee kehysjärjestelmät kauttaaltaan hyvin raskaiksi joka vaikuttaa paljolti kykyyn palvella kyselyjä nopeasti. Hänen omat manuaalisesti säädetyt kyselynsä Twitter mashup sovellukseen palauttivat 280 tulosta sekunnissa kun Apachen puhdas HTML-vastausnopeus oli samassa ympäristössä 600 vastausta sekunnissa. Hänen testaamat PHP-pohjaiset kehysrakenteet toimivat vaihtelevasti nopeuksilla 120 - 8 tulosta sekunnissa. (Peterson 2008, hakupäivä 26.10.2011.) Codeigniter sijoittuu kuitenkin nopeudeltaan kehysrakenteiden kärkeen (Golovanov 2011, hakupäivä 26.10.2011).

2.3 JavaScript ja DOM

JavaScript ja DOM (Document Object Model) ovat kaikilla yleisimmillä web-selaimilla suositusten mukaisesti tuettuja tekniikoita, jotka useasti liitetään toisiinsa. Niiden yhdistelmä mahdollistaa web-sivujen ulkoasun muokkaamisen selainikkunassa dynaamisesti ja on perusta suurelle osalle web-sovellusten käyttöliittymistä.

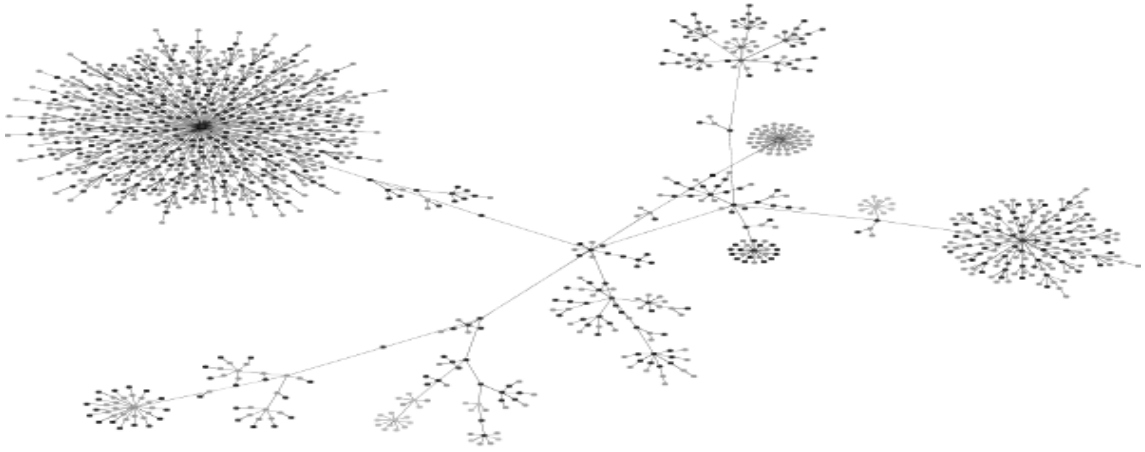
W3C on määritellyt DOM-tekniikan hyvin kattavasti ehdotuksessaan selainten tavasta muotoilla dokumentteja. Ohjelmoijan kannalta DOM voidaan ajatella rajapintajärjestelmänä, jonka avulla mahdollistetaan yhteys sovelluksen esittämään dokumenttiolioon. (Hégaret, Whitmer & Wood 2009.) Rajapintajärjestelmän ydin muodostuu kolmesta kerroksesta, jotka kukin kerrallaan laajentavat sen toiminnallisuutta ja kokoa (Richards 2006, 181). Dokumenttiolio syntyy, kun HTML-määritelmän mukaisesti muotoiltu tietokokonaisuus käsitellään selaimessa web-sivuksi. Tämä muodostettu olio sijoitetaan esitettäväksi seuraavaksi luotavaan ikkunaolioon, joka edustaa selaimenikkunaa omine metodeineen. Aktiivinen dokumentti elää ja on muokattavissa dynaamisesti sen solmuja manipuloimalla. (Keith & Sambells 2010, 31.)

DOM-dokumentteja puuna kuvattaessa jokaista yksittäistä kohtaa kutsutaan solmuksi. Solmu ja sen määritykset on oikeastaan koko rajapintajärjestelmän ydin ja kaikki muut solmutyypit ovat sen perillisiä. Solmu luokan määrityksiä ovat muun muassa lapset, vanhemmat, sekä edeltävä ja seuraava sisar, mikä riittää selvittämään mikä on sen relaatio suhteessa koko dokumenttiin. (Richards 2006, 181 - 200.) Kuviossa 3 DOM-puussa esitetään HTML-kielen taulukkoa vastaava rakenne jonka kaikki <td>-elementit sisältävät tekstisolmun.



KUVIO 3. DOM-puu

DOM-järjestelmän haasteena on nykyisen standardin solmutyyppien rajallisuus. Solmutyyppien määrä on verrannollinen tarvittavien solmupuiden syvyyteen, jolloin puurakenne tulee raskaaksi lukea. Puujärjestelmän syvyys vaikuttaa näin laskentatehon ja muistin vaatimuksiin. Tulevaisuudessa HTML5-standardi voi litistää DOM-puuta. (Irish 2009, hakupäivä 2.11.2011.) Kuviossa 4 näkyy Java-sovelmalla generoitu näkymä oamk.fi sivuston opiskelija intran DOM-puusta.



KUVIO 4. Oamkin opiskelija intran DOM-puu.

JavaScript on web-ohjelmointiin suuntautunut laajasti tuettu tulkattava ohjelmointikieli. Sen syntaksi perustuu Javaan mutta muuta yhteistä näillä ei nimeä lukuun ottamatta ole. JavaScript toimii selaimissa ilman kääntämistä, koska se suunniteltiin web-sivuihin integroitavaksi. Javascriptin kaikki muuttujat ovat olioita ja se tukee prototyyppistä perimistä, joka tarkoittaa, että oliot periytyvät suoraan olioista. Prototyyppisyyden vuoksi olioita on myös mahdollista muokata milloin tahansa ohjelmassa esimerkiksi lisäämällä niihin ominaisuuksia. Nämä ominaisuudet periytyvät myös tämän mahdollisille lapsille. (Crockford 2007.)

Uusien web-tekniikoiden, kuten HTML5 ja muun muassa sen WebGL-rajapinnan vaikutuksesta JavaScript moottorit ja selaimet ovat muuttuneet hyvin vahvoiksi ohjelmointiympäristöiksi. Edistykselliset esimerkit näiden tekniikoiden mahdollisuuksista, kuten Google Maps, Flickr, ja Amazonin "Search Inside This Book" ovat nostaneet JavaScriptin suosiota huomattavasti. Suurta suosiota seurannut sovelluksien nopea kehitystahti sisältää paljon ratkaisuja, joissa tunnetut mallit ja hyvät tavat on unohdettu. JavaScriptin käsittely olio-pohjaisena ohjelmointikielenä suunnittelumalleja ja arkkitehtuuriratkaisuja hyödyntäen auttaa kehittämään selainsovelluksia samaa modulaarisuutta käyttäen kuin palvelinpuolen sovelluksissa. (MacCaw 2011, 1–2.)

Eri selaimet tulkitsevat DOM-mallia ja JavaScript-koodia eri tavoilla, mikä johtaa yhteensopivuusongelmiin. Ohjelmoijat voivat kuitenkin huomioida tämän ohjelmakoodissaan tai käyttää erillisiä kirjastoja, joissa selainversioiden tarkistaminen on sisäänrakennettu.

Teknologian tarkoitus on palvella ihmisten tarpeita, ja sovelluksilta vaaditaan aina parempaa käytettävyyttä, saatavuutta sekä hyvää suorituskykyä. Käytettävyyttä yleensä parantavat Java-appletit ja Flash-tekniikkakaan eivät pysty puhtaasti täyttämään näitä kriteereitä web-sovellusten ympäristönä. Niiden vaatiman erillisen lisäosan asentaminen tuottaa vaivaa ja voi estyessään myös vaikuttaa saatavuuteen. Tämän johdosta monessa tapauksessa standardien DOM- ja JavaScript-tekniikoiden yhdistelmä on käyttäjille mieluisampi. Tämäkään toteutus ei ollut aluksi ongelmaton, sillä käyttökokemus häiriintyi aina kun käyttäjän täytyi saada uutta tietoa palvelimelta. Sivun lataaminen on suurimpia haittoja verrattuna työpöytäsovelluksiin ja siihen auttaa Ajax. (Darie, Bogdan, Chereches-Tosa & Bucica 2006, 13–14.)

