POHJOIS-KARJALAN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

Janne Lehikoinen

DYNAAMISEN TIEDONHALLINTAVERKKOSOVELLUKSEN SUUNNITTELU JA TOTEUTUS

Opinnäytetyö
Elokuu 2011

POHJOIS-KARJALAN
AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Elokuu 2011
Tietotekniikan koulutusohjelma

Karjalankatu 3
80200 JOENSUU
p. (013) 260 6800

Tekijä
Janne Lehikoinen

Nimeke
Dynaamisen tiedonhallintaverkkosovelluksen suunnittelu ja toteutus

Toimeksiantaja
Oy Silvadata Ab

Tiivistelmä

Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa tiedonhallintasovellus verkkopalvelun tietosisällön hallintaa varten sekä perehtyä nykyaikaisiin dynaamisiin verkkosovellustekniikoihin. Projekti toteutettiin toimeksiantona Oy Silvadata Ab:lle.

Lähtökohtana oli kehittää uudempi versio aikaisemman sovelluksen tilalle, joka toteutti vain osan työlle määritetyistä vaatimuksista. Vanhaa sovellusta sekä siitä saatuja käyttäjäkokemuksia käytettiin suunnittelun pohjana. Tärkeimmät vaatimukset sovellukselle olivat tiedonhallinnan toteuttaminen sekä muiden toiminnallisuuksien käytettävyyden parantaminen.

Työn tutkimusosassa tarkastellaan erilaisia verkkosovellustekniikoita, jotka mahdollistavat dynaamisten toiminnallisuuksien toteuttamisen. Näiden avulla työssä määritelty sovellus saatiin toteutettua. Avoimen lähdekoodin tekniikoita on paljon saatavilla ja niihin perehtyminen auttoi sovelluksen suunnittelussa ja toteutuksessa.

Tuloksena saavutettiin määrityksien mukainen tiedonhallintasovellus. Sovellus saatiin testattua ja asennettua käyttöön lopulliseen palvelinympäristöön. Käyttöönottoa ei toteutettu työn aikana, joten uudesta järjestelmästä ei vielä ole saatu käyttäjäkokemuksia. Sitä ennen sovellukseen toteutetaan jatkokehityksenä karttakoordinaatiston muunnos.

Kieli
suomi

Sivuja 34

Asiasanat
dynaaminen, tiedonhallinta, verkkosovellus

NORTH KARELIA
UNIVERSITY OF APPLIED SCIENCES

THESIS
August 2011
Degree Programme in
Information Technology
Karjalankatu 3
FIN 80200 JOENSUU
FINLAND
Tel. 358-13-260 6800

Author
Janne Lehikoinen

Title
Designing and Implementing a Dynamic Data Management Web Application

Commissioned by
Oy Silvadata Ab

Abstract

The purpose of this thesis was to design and implement data management software for web service's content management and to get familiar with modern dynamic web application technologies. The project was commissioned by Oy Silvadata Ab.

The starting point of this thesis was to develop a newer version of an earlier application, which performed only a part of the specified conditions. The earlier version of the application and the end user experiences were used to design a new version. The main requirements for the application were to implement a data management and to improve the usability of other functionalities.

The research section studies various web application technologies, which enable the implementation of dynamic functionalities and, which make possible to implement the defined application. There are many open source technologies available and familiarizing with them helped in designing and implementing a new application.

As a result of the definition the data management software was created. The application was tested and installed successfully in the final server environment. The introduction is taking place later, so any user experiences have not yet been obtained. Before that the application is further developed with the map coordinate conversion.

Language
Finnish

Pages 34

Keywords
dynamic, data management, web application

Sisältö

Termit ja lyhenteet

AJAX	Asynchronous JavaScript And XML - tekniikka dynaamisten verkkosovelluksien kehittämiseen
API	Application Programming Interface - ohjelmistorajapinta, jonka avulla ohjelmat voivat keskustella keskenään
ASP	Active Server Pages - Microsoftin kehittämä palvelinpuolen ohjelmointikieli dynaamisten verkkosivujen tuottamiseen
CGI	Common Gateway Interface - rajapinta, jossa selain välittää tietoa palvelimella suoritettavalle prosessille
CSS	Cascade Style Sheets - tyylikieli, jonka avulla määritetään verkkosivun tyyliohje
DHTML	Dynamic Hypertext Markup Language - termi tekniikoille, joilla lisätään verkkosivuun toiminnallisuutta
DOM	Document Object Model - puumainen tietorakenne, joka mahdollistaa verkkosivun sisällön muokkauksen
Flash	Macromedian kehittämä ohjelmointikieli, jolla voidaan toteuttaa verkkosivuun liitettävä ohjelma
GIS	Geographical Information System - paikkatietojärjestelmä, jonka avulla voidaan esittää paikkatietoa
HTML	HyperText Markup Language - standardoitu verkkosivujen kuvaukseen käytetty merkintäkieli
IP	Internet Protocol - protokolla, joka huolehtii tietoliikennepakettien toimittamisesta perille internetverkossa
JavaScript	Netscapen kehittämä komentosarjakieli, jonka avulla saadaan verkkosivulle dynaamista toiminnallisuutta

jQuery	JavaScript-kirjasto, joka sisältää suuren määrän hyödyllisiä ja yleisesti käytettäviä toiminnallisuuksia
JSP	Java Server Pages - Sun Microsystemsin kehittämä menetelmä upottaa Java-koodia verkkosivuun
MVC	Model-View-Control - graafisen käyttöliittymän kolmeen osaan jaettava arkkitehtuurimalli, tarkoituksena erottaa käyttöliittymä sovellustiedostoista
PDO	PHP Data Objects - tietokantarajapinta, joka mahdollistaa tietokanta riippumattoman tietojenkäsittelyn
SQL	IBM:n kehittämä relaatiotietokantojen käyttöä ja hallintaa varten kehitetty kyselykieli
SSI	Server Side Includes - tekniikka, jolla lisätään verkkosivuun palvelinpuolella suoritettavia komentosarjoja
SSL	Secure Sockets Layer - salausprotokolla, jolla voidaan suojata tietoliikenne salaamalla
UML	Unified Modeling Language - Object Management Groupin kehittämä graafinen mallinnuskieli
UNIX	Laitteistoriippumaton käyttöjärjestelmä, jolla on avoimen lähdekoodin kehitykseen perustuva alkuperä
WFS	Web Feature Service - rajapinta, jolla haetaan paikkatietojen ominaisuustietoja XML-muotoisina sanomina
WMS	Web Map Service - rajapinta karttakuvien hakuun, palauttaa vastauksena kuvat pyydetyltä alueelta
XHR	XMLHTTPRequest - objekti, joka mahdollistaa asynkroniset pyynnöt selaimen ja palvelimen välillä
XML	eXtensive Markup Language - merkintäkieli, jolla kuvataan rakenteellista merkkauskieltä

1 Johdanto

Nykyaikaiset verkkosovellukset sisältävät suuria määriä tietoa erilaisissa tietorakenteissa. Yleensä sovelluksen tarvitsemat tiedot varastoidaan relaatiotietokantaan, josta tietoja voidaan hallita erilaisilla hallintasovelluksilla. Tämä on kuitenkin haastavaa, varsinkin kun tietosisältö kasvaa suureksi ja tietokannan hallinnan kautta vain järjestelmän hallinnoijat pääsevät tietoihin käsiksi. Tietojen hallitsemista varten verkkosovelluksissa käytetään tiedonhallintaa, jonka avulla hallitaan sovelluksen sisältämiä ja sovellukseen siirrettyjä tietoja.

Erilaisia tiedonhallintasovelluksia on saatavilla laajalti niin kaupallisena kuin myös avoimen lähdekoodin jakeluina. Työssä tehtävä sovellus tehtiin kuitenkin toimeksiantajan vaatimusmäärittelyiden mukaisesti ja on vaatimuksiltaan erilainen, mitä valmiit järjestelmät mahdollistaisivat. Sovellus sisältää tietosisällön hallinnan lisäksi myös muita toiminnallisuuksia.

Opinnäytetyön toimeksiantajana toimii Oy Silvadata Ab, joka on yksityismetsätalouden tietojärjestelmiin erikoistunut yritys. Yritys on toiminut tietojärjestelmien parissa jo yli kahdenkymmenenviiden vuoden ajan. (Silvadata 2011.) Toimeksiannon perusteena on suunnitella ja toteuttaa metsänomistajien verkkopalvelua SilvaNettiä varten tiedonhallintasovellus, jolla siirretään ja hallitaan tietoja.

Työn haasteena on saada toteutettua monipuolinen ja käytännöllinen sovellus tiedonhallintaan sekä perehtyä tarkemmin erilaisiin menetelmällisiin valintoihin sovelluksen toteutustekniikoissa. Sovelluksesta pyritään tekemään mahdollisimman dynaaminen tarjoten käyttäjälle helppokäyttöinen ja toimiva käyttöliittymä. Sovellus toimii pääasiassa metsänhoitoyhdistysten toimihenkilöiden työkaluna, jolloin sen käytön on oltava sujuvaa ja tehokasta.

