Opinnäytetyö (AMK)

Tietotekniikka

Internet-tekniikka

2011

Jussi Mäkilä

WWW-SOVELLUKSEN TOTEUTUS MVC-ARKKITEHTUURILLA

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

Turun ammattikorkeakoulu

Tietotekniikka | Internet-tekniikka

Kesäkuu 2011 | Sivumäärä 27

Ohjaajat: Ins. Olli Ojala, TKL Juha Nikkanen

Jussi Mäkilä

WWW-SOVELLUKSEN TOTEUTUS MVC-ARKKITEHTUURILLA

Opinnäytetyön aiheena oli toteuttaa eräälle yritykselle myynnin- ja varastonseurantaohjelma. Tämä ohjelma piti toteuttaa käyttämällä PHP-ohjelmointikieltä ja MySQL-tietokantaa. Ohjelma tulisi ajettavaksi Internet-palvelimelle. Ohjelmointi toteutettiin käyttämällä hyväksi MVC-arkkitehtuuria ja olio-ohjelmointia.

Työ suunniteltiin yhteistyössä kohdeyrityksen kanssa. Vaatimusmäärittelyn ja käyttötapausten avulla saatiin suunniteltua vaadittavat tietokantarakenteet ja oliot. Ohjelmointi tehtiin MVC-arkkitehtuurin mukaisesti erottamalla liiketoimintalogiikka eli mallit käyttöliittymästä eli näkymistä. Näkymät ja mallit yhdistettiin toisiinsa ohjaimilla. Nämä ohjaimet säätelevät tiedon kulkua käyttäjän ja ohjelman välillä, mutta myös mallin ja näkymän välillä.

Tuloksena saatu ohjelma on helposti ylläpidettävä ja jatkokehitettävä sen selkeän rakenteen ja olio-pohjaisuuden ansiosta. Käyttöliittymää voidaan muokata tarpeen mukaan, ilman että se vaikuttaa liiketoimintalogiikkaan. Tämä mahdollistaa helpon jatkokehityksen ja kehitystyön hajauttamisen eri osa-alueisiin. Työn toteuttaminen käyttämällä MVC-arkkitehtuuria on myös huomattavasti nopeampaa kuin perinteisillä tekniikoilla, varsinkin jos apuna käytetään jotain ohjelmistokehystä.

ASIASANAT:

WWW-ohjelmointi, ohjelmistokehitys, järjestelmäarkkitehtuuri, PHP, JavaScript

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Internet Technology

June 2011 | 27 pages

Instructors: Olli Ojala, B. Eng., Juha Nikkanen, Lic. Tech., Principal Lecturer

Jussi Mäkilä

DEVELOPING A WEB APPLICATION USING THE MVC ARCHITECTURE

The goal of this thesis was to produce an application to monitor sales and storage for a certain company. This application was to be made using the PHP programming language and a MySQL database. The application was going to be a web application, run on a web server. The application was developed using the MVC architecture.

The project was designed in cooperation with the company. A software requirements specification and use cases were used to design the needed data structures and objects. The programming was made in accordance with the MVC paradigm, separating business logic from presentation. The view and the models were connected to each other via controllers. These controllers not only control the flow of information between the model and the view, but also between the user and the application.

The product was a program that is easily maintained and developed further due to its concise structure and object oriented design. The user interface is easily modified according to future needs without any impact on the business logic. This makes future development easier by giving the possibility to distribute the development work on different aspects of the application. Using the MVC architecture to develop applications is also a lot faster than using traditional methods, especially when using an application framework.

KEYWORDS:

Web development, application framework, system architecture, PHP, JavaScript

SISÄLTÖ

LYHENTEET

Ajax	(*Asynchronous Javascript And Xml*) Joukko tekniikoita, joilla verkkosivuista saadaan vuorovaikutteisempia.
CRM	(*Customer Relations Manage*) eli asiakkuudenhallinta on käsite, joka sisältää asiakaslähtöisen ajattelutavan yrityksessä sekä siihen liittyvät tietojärjestelmät.
CSS	(*Cascading Style Sheets*) Www-dokumenteille kehitetty tyyliohjeiden laji.
DOM	(*Document Object Model*) Malli kuvata verkkosivun rakennetta hierarkkisesti oliohin tai elementeihin perustuen.
ERP	(*Enterprise Resource Planning*) eli toiminnanohjausjärjestelmä on yrityksen tietojärjestelmä, jolla voidaan yhdistää monia eri yrityksen sisäisiä toimintoja.
HTML	(*HyperText Markup Language*) Merkintäkieli, jolla kuvataan hyperlinkkejä sisältävää tekstiä, eli hypertekstiä. Käytetään pääasiassa www-sivujen koodaamiseen.
HTTP	(*HyperText Transfer Protocol*) Protokolla jota selaimet ja www-palvelimet käyttävät tiedonsiirtoon.
JSON	(*JavaScript Object Notation*) JavaScript-tietorakenteiden merkintätapa merkkijonona.
MVC	(*Model-View-Controller*) Ohjelmistoarkkitehtuuri, jonka tarkoituksena on erottaa käyttöliittymä sovellusaluetiedosta.
PHP	(*PHP: Hypertext Preprocessor*) Ohjelmointikieli, jota käytetään erityisesti Web-palvelinympäristöissä dynaamisten web-sivujen luonnissa.
POST, GET	HTTP-protokollaan kuuluvia metodeja tai verbejä, joita käytetään tiedon siirrossa.
REST	(*Representational State Transfer*) Ohjelmistoarkkitehtuuri hajautettua hypermediaa, kuten World Wide Webiä, varten.
SQL	(*Structured Query Language*) Kyselykieli, jolla relaatiotietokantaan voidaan tehdä muutoksia.
XML	(*eXtensible Markup Language*) Merkintäkieli, jossa tiedon merkitys on kuvattavissa tiedon sekaan.

1 JOHDANTO

Opinnäytetyössä perehdytään erään yrityksen kanssa yhteistyössä suunniteltuun varaston- ja myynninseurantaohjelmaan, joka toteutettiin käyttämällä hyväksi MVC-arkkitehtuuria. Työn tarkoitus oli toteuttaa PHP-ohjelmointikielellä ohjelma, joka tultaisiin sulauttamaan myöhemmin suurempaan ohjelmakokonaisuuteen.

Tehtävänä oli toteuttaa tehokkaasti ja mahdollisimman selkeästi ohjelma, jonka jatkokehitys olisi mahdollista ilman suurta vaivaa. MVC-arkkitehtuuri valittiin, koska se on hyvin hallittava ja siihen on uuden sisällön lisääminen helppoa. MVC-arkkitehtuuri on ohjelmistoarkkitehtuurityyli, jonka tarkoituksena on erottaa käyttöliittymä liiketoimintalogiikasta. Käyttöliittymä eli näkymä on erillään liiketoimintalogiikkaa suorittavasta mallista. Näitä kahta yhdistää ohjain, jolla käyttäjän komentoja ja ohjelman palauttamia tuloksia ohjataan.

Työ toteutettiin käyttämällä PHP-ohjelmointikieltä, ja tiedon tallentamisessa käytettiin MySQL-tietokantaa. Käyttöliittymän toteutukseen käytettiin HTML- ja JavaScript-kieliä. Osa käyttöliittymän toiminnoista toteutettiin käyttämällä Ajax-toiminnallisuutta. Nämä tekniikat valittiin, koska ohjelmakokonaisuus, johon tämä ohjelma liitetään, on toteutettu samoilla tekniikoilla. Integrointi ei ole sujuvaa, jos ohjelmistoalustat ovat erilaiset. Ohjelmaa ajetaan Internet-palvelimella, joka tuo mukanaan useita etuja verrattuna perinteisiin työpöytäsovelluksiin. Tällaisia palveluja kutsutaan pilvipalveluiksi.