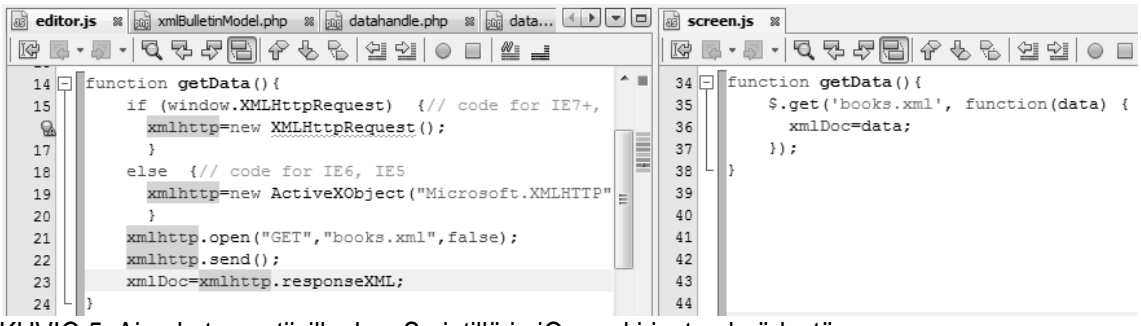
Ajax (Asynchronous JavaScript And XML) on tekniikka, jolla voidaan parantaa web-sovellusten vuorovaikutteisuutta. Vaikka JavaScript on tekniikan olennaisin toteuttaja, on Ajax termistä tullut yleispätevä kaikille web-sivun osittaiseen päivittämiseen tarkoitetuille toiminnoille. Tällainen perustuminen jo olemassa olevaan yleisesti tuettuun selain-teknologiaan, sekä periaatteessa

vapaaseen toteutustapaan on luonut Ajaxille vahvan jalansijan web-kehittäjien työkaluna. (Asleson & Schutta 2006, 14–17.)

Ajax-rajapinta, kuten monet muutkin JavaScriptin toteutukset eri selaimilla vaihtelevat niiden käyttämän moottorin mukaan. Tämä tarkoittaa, että jo ennestään haasteellisena pidetyssä ohjelmointiympäristössä koodissa joudutaan huomioimaan useita tapoja saman asian tekemiseen. Useasti nämä toiminnot ja tarkistukset ovat kuitenkin jollain tavalla uudelleen käytettäviä ja kerättävissä olevia, jolloin niistä voidaan muodostaa kirjastoja.

jQuery on avoimen lähdekoodin JavaScript-kirjasto, joka helpottaa HTML-dokumenttien dynamiikan kehittämistä. Sen avulla on mahdollista hyödyntää web-teknologioita turvallisesti ilman selainriippuvaisuuksista johtuvia ongelmia. Kirjaston tueksi on tehty useita lisämoduuleita, kuten jQuery UI joka on teemoitettu kokoelma erilaisista animaatioista ja graafisista komponenteista käyttöliittymiin liitettäväksi. (jQuery Board 2011, hakupäivä 7.11.2011.) JavaScript-kirjastot on rakennettu helpottamaan web-ohjelmointia; niillä voidaan toteuttaa esimerkiksi virheenkäsittelyä, joka on muuten hyvin haastavaa eri selainversioiden vuoksi. Kirjastojen avulla yleisiin tehtäviin saadaan myös helpompi syntaksi jonka avulla nopeutetaan työskentelyä ja vähennetään virheiden mahdollisuutta. (Lengstorf 2010, 3.) Kirjastojen käyttö kuitenkin hidastaa ohjelmien toimintaa jonkin verran (Irish 2009, hakupäivä 2.11.2011).

Kirjaston tuoma etu esimerkiksi jQueryn ja natiivin JavaScriptin välillä on huomattavasti selkeämpi ohjelma koodi. Kirjaston ".get"-funktio sisältää automaattisen selaintarkistuksen, datan muotoilun, välimuistin tarkistuksen ja monia muita sovelluskehityksen kannalta hyödyllisiä toimintoja. GPL-lisenssin etuna on myös, että toiminnot on varmennettavissa suoraan lähdekoodista. Alla olevassa kuviossa 5 koodi esimerkit natiivin JavaScriptin ja jQueryn tavoista tehdä yksinkertainen Ajax-pyyntö. $-merkki on tässä tapauksessa vain toinen nimi jQueryn kutsumiseen. (Resig 2011, hakupäivä 7.11.2011.)



KUVIO 5. Ajax-kutsu natiivilla JavaScriptillä ja jQuery-kirjastoa hyödyntäen.

Eri tietojärjestelmien välisessä kommunikoinnissa tiedon lähettäminen ja vastaanottaminen vaatii yhteisesti ymmärretyn merkitsemis- ja tulkitsemistavan. Web-järjestelmien välillä tämä yleensä tarkoittaa jonkin merkistöstandardin alaista merkkijonoa. Jotta tätä tietoa voidaan ohjelmallisesti käsitellä, täytyy tähän merkkijonoon sisällyttää tieto eri tietoelementtien rajoista ja merkityksistä. Näiden rajapisteiden ja merkityksien kokoamista eritavoilla käsiteltäviksi alkioksi ja niitä yhdisteleviksi kokoelmiksi kutsutaan jäsentämiseksi (engl. parsing).

Koska laajojen kokonaisuuksien täsmällinen kuvaus vaatii yleensä useita tasoja, on tällaisten tieto-olioiden muuntaminen merkkijonoksi ja takaisin monimutkaista sekä ohjelmoijalle aikaa vievää työtä. Jäsentämiseen on kehitetty monia valmiita työkaluja eri järjestelmiin ja niiden toiminta nojaa johonkin standardoituun, tai muuten ennalta ilmoitettuun merkintäohjeeseen. Avoimen lähdekoodin web-sovelluskehityksen kannalta olennaisimmat alustariippumattomat merkintämallit ovat XML ja JSON.

XML on joustava ja luettava merkintäformaatti rakenteellisen tiedon esittämiseen. Se pohjautuu HTML:n tavoin myös SGML-formaattiin. (Quin 2011 Hakupäivä 5.11.2011.) HTML:n keskittyessä enemmän tiedon näyttämiseen ja muotoiluun, XML keskittyy sen kuvaamiseen rakentamisen, tallentamisen ja siirtämisen näkökulmista (Darlington 2005, 291).

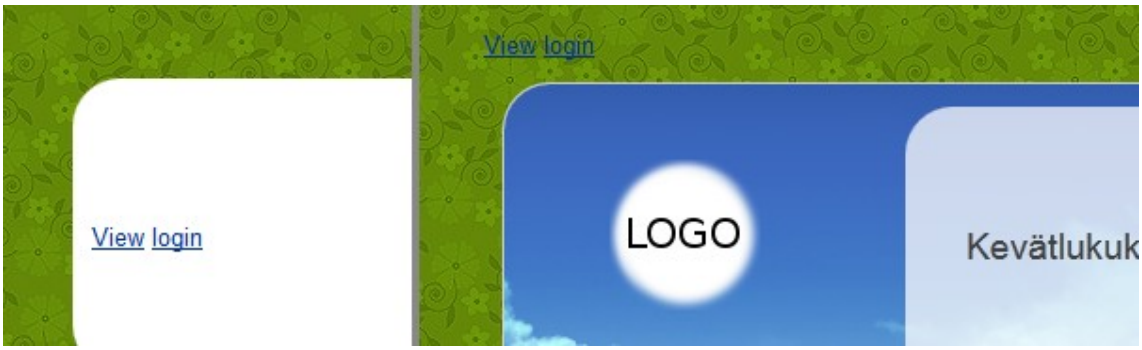
JSON (JavaScript Object Notation) on yksinkertainen tiedonsiirtoon suunniteltu tekstiformaatti, jota on helppo käyttää JavaScript-ohjelmissa. Ohjelmoinnissa sen kautta luotuja olioita on helppo hyödyntää perinteistä pistenotaatiota käyttäen. Nimestään ja JavaScript perustastaan huolimatta JSON on JavaScriptistä riippumaton, eli sitä voidaan käyttää myös muilla ohjelmointikielillä. XML formaattiin verrattuna JSON on tiiviimpi, joka tarkoittaa tiedon vaatiman tilan olevan pienempi. Tästä on hyötyä etenkin suuria tietomääriä verkossa siirtäessä. (Asleson & Schutta 2006, 69–71.)