Dynaamisten verkkosovellusten kehitykseen on tarjolla monia eri tekniikoita. Työn tutkimusosassa tutustutaan tarkemmin näihin ja valitaan oikeat tekniikat sovelluksen toteuttamiseen. Työssä käytetään mahdollisimman paljon nykyaikaisia oliopohjaisia ohjelmistoprojektin suunnittelu- ja kehitysmenetelmiä.

2 Dynaamisen verkkosovelluksen kehittäminen

Verkkosovelluskehitys on muuttunut merkittävästi, koska nykyään verkkopalvelut eivät ole enää selattavia sivustoja vaan toiminnallisia sovelluksia. Sivustoihin on aikaisemmin lisätty vain pieniä määriä toiminnallista koodia, esimerkiksi jokin pienimuotoinen prosessi, mikä on toteutettu JavaScriptillä.

Nykyisin verkkosovelluksen suunnittelun lähtökohtana on toteuttaa sovellus kokonaan toiminnalliseksi, mikä asettaa haasteita suunnittelulle ja kehitykselle. Sivuston ja käyttäjän vuorovaikutus ei ole pelkästään selailua ja linkkien klikkailua staattisilla sivustoilla, vaan sovellusmaisempaa dynaamisella käyttöliittymällä. Tämä vaatii uusia keinoja sovelluksen toteuttamiselle, koska sovelluksen käyttöä on mallinnettava suunnittelun lähtökohtana.

2.1 Verkkosovellusten kehittyminen

Verkkosivut olivat alkujaan ainoastaan staattisia sivuja. Palvelin lähetti käyttäjän pyytämän resurssin takaisin käyttäjälle. Sivut eivät sisältäneet mitään liikkuvaa, mikä oikeastaan oli osalle sivustoista hyvä asia. Verkkosivut ovat sähköiseen muotoon konvertoituja dokumentteja, joita jaetaan verkkopalvelimilta. (Asleson & Schutta 2007, 3.)

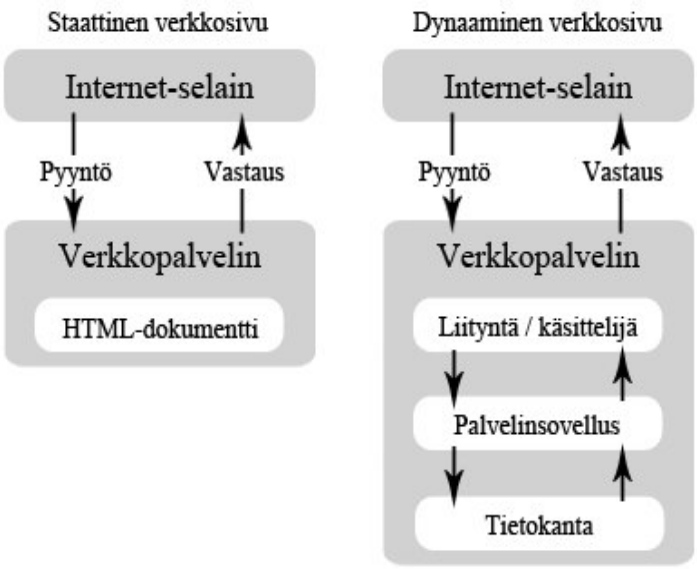
Staattisten verkkosivujen kehittyminen monipuolisiksi verkkosovelluksiksi on alkanut jo 2000-luvun alkuvuosina. Verkkosivut eivät sisältäneet juurikaan toiminnallisuutta, mikä oli riittävää ennen kuin tuli tarvetta kehittää verkkosivuista yleiskäyttöisempiä ja monipuolisempia sovelluksia, millä voidaan hoitaa erilaisia palveluita, kuten verkkokauppaa tai pankkiasiointeja. (Falck 2011, 20.)

Nykyisiin dynaamisiin sovelluksiin pyritään tekemään enemmän käyttäjän ehdoilla toteutettavia toiminnallisuuksia, esimerkiksi tietokannasta haetun tietolistauksen lajittelua. Verkkosovellukset eivät ole enää vain dokumentteja, joita selataan ja luetaan, vaan verkkopalveluita ja sovelmia, mitkä toteuttavat käyttäjän tekemiä toimintoja. Tekniikan avulla desktop-tyyppisiä sovelluksia voidaan rakentaa verkkosovelluksina.

2.2 Dynaamiset tekniikat

Dynaamisten tekniikoiden avulla voidaan toteuttaa monipuolisia verkkosovelluksia, jotka sisältävät erilaisia palveluja ja toimintoja. Sivuston sisältö ja rakenne voivat muuttua jatkuvasti tai käyttäjän suorittamien toimintojen seurauksena. Toiminnallisuudet ovat käyttäjäkohtaisesti määriteltävissä, mikä antaa käyttäjälle mahdollisuuden suorittaa sivustolla erilaisia toimenpiteitä. (2KMediat 2011.)

Verkkosovelluksen toimintamalli muuttuu dynaamisuuden myötä siten, että palvelin ei palauta pyydettyä resurssia tai sivua suoraan HTML-dokumenttina, vaan muodostaa sen pyynnön mukaisesti (kuvio 1). Palvelimella suoritettavat pyynnöt ohjaavat selaimelle palautettavaa vastausta toteuttamalla ensin pyynnön määrittämän prosessin. (Smooth Step 2011.)



Kuvio 1. Staattisen ja dynaamisen verkkosivun toimintamalli (The Code Project 2011).

Dynaamisten verkkosovellusten toteuttamiseen on kehitetty erilaisia tekniikoita ja käytettävät teknologiat kehittyvät jatkuvasti eteenpäin. Verkkosovelluksia kehitettäessä pyritään yhä itsenäisempiin sovelluksiin sekä visuaalisesti että toiminnallisesti. (2KMediat 2011.)

Tekniikat jakautuvat selain-, eli asiakaspuolen sekä palvelinpuolen tekniikoihin. Selainpuolen tekniikoilla toteutetaan sivuston käyttäjälle näkyviä toiminnallisuuksia ja palvelinpuolen tekniikat suorittavat rajapintojen kautta sovelluksen toimintalogiikkaa.

2.2.1 DHTML

Ensimmäisiä tekniikoita dynaamisemman verkkosivun tekemiseen olivat HTML-lomakkeet, joihin lisättiin toiminnallisuutta erilaisilla skriptikielillä. Dynaaminen HTML on käytännössä tyylimäärittelyjen ja skriptien yhdistelmä, mikä muuttaa elementin ominaisuuksia tapahtumamääritteen yhteydessä. Enimmäkseen käytettyjä tekniikoiden yhdistelmiä ovat CSS-tyylimäärittelyt, JavaScript-koodi ja DOM-tietorakenne. Tekniikoiden avulla verkkosivusta saadaan elävämpi ja toiminnallisempi tarjoten käyttäjälle mahdollisuuden suorittaa erilaisia toimintoja verkkosivuilla. (Asleson & Schutta 2007, 11.)

Verkkosivulla olevan elementin ominaisuuksia voidaan muuttaa dynaamisen HTML:n avulla määrittämällä HTML-tagiin tapahtumamäärite, joka toteuttaa JavaScript-koodin. Esimerkiksi muutetaan elementin tyyliominaisuuksia, kun käyttäjä valitsee hiirellä elementin, jolloin onclick-tapahtumakäsittelijässä selain suorittaa JavaScript-koodin.

Dynaamisella HTML:llä voidaan myös kuvata SSI-komentoja käyttäviä verkkosivuja, joissa voidaan sisällyttää tiedosto toiseen tiedostoon tai lisätä tiedostoon suoritettava skripti. SSI tarjoaa tehokkaan tavan hajauttaa toiminnallisuutta useampiin tiedostoihin. (2KMediat 2011.)

2.2.2 CGI

Verkkosovelluksia alettiin kehittää palvelimella ajettavina ohjelmina, mitkä suorittavat tietyn toiminnon käyttäjän pyytäessä sitä. Sovellusta käytetään internetselaimen avulla, mikä visualisoi palvelimen lähettämän tiedon. Ensimmäisiä dynaamisen tiedonhaun mahdollistamia tekniikoita oli CGI, jonka avulla voidaan luoda yhteys verkkosovelluksen ja tietokannan välille ja tuoda käyttäjän näkyville tietokantahaun tulokset. Palvelimella suoritettavien tiedostojen pääperiaatteena oli, että HTML-sivun sisältö tuotettiin vastineeksi käyttäjän antamaan syötteeseen. Kutsuessa CGI-ohjelmaa ei tiedostoa ladata palvelimelle, vaan ohjelma suoritetaan palvelimella ja palautetaan vastaus selaimelle. CGI-ohjelmat voidaan toteuttaa millä tahansa ohjelmointikielellä. (Asleson & Schutta 2007, 4.)