2 ARKKITEHTUURI

2.1 Olio-ohjelmointi

Olio-ohjelmointi on yksi nykyaikaisen ohjelmoinnin perusteista. Se on lähestymistapa ohjelmointiin, jolla työtä saadaan jäsenneltyä ja yksinkertaistettua. Käyttämällä olio-ohjelmointia voidaan ratkaista monia ohjelmoinnin yleisiä ongelmia ja ohjelman suunnittelu helpottuu huomattavasti. [1]

Ideana on jakaa tai jäsennellä ohjelman suunnittelu käsitteellisiin olioihin (engl. object). Nämä oliot ovat pieniä ohjelman osia, jotka voivat omata erilaisia toimintoja ja ominaisuuksia. Olio voidaan nähdä erillisenä yksikkönä, joka suorittaa tehtäviä ohjelmassa tai varastoi tietoa. Tällainen voi olla esimerkiksi kirjasto-ohjelman tapauksessa kirja, jolla on ominaisuuksia, kuten kirjoittaja, julkaisuvuosi ja nimi. Toinen olio voisi olla lainaaja, jolla on ominaisuuksina henkilötietoja ja toimintoina vaikkapa lainaaminen ja palauttaminen. Oliolla on usein sekä ominaisuuksia että toimintoja. Ohjelmoitaessa nämä oliot merkitään *luokiksi* ja niille luotavat ominaisuudet ovat muuttujia tai *attribuutteja* ja toiminnot funktioita tai *metodeja* (Ohjelma 1). [2]

Ohjelma 1. Esimerkki luokasta PHP-kielellä toteutettuna. Luokalla ”Oppilas” on attribuutti ”arvosana” ja metodi ”naytaArvosana”, joka näyttää oppilaan arvosanan.

```
class Oppilas {
  public var $arvosana = 8;
  public function naytaArvosana() {
    echo $this->arvosana;
  }
}
```

Yksi tärkeä ajatus olioiden takana on niiden vuorovaikutus toisten olioiden kanssa. Näistä vuorovaikutustavoista merkittävin on perinnöllisyys. Olioille voidaan luoda lapsiolioita, joille periytyy niiden vanhemmasta attribuutteja ja metodeja (Ohjelma 2). Tämä helpottaa ohjelmointityötä huomattavasti, koska perustoiminnot voidaan antaa yhdelle oliolle ja sitten luoda tälle lapsia, jotka erikoistuvat tiettyihin

toimintoihin tai osa-alueisiin, mutta joilla on kuitenkin kaikki perustoiminnot vanhemmastaan periytyneinä. [2]

Ohjelma 2. Tässä PHP-esimerkissä luokka ”Oppilas” perii luokan ”Ihminen” attribuutit ja metodit samalla myös laajentaen alkuperäistä luokkaa.

```
class Ihminen {
   public var $ika = 23;
   public var $nimi = ”Matti Meikäläinen”;
   public function muutaNimi ($nimi) {
      $this->nimi = $nimi;
   }
}

class Oppilas extends Ihminen {
   public var $arvosana = 8;
   public function naytaArvosana() {
      echo $this->arvosana;
   }
   public function muutaIka($uusiika) {
      $this->ika = $uusiika;
   }
}
```

Olio määritetään luokalla, jonka sisään määritetään ne attribuutit ja metodit, joita sen halutaan omistavan. Kun jossain ohjelman osassa halutaan käyttää tätä oliota, se pitää luoda. Olio luodaan tallentamalla se uuteen muuttujaan. Tätä muuttujaa voidaan tämän jälkeen manipuloida käytettävän ohjelmointikielen mahdollistamilla tavoilla.

Kaikenlaiseen ohjelmointiin tämä tekniikka ei kuitenkaan sovellu. Olio-ohjelmointi on parhaimmillaan tehtäessä suuria ohjelmia, joissa lähdekoodia muodostuu paljon ja ohjelman organisointi on perinteisillä tavoilla vaikeaa. Jos ohjelman koodiosiot ovat lyhyitä, on perinteinen funktioihin ja suoriin käskyihin perustuva tapa usein helpompi ja nopeampi toteuttaa.

2.2 Asiakasohjelma-palvelinmalli

Asiakasohjelma-palvelinmalli on tietojenkäsittelyssä käytetty hajautettu sovellusmalli, jossa sovelluksen työt jaetaan palvelimen ja asiakasohjelman välille. Palvelimella ajetaan palvelinsovellusta, jossa ohjelman resurssit sijaitsevat ja jossa ohjelman liiketoimintalogiikka yleensä suoritetaan. Asiakasohjelma tekee kutsuja

palvelimelle ja pyytää resursseja, jotka asiakasohjelma sitten esittää käyttäjälle. Käyttäjä ei ole suoraan tekemisissä palvelimen kanssa, vaan kaikki toiminta tapahtuu asiakasohjelman välityksellä. Yhtä palvelinta kohden voi olla useita asiakasohjelmia, jotka voivat käyttää palvelimen resursseja samanaikaisesti. Kommunikointi näiden kahden osapuolen välillä tapahtuu usein tietoverkon välityksellä. [3]

Internet-sovellukset ovat hyvä esimerkki asiakasohjelma-palvelinmallista. Palvelinohjelmaa ajetaan HTTP-palvelimella jossain päin Internetiä, ja käyttäjä tekee kutsuja asiakasohjelman eli selaimen välityksellä.

2.3 REST-arkkitehtuuri

REST (Representational State Transfer) on eräänlainen ohjelma-arkkitehtuuri, jota käytetään hajautetussa hypermediassa, kuten World Wide Webissä. REST-arkkitehtuuri nojaa vahvasti asiakasohjelma-palvelinmalliin, jossa asiakasohjelmat lähettävät pyyntöjä palvelimelle joka käsittelee ne ja palauttaa vastauksen asiakasohjelmalle. Nämä pyynnöt ja vastaukset ovat rakenteeltaan eräänlaisia esityksiä resursseista. Resurssi voi olla mikä tahansa tieto, jota voidaan jollain järkevällä tavalla kuvata ja esitys siitä on usein jokin dokumentti, joka käsittelee tämän resurssin tilaa. [4] Paras esimerkki tästä on Internet-sivu. Sovellus on oletuksena levossa ja muuttaa tilaansa vain saatuaan pyynnön asiakasohjelmalta. Internet-sovellusten tapauksessa tämä tarkoittaa sitä, että jos asiakasohjelma, eli selain, ei päivitä sivua, on taustalla toimiva palvelinsovellus levossa eikä tietoinen mistään käyttäjän mahdollisesti tekemistä muutoksista. Käyttäjä voi tehdä muutoksia sivulla, mutta ennen kuin hän päivittää sivun tavalla tai toisella, mitään ei tallennu. REST-arkkitehtuuria kehitettiin samanaikaisesti HTTP-protokollan kanssa, joka sisältää monia sille ominaisia komponentteja, mutta sitä voidaan käyttää myös muilla sovellustason protokollilla [5]. Www-sovelluksen tapauksessa on käytössä yleensä HTTP-protokolla.

Asiakasohjelma kommunikoi palvelimen kanssa eräänlaisten *verbien* välityksellä. Nämä verbit ovat määritetty osaksi HTTP-protokollaa, jolla www toimii. HTTP-

protokolla sisältää tuen useille verbeille, mutta täydellistä REST-tukea ei kuitenkaan yleensä ohjelmille luoda. Yleisimmin käytetään vain GET- ja POST-verbejä muiden verbien ollessa huonosti tuettuja eri selainten ja ohjelmointikielien puolesta. GET-verbillä noudetaan tietoa palvelimelta näytettäväksi käyttäjälle. POST-verbillä muutetaan palvelimen tilaa esimerkiksi lisäämällä tietokantaan uutta tietoa tai muuttamalla sitä. [4]

2.4 MVC-arkkitehtuuri

MVC (Model-View-Contoller), tai suomeksi malli-näkymä-ohjain, on ohjelmointiarkkitehtuuri, jossa ohjelma jaetaan selkeisiin loogisiin osiin [6]. Tämä jakaminen, kuten moni muukin nykyaikaisen ohjelmoinnin ajatusmalleista, noudattaa ideologiaa, jonka perusteella tietokoneohjelma tulisi jakaa sellaisiin osiin että päällekkäisyydet saataisiin minimoitua (engl. separation of concerns) [7]. Näin jotain ohjelman osaa voidaan tutkia erillään muista osista. Tämä helpottaa monimutkaisten ohjelmien hallittavuutta ja kehittämistä. MVC-arkkitehtuurissa ohjelma jaetaan kolmeen loogiseen osaan; malliin, näkymään ja ohjaimeen [8]. Itse asiassa nämä ovat usein käytännössä kokoelmia useita malleja, näkymiä ja ohjaimia. MVC-arkkitehtuuri toimii erityisen hyvin www-sovellusten kehityksessä, koska nämä kyseiset osat on niissä helppo erottaa toisistaan. Www-sovelluksissa myös verrattain usein syntyy tarvetta käyttöliittymän muokkaamiseen jälkeenpäin. Tähän on MVC-arkkitehtuuri myös omiaan, sillä käyttöliittymä on erillään muusta toiminnallisuudesta ja näin helposta muokattavissa.