3 SÄHKÖISEN ILMOITUSTAULUN TOTEUTTAMINEN

Sovelluksen rakentaminen lähti tekniikoiden ja mallien sisäisten toimintatapojen, sekä ulkoisten rajapintojen määrittelystä. Näiden soveltuvuutta piti edelleen verrata toiminnallisiin vaatimuksiin kunnes saatiin varmistus ehtojen jonkin asteisesta täyttymisestä. Sovelluksen tuli tarjota selaimen välityksellä toimiva helposti ylläpidettävä tiedotusnäyttö. Toimintojen ohjelmoinnissa oli otettava huomioon ohjelmistokehyksen käyttö, erillisen tietokannan välttäminen, Ajax-tekniikka sekä käytettävyys. Haasteena oli toteuttaa järjestelmä siten, että se noudattaa selkeää rakennetta ja on laajennettavissa tai muunnettavissa myöhemmin. Liiketoiminnan vaatima laaja yhteensopivuus eri selaimissa tai tietoturva ei ollut työn kannalta kriittisessä asemassa. Julkislaitoksen yleinen ilmoitustaulu ei todennäköisesti sisällä salassa pidettävää materiaalia, tai ole muuten turvallisuusnäkökulmasta kriittinen toiminto.

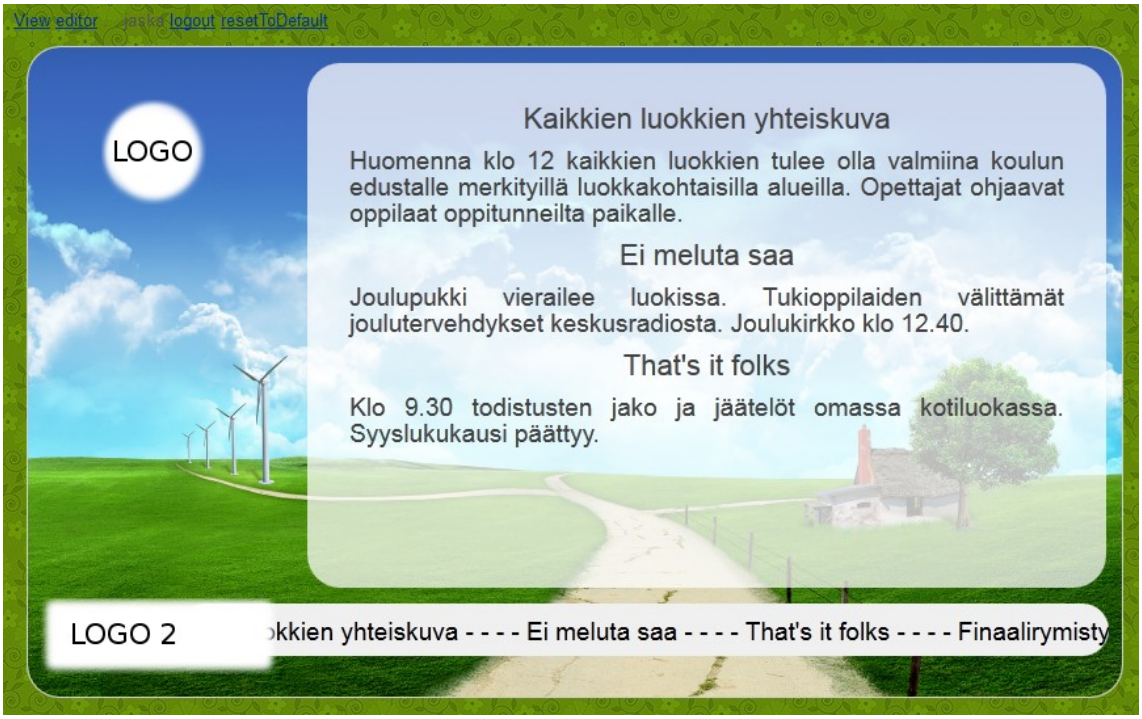
3.1 Käyttöliittymä

Sovelluksen käyttäjäryhmät ovat katselijat ja ylläpitäjät. Molemmille on erilaiset näkymät sovellukseen. Katselijoiden näkymään ei ole liitetty minkäänlaista pääsynvalvontaa, koska sen kautta tietoa ei pysty muokkaamaan. Sovelluksessa on kolme erilaista käyttöliittymää, jotka ovat tiedotteiden katselu, -hallinnointi ja kirjautuminen. Niillä kaikilla on yhtenäinen tausta, fontin tyyli, lohkomuotoilu sekä navigointikomponentti, joka mukautuu kirjautumistilan perusteella. Kuviossa 6 on nähtävissä nämä käyttöliittymille yhteiset elementit sekä muunnelma, jossa navigointi (View login) on siirretty taustalle ja sivutaustan päällä olevaan lohkoon on lisätty grafiikkaa.



KUVIO 6. Käyttöliittymän peruselementit.

Tiedotteiden katseluun tarkoitetusta käyttöliittymästä on interaktiivisuus rajattu pois projektiresurssien perusteella, vaikka jonkinlaisen selaustoiminnon toteuttaminen olisi periaatteessa ollut perusteltua. Käyttöliittymä jää siten näennäisesti staattiseksi, vaikka käyttääkin Ajax-tekniikkaa itsensä päivittämiseen. Sen sisältö muodostuu kolmesta kokonaisesta tiedotteesta kerrallaan kaikkia tiedotteita iteroiden sekä kaikkien tiedotteiden otsikoista muodostuvasta nopeasta yhteenvedosta, jota kuljetetaan liittymän alaosassa. Käyttöliittymän sommittelussa on käytetty pohjana tv-uutisten näkymiä. Kuviossa 7 näkyy liittymä-elementit ja niiden muotoilua, kuten hieman läpinäkyvä pohja kokonaisten tiedotteiden taustana sekä otsikot puhtaalla pohjalla.



KUVIO 7. Ilmoitustaulun katseluliittymä

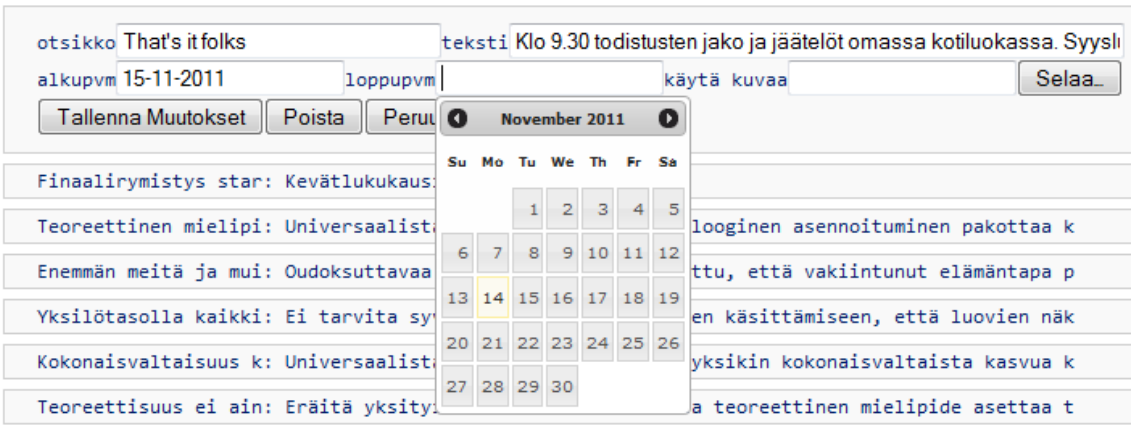
Kirjautumisliittymä on perusulkoasuun upotettu hyvin pelkistetyn näköinen HTML-lomake, joka sisältää mahdollisen virhetiedon, kentät tunnuksen ja salasanan syöttämiseen sekä lähetä-painikkeen. Lähetyksen ja lomakkeelle syötettyjen tietojen oikeellisuuden varmistuksen jälkeen käyttäjä ohjataan palvelinsovelluksen oletusohjaimeen. Varmistuksen epäonnistumisesta asetetaan virhetieto sovelluskehyksen avustuskirjastoon ja uudelleen ajetaan kirjautumisliittymä.