2.2.3 ASP, JSP, PHP

Verkkopalvelinympäristöön dynaamisten verkkosovellusten toteuttamiseen on käytettävissä monia eri palvelinpuolen ohjelmointikieliä. ASP, JSP ja PHP ovat komentosarjakieliä, joiden ohjelmakoodi käännetään palvelimella sovelluksen suorituksen aikana. Näiden kielien avulla tehdyillä skripteillä on mahdollista palauttaa palvelimelta selaimelle muutakin kuin vain staattinen HTML-sivu. Sivua voidaan muokata dynaamisesti selaimen lähettämillä parametreillä, jolloin palvelin suorittaa pyydetyn prosessin ja lähettää selaimelle ladattavaksi sivun, mikä näyttää käyttäjälle normaalilta staattiselta verkkosivulta. Prosessissa voidaan esimerkiksi suorittaa tietokantahaku, jossa verkkosivulle haetaan listaus haettavista tiedoista. (Asleson & Schutta 2007, 5.)

Näiden tekniikoiden heikkoutena on kuitenkin sovelluksen vuorovaikutuksen heikkous. Käyttäjän suorittaessa toimintoa selain lähettää palvelimelle pyynnön toiminnon prosessoinnista, jonka jälkeen palvelin palauttaa selaimelle sivun kokonaisuudessaan uudestaan. Tämä tekee sovelluksen käytöstä hidasta ja raskasta sekä käyttäjälle epämiellyttävän, koska sivua ladataan jatkuvasti uudestaan. Tekniikat eivät myöskään mahdollista dynaamisten toiminnallisuuksien toteuttamista, koska toiminnon suoritus tehdään ainoastaan palvelimella eikä selainsovelluksessa.

2.2.4 Flash ja JavaScript

Vuorovaikutusta käyttäjän ja sovelluksen välillä voidaan lisätä käyttämällä selainpuolen ohjelmointikieliä, kuten Flash ja JavaScript. Käyttäjä voi suorittaa sivuston toiminnallisuuksia ilman palvelimen ja selaimen välistä kommunikointia. Verkkosivuista saadaan näin aidosti dynaamisia sovelluksia.

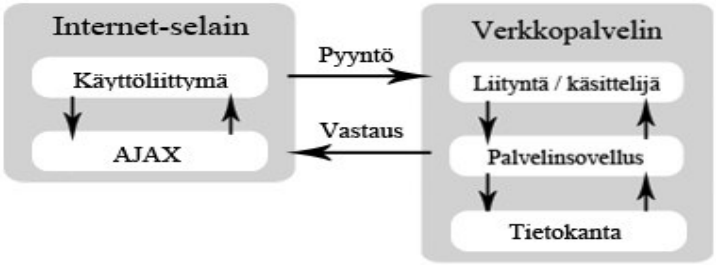
Toiminnallinen koodi lisätään osaksi HTML-sivua. Komentojonot suoritetaan selainpuolen sovelluksessa, jolloin selain lukee ja suorittaa koodin samanaikaisesti. Käyttäjän työasemalla on oltava kyseisten kielien plugin-osat, jotta internet-selain pystyy tulkitsemaan sivuston komentojonoja. Toimintojen tueksi tarvittavat prosessit hoidetaan lähettämällä rajapintojen kautta pyyntö palvelinsovellukselle. Pyyntöjen lähettämiseen ja vastauksen käsittelyyn tekniikat tarjoavat hyvät mahdollisuudet. (Korpela 2011.)

2.2.5 Ajax

Monipuolisempien verkkosovellusten tekemisen mahdollistava selainpuolen ohjelmointikieli Ajax on yhdistelmä edeltäviä tekniikoita. Keskeisimmät komponentit ovat JavaScript sekä XMLHTTPRequest-objekti, joka loi perustan Ajaxin kehitykselle. Tekniikan etu onkin että se rakentuu olemassa olevien tekniikoiden varaan, jolloin kehittäjien ei tarvitse opetella uusia ohjelmointikieliä. Ajax toimii nykyaikaisissa selaimissa ilman erillisiä plugin-osia parantaen verkkosovellusten toimintavarmuutta. (Oracle Technology Network. 2011.)

Ajaxin avulla voidaan lähettää pyyntöjä palvelimen suoritettavaksi samalla kun käyttäjä suorittaa sovelluksen muita toimintoja. Palvelinpyynnöt lähetetään taustalla käyttäjän näkymättömissä, jolloin välitettävät tiedot eivät näy ulospäin. Tämä on kehitysaskel verkkosovelluskehityksessä, koska sovellukset voidaan suunnitella toimimaan asynkronisesti. Näin voidaan toteuttaa toimintoja, jotka ovat olleet mahdollisia vain desktop-sovelluksissa. Käytännössä voidaan suorittaa prosessi, esimerkiksi tarkastaa tiedon oikeellisuus samanaikaisesti muita toimintoja suorittaessa. (Asleson & Schutta 2007, 15.)

Ajax-kerros lisätään dynaamisen verkkosivun toimintamallin selainpuolen sovellukseen (kuvio 2). Kaikki pyynnöt ja tiedonvälitys kulkevat tätä kautta. Ajax hoitaa käyttäjän tekemät toiminnot selainpuolella ja tarvittaessa lähettää pyyntöjä palvelinsovellukselle ja käsittelee ne tuoden haetun tiedon käyttäjän näkyville. (Sun Microsystems 2011.)



Kuvio 2. Ajaxin toimintamalli (The Code Project 2011).

Ajaxin käyttö on suhteellisen yksinkertainen jo olemassa olevien tekniikoiden hyödyntämisen vuoksi. Verkkosovelluksia kehitettäessä tarvitaan myös muitakin verkkosovellustekniikoita, kuten käyttöliittymää, palvelinyhteyksiä sekä palvelinpuolen skriptejä varten. Näiden yhdistelmä luo pohjan dynaamiselle verkkosovelluskehitykselle.

2.3 Palvelinyhteydet ja XMLHTTPRequest

Verkkosivut ladataan HTTP-protokollan avulla pyyntö-vastaustyyppisenä, jolloin palvelin palauttaa joko pyydetyn tietosisällön tai virheilmoituksen. Staattisten verkkosivujen lataamiseen käyttötarkoitus soveltuu hyvin, mutta dynaamisia verkkosovelluksia varten synkroninen tiedonsiirtotapa on ongelmallinen, koska se vaatii koko selainnäkymän päivittämisen uudestaan jokaisen pyynnön jälkeen. (Peltomäki & Nykänen 2006, 296.)

Dynaamisissa verkkosovelluksissa palvelinyhteydet voidaan toteuttaa XMLHTTPRequest-objektilla, joka käyttää tiedonsiirtoon JavaScriptiä viestien HTTP-protokollan kanssa. Tekniikan ansiosta palvelimen ja selainsovelluksen välinen kommunikointi on mahdollista toteuttaa käyttäjälle näkymättömänä asynkronisella tiedonsiirrolla. (Apple Developers 2011.) Objekti sisältää saapuneen pyynnön käsittelyyn erilaisia ominaisuuksia, mitkä on esitelty taulukossa 1. Objektin sisältämät pyynnön suorittamiseen käytettävät metodit on esitelty taulukossa 2.

Tietoa voidaan siirtää selaimelta palvelimelle kahdella tavalla, joko POST- tai GET-metodin avulla. GET-metodissa tieto siirtyy URL-osoitteen mukana ja POST-metodissa tieto siirtyy HTTP-otsikkotietojen mukana, jolloin tietoa ei näytetä käyttäjälle missään vaiheessa tiedonsiirtoa. (W3C 2011.)

Taulukko 1. XMLHTTPRequest-objektin ominaisuudet (W3C 2011).

readyState	objektin tila
responseBody	saapunut data taulukkona
responseText	saapunut data merkkijonona
responseXML	saapunut data xml-muodossa
status	HTTP-statuksen koodi
statusText	HTTP-statuskoodin selite
onreadystatechange	tapahtumakäsittelijä

Taulukko 2. XMLHTTPRequest-objektin metodit (W3C 2011).

abort()	keskeyttää suoritettavan pyynnön
getAllResponseHeaders()	palauttaa HTTP-otsakkeiden arvot
getResponseHeader("nimi")	palauttaa valitun HTTP-otsakkeen arvon
open("method", "osoite", async)	avaa yhteyden
send(data)	lähettää pyynnön palvelimelle
setRequestHeader("nimi", "arvo")	asettaa pyynnölle otsikkotiedon

2.3.1 Pyyntöjen lähettäminen

Palvelinpyynnön lähettäminen aloitetaan luomalla XMLHTTPRequest-objekti ja alustamalla se. Vastauksen käsittelyä varten objektin onreadystatechange-tapahtumakäsittelijälle määritetään funktio, joka suoritetaan palvelimen vastattua pyyntöön. Funktion avulla voidaan esimerkiksi tutkia onko pyyntö suoritettu kokonaan. Pyyntö lähetetään palvelimelle avaamalla yhteys palvelimeen, asettamalla halutut HTTP-otsakkeet sekä määrittämällä lähetettävät parametrit send-metodille. (W3C 2011.)