Näkymä käsittää käyttöliittymän ja siihen liittyvän koodin. Periaatteessa kaikki, minkä käyttäjä näkee, kuuluu näkymän piiriin. Käyttäjä pystyy vaikuttamaan ohjelman toimintaan vain näkymän kautta. Malli sisältää ohjelman liiketoimintalogiikan. Tähän kuuluu usein esimerkiksi tietokantojen hallinta, eli resurssien nouto ja muokkaaminen. Ohjain toimii näiden kahden välissä ja välittää käyttäjän antamat ohjeet näkymästä mallille ja taas mallilta käyttäjälle näkymään. Perusideana on, että ohjaimet ovat mahdollisimman kevyitä, eivätkä sisällä logiikkaa enempää kuin mikä on pakollista toiminnan ohjaamiseen. [8]

2.5 Ohjelmistokehykset

Nykyään monet monimutkaisemmat www-sovellukset toteutetaan käyttäen jotain ohjelmistokehystä. Ohjelmistokehys on alusta, jonka päälle sovellus kehitetään. Se yleensä sisältää paljon valmista koodia, jota voidaan käyttää nopeuttamaan sovelluksen kehitystä ja koodin kirjoittamista. [9] Varsinkin ohjelmistoyrityksille, joissa sovelluksia luodaan lyhyellä aikavälillä – ja usein samanaikaisesti – tällaiset ohjelmistokehykset ovat tärkeä apu. Suurin osa ohjelmistokehyksistä on olio-pohjaisia ja monet niistä käyttävät MVC-arkkitehtuuria. Huono puoli monissa ohjelmistokehyksissä on niiden raskaus ja se, että ne saattavat sisältää paljon turhaa koodia ja toiminnallisuutta. Uuden ohjelmistokehyksen opetteleminen vie myös usein paljon aikaa [9].

Lähtökohtana koko työssä oli, että ohjelman tulee olla mahdollisimman taloudellinen ja kevyt ja sen tulisi sisältää mahdollisimman vähän redundanttia ja turhaa koodia. Monet valmiit ohjelmistokehykset sallivat myös näkymän keskustelemisen suoraan mallin kanssa. Tässä työssä kuitenkaan ei tätä lähestymistapaa käytetty. Ideana oli, että kaiken toiminnan pitää kulkea ohjaimen kautta, jotta ohjelma pysyy mahdollisimman hallittavana. Näistä syistä työssä ei otettu käyttöön valmista ohjelmistokehystä. Ohjelmistokehyksen mallia tosin sovellettiin tietyssä mielessä. Ohjelman rakenne suunniteltiin vastaamaan ohjelmistokehyksiä sillä tavalla, että koodi olisi mahdollisimman helposti uudelleenkäytettävää, ja ohjelman liiketoimintalogiikkaa koodatessa olisi mahdollisuus käyttää valmiita yhteisiä toimintoja ja tietovarastoja. Käytännössä tämä on yksinkertainen järjestelmä, jossa ohjelman eri osat on jaettu eri tiedostoihin ja eri kansioihin sovelluksen suoritushakemistossa. Valmiita toimintoja ovat mm. tietokannan kyselyt ja näkymän eri komponenttien luominen.

Pitää kuitenkin ottaa huomioon että tällainen ohjelmistokehys ja MVC–järjestelmä koskevat vain palvelimella ajettavaa ohjelmaa. Asiakasohjelmassa, eli selaimessa, toimii myös omaa ohjelmalogiikkaa, kuten JavaScript-komentosarjat ja Ajax-toiminnallisuus [10]. Nämä eivät toimi samalla periaatteella kuin palvelimen REST-tyyppinen ohjelmisto, eikä niitä voida myöskään ajatella samanlaiseksi

kokonaisuudeksi. Selaimen ohjelmisto voi pyytää ja muuttaa resursseja ilman käyttäjän nimenomaista komentoa [10]. Selaimella toimiva logiikka yleensä käsittää lähinnä käyttöliittymän muokkaamiseen liittyviä toimintoja, eikä niinkään tärkeitä ohjelman toimintoja. Selaimessa ajettavien komentosarjakielien tietoturva on heikkoa ja tärkeät toiminnot on siitä syystä järkevä pitää siitä erillään [11].

2.6 Ajax

Ajax-lyhenne tulee englannin kielen sanoista *Asynchronous JavaScript And XML*. Ajax on käytännössä nimitys JavaScript-kielellä toteutettavista pyynnöistä, jotka toimivat asynkronisesti muiden HTTP-pyyntöjen kanssa. Eli toisin kuin normaalit HTTP-pyynnöt, Ajax-pyynnöt eivät vaadi sivun päivitystä suorituksessaan. Ajax noudattaa samoja REST-arkkitehtuuriin kuuluvia tekniikoita ja tapoja, mutta toimii normaalien sivunpäivitysten taustalla käyttäen hyväkseen JavaScriptin XMLHttpRequest-oliota. Pyyntö on periaatteessa samanlainen kun normaali HTTP-pyyntö, mutta se suoritetaan taustalla, ilman että käyttäjälle näytetään mitään muutosta sivulla. [12]

Alun perin ideana on ollut, että pyyntö palauttaa datan aina XML-muodossa [12], mutta se ei ole pakollista ja nykyään harvoin totta. XML-koodin tulkitseminen JavaScriptin ymmärtämiksi tietorakenteiksi on työlästä ja monimutkaista. On paljon tehokkaampaa palauttaa suoraan HTML-koodia, jolla voidaan suoraan korvata osia sivusta, tai JSON-notaatiolla toteutettu merkkijono, joka voidaan tulkita JavaScriptissä suoraan olioiksi tai muiksi tietorakenteiksi.

JavaScriptin ja Ajaxin avulla voidaan välttää REST-arkkitehtuurin rajoitukset, jotka tekevät muutosten tallennuksen ilman sivun uudelleenlatausta mahdottomaksi. JavaScriptillä on mahdollista seurata käyttäjän tekemiä muutoksia sivulla reaaliajassa ja sitten toteuttaa taustalla Ajax-pyyntöjä palvelinsovellukselle näin muokaten ohjelman tilaa. Ohjelman tilan muutokset on mahdollista näyttää käyttäjälle välittömästi käyttämällä JavaScriptiä muokkaamaan verkkosivun DOM:a. Sivun uudelleenlatausta ei tarvita, ja käyttäjälle välittyy entistä rikkaampi ja reagoivampi kokemus ohjelman käyttöliittymästä. [12]

Ajax-tekniikka on nykyään hyvin suosittu ja lähes kaikki suuret nykyaikaiset sivustot käyttävät sitä jotenkin hyväkseen. Esimerkiksi Facebook käyttää Ajax-tekniikkaa lukuisissa eri toiminnoissaan.

2.7 Pilvimalli

Käyttöympäristöä, jossa ohjelmistot ajetaan jossain päin Internetiä eli "pilvessä", kutsutaan nykyajan trendien mukaisesti pilvimalliksi. Käyttäjät käyttävät tällaisia ohjelmia jonkin asiakasohjelman kautta, tässä tapauksessa Internet-selaimen. Koska ohjelma toimii Internetissä, asiakasyritysten käyttävät voivat käyttää ohjelmaa maantieteellisestä sijainnistaan riippumatta. Samaa ohjelmaa voi myös käyttää moni käyttäjä samanaikaisesti, ilman että tehtävät toiminnot menevät sekaisin. [13]

Sijainnista riippumattomuuden ja monen käyttäjän tukemisen lisäksi tällä mallilla on muitakin etuja. Yrityksen ei tarvitse sijoittaa pääomaa palvelimiin, sillä ohjelmaa ajetaan kolmannen osapuolen laitteistoilla. Päätelaitteille ei tarvitse asentaa mittavia ohjelmistoja. Asiakasohjelmat ovat usein pieniä ja monesti jopa osana käyttöjärjestelmää.

Pilviohjelmat toimivat SaaS (Software as a Service) -periaatteella ja niiden hinnoittelu on usein käytön mukaan [13]. Hinnoittelu voi olla kuukausittainen, käyttäjien määrän mukaan määräytyvä, resurssien käytön mukaan määräytyvä tai jokin näiden yhdistelmä. Joka tapauksessa, hinnoittelutapa mahdollistaa yritykselle ennakoitavan maksusuunnitelman. Yrityksen ei tarvitse sitoa suurta määrää pääomaa ohjelma- ja laitteistohankintoihin kun maksu tapahtuu osissa.