Tiedotteiden hallinnoinnista vastaavan käyttöliittymän perusasetelma koostuu tiedotteen luomislomakkeesta ja tiedotteiden lyhennelmistä, jotka esitetään käyttöliittymässä listana. Lomake sisältää kentät tiedotteen otsikon, leipätekstin, alku- ja loppupäivämäärän syöttämiseen. Päivämäärien syöttämisen avustamiseksi ohjelma tarjoaa JavaScriptin varassa olevaa visuaalista

kalenteri-moduulia, jonka avulla on mahdollista muodostaa oikeanmuotoinen päivämäärä osoittimella valitsemalla. Aikarajoitukset eivät ole pakollisia ja tyhjäksi jätetyt kentät tarkoittavat tiedotteen pysyvän esityksessä toistaiseksi. Ohjelma esitäyttää alkupvm -kenttään sen hetkisen päivämäärän, mutta sillä ei ole merkitystä ohjelman suorituksen kannalta ja hyödyt ovat lähinnä visuaaliset ja käsittelyn suhteen informatiiviset. Lomakkeen alla on kokoelma tiedotteiden lyhennelmistä, jotka on muotoiltu Codeigniter code -elementin suuntaisesti. Lyhennelmä esittää katkaistut merkkijonot tiedotteiden otsikoista ja leipätekstistä. Kuviossa 8 on käyttöliittymä perusmuodossaan.

KUVIO 8. Hallinnointiliittymä

Kuviossa 9 on esitetty jo olemassa olevan tiedotteen muokkaamistilanne, jossa kursori on loppupvm-kentässä ja siten jQuery UI-kalenteri komponentti näkyvissä. Tiedotteen muokkaaminen alkaa valitsemalla sen lyhennelmä listalta osoittaen, jolloin ohjelma sulkee lyhennelmän ja aukaisee tilalle tiedot lomakkeella. Lomake sisältää tietojen lisäksi toimenpide-painikkeet tallennukselle, poistamiselle ja muokkaustilan perumiselle.



KUVIO 9 Ilmoitustenhallintapaneelissa vanhan tiedotteen muokkaaminen päivämäärä kentässä

Käyttöliittymän suunnittelussa ei tarkoituksellisesti pohjauduttu mihinkään erilliseen teoriaan tai malliin vaan luotettiin omien kokemuksien myötä kehittyneeseen intuitioon. Ulkoasu kehittyi työn aikana ja sitä korjattiin muun testauksen yhteydessä, jos epäkohdat tuntuivat häiritseviltä. Pieniä virheitä ei huomioitu, koska sovelluksen visuaalinen ilme oli toissijainen työn kannalta. Tyylit sovellukselle määriteltiin 800 x 600 resoluutiota ajatellen ja tätä aluetta on pyritty käyttämään mahdollisimman tehokkaasti. Tietojen esittämiseen ja syöttämiseen harkittiin myös taulukko-muotoista asettelua, joka olisi myös mahdollistanut tiiviin käyttöliittymän. Vaihtoehtona se olisi ollut todennäköisesti visuaalisesti hieman raskaampi ja sekavampi.

3.2 Tiedotteiden tallentaminen

Toteutuksessa käytetty tiedosto bulletinfile.xml on palvelimella sijaitsevan tiedostojärjestelmän hallinnassa oleva XML-muotoinen tekstitiedosto. Tässä toteutuksessa yksittäinen tiedosto on riittävä tiedotteiden tallentamiseen, eikä johda lisenssirajoituksiin. Tiedoston sisältö on merkkijono, jota käytetään mallina luodessa ohjelmistossa käytettävä tietokokonaisuus, sen toiminnallisuuden katsotaan tapahtuvan vain osana sovellusta. Tiedosto ei näin ollen tarvitse erillistä moottoria.

Tiedoston käytössä tietovarastona on muutamia tapauskohtaisia määrittelyongelmia. Tiedoston luku- ja kirjoitusoikeudet tulee asettaa kullekin sen käyttäjälle, kuten esimerkiksi palvelimelle ja PHP-moduulille. Tiedostoihin voidaan vaikuttaa myös hakemistorajauksilla, esimerkiksi sijoittamalla rajoitettavaksi luokiteltu tieto jollain tavalla suojattuun tai eristettyyn hakemistoon (The Apache Software Foundation 2011, hakupäivä 1.11.2011.) Toisaalta jos tiedostoon pääsy

sallitaan suoraan web-palvelimelta, voidaan tätä hyväksikäyttää luodessa suorempia pyyntöjä selainsovelluksista. Tässä työssä päädyttiin käyttämään kehysrakenteen mallia kaikessa tiedostonkäsittelyssä, jolloin polku ei kahdennu ohjelmakoodissa. Kehysrakenteessa käytetty hakemistokohtainen suoja ei siten myöskään hidasta erikseen palvelimen toimintaa.

Tiedoston muotoileminen XML- tai JSON-standardin mukaisesti mahdollistaa sen lukemisen suoraan olioksi monilla siihen tarkoitetuilla työkaluilla. PHP sisältää periaatteessa tuen molempien käsittelemiseen, mutta XML-tuki on laajempi ja dokumentoidumpi. Palvelinsovelluksessa tietoa hallitsevan mallin tekeminen aloitettiin yksinkertaisimmilla metodeilla, kuten palauta koko sisältö ja lisää uusi tiedotus. Ensimmäinen metodi ei vaatinut minkäänlaista jäsentämistä palvelimella, mutta lisätessä tietoa oli selkeää, että aikataulun vuoksi kannatti käyttää jotain työkalua.

SimpleXML on PHP -laajennus, jolla on helppo lukea XML-olio suoraan tiedostosta ja suorittaa perustoimintoja, kuten puussa liikkuminen, tiedonhaku, lukeminen ja lisääminen. Siirryttäessä astetta monimutkaisempaan toiminnallisuuteen, kuten muokkaamiseen on sen käyttäminen epäkäytännöllistä esimerkiksi elementtien manipuloinnin tuen puuttuessa. PHP DOM sisältää laajemman ja käytännöllisemmän toiminnan tuen ohjelmoijalle. Tässä työssä sitä käytetään bulletinfile.xml-tiedoston tiedotteiden muokkaamiseen ja poistamiseen mallin sisällä, kuten käytetään myös edeltäviä tekniikoita niiltä osin kuin ne olivat sopivia.

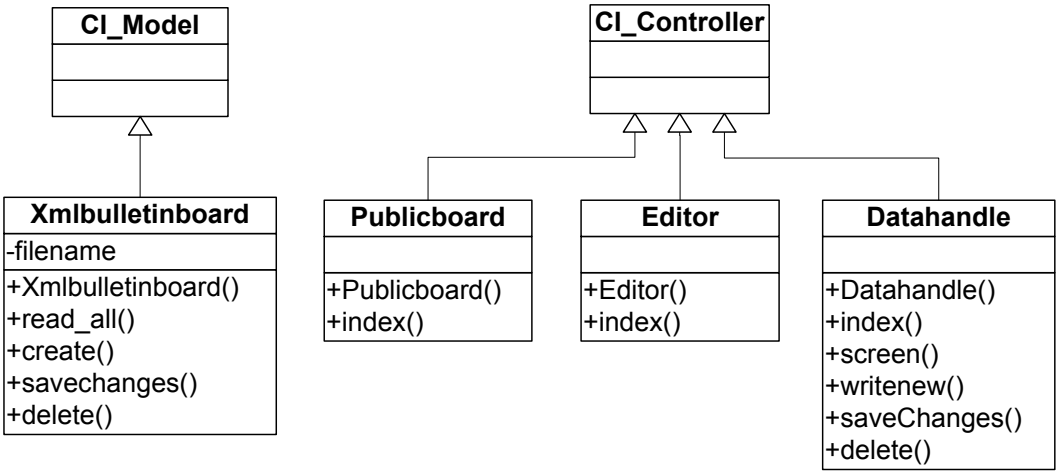
3.3 Arkkitehtuuri

Sovelluskehyksen tapa toteuttaa noudattamaansa arkkitehtuuria oli huomioonotettava kriteeri, jolla rajattiin mahdollisia web-kehysrakenteita. Kehyksen tuli tukea MVC-mallin hyödyntämistä, mutta samalla oli epävarmaa miten paljon oli varaa käyttää aikaa kehyksen toteutustavan opiskeluun. Kehyksen tuli sallia poikkeava toteutus, jotta projektin onnistuminen ei vaarannu tarpeettoman paljoa mahdollisista virhearvioista kehyksen käytön opettelunopeudesta. Samalla kehyksen toteuttaman mallin tuli mielellään olla selkeästi havaittavissa käytännön tasolla.