POST- ja GET-lähetystapojen osalta palvelimelle lähetettävä pyyntö on samanlainen lukuun ottamatta parametrien välitystä. GET-metodissa nimi-arvoparit liitetään osoitteeseen ja POST-metodissa kyselyrivi lähetetään send-metodin mukana. (Oracle Technology Network. 2011.)

XMLHTTPRequest-objekti on riippuvainen käytettävästä internet-selaimesta. Internet Exploreria varten tarvitaan ActiveX-tekniikkaa käyttävä objekti, kun taas Safari- ja Mozilla-selaimet käyttävät JavaScript-luokan XMLHTTPRequest-objektia. (Oracle Technology Network. 2011.)

2.3.2 Vastausten käsittely

Palvelinpyynnön palauttaman vastauksen käsittely aloitetaan tutkimalla onko pyyntö suoritettu kokonaisuudessaan XMLHTTPRequest-objektin readyState-ominaisuuden avulla. ReadyStaten tilat on esitelty taulukossa 3. Tämän jälkeen luetaan palvelimen palauttama vastauskoodi, josta nähdään onko pyynnön suorittaminen onnistunut. Onnistuneen pyynnön palauttama data saadaan ulos kolmessa eri muodossa, esimerkiksi merkkijonomuotoisena responseText-ominaisuudesta. Vastaanotettu data käsitellään halutulla tavalla JavaScriptin ja DOM:n avulla ja asetetaan se käyttäjän näkyville. (W3C 2011.)

Taulukko 3. ReadyStaten tilat, jotka kertovat pyynnön suoritusvaiheen (W3C 2011).

0	alustamaton objekti
1	odottaa latausta
2	pyyntö lähetetty ja suoritus käynnissä
3	data lähetetty ja suoritus käynnissä
4	suoritus valmis

Palvelimen palauttamat HTTP-statuskoodit kertovat pyynnön vastauksen onnistumisesta. Koodi 200 kertoo pyynnön onnistuneen, kuten kaikki muutkin 2xx-alkuiset. 3xx-alkuiset koodit kertovat sisällön sijaitsevan muualla kuin pyynnössä määritetyssä sijainnissa. 4xx-alkuiset koodit ilmaisevat selainpuolen sovelluksen ja 5xx-alkuiset koodit palvelimen tekemistä virheistä. Vastauskoodien avulla voidaan toteuttaa erinomaisesti virheenkäsittelyä, esimerkiksi koodi 403 - käyttäjällä ei ole oikeuksia kyseiseen dokumenttiin. (W3C 2011.)

3 Tavoitteet ja lähtökohta

Työn tavoitteena oli tehdä toimeksiantajayritykselle SilvaNetti Admin-tiedonhallintasovellus, jolla hallitaan SilvaNetti-verkkopalvelun tietosisältöä. Sovelluksesta on toteutettuna aikaisempi versio, jolla lähinnä ladataan verkkotietokantaan palvelun tarvitsemat tiedot. Palvelun kasvaessa on tullut tarvetta päästä hallitsemaan tietosisältöä paremmin. Aikaisemman version käyttökokemukset ja kehitysehdotukset tulevat määrittelemään uuden sovelluksen suunnittelua. Sovellus toteutetaan kokonaan uudeksi versioksi.

Tiedonhallintasovellus toteutetaan verkkopalveluna, jolloin sen käyttö on mahdollista kaikkialta. Sovellusta käyttävät pääosin metsänhoitoyhdistyksien toimihenkilöt, jotka siirtävät metsänomistajien metsäsuunnitelmia palveluun sekä hallitsevat siirrettyjä tietoja. Palvelun tiedot päivitetään vähintään vuosittain. Sovelluksen on tarkoitus helpottaa tietosisällön hallintaa.

3.1 Vaatimusmäärittely

Ensisijainen toiminnallisuus sovellukselle on metsäsuunnitelma ja metsävaratietojen siirtäminen SilvaNetti-palveluun. Palvelun tiedot sijaitsevat erillisessä SQL-relaatiotietokannassa verkkopalvelimella. Toiminnallisuuden tehtävänä on lukea ja siirtää Silvadatan metsävaratietojen hallintaohjelmalla muodostettu siirtopaketti tietokantaan. Siirtopaketti sisältää valitun metsänomistajan metsäsuunnitelman metsävaratietoineen sekä paikkatiedon ja karttakuviot. Tietojen siirtovaiheessa lisätään myös SilvaNetti-käyttäjätili metsänomistajalle.

Tietojen lataamista varten organisaatioiden tarvitsee lisätä pääkäyttäjätunnuksilla sovellukseen metsätoimihenkilöille oma käyttäjätili, jonka avulla toimihenkilö pääsee kirjautumaan sovellukseen. Tällä hallitaan metsänomistajia vastuu ja toimialoittain sekä saadaan aina oikea yhteyshenkilö metsänomistajan tietojen hallintaan. Toimihenkilöt voivat hallita vain itse lataamiaan metsänomistajien tietoja.

Tietokantaan siirrettyjen tietojen hallitsemista varten sovelluksessa on oma osionsa, missä voidaan valita haluttu metsänomistaja, jonka tietoja halutaan muokata, päivittää tai poistaa. Tietoja voidaan muokata metsäsuunnitelma- tai metsätilakohtaisesti. Kaikki palveluun ladatut metsänomistajan tiedot ovat näkyvissä tässä näkymässä. Näkymän kautta voidaan myös siirtyä valitun metsänomistajan SilvaNetti-tiliin.

Sovelluksen info-osiossa näytetään kyseisen organisaation palveluun lataamat tiedot kokonaisuudessaan. Tiedoissa ilmoitetaan palveluun ladattujen metsänomistajien määrä sekä ladattujen karttakuvioiden kokonaispinta-ala. Asiakaspeittävyys on havainnollistettu karttanäkymässä, jossa näytetään esimerkiksi metsänhoitoyhdistyksen toimialue kartalla ja sen sisällä palveluun ladatut metsäkuviot.

3.2 SilvaNetti

SilvaNetti on metsänomistajia varten perustettu verkkopalvelu, jossa pääsee näkemään ja hallitsemaan omaa metsäomaisuuttaan verkossa. Sisältö muodostuu metsänhoitosuunnitelmasta sekä metsävaratiedoista, mitkä antavat käyttäjälle kuvan omasta metsäomaisuudesta ja sen toteutustilasta. (SilvaNetti 2011.) Palvelu on toteutettu Flash-tekniikalla ja se toimii verkkoyhteyden varassa. SilvaNetti on palkittu Pitney Bowes Meridian Awards 2010 tuotekehityskilpailun toisella sijalla.

Palvelun kautta on muun muassa mahdollista lähettää metsänhoitajalle metsänhoitopyyntöehdotuksia, viestiä oman metsätoimihenkilön kanssa, tulostaa oman metsätilansa kartta sekä lisätä metsäkuvioille omia valokuvia ja muistiinpanoja. SilvaNetissä on ladattuna jo yli kahdentuhannen metsänomistajan metsäsuunnitelmatiedot sisältäen lähes kolmesataatuhatta hehtaaria kuviotietoja ominaisuuksineen.

3.3 Sovelluksen nykytilanne

Aikaisemmin tekemässäni sovellusversiossa on toteutettu tietojen siirtäminen tietokantaan sekä käyttäjätilien hallinta. Sovellus on havaittu käytössä toimivaksi tietojen siirron osalta, mutta tietojen hallinta on ollut puutteellista. Myös sovelluksen käytettävyydessä on laajalti parannettavaa.

Tietojen lataaminen haluttiin alkujaan toteuttaa erillisellä ohjelmalla, koska tiedettiin sen mahdollistavan paremmin tietojen hallinnan ja päivittämisen. Myös Silvadatan metsävaratietojen hallintaohjelmasta ulos saatava siirtopaketti sisältää paljon dataa, jota täytyy konvertoida ennen tietokantaan lukemista.

4 Suunnittelu

Sovelluksen suunnittelun lähtökohtana oli kehittää edellisen version ominaisuuksien lisäksi tiedonhallintaa sekä huomioida saadut käyttäjäkokemukset ja kehitysehdotukset. Parempi käytettävyys, toimivuus sekä tietojen päivittämiseen liittyvät parannukset nousivat esiin. Nämä ominaisuudet yhdessä toimeksiantajan vaatimusmäärittelyn kanssa toimivat suunnittelun pohjana. Tarkoitus on myös tutkia voisiko tietoturvaa parantaa vielä jollain tapaa entisestään, koska sovelluksessa käsitellään metsänomistajien metsävaratietoja, joita ei haluta päätyvän vääriin käsiin.

Vaatimusmäärittelyjen pohjalta ennen suunnittelua tehdään analyysivaihe, jossa tarkoitus on tunnistaa ja kuvata sovelluksen tarvitsemat luokat ja tietorakenteet sekä miettiä kuinka käyttötapaukset ja toiminnallisuudet ovat toteutettavissa kuvatuilla luokilla. Määritettyä toiminnallisuutta mallinnetaan UML-kaavioilla, jotka tarjoavat sopivat mallit oliopohjaisten sovellusten suunnitteluun. Mallintamisen avulla saadaan sovelluksesta ymmärrettävämpi ja hallittavampi sekä se tukee varsinaista ohjelmointityötä toteuttamisvaiheessa.