3 TEKNIIKKA

3.1 PHP

PHP (rekursiivinen akronyymi sanoista *PHP: Hypertext Preprocessor*) on laajalle levinnyt komentosarjakieli, joka soveltuu erityisen hyvin www-sovellusten ja

dynaamisten verkkosivujen ohjelmointiin [14]. PHP on palvelinpuolen ohjelmointikieli ja sitä ajetaan www-palvelimen yhteydessä. PHP on komentosarjakieli, joka tarkoittaa sitä, että lähdekoodi tulkitaan suorituksen aikana. Koodia ei näin ollen tarvitse kääntää ensin konekieliseksi binääritiedostoksi. Tästä on hyötyä varsinkin ohjelman kehitys- ja testausvaiheessa, kun muutokset ohjelmaan voidaan tehdä nopeasti siirtämällä vain uusi lähdekooditiedosto palvelimelle. PHP:ta voidaan myös upottaa HTML-koodin sekaan, joka tekee siitä oivallisen juuri www-sivujen ohjelmoimisessa (Ohjelma 3). PHP on ilmainen, ja se on julkaistu *PHP License* -ohjelmistolisenssin alaisuudessa. [15]

Ohjelma 3. Esimerkki PHP-koodista sulautettuna HTML-koodin keskelle. Ohjelma piirtää ruudulle tekstin ”Hello World”. PHP-koodi on erotettavissa lähdekoodista ”<?php”- ja ”?>”-merkinnöin.

```
<html>
  <head>
    <title>Minun sivuni</title>
  </head>
  <body>
    <p><?php echo ”Hello World”; ?></p>
  </body>
</html>
```

PHP oli alun perin tanskalaisen Rasmus Lerdorfin kehittämä kokoelma Perl-kielen komentosarjoja, jotka hän oli suunnitellut oman kotisivunsa ylläpitoon. Myöhemmin jatkokehitettyään ideaa hän julkaisi kokoelman nimellä ”Personal Home Page/Forms Interpreter”. Kaksi israelilaista kehittäjää, Zeev Suraski ja Andi Gutmans, paransivat ohjelman jäsennintä ja myöhemmin kehittivät *Zend Enginen*, josta tuli PHP:n ydin versiosta 4 eteenpäin. [16]

Tähän työhön valittiin PHP sen ilmaisuuden, matalan oppimiskynnyksen sekä hyvä tuen vuoksi. PHP on, kuten tässäkin tapauksessa, usein osana Linux-palvelimia jo valmiiksi. Erillistä tukea ohjelmointikielelle ei siis tarvinnut asentaa. PHP on yksi tämän hetken käytetyimpiä www-ohjelmointikieliä, ja sen dokumentointi ja käyttäjäyhteisö ovat pitkälle kehittyneitä.

3.2 MySQL

MySQL on suosittu relaatiotietokannan hallintajärjestelmä. MySQL käyttää IBM:n kehittämää SQL-kyselykieltä, jolla voidaan tehdä relaatiotietokantaan hakuja ja muutoksia. MySQL-tietokanta on suomalaisen Michael Wideniuksen ja ruotsalaisen David Axmarkin luoma. [17]

Tietokanta on välttämätön osa tätä opinnäytetyötä. Periaatteessa mikä tahansa SQL-tietokanta kävisi, sillä kirjoitusasu on vakio kaikissa SQL-kieltä käyttävissä tietokannoissa. MySQL valittiin sen ilmaisuuden ja hyvän tuen vuoksi. PHP:ssä on valmiina oma rajapinta MySQL-tietokannan hallintaan, joka mahdollistaa helpon ja hyvin dokumentoidun tavan hallita tietokantaa [18]. MySQL on myös yleensä Linux-pohjaisissa palvelimissa valmiiksi asennettuna, tai se on niihin helppo lisätä.

3.3 JavaScript

JavaScript on yleisesti käytetty Internet-selaimessa ajettava komentosarjakieli. Sen kehitti alun perin Netscape Communication Corporation nimellä Mocha. Markkinointisyistä nimi lopulta muotoutui JavaScriptiksi. Java-ohjelmointikielen kanssa JavaScriptillä ei ole kuitenkaan ole yhteyttä. JavaScript on nykyään kehittynyt oliopohjainen komentosarjatyyppinen ohjelmointikieli. Siinä on myös vaikutteita funktionaalisista ohjelmointikielistä, kuten esimerkiksi lamda-funktiot. JavaScriptit toiminnallisuus ja ominaisuudet riippuvat käytettävästä selaimesta. JavaScript 1.5 on viimeisin standardi, jota lähes kaikki selaimet tukevat. Lisäksi monet uudet selaimet tukevat kieleen tehtyjä laajennuksia. [10]

3.3.1 JavaScriptin käyttö

JavaScript on tärkeä osa Web 2.0:aa ja sitä myöhempiä teknologioita, jotka määrittävät Word Wide Webiä asiakaskeskeisenä kokonaisuutena. JavaScript on tärkeimpiä rikkaan käyttäjäkokemuksen ja reaaliaikaisen käyttöliittymän mahdollistavia tekniikoita. Tuleva HTML 5 -standardi siirtää osan nykyisen JavaScriptin käyttöliittymään liittyvistä töistä suoraan verkkosivun HTML koodin tehtäväksi. Samalla uusi standardi tosin tuo JavaScriptiin lukuisia uusia

ominaisuuksia [20]. JavaScript tulee näin ollen olemaan tärkeä osa verkkosivuja vielä tulevaisuudessakin.

DOM (Document Object Model) on yleinen tapa kuvata HTML-sivun rakennetta hierarkkisesti erilaisin olioin tai elementein [19]. JavaScriptillä on mahdollista manipuloida DOM:a reaaliajassa, eli esimerkiksi muuttaa sivun sisältöä, ilman sivunlatauksia [10]. Tämä mahdollistaa entistä huomattavasti reagoivamman ja käyttäjäystävällisemmän kokemuksen käyttäjälle.

3.3.2 jQuery

jQuery on Internetin käytetyin JavaScript-kirjasto, jonka tarkoituksena on helpottaa ja nopeuttaa ohjelmointia JavaScript-kielellä [21]. Suuri osa JavaScriptillä tehtävästä ohjelmoinnista on tapahtumien käsittelyä. Tapahtumilla seurataan käyttäjän tekemiä muutoksia sivuun, jotta niihin voidaan reagoida ilman käyttäjän suorittamaa sivun uudelleenlatausta. jQuery tarjoaa hyvät oikotiet monille muuten työläille toistuville algoritmeille varsinkin tapahtumien hallinnassa (Ohjelma 4).

jQuery helpottaa myös verkkosivun DOM:n manipulointia eli muutosten tekemistä sivun rakenteeseen tai sisältöön. Eri elementtejä pystytään kutsumaan CSS-tyyppisillä valitsimilla, jotka ovat yksinkertaisia ja selkolukuisia. Nämä valitsimet toimivat hierarkkisesti eli samoin kuin verkkosivut rakentuvat.

Ohjelma 4. Esimerkki jQueryllä toteutusta koodista, jossa tunnuksella ”navi” olevan elementin sisäisiin linkkeihin liitetään käyttäjän klikkauksia tarkkaileva kuuntelija. Alla on sama toiminnallisuus toteutettuna perinteisellä tavalla.

```
// jQuerylla toteutettuna
$('#navi a').click(function () {
  alert('klikkasit minua');
});


// ilman apukirjastoja
var links = document.getElementById('navi').getElementsByTagName('a');
for(var i=0;i< links.length;i++) {
  links[i].onclick = function() {
    alert('klikkasit minua');
  };
};
```

jQueryyn saatavilla oleva liitännäinen, *jQuery UI*, sisältää myös erilaisia valmiita elementtejä käyttöliittymään ja työkaluja niiden animointiin. [22]

3.4 Smarty

Smarty on PHP–ohjelmointikielelle suunnattu mallinejärjestelmä. Mallinejärjestelmä on käytännöllinen halutessa erottaa esityslogiikkaa toimintalogiikasta. MVC-järjestelmässä mallinejärjestelmä on siksi hyvä vaihtoehto näkymän toteuttamiseen. Www-sovellusten esityskielenä on aina HTML-kieli, jonka selain tulkitsee näytettäväksi sivuksi. Smarty tuottaa HTML-sisältöä ohjelmoijan tekemien valmiiden mallineitten pohjalta käyttäen mm. tarkkeita. Tarkkeet ovat muuttujia ja erilaisia ohjausrakenteita, joita käytetään mallineita rakennettaessa. Mallineet ovat periaatteessa HTML-sivuja, joissa on korvattu dynaamista sisältö Smartyn käyttämillä tarkkeilla (Ohjelma 5). Toimintalogiikan puolella sijoitetaan arvoja tarkkeille, ja kun sivu luodaan selaimelle, käytetään tarkkeiden tilalla sijoitettuja arvoja. [23]

Ohjelma 5. Ote Smartyn mallineen lähdekoodista. Esimerkissä Smarty tuottaa foreach-silmukalla rivejä taulukkoon. Smartyn osiot on merkitty HTML-koodiin { ja }-merkkien väliin.

```
<table>
  <tr>
    <th>Nimi</th><th>Hinta</th><th>Ostopäivä</th>
  </tr>
  {foreach $rivit as $rivi}
  <tr>
    <td>{$rivi.nimi}</td><td>{$rivi.hinta}</td><td>{$rivi.aika}</td>
  </tr>
  {/foreach}
</table>
```

Vaikka PHP–kielen yksi tärkeistä ominaisuuksista on mahdollisuus upottaa PHP–koodia HTML-koodin sekaan ja periaatteessa toteuttaa vastaavat ominaisuudet suoraan, on PHP–koodin erottaminen esityslogiikasta perusteltua. Koodi saadaan selkeämmäksi ja helpommin hallittavaksi. Uusien sivujen luominen mallineiden avulla on helpompaa ja muokkaaminen selkeämpää.