Kehysrakenteeksi valittiin Codeigniter selkeiden ohjeiden, avoimen lähdekoodin ja yleisen suosion perusteella. Sen noudattama löyhä MVC-versio toimi pohjana järjestelmän rakenteelle, mutta on myös mahdollistanut toiminnallisten poikkeuksien käyttämisen kehyksen opettelussa. Kehyksen joustavuus lisäsi varmuutta projektin loppuun saamisesta ja edisti siten myös työn

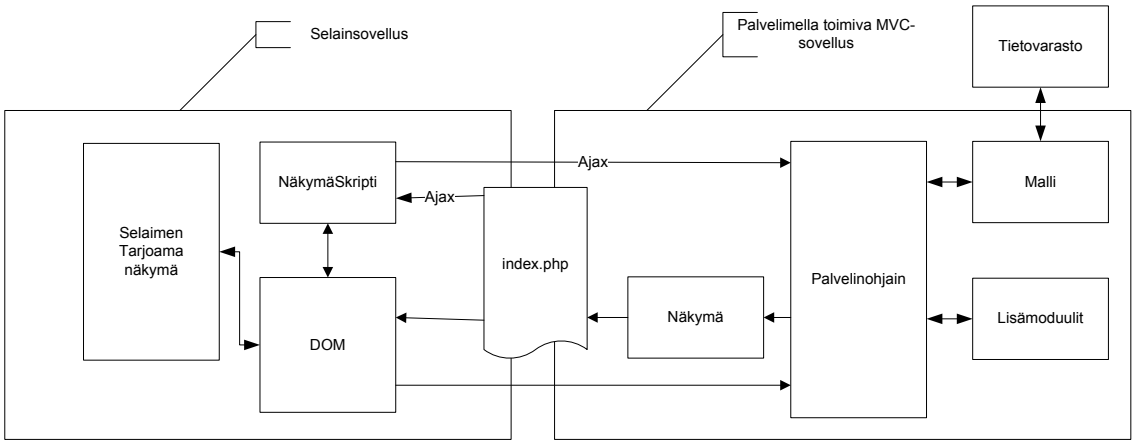
tekemistä. Passivisen näkymän-malli on Codeigniter-kehyksessä selkeästi esillä ja sen käyttämiseen voi siirtyä vaiheittain.

Sovelluskehyksessä ohjelmointi keskittyi odotettua paremmin suoran toiminnallisen lisäarvon tuottamiseen eikä ympäristön rakentaminenkaan ollut vaikeaa. Kuvio 10 esittää sovellukseen toteutuneet luokat metodeineen. Xmlbulletinboard luokan toteutus käsittelee tallenteiden hallinnoimisen ja siihen on sisäänrakennettu tiedoston määritykset. Xmlbulletinboard voi myös luoda XML-määritteet sisältävän tiedoston tarvittaessa uuden tiedotteen tallennuksen yhteydessä. Publicboard luokka sisältää määritykset katseluliittymän muodostamiseen, jotka rajoittuvat avusteiden ja liittymänäkymien kutsumiseen. Myös Editor on toiminnallisesti hyvin yksinkertainen ja tiedotteidenhallintaliittymän toiminnallisuus muodostuvatkin vasta sen lataamien näkymien myötä. Datahandle toimii rajapintana katselu- ja hallinnointinäkymien sekä Xmlbulletinboard metodien välillä. Sen on CRUD-toimintojen lisäksi tarkoitus tarjota kaksi erilaista lukumetodia, joista toinen julkisempaan käyttöön.



KUVIO 10. Kehykseen toteutetut Luokat

Järjestelmän eri toimintoja suunniteltaessa päädyttiin siihen, että sovellusympäristön käyttötapausjohtoiset toiminnot, kuten kirjautuminen ja navigointi toteutetaan puhtaasti palvelintekniikalla. Selaintekniikan vastuulla on ydintoiminnan käyttöliittymien tukeminen, kuten avustaa tiedotteiden hallintaa. Tämän tarkoituksena oli kohdistaa resursseja vaatimusmäärittelyjen mukaisesti CRUD-toimintojen käytettävyyteen ja palvelintekniikan laadukkuuteen. Näin syntyneen järjestelmän luonnollinen jaottelu MVC-kehysrakenteen ja toimintaympäristön huomioiden voidaan kuvata kuten kuviossa 11.



KUVIO 11. Järjestelmäarkkitehtuuri

3.4 Ajax

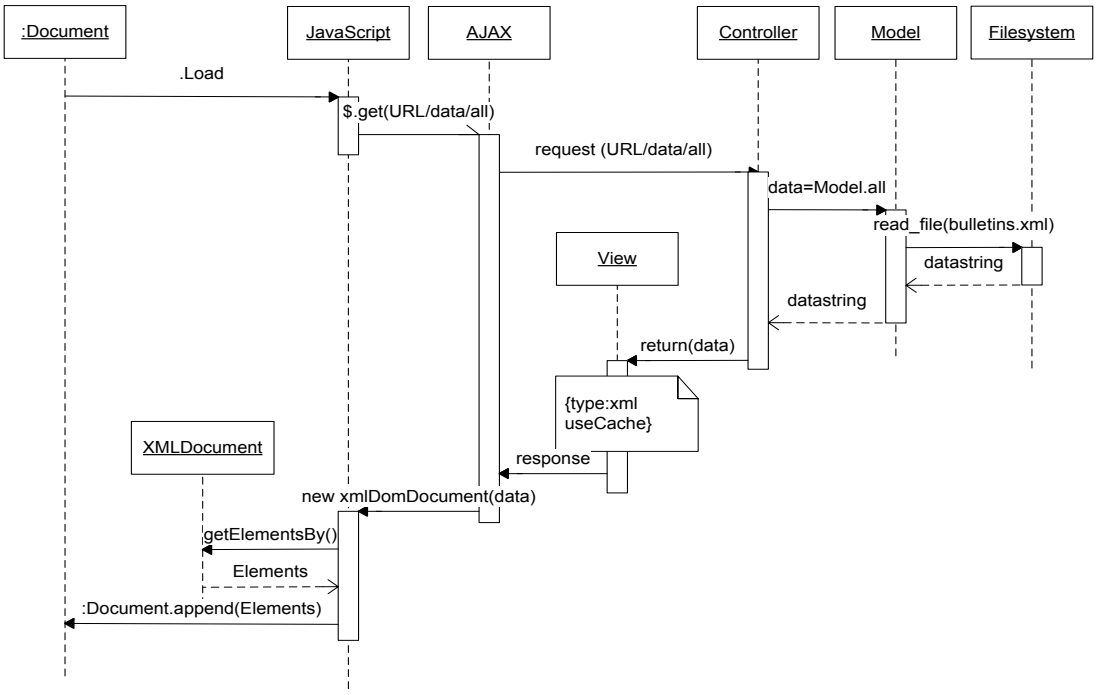
Sovelluksessa Ajax-tekniikan tarkoituksena on piilottaa käyttäjältä tiedotteiden hallinnointiin liittyvä sovellustekninen toiminta. Käyttöliittymien sisällä toimivat katselu ja hallinnointi JavaScript-sovellukset toteuttavat tiedotteiden lataamisen ja esittämisen omina yksiköinään muun sivun valmistuttua. Kehykseen toteutettu Datahandle saa kutsuja ainoastaan Ajax-tekniikan kautta ja sen palauttamat näytöt ovat XML-muotoisia merkkijonoja. Käytännössä sen kautta siis kulkee kaikki itse tiedotteiden käsittely sovelluksessa.