4.1 Tekniset vaatimukset ja tietoturva

Palvelinympäristössä toimiva sovellus toteutetaan selainpuolen asiakassovelluksena ja sovelluksen tarvitsemat toiminnallisuudet suoritetaan rajapintojen kautta palvelimella ajettavina skripteinä. Kaikki tiedonsiirto tapahtuu salattua SSL-yhteyttä käyttäen, joten mitään selväkielistä dataa ei missään vaiheessa siirretä.

Käytettävät palvelimet ovat Unix-pohjaisia luotettavuuden, joustavuuden ja hankinnan helppouden vuoksi. Unix-palvelimien konfigurointi eroaa Windows-palvelimista, koska oikeastaan kaikki konfigurointi tapahtuu komentoikkunassa, eikä graafisessa näkymässä. Palvelinarkkitehtuurina käytetään kahden palvelimen järjestelmää, ensimmäinen palvelin on tietokantaa ja palvelinsovelluksia varten sekä toinen on sovelluspalvelin, jossa ajetaan selainpohjaista sovellusta julkisella IP-osoitteella. Palvelimien väliset yhteydet suoritetaan suojattuja yhteyksiä ja rajapintoja käyttäen. Näin saadaan myös erotettua tietokanta ja tietosisältö pois julkiselta palvelimelta.

Sovelluksen tulee olla mahdollisimman hyvin suojattu väärinkäytöksiltä. Tästä huolehditaan vahvalla käyttäjähallinnalla, sovelluksen ja lähdekoodin tietoturvalla sekä suojatuilla tietoliikenneyhteyksillä.

4.2 Käyttöliittymä

Lähtökohtana ei ole tehdä niinkään näyttävä vaan toimiva ja tehokas käyttöliittymä, koska kyseessä on hallintasovellus, jonka avulla hallitaan isomman verkkopalvelun tietosisältöä. Sovelluksen käyttöliittymä toteutetaan periaatteella desktop-sovellus verkkosovelluksena lisättynä dynaamisilla ominaisuuksilla ja elementeillä.

Käyttöliittymän pohjana käytetään jQuery User Interface -kirjastoa, jossa on erilaisia verkkosivuilla tarvittavia komponentteja, kuten valintalistoja, painikkeita, sivupohjia ja dialog-ilmoituksia (katso jQuerylla toteutetun käyttöliittymän kuva luvusta 6). jQuery mahdollistaa myös monipuolisten CSS-tyylipohjien luonnin sivuston käyttöön. Kaikki elementit ja tyylit ovat muokattavissa halutun laisiksi. (jQuery 2011.)

Sovelluksen ulkoasuksi muodostui dialog-tyylinen internetsovellus, jota voi raahata ja venyttää haluttuun kohtaan selainnäkymässä. Toiminto on käytännöllinen, koska metsätoimihenkilöillä on käytössään yleensä pieniresoluutioisia maastomikroja, millä he työskentelevät asiakaskäynneillä ja maastossa.

4.3 Toiminnallisuuksien mallintaminen UML-kaavioiden avulla

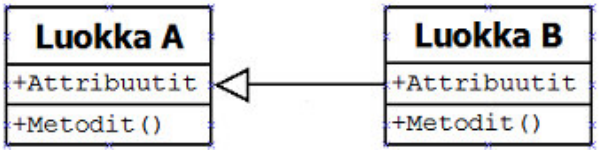
Sovelluksen toiminnallisuus suunniteltiin mallintamalla käyttötapaukset ja toiminnot sekä suunnittelemalla tarvittavat luokat ja niiden ominaisuudet, metodit sekä luokkien väliset yhteydet. Mallintaminen toteutettiin oliosuuntautuneella suunnittelulla käyttäen UML-kaavioita. Oliomallinnuksen avulla saadaan suunniteltua tehokkaasti sovelluksen pysyvä staattinen rakenne, eli tietosisältö ja luokkamallit sekä dynaaminen rakenne, eli sovelluksen käyttötapaukset ja suoritettavat toiminnallisuudet.

Mallinnus ja kaaviot tekevät sovelluksen rakenteesta hallittavamman, mikä tukee varsinkin jatkokehitystä sovelluksen toiminnallisuuden muuttuessa sekä toteutetun sovelluksen ymmärtämistä. Mallien avulla saadaan yhtenäinen käsitys vaatimuksista, suunnittelusta ja toteutetusta toiminnallisuudesta.

4.3.1 Olio- ja luokkamallinnus

Sovelluksen luokat ja niiden sisällöt sekä luokkien väliset suhteet kuvataan luokkakaavioilla. Analyysivaiheessa vaatimusmäärittelyjen pohjalta määritetyt luokat piirretään kaavioksi, jossa luokat ominaisuuksineen ovat kuvattuina. Luokkien väliset yhteydet kuvataan myös kaavioissa. Nämä kaaviot toteuttavat oliomenetelmällisen suunnitteluvaiheen staattisen mallin, jonka pohjalta muut kuvaukset rakentuvat.

Luokkakaavion luokissa määritetään luokan käyttämät attribuutit ja metodit. Luokat esitetään graafisina taulukoina. Kuvion 3 esimerkissä esitetään kaksi luokkaa, missä luokka B perii luokan A ominaisuudet.

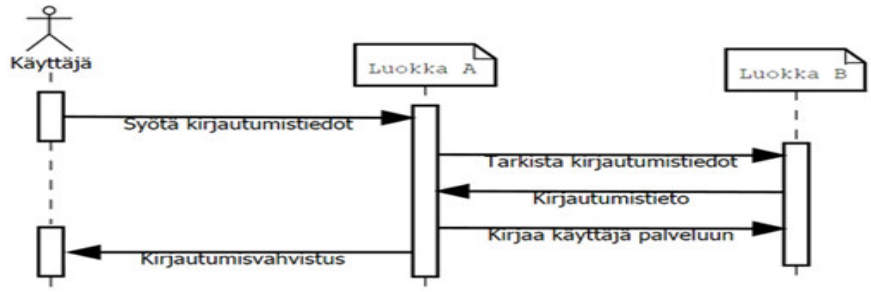


Kuvio 3. Luokkakaavion luokat ja niiden välinen yhteys.

4.3.2 Käyttötapauksien ja toiminnallisuuden kuvaaminen

Suunnitteluvaiheen dynaamisessa mallissa suunnitellaan toiminnallisuuksien toteuttaminen kuvatuilla luokkakaavioilla. Toiminnallisuuksien suunnitteluun ja kuvaamiseen käytettävillä sekvenssikaaviolla kuvataan tietty toiminto tai tapahtumasarja yhdistämällä käyttötapaus luokkakaavioon.

Sekvenssikaaviossa toiminnon tarvitsemat luokat ovat kuvattuina pystyviivoina sekä toimintopyynnöt eli metodikutsut niiden välisinä nuolina. Luokkien kontrollit kuvataan paksunnettuna jaksona. Kuvion 4 esimerkissä esitetään luokkakaavioon yhdistetty sekvenssikaavio käyttäjän kirjautumisesta. Luokka A kuvaa sovellusta ja luokka B toiminnallisuutta.



Kuvio 4. Sekvenssikaavio käyttäjän kirjautumisesta järjestelmään.

5 Tiedonhallintasovelluksen toteutus

Toteutus aloitettiin määrittämällä sovelluksen kehitystyö pienempiin osakokonaisuuksiin, mitä olivat kehitysympäristön pystytys, palvelinlaitteiston konfigurointi, tietokanta, rajapinnat, palvelinsovellus sekä selainsovellus. Sovelluksen tuottaminen on kuitenkin laaja ja monimutkainen tehtävä, joten onnistuneen toteutuksen varmistamiseksi kehitystyö on syytä jaotella osiin.

Määrittelyjen pohjalta suunniteltujen mallinnuksien mukaan toteutettiin sovelluksen luokat ja niiden ominaisuudet. Toteutuksen aikana luokat muuttuivat vielä jonkin verran, koska tarvittavia ominaisuuksia määräytyi lisää. Toteutusprosessi tehtiin siis iteratiivisesti, eli pienemmissä osissa toistaen prosessia.

5.1 Kehitysympäristö

Sovelluksen kehitys tapahtui varsinaisessa palvelinympäristössä, koska se helpotti toiminnallisuuksien yksikkö- ja moduulitestausta, mitkä tehdään toteutusvaiheen aikana. Toiminnallisuudet saatiin testattua sovelluksen lopullisessa ympäristössä, eikä esimerkiksi kehityspalvelimella, joten järjestelmän toiminta tuli testattua ja konfiguroitua valmiiksi kehityksen aikana.

Sovelluksen kehityksessä käytettävät tekniikat ja menetelmät pohjautuvat kaikki avoimen lähdekoodin jakeluihin, joten myös kehitysympäristöksi valitsin Eclipsen, joka on Java-pohjainen ohjelmistokehitin. Ohjelmistokehitin suorittaa koodin oikeellisuuden tarkistuksen kirjoitusvaiheessa, joten koodin tuottaminen on helpompaa ja nopeampaa, koska mahdolliset virheet huomataan näin helpommin. Kehitin neuvoo myös syntaksissa ehdottomalla valittavissa olevia ominaisuuksia.