4 TOTEUTUS

Opinnäytetyö toteutettiin avoimen lähdekoodin NetBeans–ohjelmistolla, joka on Sun Microsystemsin perustama projekti. NetBeans on ohjelmistokehitysympäristö, joka on pääasiassa tarkoitettu Java-ohjelmointiin, mutta josta on saatavilla myös PHP:tä tukeva paketti. [24] Ohjelmoinnissa käytettiin apuna Subversion–versionhallintajärjestelmää. Versionhallinta oli tässä tapauksessa käytössä lähinnä varmuuskopiointitarkoituksessa, sillä kehittäjiä projektissa oli vain yksi. Suurimpia versionhallinnan etuja on ohjelmointityön hajauttaminen useille ohjelmoijille. Työkoneelle oli asennettuna tarvittavat ohjelmistot ja palvelimet, jotka vastasivat kohdeympäristöä mahdollisimman tarkasti. Ohjelman kehittäminen, virheiden etsiminen ja testaaminen tapahtui ohjelmoijan työkoneella. Valmiit versiot siirrettiin yrityksen palvelimelle FTP-tiedostonsiirtoprotokollan avulla. Käyttäjät suorittavat viimeisen testauksen asiakkaan palvelimella.

4.1 Projektin suunnittelu

Projektia suunniteltiin ja toteutettiin yhteistyössä yrityksen johdon kanssa. Vuosien kokemus alalta ja selkeät ideat ohjelman vaatimuksista olivat valtava apu ohjelman tekemisessä. Projektin alussa yrityksen toiveiden mukaan luotiin prototyyppisovellus, jossa voitiin testata erilaisia ideoita käyttöliittymän ja toiminnallisuuden suhteen. Tämä sovellus pidettiin hyvin yksinkertaisena tekniseltä toteutukseltaan, koska tarkoituksena oli lähinnä kehittää ideoita ja muodostaa vaatimusmäärittely sekä käyttötapaukset lopullista ohjelmaa varten. Kun prototyyppiä oli testattu ja kehitetty riittävästi, alkoi varsinaisen sovelluksen suunnittelu. Suurin osa prototyypin ominaisuuksista sisällytettiin sovellukselta vaadittaviin ominaisuuksiin ja joitain uusia ominaisuuksia lisättiin vielä tässä vaiheessa.

Samalla ideoitiin ulkoasua ja käyttöliittymää. Prototyypin ulkoasua oli erittäin karu ilman muotoiluja tai grafiikoita. Varsinaiseen sovellukseen suunniteltiin nykyaikaista ja myös esteettisesti miellyttävää ulkonäköä, sillä ohjelmaa tullaan

myymään kaupallisesti. Usein ulkonäkö on asiakkaalle tärkeimpiä ominaisuuksia uutta ohjelmaa valittaessa. Ulkoasun suunnittelussa käytettiin apuna jQuery UI -paketin mukana tulevaa CSS-kehysrakennetta. Tämä käytännössä sisältää erilaisia valmiita helposti muokattavia ulkoasutyylejä, joita on helppo sisällyttää verkkosivuihin tai niiden osiin. Nämä tyylit on suunniteltu siten, että ne antavat mielikuvan perinteisestä työpöytäsovelluksesta, joihin käyttäjät ovat tottuneet. Näin saadaan siirtymä www-sovellukseen mahdollisimman helpoksi.

4.1.1 Vaatimusmäärittely

Kun alkuperäistä prototyyppiä oli kehitetty ja ideoitu, laadittiin vaatimusmäärittely ohjelmalle. Mitään virallista vaatimusmäärittelydokumenttia ei laadittu, vaan suunnitelmat tehtiin pääpirteittäin muistiinpanojen ja suullisten ideoiden muodossa.

Ohjelmassa tulisi olemaan käyttöliittymä, jolla voitaisiin erilaisia kriteereitä valitsemalla tulostaa ruudulle ja paperille raportteja joko varastosta tai myynnistä. Käytännössä kyse on monimutkaisesta hakukoneesta, jolla voidaan hakea tietokannasta tuotteiden varasto- ja myyntitietoja. Tulokset tulostuvat listaksi, jota voidaan järjestää näkyvien ominaisuuksien perusteella. Ohjelma tukee vain yksittäin myytäviä yksilöityjä tuotteita. Sitä voidaan näin ollen käyttää lähinnä vähittäiskaupassa, jossa myydään esimerkiksi ajoneuvoja tai työkoneita jotka on yksilöity rekisteritunnuksilla tai muilla tavoilla. Alkuperäinen ohjelma kehitettiin autokauppaa silmällä pitäen.

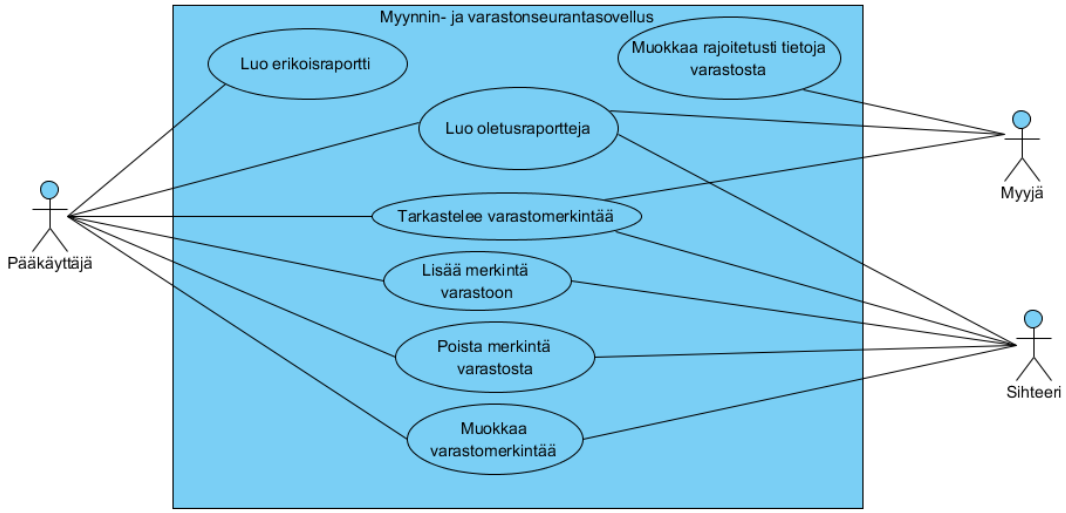
Vaatimuksina annettiin kriteerit, joilla raportteja luodaan, sekä listan tulostaminen ja järjestäminen. Ohjelmassa tulisi olla myös valmiita raportteja, joilla saataisiin tulostettua nopeasti eniten käytettyjä raportteja. Näitä ovat myynti- ja ostoraportti, sekä nykyinen varastotilanne. Myynti- ja ostoraporttiin tulisi antaa aikaväli, jolta raportti halutaan. Käyttöliittymän osalta vaadittiin että ostosta ja myynnistä haettaessa tulisi kaavakkeiden olla eritelty selkeästi. Lisäksi haluttiin mahdollisuus palata raportin listauksesta kriteerien syöttösivulle niin, että syötetyt kriteerit pysyvät näkyvissä ja näin ollen niitä olisi helppo muuttaa.

Jokainen rivi raportissa on yksi tuote ja tuotteen tietoja on pystyttävä muokkaamaan.

Koska ohjelma tulee osaksi suurempaa pääjärjestelmää, annettiin myös ehdot, joilla ohjelman tulee sulautua pääjärjestelmää. Ohjelma tulee käyttämään pääjärjestelmän tuotetietokantaa ja käyttäjien tiedot saadaan pääohjelmasta muuttujana. Ulkoasu suunniteltiin myös pääjärjestelmän mukaiseksi. Istunnot pääjärjestelmä tulee hallitsemaan automaattisesti, joten niihin ei tässä yhteydessä tarvinnut paneutua.