Työssä on natiiveja XMLHttpRequest kutsuja, sekä jQuery-kirjaston Ajax-funktiota. Puhtaaseen parametrittömään lukuoperaatioon sovellus käyttää JavaScriptiä, mutta kaikki tietoa sisältävät kutsut kulkevat jQueryn läpi välttääkseen mahdollisia yhteensopimattomuuksia. Ajax-toiminnot eivät sinänsä vielä olleet riittävä syy ottaa käyttöön erillistä JavaScript-kirjastoa, mutta käytettävyyden ja selainyhteensopivuuksien parantamisen kanssa se oli jo hyvin perusteltua. Hyvän dokumentaation, käyttöliittymäkomponenttien tason ja yleisen suosion perusteella sovelluksessa päätettiin käyttää jQuery-kirjastoa.

3.5 Sovelluslogiikka

Sisällönhallintajärjestelmän ydinprosessi on tiedon jakaminen. Sen toteuttamiseen eri konteksteissa, kuten kohteissa, tavoissa ja ajassa tarvitaan eri tekniikoita. Tämän työn kannalta olennaisimpia ovat esittämisen, tiedonsiirron ja -ylläpidon näkökulmat. Tiedonhallinnan mekaniikan rinnalla käyttökokemus on keskeisessä roolissa sovelluksen toiminnassa.

Katselu-sovellus käynnistyy palvelimelle saapuvasta pyynnöstä, jonka sovelluskehyksen reititys ohjaa URI:n perusteella näyttökontrollerille. Tämä ajaa palautettavaksi otsikko- ja runkonäkymät. Vastauksena pyynnön lähettäneeseen selaimeen saapuu html-tiedosto liiteosoitteineen, joista latauksen valmistuttua luodaan DOM-olio ja aktivoidaan JavaScript sovellus. Kuviossa 12 näkyy sovelluksessa seuraavaksi ajettavan tiedotteiden hakemisen sisältämä tietoliikenne. Ensimmäiseksi nyt aktiivinen sovellus lähettää jQueryn Ajax-metodin kautta palvelimelle pyynnön, jonka URI sisältää parametrina reititysohjeen tietoyhteyttä hallitsevalle kontrollerille. Tietoyhteyskontrolleri kutsuu malliolion katselumetodia, jonka palauttama arvo lähetetään näkymän kautta vastauksena selaimelle. Saapuva vastaus käännetään paluuarvosta uudeksi XMLDOM-olioksi, jonka sisältämistä elementeistä solmuineen luetaan tiedotteet ja esitetään niitä osana sivuja edustavaa DOM-oliota. Tämän jälkeen JavaScript-sovellus käy läpi tiedotteita XMLDOM-oliota silmukoiden ja päivittää olion toistamalla palvelin pyynnön sopivan ajan kuluttua.



KUVIO 12. UML-sekvenssikaavio JavaScript-ohjelman automaattisesta alustus-funktiosta

Kuvion 12 selainohjelman tiedonhallinnasta vastaava JavaScript-ohjelma kutsuu palvelimen tietomoduulia ja päivittää sivua. Kuvassa ei näy ohjelmoijalta piilotettuja objekteja tai avusteita, vaikka ne olisivatkin järjestelmän suunnittelun kannalta kriittisiä. Sekvenssiin kuitenkin vaikuttaa sisäänrakennettu reititys ja avustetoiminta Codeigniter -kehyksessä, sekä jQuery -kirjaston tarkastus funktiot.

Myös tiedotteiden hallinnointitoiminnot noudattavat samankaltaista kaavaa. Tiedotteiden muokkaamiset kulkevat Ajax POST-pyynnön kautta sen ollessa parempi tiedon lähettämiseen. jQueryn append metodi ei myöskään enää riitä käyttöliittymään tehtäviin muutoksiin, mutta muuten erot tietoliikenteen osalta ovat vähäiset.

3.6 Yleisiä periaatteita

Koodia kirjoittaessa toiminnallisuuden rakentaminen lähti usein hyvin konkreettiselta tasolta, jossa pyrittiin saamaan välittömästi näkyviä tuloksia. Erilaiset käyttöliittymän muutoksesta johdetut tapahtumat ja vastaukseksi annettavat palautteet kirjoitettiin hyvin suoraviivaisesti eteneväksi joukoksi tarkkoja rajapintakutsuja. Lähtökohtaisesti ne sisälsivät myös useita

rinnakkaisia ja sisäistettyjä kontrollirakenteita. Tällainen ohjelmointikäytäntö ei selkeästi kuitenkaan pysty kannattelemaan kestävää ohjelmistokehitystä ja sitä pyrittiin parantamaan muutamilla hyvillä ohjelmointikäytännöillä, kuten abstrahoinnilla, jossa yksinkertaisempia käskyjä kootaan yhteen ja kääritään korkeamman yhteiskäsitteen alle.

Ohjelmistokoodin muodostamisessa pyrittiin käyttämään yhtenäisesti Banner-sisennyssääntöä. Sen mukaan lohkon aloittava kaarisulku on kontrollirakenteen aloituslausekkeen kanssa samalla rivillä ja on ainoa sisentämätön. Banner-sisennyksen etuna on selkeämpi tapa osoittaa lohkojen aloitus- ja lopetusrivi, joka pääsääntöisesti nopeuttaa koodin luettavuutta. Tällainen lohkojen yksinkertainen visuaalinen muoto voidaan ymmärtää yhdeksi alkeellisemmaksi abstraktion kuvaustavaksi. Visuaalisten muotojen yhdistelmä antaa nopeasti myös vihjeitä ohjelmalohkojen koheesiosta, eli kuinka tarkasti ne toteuttavat yhden toiminnon periaatetta. Lohkojen sisältämien kerrostumien kasvaessa suureksi sisällön ymmärtäminen vaikeutui selkeästi ja siten myös kehittämisnopeus hidastui. Kehitysnopeuden laskiessa huomattavasti näiden kerrostumien vuoksi alettiin aktiivisesti hakea toistuvia toiminnallisuuksia ja aputoimintoja. Näitä refaktoroimalla pystyttiin hallitsemaan sovelluksen kehitysnopeutta ja osin ne johtivat myös aiemmin heikosti käsiteltyjen ohjelmointitapojen korostumiseen. Toimintojen jatkuva uudelleen nimeäminen ja toistuvien toimintojen kääriminen itsenäisiksi yksiköiksi vauhdittui ja kaavamaistui nopeasti osaksi, koska käytetty ohjelmointiympäristö sisälsi graafisen työkalun sen turvalliseen suoritukseen.

Ohjelmistokoodin muuttumisnopeus aiheutti jonkinasteista epävarmuutta ja sitä pidäteltiinkin jonkin verran, koska pelättiin menetettävän jokin olennainen ohjelman osa. Monien rajapintojen ollessa vielä heikosti tunnettuja kyky niiden toteutuksen toistamiseen koettiin epävarmaksi. Samalla ei luotettu myöskään varmuuskopioiden kykyyn taltioida ohjelmistokoodin juuri oikeita kehitysvaiheita. Ongelman ratkaisuksi otettiin käyttöön versionhallinta, johon pystyi tallentamaan kehitysvaiheita paremmin. Valitsin käyttööni Mercurial ohjelmiston, koska se oli käyttämääni ohjelmointiympäristöön sisäänrakennettu avoimen lähdekoodin projekti, joka tukee hajautettua sovelluskehitystä. Sen avulla eri kehityshaarojen ja sovellusversioiden välillä siirtyminen on nopeaa sekä muutosten seuraaminen helpottuu. (Mercurial community 2011, hakupäivä 14.12.2011.) Ohjelmointiympäristön suora tuki mahdollisti versionhallinnan vaivattoman käytön, jolloin se voitiin suorittaa aina tarvittaessa ilman työskentelyn keskeytystä. Näin saavutettu runsaampi ja oikea aikaisempi versiohistoria piti suuremmalla varmuudella sisällään kadonneet koodin osat, kuten nimeämisessä vaihtuneet viitteet tai vaikka testauksessa pois kommentoidut toiminnot.