Eclipseä muokkasin hieman tehokkaammaksi kehitystyökaluksi lisäämällä siihen saatavia plugin-moduuleita. PHP-syntaksituki saatiin lisättyä PHPeclipsellä, PHP-koodin debuggaus Zend Debuggerilla, tietokannan hallinta DBViewerillä sekä testausominaisuudet PHPUnitilla. Eclipseä voi myös muokata ohjelman asetuksia muuttamalla, esimerkiksi syntaksin värityksen osalta.

Sovelluksen kehityksessä tuotettuja tiedostoja hallitaan versionhallinnan avulla. Palvelimelle asennettu Subversion-versionhallintajärjestelmä mahdollistaa tiedostojen muokkaamisen ja hallinnan verkon yli. Versionhallinnassa sinne tuotetut tiedostot kaikkine edeltävine versioineen säilyvät hallitusti tallessa ja ovat saatavissa kaikkialta.

5.2 Palvelinjärjestelmä

Sovellus toimii kahden palvelimen järjestelmässä, missä on tietokanta- ja sovelluspalvelin. Palvelimet toimivat unix-alustalla CentOS-käyttöjärjestelmällä varustettuina. Unix-pohjaisuus asetti konfiguroinnissa hieman haasteita, mutta asennukset saatiin tehtyä asennettavien ohjelmien ja palvelujen manuaalien avulla. Kaikki palvelimille asennetut ohjelmat ja palvelut perustuvat avoimen lähdekoodin jakeluihin.

Tietokantapalvelimelle asennettiin PostgreSQL-relaatiotietokanta, johon lisättiin paikkatieto-ominaisuudet asentamalla PostGIS-laajennus. Palvelinsovellusten suorittamista varten palvelimeen konfiguroitiin PHP sekä PHP Data Object tietokantahakuja varten. SilvaNettiin siirrettävät paikkatietoa sisältävät siirtotiedostot ovat Mapinfon TAB- ja MIF-formaateissa. Näitä varten asennetun FWToolsin avulla tiedostot voidaan lukea tietokantaan kyselytyyppisesti. Kyselyn parametreillä voidaan myös vaikuttaa esimerkiksi haluttuun koordinaattijärjestelmään.

Karttatoiminnallisuuksia varten palvelimeen asennettiin ja konfiguroitiin karttamoottoriksi MapInfon MapXtreme WMS- ja WFS-rajapinta palveluineen. Palvelin viestii sovelluspalvelimen kanssa suojatulla sisäisellä yhteydellä, joten ulkopuolelta tietoihin ei pääse suoraan käsiksi.

Sovelluspalvelin toimii sovelluksen moottorina, joka suorittaa asiakkaille tarjottavia sovelluksia ja palveluita. Palvelimelle asennettuja palveluita ovat Apache HTTP-palvelin, PHP sekä rajapintoja ja tietoyhteyksiä varten tarvittavat yhteys- ja suojausominaisuudet. Sovelluspalvelimella suoritetaan ainoastaan sovelluksen selainpuolen asiakassovellusta. Palvelin viestii tietokantapalvelimen kanssa tarvittavien palveluiden ja tietojen saamiseksi.

5.3 Tietokanta

Tietosisältö rajoitti käytettävän tietokannan valintaa siinä määrin, että sovellus käsittelee ja tallentaa tietokantaan paikkatietoja, joten tallennuksen tulisi olla mahdollista paikkatietoa tukevaan tietotyyppiin. Tällaisen ratkaisun toteutti PostgreSQL-tietokanta PostGIS-laajennuksella, joka lisää tuen geometry-tietotyypille. Geometry-tietotyyppiin voidaan tallentaa mm. piste-, viiva-, ympyrä- ja polygoni-muotoisia paikkatietoja.

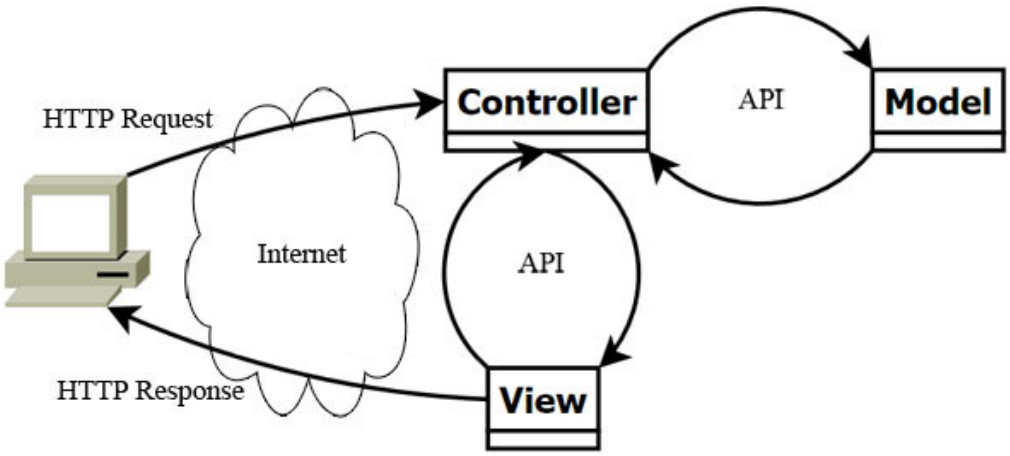
Tietokanta toteutettiin palveluun siirrettäviä tietorakenteita vastaavana, koska tietojen siirtoon käytettävä FWTools lukee tiedostot niiden rakennetta vastaavina tietokantaan. Tietorakenne on myös todettu jo vuosia käytössä olleena toimivaksi, joten mitään suurempaa syytä sen uudistamiseen ei ollut.

Tietokannan hallintaa varten palvelimelle asennettiin pgAdmin-hallintasovellus. Sovelluksen kautta voidaan hallita käytettävän tietokannan rakennetta ja tietosisältöä sekä suorittaa erilaisia kyselyitä.

5.4 Sovellusarkkitehtuuri

Sovelluksen tietorakenteet toteutettiin MVC-mallin mukaisena. Mallissa sovellusarkkitehtuuri jaetaan kolmeen osaan, logiikasta vastaavaan malliin, tiedon esityksestä vastaavaan näkymään ja toiminnasta vastaavaan ohjaimeen (kuvio 5). MVC-malli sopii erinomaisesti käytettäväksi dynaamisen verkkosovelluksen graafiselle käyttöliittymälle. (Microsoft Development 2011.)

Model, eli malli toimittaa tietoja ohjaimelle ja näkymälle. Se on ainoa osa, mikä on yhteydessä tietokantaan. Kaikki tiedonvälitys tapahtuu mallin kautta. View, eli näkymä huolehtii mallilta saadun tiedon esittämisestä tarvittavassa muodossa, yleisimmin HTML-koodattuna. Controller, eli ohjain ottaa vastaan ja suorittaa sovelluksen pyynnöt erilaisilla ohjausparametreillä. (Sun Microsystems 2011.)



Kuvio 5. MVC-mallin tiedonvälitys (Microsoft Development 2011).

MVC-mallin avulla sovelluksen muutostarpeet kohdistuvat yleensä vain yhteen tai kahteen osaan, mikä helpottaa kehitystyötä. Malli hyödyntää oliopohjaista lähestymistapaa, jonka käyttöä tukee valmiit frameworkit. Sovelluksen arkkitehtuuri oli kuitenkin sisällöltään kohtuullisen kokoinen, joten päädyin tekemään oman frameworkin PHP:llä.

5.5 Palvelinsovellus ja rajapinnat

Palvelinpuolen sovelluksen ja suoritettavat skriptit toteutin PHP:llä, joka on monipuolinen ohjelmointikieli verkkosovelluskehitykseen. Selainsovellus kutsuu skriptejä rajapintojen yli palvelinsovelluksen controllerilta, eli ohjaimelta, joka vastaanottaa ja käsittelee pyynnön ja toimittaa sen MVC-mallin mukaisesti takaisin selaimelle. Palvelinsovellus toteuttaa sovelluksen toiminnallisuuden ja tietojenkäsittelyn.

Tietokannan käsittely on toteutettu palvelinsovelluksella käyttäen PHP Data Objectia, joka on tietokantariippumaton rajapinta. PDO on joukko erilaisia luokkia, joiden ominaisuuksien ja metodien avulla tietokantaa hallitaan. Kyselyt voidaan tehdä valmisteltuina, mikä parantaa tietoturvaa ehkäisemällä SQL-injektioita sekä tarkistamalla kyselyn oikeellisuuden. PDO tukee myös erinomaisesti virheenkäsittelyä, joka on tärkeää tämänkaltaisissa verkkosovelluksissa, koska mitään käsittelemätöntä tapahtumaa ei haluta jättää sovelluksen toimintalogiikkaan.