4.1.2 Käyttötapaukset

Kun vaatimukset olivat selvät, alettiin selvittää erilaisia käyttötapauksia. Nämä ovat erilaisia tapoja tai tilanteita joilla käyttäjät ohjelmaa käyttävät. Ohjelmaa voidaan käyttää eri käyttöoikeuksilla ja ylemmillä oikeuksilla on mahdollista mm. muokata tuotteita. Kuva 1 esittää käyttötapaukset UML-kaaviona. Käyttäjäryhmiä on kolme: pääkäyttäjä, myyjä ja sihteeri. Pääkäyttäjä on yrityksessä korkeimmissa johtotehtävissä oleva käyttäjä. Pääkäyttäjällä on täydet oikeudet, eli hänellä on kaikki mahdolliset toiminnot käytössään. Myyjä on henkilö, jonka pääasiallinen tehtävä on myydä tuotteita. Myyjä pääsee näkemään varastoa rajoitetusti, eli hän ei esimerkiksi pysty tekemään erikoisraportteja. Myyjä näkee ainoastaan joitain valmiita raportteja, eikä näissäkään näe kaikkia mahdollisia kenttiä, kuten muiden myyjien myyntejä. Myyjällä on myös oikeus muokata tuotteita rajoitetusti. Tuotteissa saattaa olla tietoja, joissa tarvitaan myyjän erikoisosaamista tai tietoja, jotka ovat vain myyjän tiedossa. Tällaisia ovat esimerkiksi ajoneuvokaupassa pyyntihinta. Sihteerin tehtävä on merkitä lähinnä ostoja ja myyntejä varastoon. Myyjät toimittavat sihteerille tiedot myynneistä tai ostoista ja sihteeri kirjaa ne varastoon keskitetysti. Sihteerillä on varaston muokkaukseen täyden oikeudet, mutta ei raporttien luomiseen.



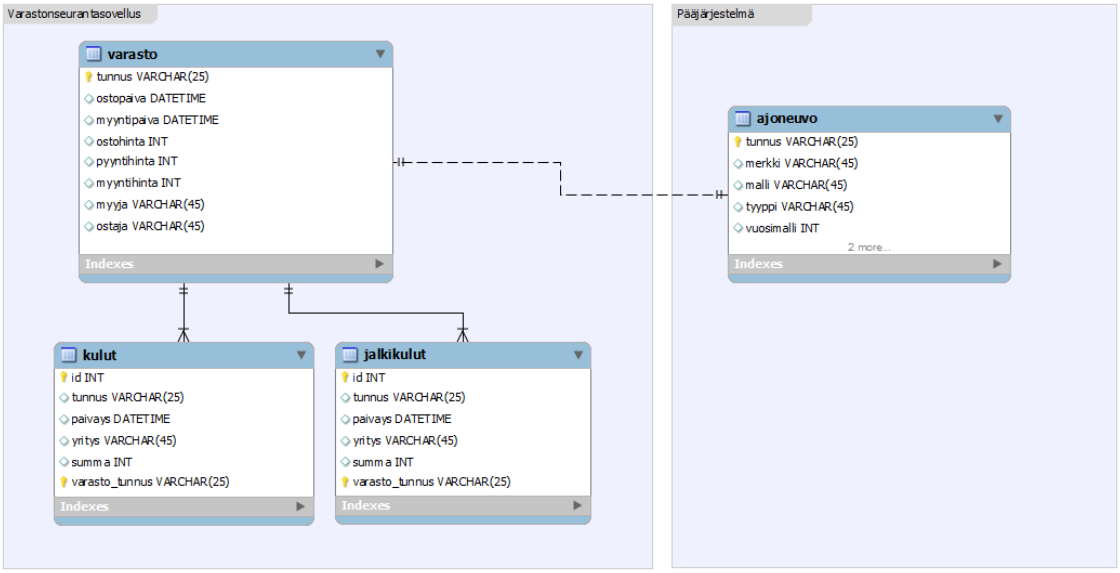
Kuva 1. Työn suunnittelussa käytetyt käyttötapaukset UML-kaaviona.

4.2 Mallien toteuttaminen

Mallit ovat sovelluksen ”back-end” eli se liiketoimintalogiikka, joka toimii sovelluksen taustalla käyttäjälle tietämättömänä. Mallit toteuttavat kaikki ohjelman tärkeät toiminnot, jotka eivät liity käyttöliittymään. [8] Tämä kattaa tuotteiden lisäämisen, muokkaamisen ja poistamisen, sekä erilaisten raporttien tuottamisen. Mallit tuottavat pyydetyn datan yksinkertaisina tietorakenteina, kuten hajautustauluina. Tietorakenteet ohjataan eteenpäin ohjaimen avulla käyttöliittymälle, joka muodostaa niistä käyttäjälle kaavakkeita ja listoja.

4.2.1 Tietokannan rakenne

Kaikki tuotteet ja varastomerkinnät tallennetaan tietokantaan. Tietokanta suunniteltiin käsittämään kaksi pääkäsitettä; tuote ja varastomerkintä. Ajoneuvo-taulu sisältää pelkästään tuotteiden tietoja. Tämä mahdollistaa olemassa olevan tuotetietokannan käytön ja tuotteiden tietojen haun erilaisista rekistereistä rajapintoja hyväksi käyttäen, kuten esimerkiksi ajoneuvon tiedot Liikenteen turvallisuusviraston palvelusta.



Kuva 2. Tietokannan rakenne esitettynä kaavioina. Kulut ja jälkikulut ovat sidottu varasto-tauluun vierasavaimen avulla. Jos varastomerkintä poistetaan, siihen liittyvät kulut ja jälkikulut poistuvat myös.

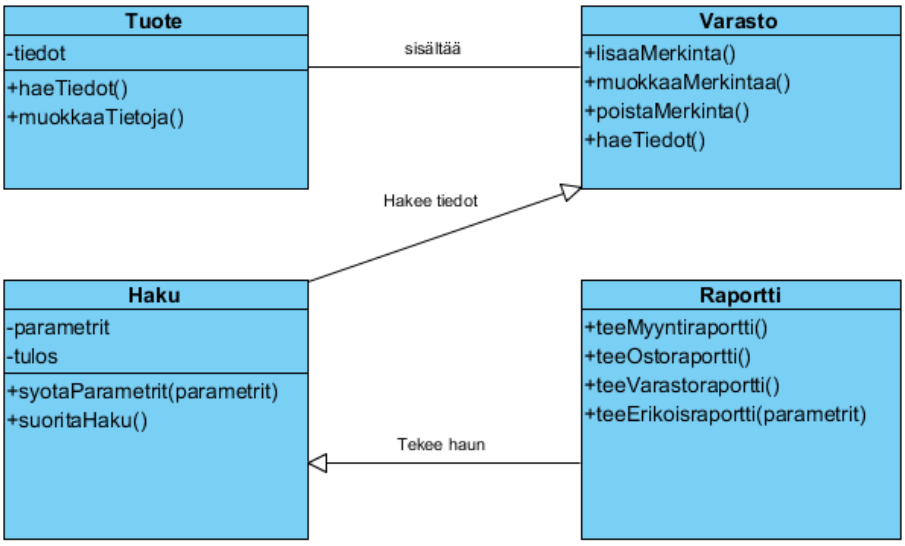
Kuva 2 esittää tietokannan rakenteen kaaviona. Varasto on taulu, jossa tuotteen yhteys varastoon esitetään. Tämä kuvaa siis tuotteen varastomerkintää. Varastomerkintä sisältää varaston ja myynnin kannalta olennaiset tiedot. Näitä ovat mm. ostopäivä, myyntipäivä, ostohinta, myyntihinta ja pyyntihinta. Näillä perustiedoilla voidaan laskea erikoisempia ja myynninseurannan kannalta mielenkiintoisia tietoja, kuten myyntikate, varastopäivät, kateprosentit jne. Varastoon on myös liitetty kulut ja jälkikulut, joita tuotteisiin on tullut tai tulee myynnin jälkeen. Nämä tiedot ovat osa varastomerkintää, mutta ne tarvitsee tallentaa omiin tauluihinsa, sillä niitä voi olla useampia yhtä varastomerkintää kohden.

4.2.2 Olioiden suunnittelu

Mallit ovat käytännössä olioita, jotka luodaan ohjelman ajon aikana tarvittaessa. Esimerkiksi muokatessa varastossa tuotetta, luodaan tuote-olio, jonka muokkaus-metodilla muutetaan tuotteen tilaa. Tila tallennetaan tietokantaan yhtenä tietueena. Tuote-olio ei suoranaisesti vastaa yhtä tuotetta vaan pikemminkin tuotteiden hallinnointiyksikköä. Jos tuotetta muokataan tai tuote lisätään,

annetaan muokattavan tuotteen tiedot oliolle. Oliota luotaessa ei siis määritetä mitä tuotetta olio vastaa. Tämä mahdollistaa sen, että samalla oliolla voidaan muokata eri tuotteita tai hakea eri tuotteista tietoja, ilman että luodaan aina uusia olioita.

Kuten kuvassa 3 esitetään, pääasialliset oliot ovat tuote, varasto ja haku. Tuote-oliolla pystyy muokkaamaan ja hakemaan tuotteen teknisiä tietoja. Tämä on linkitetty varasto-olioon, jolla muokataan tuotetta varastossa. Tuote-oliota ei koskaan käytetä yksinään, vaan sitä käytetään varasto-oliosta käsin. Käyttäjän ymmärtämän tuotteen tiedot koostuvat tuotteen teknisistä tiedoista sekä varastomerkinnän tiedoista yhdistettynä. Käyttäjä muokkaa tai lisää tuotteita kaavakkeella, jossa ovat samassa sekä tuotteen tekniset, että varaston hinta- ja aikatiedot.