4 POHDINTA

Projektikokonaisuus oli hyvin kattava normaalin tuotekehityksen näkökulmasta lukuun ottamatta liiketalouden osa-alueita. Siinä käytiin läpi suuri joukko erilaisia työvaiheita, mutta niiden kulku ei sitoutunut mihinkään tiettyyn kehitysmalliin. Todennäköisesti työskentely käytti väljästi useaa eri kehitysmallia päällekkäin, joista tunnistettavissa jollain tasolla ovat spiraali-malli ja ketterät-menetelmät. Tämä saattoi johtua siitä, että nämä olivat laajimmin tunnettuja tekniikoita itselleni.

Sovelluksen toiminnallisten tavoitteiden mukainen sähköinen ilmoitustaulu toteutui ja odotukset sen käytettävyyden tasosta ylittyivät. Kaikki alkuperäisiin vaatimuksiin perustuva ydintoimintojen joukko onnistui ja niiden visuaalista laatua vietiin mielestäni kaupallisten tuotteiden tasolle. Alun perin huonosti tunnettu XML- ja DOM-tekniikoiden yhdistäminen tiedonvarastointiin pystyttiin soveltamaan ilman pelättyä manuaalisten merkkijono-lukijoiden luomista. Sovelluksen hallinnallisten tavoitteiden ympäristöriippumattomuuskin toteutui vielä raportin viimeistelyvaiheessa.

Työn tuloksena syntynyt sovellus havainnollistaa MVC-mallin laajoja soveltamismahdollisuuksia. Palvelinsovelluksen näkymä toteuttaa sivun selaimeen, jonka sisällä skripti kutsuu palvelimella sijaitsevaa ohjainta käsitelläkseen liiketoiminta olioita. Tällainen ratkaisu vaikuttaa luonnolliselta ottaen huomioon, että Ajax-toteutuksella haetaan parempaa käyttökokemusta. Se ei myöskään riko MVC-mallin teoriaa näkymien sisältäessä rajapintatoimintoja, kuten mallin seuraaminen. Kuitenkin jos tarkoitus olisi keventää palvelinkuormaa, voisi olla järkevää muodostaa selainsovellus itsenäisenä ja tarjota sille oma malli, kuten Presentation Model-arkkitehtuurissa. Tulevaisuudessa HTML5-version paikallisten tietovarastojen myötä tällainen puhtaampi jaottelu tulee luultavasti yleistymään.

Sovelluksen suunnittelussa yksi huomioinnin kohta oli mahdollisten ulkoisten riippuvaisuuksien rajoittaminen. Tuotteen riippuvaisuudet, eli vaaditut resurssit voidaan ajatella käänteisesti verrannollisina sen houkuttelevuuteen. On vaikea arvioida miten paljon halu tutkia ajax-tekniikkaa ja sovelluskehityksen toimintaa vaikuttivat siihen, että ilmoitustaulu on toteutettu pilvitekniikoilla. Näinpä voidaankin ajatella ongelman ja ratkaisun löytäneen toisena. Joka tapauksessa ilman lisäosia yleisimmillä selaimilla toimivan pilvisovelluksen voidaan ajatella olevan organisaatiossa

helposti ja laajasti saatavilla. Tällainen käytön mahdollistaminen kaikilla verkon koneilla tukee käytettävyyttä.

JavaScriptin tuoma odotettu käytettävyyden lisäarvo sovellukseen toteutui moninkertaisesti. Alkuperäisenä odotuksena oli, että sen käyttäminen perustoimintojen ulkopuolella vaatisi liikaa kehitysresursseja, mutta jQuery-kirjaston ottaminen mukaan nopeutti ja inspiroi työntekoa. Viimeisimmässä versiossa käyttöliittymä on kauttaaltaan jo tavalla tai toisella sen varassa. Näin syntynyt tuote parantaa käytettävyyttä ja käyttökokemusta huomattavasti, sillä sovelluksessa tapahtuvat muutokset erottuvat animaatioiden ja grafiikan avulla selkeästi ilman käyttäjän tarvetta tarkoituksellisesti tutkia koko dokumenttia. Luultavasti CSS3-version ja muiden uusien tekniikoiden avulla sisältöä ja toimintojen vaikutuksia voidaan entisestään nostaa esille, esimerkiksi grafiikan elävöittämisellä.

Työharjoittelussa luodun prototyypin luomat jatkokehitysideat kattoivat suuren osan opinnäytetyön vaatimusmäärittelyistä. Niiden lisänä toimivat opinnäytetyön muut määritykset hidastivat sovelluskehitystä aluksi huomattavasti, mutta paljolti niiden ansiosta lopullinen tuote ja sen lähdekoodi ei ole välttämättä täysin kertakäyttöinen. Työn jatkokehitykseen ilmenikin useita parannuksia suoraan ydintoimintoihin, mutta myös tukeviin ratkaisuihin, kuten tiedotteiden arkistointiin. Tärkeimpiä lisäyksiä ydintoimintoihin tulevat olemaan mahdollisuus kuvien käyttöön, tärkeiden tiedotteiden korostamiseen sekä esityksen kiertonopeuden ja -vaiheen ilmentäminen. Sovellus voisi myös olla sopivassa vaiheessa julkaista se avoimen lähdekoodin projektiksi, jolloin siitä voisi hyötyä useammat ihmiset.

Sovelluksen käyttöliittymän muodostavan DOM- ja JavaScript-tekniikan kanssa työskentely oli minulle entuudestaan täysin tuntematonta. Niiden yhteensovittaminen sovelluksen kehityksessä vei hyvin paljon aikaa ja jopa vaaransi projektin onnistumisen. Suurin syy tähän oli ehkä liian teoreettisten lähteiden valinta etenkin DOM-rajapintaa opiskellessa. DOM ja JavaScript omina yksiköinään vaikuttivat hyvinkin selkeiltä, mikä saattoi antaa virheellisen mielikuvan niiden sovittamisen helppoudesta.

Palvelinympäristössä sovelluslogiikan työstäminen oli yllättävän helppoa, eivätkä siellä ennestään tuntemattomatkaan konseptit, kuten kehysrakenne ja merkkipohjaisen tiedoston käyttö tallentamiseen hidastaneet kohtuuttomasti. Toki niissä oli omaa problematiikkaa, kuten URI-reitityksen rooli sovelluslogiikassa ja tiedon muuntaminen käsiteltäväksi objektiksi, mutta aiheista löytyvä dokumentaatio kiteytti asiat nopeasti. Todennäköisesti palvelinsovellusten kanssa työskentelyyn tottuneempana tunnen sen termistöä ja käyttämiä tekniikoita kohtuullisesti, joka

nopeutti tietolähteiden käsittelyä. Kehysrakenteena Codeigniter toimi sovelluksessa hyvin. Se ohjasi toteuttamaan ohjelmoinnissa loogisesti rakentuvaa modulaarisuutta kuitenkaan sitä pakottamatta. Kehysrakenteessa oli tarjolla hyvin käytännöllisiä avustetoimintoja, mutta toisaalta ne eivät olleet kovin kattavia.

Vaikka työskentelytavat ja työkalut eivät ole tutkimuksellisten tavoitteiden kannalta oleellisia, koen ne kuitenkin hyvin tärkeiksi sovellusohjelmointiosuuden kannalta. Version hallinnan käyttöönotto vaikutti ehkä hieman yllättäen eniten henkisellä tasolla poistamalla jatkuvan epävarmuuden ja ylimääräisen varmistelun ohjelmointi ratkaisujen muutoksissa. Sen avulla koodin toiminnallisuuden edistäminen ja luettavuuden ylläpitäminen parantui, koska koodia voi muokata huolettomammin versioiden sisältäessä muutoshistorian sekä mahdollisesti väärinperustein poistetut koodit ja kommentoinnit.

Opinnäytetyöprosessin aikana nousi esiin useita huonosti tuntemiani sovelluskehitykseen liittyviä konsepteja. Esimerkiksi sovelluksen JavaScript ei ole puhtaasti sijoitettu omaksi yksikökseen kuten "Huomaamaton JavaScript-ohjelmointitapa" (eng. Unobtrusive JavaScript) esittää, vaan sitä on sisällytetty HTML-merkinnän joukkoon. Tosin kyseiset ongelmakohdat ovat selain-skriptin itsensä generoimia. Samoin vasta riittävän refaktoroinnin perusteella koheesion parantuessa huomasin mahdollisuuden parempaan virheenkäsittelyyn, kuten try- & catch-lohko rakenteet ja siitä edelleen tutustuin testivetoiseen ohjelmointitapaan.