Karttaikkunaa varten sovellukseen haetaan karttakuvat ja ominaisuustiedot WMS- ja WFS-rajapintojen kautta. Selainsovellus ottaa yhteyden tietokantapalvelimen karttamoottoriin, joka palauttaa taustakartan halutulta alueelta ominaisuustietoineen.

5.6 Selainsovellus

Selainsovelluksen käyttöliittymä on toteutettu dynaamisilla HTML-elementeillä sekä JavaScriptillä, joten käyttöliittymän kaikki toiminnallisuus tapahtuu selaimessa suoritettavana. Sovelluksen Front Controller-ohjaimessa käytettiin ainoastaan palvelimella suoritettavaa PHP:tä.

Front Controller ohjaa sovelluksen toimintaa selainpuolella ottaen yhteyttä palvelinsovellukseen tietojenhakua ja muita prosesseja varten. Se on ainoa julkisesti saatavissa oleva sovellustiedosto, mikä alustaa ja ohjaa sovellusta toimimaan yhteydessä palvelinsovelluksen kanssa sisällyttäen siihen muita toiminnallisuuksia.

Ohjaimen lisänä käytettiin jQuery-JavaScript kirjastoa hoitamaan sivuston dynaamiset toiminnallisuudet. jQueryn valmiita ominaisuuksia ja elementtejä käyttämällä käyttöliittymästä saatiin tehtyä erittäin monipuolinen ja dynaaminen. Elementit mahdollistivat toteuttaa toiminnallisia lomakkeita ja muita rakenteita.

Karttakäyttöliittymä toteutettiin Openlayersillä, joka on avoimen lähdekoodin JavaScript-kirjasto karttatietojen näyttämiseen internet-selaimessa. Karttaikkuna sisältää perustoiminnot kartan selaukseen ja tietojen näyttämiseen. Openlayers viestii karttamoottorin rajapintojen kanssa tarjoten käyttäjälle taustakartan halutulta alueelta sekä siihen liittyvät kuvio- ja ominaisuustiedot.

5.7 Testaus

Sovelluksen kehitysvaiheessa tehtiin yksikkö- ja moduulitestausta PHPUnitilla, joka on Eclipseen saatava lisäominaisuus. Testitapauksista tehdään erillinen luokka, jonka avulla testattavat luokat tai funktiot ajetaan phpunit-komennoilla läpi.

Testiluokka sisältää erilaisia assert-metodeita eri parametreineen, millä testataan suorituksen toimintaa. Testattavan funktion ominaisuudet, parametrit ja paluuarvot testataan kaikki läpi erilaisilla variaatioilla. Testiajon tuloksena saadaan listaus läpimenneistä ja hylätyistä testitapauksista, joiden on oltava yksiselitteisiä vertaamalla funktion paluuarvoa odotettuun arvoon. Testauksen avulla saadaan minimoitua virheitä jo sovelluksen kehitysvaiheessa.

Alla olevassa esimerkissä malli PHPUnit-luokan assertTrue-metodin käytöstä, mikä testaa onko boolean-arvo ”true”. Testiluokassa testataan funktion palauttamaa arvoa eri parametreillä. Funktio palauttaa arvon ”true”, jos parametrina välitettävä tieto on mallia ”integer”. Testin suorittamisen jälkeen ilmoitetaan suoritetuista ja epäonnistuneista testeistä. (PHPUnit 2011.)

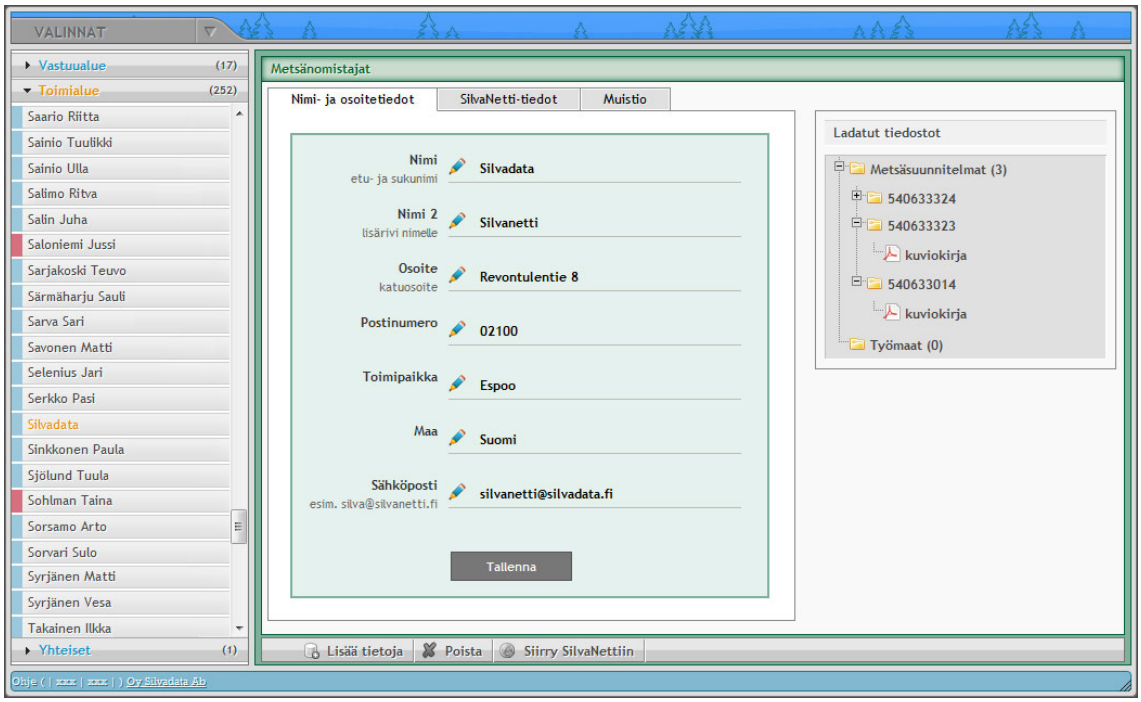
```
class UnitTest extends PHPUnit {
    $reference = new MainClass();
    $this->assertTrue($reference->function());
    $this->assertTrue($reference->function(1));
    $this->assertTrue($reference->function(“a”));
    $this->assertTrue($reference->function(true));
}

FAILURES!
Tests: 4, Assertions: 4, Failures: 3.
```

Käyttöliittymän JavaScript-koodin testaus täytyy toteuttaa ajon aikana suoritettavana debuggauksena. Tähän tarkoitukseen on saatavilla paljon erillisiä lisäohjelmia, kuten Firebug Mozilla Firefox-selaimeen. Firebugin avulla suoritettavat JavaScript-lauseet voidaan debugata läpi tarkastamalla niiden suoritusta ajonaikana. Debuggerista on nähtävissä mahdolliset virheet ja varoitukset sijaintitietoineen, joiden avulla ne ovat löydettävissä.

6 Tulokset

Tiedonhallintasovelluksen kehittämisessä tärkein ominaisuus oli saada toteutettua tiedonhallinta sovelluksen aikaisempien ja uusien määriteltyjen ominaisuuksien lisäksi. Tietojenhallinnasta tuli selkeä ja helposti käytettävä. Erilaiset hallittavat tiedot ovat nyt sijoiteltuina sovelluksessa omiin osioihinsa. Käyttöliittymän jQuery-elementtien dynaamiset toiminnot parantavat käytettävyyttä merkittävästi, kuten esimerkiksi metsänomistajan valinta, mikä on toteutettu accordion-listauksella, missä voidaan jakaa tiedot useisiin selattaviin osioihin (kuva 1). Kaikki sovelluksen elementit ovat toteutettu jQuerylla, joten tietojen hallinta on sujuvaa sekä myös toiminnallista ja näyttävää dynaamisuuden ansiosta.



Kuva 1. Metsänomistajan tietojen hallinta.

Sovelluksen avulla tiedonsiirto ja muut aikaisemmassa versiossa olleet toiminnallisuudet saivat parannuksia. Tiedonsiirrossa käyttäjä pääsee paremmin näkemään ja vaikuttamaan siirtotietojen sisältöön, kuten muokkaamalla tai poistamalla osaa tietosisällöstä.

Käyttöliittymä toteutui sovellusmaiseksi, missä toiminnot ovat omissa näkymissään. Eri toimintoja vaihdettaessa haetaan sivuun vain haluttu sisältö, eikä päivitetä koko näkymää uudestaan. Uskoisin myös käyttöliittymän toteuttavan käyttäjien kehitysehdotukset paremmasta ja selkeämmästä käytettävyydestä.

Syötettävien lomaketietojen tarkistus selaimessa suoritettavan JavaScript-koodin avulla saatiin käyttäjäystävällisemmäksi dynaamisten toimintojen avulla. Syötteet voidaan tarkistaa samanaikaisesti muita toimintoja suoritettaessa, jolloin käyttäjän ei tarvitse arvailla mitä tietoja lomakkeelle täytyy syöttää. Dialogimaiset sovelluksen päälle avautuvat lomakkeet ovat myös näyttäviä ja toimivia sekä sovelluksen toiminnan kannalta käytännöllisiä, koska käyttäjää ei tarvitse ohjata erilliselle sivulle (kuva 2).