Kuva 3. Työn suunnittelussa käytetty kaavio olioista ja niiden yhteyksistä toisiinsa.

Haku-olioilla luodaan hakuja, jotka sitten monimutkaisilla tietokantakyselyillä tuottavat haettavat tiedot tuotteista listana. Haku jakautuu eri olioihin. Kaiken perustana on yleistetty abstrakti hakuolio, joka määrittelee tietokantahaun perusmetodit. Tästä periytetään erikoistuneemmat haut, kuten tässä tapauksessa raportit. Tämän kaltainen järjestely mahdollistaa myöhemmin muidenkin samaan järjestelmään kuuluvien ohjelmien hakujen suorittamisen samalla tekniikalla.

Raportti-olio luodaan ja sille annetaan käytettävät hakuparametrit ja tiedot, jotka halutaan raportin näyttävän. Tämän jälkeen suoritetaan haku ja olio palauttaa halutut tiedot hajautustauluna. Tästä taulusta sivun ohjain rakentaa toisen mallin avulla taulukkorakenteen, joka voidaan viedä selaimelle HTML-koodina. Käyttäjä näkee tuloksen selkeänä taulukkona riveineen ja soluineen.

4.3 Ohjaimien toteuttaminen

Tärkein ja eniten ohjauslogiikkaa vaativa osa sovelluksesta on ohjaimet, jotka koordinoivat malleista saatavan raa'an datan virtaa näkymään käyttäjän nähtäväksi. Koska sovellus käyttää päätoiminnoissaan REST-arkkitehtuuria, joka kerta kun halutaan jotain tietoa välittyvän käyttäjältä palvelimelle ja päinvastoin, pitää sivu ladata uudestaan [4]. Tämän toiminnon suorittaa ohjain. Ohjain sieppaa käyttäjän aiheuttamat tilamuutokset ja tulkitsee niiden avulla mitä olioita malleista luodaan ja mitä metodeja käytetään milläkin parametreilla. Tämän jälkeen ohjain valitsee mikä näkymän malline valitaan ja sijoittaa mallineen käyttämät tarkkeet. Lopulta ohjain käskee näkymän muodostaa valmis sivu käyttäjälle.

Jossain MVC-toteutuksissa myös ohjaimet toteutetaan olioina. Tästä ei kuitenkaan ole varsinaisesti hyötyä tällaisessa sovelluksessa ja se olisi vain monimutkaistanut ohjelman suoritusta. Ohjaimet toteutettiin käytännössä listana komentoja, joiden toimintaa ohjaavat ehtolausekkeet.

REST-arkkitehtuuriin mukaan käyttäjä vaikuttaa ohjelman tilaan lähettämällä selaimen avulla pyyntöjä eräänlaisten verbien avulla. Näitä ovat tässä tapauksessa GET- ja POST-verbit. Verbi sisältää aina listan muuttujista ja niiden arvoista ja nämä listat välittyvät palvelimelle pyyntöjen mukana. GET-verbiä käytetään pääasiassa tiedon hakemiseen, eli jos halutaan esimerkiksi näyttää jokin sivu tai lista tietoa. POST-verbiä käytetään kun halutaan muuttaa tietoa, eli esimerkiksi muokata varastomerkintää tai lisätä uusia merkintöjä. [4] On olemassa myös monia muita verbejä, mutta GET ja POST ovat yleisimmät käytetyt verbit. Muita verbejä ei yleensä käytetä niitä huonosti tukevien ohjelmointikielien ja selainten vuoksi.

Ensimmäinen vaihe ohjelman tilan muutoksessa, eli käytännössä sivun päivityksessä, on ns. ”bootstrap”-tiedosto, jonka nimi on index.php. Tämä tiedosto ladataan aina ensimmäisenä ja se luo kaikki pakolliset oliot, hakee asetukset sekä valitsee mikä ohjain ajetaan. Tämä tapahtuu tulkitsemalla annettuja GET-muuttujia. Yksi GET-muuttuja on aina sivun nimi, joka halutaan ladata. Muut GET- ja POST-muuttujat määrittävät mitä toimintoja ja millä arvoilla ohjain ne tekee. Esimerkiksi, onko kyse tuotteen lisäämisestä tai poistamisesta, vai ollaanko hakemassa tuotteen tietoja muokkaamista varten.

4.4 Näkymän toteuttaminen

Kun ohjain on saanut pyydetyn datan tai vahvistuksen datan muutoksesta malleilta, ohjaa se tiedon tästä käyttäjälle näkymään. Näkymää, eli tässä tapauksessa Smarty-mallinejärjestelmää, ohjataan tarkkeiden avulla. Niihin sijoitetaan tietoa ohjaimen toimesta. Ohjain sijoittaa esimerkiksi listan haetuista tuotteista yhteen näkymän tarkkeeseen, josta malline sitten muuntaa luettavaa listaa HTML-koodin avulla.

Näkymää tässä sovelluksessa hoitaa Smarty-mallinejärjestelmä. Smartyn mallineitten koodi on suurimmaksi osaksi normaalia HTML-koodia. Tarkkeet ja loogiset rakenteet ovat erotettu HTML-koodista aloitus- ja lopetusmerkeillä. Näiden merkkien avulla Smarty-järjestelmä tietää mitkä osat koodista tulee käsitellä PHP:tä hyväksi käyttäen. Merkattujen kohtien tilalle muodostetaan HTML-koodia käsittelyn tuloksien perusteella.

Smarty tukee sisäkkäisiä mallineita, jotka helpottavat mallineitten hallintaa ja kirjoittamista [23]. Koska jokainen näkymä on yksi HTML-sivu, pitää siinä olla kaikki HTML-sivulta vaadittavat komponentit. Näistä monet toistavat itseään eri sivuilla ja siksi mallineet jaettiin osiin, joissa nämä toistot otetaan huomioon. Esimerkiksi sivun yläosa ja alaosa ovat aina samanlaisia, vain keskiosa muuttuu. Mallineet rakennettiin siten, että jokaisen mallineen alussa sijoitetaan HTML-sivun yläosaa kuvaava alamalline ja alaosassa HTML-sivun alaosaa kuvaava alamalline.

Näin ollen mallineissa tarvitsee koodata joka kerta vain keskiosa, yläosan ja alaosan pysyessä aina samoina.

4.5 Käyttöliittymä

Käyttöliittymä luotiin käyttämällä apuna jQuery UI-kirjastoa. Tämä kirjasto sisältää mm. CSS-kehysrakenteen, jolla voidaan luoda nopeasti ja helposti näyttäviä käyttöliittymiä. CSS-kieli on tapa määrittää verkkosivuille tyyliohjeita [25]. Selain noudattaa tyyliohjeita ja asettaa sekä muotoilee elementit niiden perusteella sivulle. jQuery UI sisältää valmiita tyyliohjeita, joilla saadaan luotua käyttöliittymästä ikkunoihin tai widgeteihin perustuva. Näin käyttöliittymää saadaan muistuttamaan enemmän perinteisiä Windows-sovelluksia.

Käyttöliittymässä käytettiin jossain tilanteissa hyödyksi Ajaxia. Esimerkiksi ikkuna, joka avautuu käyttäjän painaessa tuoteriviä, hakee tietonsa Ajaxilla. Toinen käyttökohde ovat hakukaavakkeen alasvetovalikot, jotka ovat toisiinsa linkitettyjä. Kun käyttäjä valitsee ensimmäisestä alasvetovalikosta arvot, toiseen valikkoon täyttyy ensimmäisestä riippuvat arvot. Tiedot haetaan Ajaxilla toiseen valikkoon käyttäjän valitessa ensimmäisestä valikosta jokin arvo.

Ajaxia käyttävä toiminnallisuus vaatii tapahtumien kuuntelua. Erilaisia tapahtumia kuunnellaan lisäämällä kuunneltaviin elementteihin erityisiä kuuntelijoita. Kuuntelija rekisteröidään aina tiettyyn elementtiin kuuntelemaan tiettyä tapahtumaa. Kun tämä tapahtuma laukaistaan, suorittaa kuuntelija tapahtuman käsittelijän. Käsittelijä saa tiedon tapahtumasta ja mitä elementtiä se koskee. Näiden tietojen perusteella käsittelijä voi suorittaa mitä tahansa toimenpiteitä, kuten Ajax-kutsuja. [10] Tapahtumien rekisteröityminen ei vaadi sivun uudelleenlatausta ja näin ollen tapahtumia tarkkaillaan reaaliajassa. Tapahtumien kuuntelulla voidaan suorittaa muitakin toimenpiteitä kuin Ajax-kutsuja. Esimerkiksi taulukoita voidaan järjestää kuuntelemalla rivin otsikkoa. Kun otsikkoa klikataan, lista järjestetään otsikon aiheen mukaisesti.