Internetiä hyödyntävässä järjestelmässä sovellusmoduulien vastuiden jakaminen on tärkeä osa järjestelmän mallintamista. Moduulien alustan tunteminen mahdollistaa realistisemman suunnitelman luomisen, esimerkiksi DOM-skriptauksen haasteellisen luonteen vuoksi voi olla joskus kannattavaa luoda palvelimella valmiita komponenttiaihioita selainohjelmassa ajonaikana jalostettavaksi. Toisaalta mallinnuksessa kannattaa pyrkiä erottelemaan käyttöliittymään ja ydintoimintaan kohdistuva sovelluslogiikka, jolloin ohjelmiston ylläpidettävyys parantuu. Eiväthän nämä toisiaan poissulkevia ajatuksia ole, kunhan palvelin ohjelmiston komponentteja tuottava moduuli on riittävän itsenäinen.

Kehysrakenteiden ja kirjastojen käyttäminen on aina jonkinasteinen valinta koodin optimoinnin ja kehitysnopeuden välillä. Vaikka tässä työssä optimointi ei ole pitkälle viety ja osa toiminta nopeuteen perustuvista parannuksista on varmaankin epäolennaisia niin uskon silti, että pienistä puroista ne isotkin virrat syntyvät. Todennäköisesti optimoinnin huomiointi aina kun siihen kykenee kehittää jossain vaiheessa todellista hyötyä.

LÄHTEET

The Apache Software Foundation. 2011. Apache Tutorial: .htaccess files. Hakupäivä 1.11.2011, Http://httpd.apache.org/docs/2.2/howto/htaccess.html.

Asleson, R. & Schutta, N. 2006. Ajax - Tehokas hallinta. Jyväskylä: Readme.fi.

Porębski, B. Przystalski, K. & Nowak, K. 2011. Building PHP Applications with Symfony, CakePHP, and Zend Framework. Indianapolis: Wiley Publishing, Inc.

Bedell, K. & Turner J. 2002. Struts: kick start. Yhdysvallat: Sams Publishing.

Berners-Lee, T. 1991. www-talk from September to October 1991. Hakupäivä 2.10.2011, Http://lists.w3.org/Archives/Public/www-talk/1991SepOct/0003.html.

Berners-Lee, T. 1989. The original proposal of the WWW, HTMLized. Hakupäivä 2.10.2011, Http://www.w3.org/History/1989/proposal.html.

Cluts, N. 1996. Creating ActiveX Components in C++. Hakupäivä 2.11.2011, Http://msdn.microsoft.com/en-us/library/ms974283.aspx.

Coar, K. & Robinson, D. 2004. The Common Gateway Interface (CGI) Version 1.1. Hakupäivä 1.12.2011, Http://tools.ietf.org/html/rfc3875.

ExpressionEngine Dev Team. 2011. CodeIgniter User Guide Version 2.1.0. Hakupäivä 22.9.2011, Http://codeigniter.com/user_guide/.

Crockford, D. Yahoo. 2007. Seminaari 2007. Tekijän hallussa.

Cunningham & Cunnigham, Inc. 2011. Model View Controller. Hakupäivä 26.10.2011, Http://c2.com/cgi/wiki?ModelViewController.

Darie, C. Bogdan, B. Chereches-Tosa, F & Bucica, M. 2006. AJAX and PHP: Building Responsive Web Applications. Birmingham: Packt Publishing Ltd.

Darlington, Keith. 2005. Effective website development: tools and techniques. Harlow: Pearson Education Limited.

Elia, E. 1996. Macromedia unveils Shockwave and Director 5. Hakupäivä 2.11.2011, Http://www.faqs.org/abstracts/Computers-and-office-automation-industries/Macromedia-unveils-Shockwave-and-Director-5-Director-5-MOA-better-Xtras.html.

Engel, J. 1999. Programming for the Java virtual machine. Massachusetts: Addison-Wesley Professional.

Forlizzi ,J. & Ford ,S. 2000. The Building Blocks of Experience: An Early Framework for Interaction Designers. Proc Designing interactive systems: processes, practices, methods, and techniques. Hakupäivä 11.12.2011, Http://goodgestreet.com/docs/forlizziDIS00.pdf.

Fowler, M. 2006a. Passive View. Hakupäivä 11.11.2011, Http://martinfowler.com/eaaDev/PassiveScreen.html.

Fowler, M. 2006b. GUI Architectures. Hakupäivä 4.12.2011, Http://www.martinfowler.com/eaaDev/uiArchs.html.

Golovanov, A. 2011. Benchmark update: Cake vs. CodeIgniter vs. Kohana. Hakupäivä 26.10.2011, Http://pr0digy.com/codeigniter/benchmark-update-static-cake-codeigniter-kohana/.

Greer, D. 2007. Interactive Application Architecture Patterns. Hakupäivä 4.12.2011, Http://www.aspiringcraftsman.com/2007/08/25/interactive-application-architecture/.

Hégaret, P. Whitmer, R. & Wood, L. 2009. Document Object Model. Hakupäivä 22.9.2011, Http://www.w3.org/DOM/.

Irish, P. 2009. jQuery Anti-Patterns for Performance. Hakupäivä 2.11.2011, Http://paulirish.com/2009/perf/.

jQuery Board. 2011. jQuery: The Write Less, Do More, JavaScript Library. Hakupäivä 22. 9 2011, Http://jquery.com/.

Qian, K. Fu, X. & Tao, L. 2009. Software Architecture and Design Illuminated. Sudbury: Jones and Bartlett Publishers, LCC.

Keith, J. & Sambells, J. 2010. DOM Scripting: Web Design with JavaScript and the Document Object Model. New York: Apress.

Krasner, G. & Pope, S. 1988. A Cookbook for Using View-Controller User the Model-Interface Paradigm in Smalltalk-80. Journal of Object-Oriented Programming 1988 (Elo-Syyskuu) 26-31.

Lengstorf, J. 2010. Pro PHP and JQuery. New York: Apress.

MacCaw, A. 2011. JavaScript Web Applications. Sebastopol: O'Reilly Media, Inc.

Mercurial community. 2011. Mercurial source control management. Hakupäivä 14.12.2011, Http://mercurial.selenic.com/about/.

Myer, T. 2008. Professional CodeIgniter. Indianapolis: Wiley Publishing, Inc .

Allaire, J. 2002. Macromedia Flash MX—A next-generation rich client. Hakupäivä 6.11.2011, Http://download.macromedia.com/pub/flash/whitepapers/richclient.pdf.

NCSA 1994. NCSA Mosaic v. 2.0 alpha 1 for MS Windows - Bug/enhancement list. Hakupäivä 11.12.2011, Ftp://ftp.ncsa.uiuc.edu/Mosaic/Windows/Archive/NCSAMosaicV2.0a1.zip.

Peterson, D. 2008. Rasmus Lerdorf: PHP Frameworks? Think Again. Hakupäivä 26.10.2011, Http://www.sitepoint.com/rasmus-lerdorf-php-frameworks-think-again/#comments.

Powell, T. 2003. HTML & XHTML: the complete reference (4. painos). Europe: McGraw-Hill Education.

Reenskaug, T. 2007. The original MVC reports. Dept. of Informatics. Oslo: University of Oslo.

Resig, J. 2011. jQuery Lähdekoodi. Hakupäivä 7. 11 2011, Http://code.jquery.com/jquery-1.7.js.

Richards, R. 2006. Pro PHP XML and Web services. New York: Apress.

Rosen, R. & Shklar, L. 2003. Web application architecture: principles, protocols, and practices. Chichester: John Wiley & Sons Ltd.

Suthers, D., professori, IEEE International Conference on Advanced Learning Technologies. 2001. Luentomateriaali 30.5.2001. Tekijän hallussa.

Summers, E. 2008. A short history of Web UI programming. Hakupäivä 30.11.2011, Http://weblogs.java.net/blog/evanx/archive/2008/06/a_short_history.html.

Quin, L. 2011. XML Essentials. Hakupäivä 5.11.2011, Http://www.w3.org/standards/xml/core.