Kuva 2. Dialog-pohjaiset lomakkeet.

Lopputuloksena tiedonhallintasovelluksen toteutuksesta tuli määrityksien ja suunnitellun mukainen. Sovelluksen avulla tietosisältöä voidaan hallita selkeästi sekä myös muut toiminnot paranivat entisestään. Dynaamiset tekniikat parantavat verkkosovellusta käyttäjän näkökulmasta huomattavasti, koska enää ei jokaisen toiminnon jälkeen tarvitse odottaa prosessin suorittamista tai sivun uudelleenlatausta. Myös toiminnallisuuksien käyttömukavuus ja näyttävyys on parempi.

Sovellusta ei oteta tässä vaiheessa vielä käyttöön, koska projekti vaatii myös muutoksia SilvaNetti-palveluun uusien ominaisuuksien ja tietosisällön muuttumisen osalta. Tietosisällön hallintaa on kuitenkin kokeiltu ja testattu niiltä osin, mitä SilvaNetti tällä hetkellä mahdollistaa. Nämä ominaisuudet, kuten tietojen hallinta täyttävät määritetyt vaatimukset ja kehitysehdotukset. Loppukäyttäjiltä saatuja kokemuksia uudesta järjestelmästä ei siis tässä vaiheessa vielä tiedetä.

Työn aikana tutkitut ja käytetyt menetelmät ja tekniikat auttavat kehittämään verkkosivuista nykyaikaisempia toiminnallisia verkkosovelluksia ja palveluita. Myös projektin aikana tehdyt määritelmät ja suunnitelmat tukevat sovelluksen jatkokehitystä, jota tällaisen jatkuvasti kehittyvän verkkopalvelun kohdalla on varmasti tulevaisuudessa tulossa lisää.

7 Pohdinta

Työn tutkimusosa avarsi näkemystä nykyiselle verkkosovelluskehitykselle. Tekniikat kehittyvät jatkuvasti ja mahdollisuuksia eri toteutustavoille ja tekniikoille on todella paljon. Myös vanhempien tekniikoiden käyttöä toteutuksissa näkyy vielä nykyisin, koska monet uudet tekniikat pohjautuvat niihin.

Dynaamisen verkkosovelluksen kehittämiseen on saatavilla erittäin paljon erilaisia frameworkkeja, kirjastoja, sovelluksia ja sivupohjia, minkä avulla kehitystyö on sujuvampaa. Verkkosovelluskehitys on muuttunut hyvin paljon desktop-sovelluskehityksen kaltaiseksi.

Projektin suunnitteluvaiheessa toteutetut mallit ja kaaviot tukivat tarkoituksenmukaisesti toteutusta ja saivat sovelluslogiikasta toimivamman ja selkeämmän. Suunnittelun merkitystä monesti vähäksytään, mutta toteutuksen myötä huomasi suunnittelun tärkeyden. Luokat ja metodit olivat valmiiksi suunniteltuina, joten ohjelmointityö oli siten huomattavasti sujuvampaa. Myös sovelluksen jatkokehitys on helpompaa, kun sovelluslogiikka on nähtävissä kaavioina, eikä tarvitse lukea koko lähdekoodia läpi ymmärtääkseen toimintaa.

Sovelluksen käyttöliittymästä saatiin erittäin toimiva ja käytettävyydeltään parempi dynaamisuuden myötä. Käyttäjä voi selata, hakea, muokata ja käyttää toiminnallisuuksia ilman että minkäänlaista tiedonsiirtoa palvelimelle suoritetaan, tai että sivua jouduttaisiin päivittämään jatkuvasti uudestaan. jQueryn elementit tarjoavat dynaamiset vaihtoehdot monille erilaisille verkkosivun osille.

Verkkopalveluihin jotka sisältävät suuren määrän tietoa, on välttämätöntä kehittää tietojen hallintaa, koska suuria määriä tietoa sisältävät tietokannat ovat erittäin vaikeasti hallittavissa. Työssä toteutetun kaltainen tietosisällön hallintasovellus toteuttaa SilvaNetti-verkkopalvelun tietojen hallinnan.

Työn suunnittelu ja toteutus onnistui mielestäni hyvin. Dynaamisten verkkosovellustekniikoiden tutkiminen sekä niiden pohjalta suunnitellun ja toteutetun sovelluksen toteutus asetti työlle tarpeeksi haasteita ja sitä kautta mielenkiintoa. Verkkosivuja pystytään tekemään näyttävimmillä ja toimivammilla tekniikoilla.

Sovelluksen käyttöönoton siirtyessä myöhempään ajankohtaan jäi projektin aikana toteuttamatta käyttöönottovaihe. Monet muutosehdotukset olivat kuitenkin asiakaslähtöisiä, joten on mielenkiintoista nähdä jatkossa saadut palautteet, kun järjestelmä otetaan käyttöön. Käyttöönotto on suunniteltu tapahtuvaksi vaiheittain siten, että vanha hallintasovellus toimii uuden rinnalla jonkin aikaa. Palvelua käyttävät suurimmat organisaatiot saavat uuden sovelluksen käyttäjätestaukseen ensimmäisenä, koska näin saadaan parhaiten esille tiedonhallinnan tehokkuus hallittaessa suurta määrää palveluun siirrettyä tietoa.

Verkkopalvelut, kuten SilvaNetti, kehittyvät jatkuvasti uusien tarpeiden ja vaatimuksien mukaisesti. Projektin jatkokehitystarpeeksi ilmeni vanhan KKJ-karttakoordinaatiston muunnos EUREF-muotoiseksi. Tietokannassa paikkatiedot ovat nykyisellään vielä KKJ-muotoisina. Sovellukseen toteutetaan muunnin, joka muuntaa koordinaatiston tietoja siirtäessä, joten kaikki järjestelmään ladatut tiedot muunnetaan EUREF-koordinaatistoon. Muita jatkokehitystarpeita on kehittää edelleen viestintä-toimintoa, jolloin käydyt viestiketjut voisivat olla luettavissa SilvaNetistä. Metsänomistajalle jäisi metsäpalstakohtaisesti järjestelmään tietoja suunnitelluista ja toteutetuista toimenpiteistä.

Lähteet

2KMediat. 2011. Dynaaminen HTML ja DOM.
http://www.2kmediat.com/dhtml/johdanto2.asp. Viitattu 15.6.2011.

Apple Developers. 2011. XMLHttpRequest object.
http://developer.apple.com/internet/webcontent/xmlhttpreq.html.
Viitattu 9.6.2011.

Asleson R & Schutta D. 2007. Ajax - tehokas hallinta. Helsinki: Readme.fi.

Falck K. 2011. Photoshop on kuollut. Tietokone 30 (5), 20.

Korpela J. 2011. Web-julkaisemisen opas. JavaScript.
http://www.cs.tut.fi/~jkorpela/webjulk/3.2.html . Viitattu. 15.6.2011.

Microsoft Development. 2011. Model-View-Controller.
http://msdn.microsoft.com/en-us/library/ff649643.aspx
Viitattu 27.6.2011.

Oracle Technology Network. 2011. Asynchronous JavaScript Technology.
http://www.oracle.com/technetwork/articles/javaee/ajax-135201.html.
Viitattu 20.6.2011.

Peltomäki J & Nykänen O. 2006. Web-selainohjelmointi. Helsinki: WSOY.

PHPUnit. 2011. PHPUnit Manual.
http://www.phpunit.de/manual/3.6/en/index.html. Viitattu 30.6.2011.

Silvadata. Yritystiedot. 2011. http://www.silvadata.fi/. Viitattu 1.6.2011.

SilvaNetti. 2011. Metsänomistajan verkkopalvelu.
http://www.silvadata.fi/tuotteet/esitteet/materiaalit/SilvaNetti_esite.pdf.
Viitattu 5.6.2011.

Smooth Step. 2011. Dynamic Websites.
http://www.smooth-step.com/web-design/dynamic-websites.
Viitattu 10.6.2011.

Sun Microsystems. 2011a. Ajax Design Strategies.
http://java.sun.com/developer/technicalArticles/J2EE/AJAX/DesignStrategies/. Viitattu 9.6.2011.

Sun Microsystems. 2011b. Model-View-Controller.
http://java.sun.com/blueprints/patterns/MVC-detailed.html
Viitattu 27.6.2011.

The Code Project. 2011. Simple and Practical Introduction to Asynchronous JavaScript and XML (AJAX).
http://www.codeproject.com/KB/ajax/practicalajax.aspx. Viitattu 8.6.2011.

The jQuery Project. 2011. jQuery User Interface. http://jquery.com/.
Viitattu 22.6.2011.

The World Wide Web Consortium W3C. 2011a. XMLHttpRequest.
http://www.w3.org/TR/XMLHttpRequest/. Viitattu 21.6.2011.

The World Wide Web Consortium W3C. 2011b. Status Code Definitions.
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.
Viitattu 5.7.2011.