4.6 Ohjelman toiminta

Ohjelman looginen toiminta kulkee seuraavasti. Käyttäjä napsauttaa sivulla jotain linkkiä tai nappia. Hän on saattanut valita tätä ennen sivulla joitain valintoja, kuten erilaisia kriteerejä raporttia tulostettaessa. Tämä klikkaus lähettää HTTP-pyynnön index.php-tiedostolle, joka asettaa kaikki vaadittavat asetukset, kuten tietokantayhteydet, erilaiset valmiit muuttujat ja vakiot sekä noutaa pääjärjestelmästä tiedot käyttäjistä. Lopuksi tämä tiedosto käynnistää oikean ohjaimen ”page”-nimisen GET-muuttujan mukaisesti. Ennen kun ohjain käynnistetään, tarkastetaan että käyttäjällä on oikeus tehdä toiminto, jota hän on suorittamassa. Oikeudet on merkitty erääseen muuttujaan, joka sisältää ehtoina GET- ja POST-muuttujien eri yhdistelmiä. Näiden ehtojen toteutumista vastaa tietty oikeus, joka kirjautuneella käyttäjällä on oltava. Mikäli vaadittavat oikeudet vastaavat käyttäjän oikeuksia, suoritus jatkuu. Muussa tapauksessa suoritus keskeytyy ja käyttäjälle esitetään virheilmoitus.

Ohjain tarkistaa käynnistyttyään, mitä GET- tai POST-muuttujia lähetettiin ja mitä toimintoja nämä muuttujat aiheuttavat. Ohjain luo tarvittavat oliot malleista ja tekee tarvittavat toiminnot käyttämällä parametreina muita lähetettyjä muuttujia. Näitä ovat esimerkiksi raporttia tehtäessä säädetyt kriteerit. Kun ohjain saa vastauksen olioilta, se asettaa vastauksen tiedot näkymän tarkkeisiin. Viimein ohjain valitsee oikean mallineen ja näyttää sen, eli siirtää ohjat näkymälle.

Näkymä eli Smarty tarkastaa onko sivusta olemassa väliaikaismuistissa valmiiksi käännetty versio. Jos se löytyy, se näytetään, muuten valittu malline tarkkeineen käännetään HTML-koodiksi. Näkymä syöttää annetut tarkkeet rakennettavalle sivulle ja suorittaa kaikki ohjausrakenteet, kuten if- ja foreach-lausekkeet, joilla voidaan mm. toistaa joitain elementtejä kuten taulukon rivejä sivulle. Lopulta selaimelle lähetetään vastauksena käyttäjän alkuperäiseen pyyntöön HTML-sivu, jonka selain piirtää verkkosivuksi käyttäjän nähtäville.

4.6.1 Käyttäjätilit ja oikeudet

Ohjelma tulee olemaan osa järjestelmää, jossa eri asiakasyrityksillä ja niiden käyttäjillä on jokaisella oma käyttäjätilinsä järjestelmään ja jokaisella käyttäjätilillä on oikeutensa käyttäjäryhmästään riippuen. Nämä käyttäjätiedot ohjelma saa järjestelmältä erään muuttujan muodossa. Jokainen tehtävä toiminto tarkastetaan ensin käyttöoikeuksien puolesta. Jos käyttäjällä ei ole riittävästi oikeuksia suorittaa toimintoa, palautetaan virheilmoitus.

Ohjelma tukee kolmea eri käyttäjäryhmää: pääkäyttäjä, sihteeri ja myyjä. Pääkäyttäjällä on täydet oikeudet ja sihteerillä sekä myyjällä toisistaan poikkeavat vajaat oikeudet. Myyjällä näistä on selkeästi vähiten oikeuksia. Jokaisen käyttäjän yksittäiset oikeudet on tallennettu muuttujaan, jota vertaamalla toiminnon vaatimaan oikeuteen voidaan selvittää onko käyttäjällä oikeus kyseiseen toimintoon. Nämä toiminnot määritellään käyttäjän lähettämien GET- ja POST-muuttujien perusteella. Jokaista toimintoa vastaa tietty GET- ja POST-muuttujien yhdistelmä. Näin ollen oikeus tiettyyn toimintoon voidaan tulkita jo hyvin aikaisessa vaiheessa uuden sivun latausta, jo ennen kuin ohjaimia tai malleja on ladattu.

4.6.2 Integroiminen pääjärjestelmään

Koska ohjelma tulee olemaan osa suurempaa järjestelmää, pitää se integroida pääjärjestelmään jotenkin. Tämä oli tässä tapauksessa helppoa, sillä ohjelma on varsin omavarainen ja erillinen muusta järjestelmästä, paria kohtaa lukuun ottamatta. Ainoat osat, joita ohjelma pääjärjestelmästä tarvitsee, ovat käyttäjän tiedot ja tuotteiden tiedot. Käyttäjän tiedot tulevat pääjärjestelmältä muuttujan muodossa ja tuotteiden tiedot saadaan tietokannasta suoraan osaksi ohjelman omia tietokantakyselyjä. Taustalla on myös sisään kirjautumisen mahdollistava istuntojärjestelmä, mutta sen pääjärjestelmä hoitaa automaattisesti. Nämä yhteydet pääjärjestelmään toteutetaan jokaisen tapahtuman aloittavassa index.php-tiedostossa. Tiedoston alkuun on liitetty vaaditut tiedostot pääjärjestelmästä, jotka vuorostaan lataavat tarvitut tiedot ohjelmalle.

4.7 Ohjelman ajaminen

Ohjelmaa ajetaan yrityksen vuokraamalla palvelinkoneella, jossa sijaitsee HTTP- ja MySQL-palvelimet joita ohjelma toimintaansa tarvitsee. Ohjelman käyttöä vuokrataan asiakasyrityksille, jotka voivat käyttää sitä Internet-selaimen välityksellä. Ohjelma on www-sovellukselle luonteenomaisesti aina päällä ja käyttäjät voivat käyttää sitä ympäri vuorokauden.

4.8 Ohjelman testaaminen

Ohjelmaa ei saatu täysin valmiiksi tämän työn aikana. Ohjelmasta saatiin kuitenkin valmiiksi beetaversio, joka riittää ohjelman perusteelliseen testaukseen. Niin sanottuja yksikkötestauksia ei tämän ohjelman testaamisessa käytetty, sillä aikaa käyttäjien toteuttamaan testaukseen on tässä tapauksessa runsaasti.

Testaus suoritetaan kohdeyrityksessä käyttäjien toimesta, ja se tulee kestämään korkeintaan noin puoli vuotta. Puolen vuoden kuluttua ohjelma tullaan julkaisemaan laajalle yleisölle ja sen pitää olla hiottu julkaisukuntoon siihen mennessä.

Pieniä muutoksia saatetaan myös tehdä ohjelmaan käyttäjien havaintojen perusteella. Käyttöliittymästä oli tarkoitus tehdä mahdollisimman käyttäjäystävällinen ja tähän auttaa parhaiten käyttäjien tekemät havainnot pitemmältä aikaväliltä. Testikäyttäjiä haastatellaan säännöllisesti, esimerkiksi kuukauden välein, ja selvitetään mitkä ovat suurimmat ongelmat ohjelman käytössä.

5 YHTEENVETO

Tavoitteena oli, että sovellus saadaan tämän työn aikana siihen kuntoon, että se voidaan ottaa testikäyttöön. Tämä toteutui ja sovellus saatiin testattavaksi kohdeyritykseen, jossa sitä jatkokehitetään vielä tulevina kuukausina.

Työn tuloksena syntyi www-sovellus, joka toimii toivotulla tavalla. Sovellus toteutettiin MVC-arkkitehtuuria hyväksi käyttämällä. Tämä lähestymistapa helpotti koodin suunnittelua ja kirjoittamista huomattavasti. Suurin ja tärkein osa ohjelmasta eli liiketoimintalogiikka pysyttiin toteuttamaan käyttämällä olioita. Tämä lisää uudelleenkäytettävyyttä ja jatkokehitys on näin myös helpompaa. Käyttöliittymän erottaminen liiketoimintalogiikasta mahdollistaa käyttöliittymän tehokkaan muokkaamisen ja kehittämisen, ilman että ohjelma toiminnallisuus muuttuu oleellisesti. Ohjelman toteuttaminen käyttämällä MVC-arkkitehtuuria on myös nopeampaa kuin perinteisillä tavoilla, varsinkin jos apuna käytetään ohjelmistokehystä.

Sovellus tulee osaksi suurempaa ohjelmakokonaisuutta, mutta tämän työn aikana se ei ollut vielä valmiina testattavaksi kokonaisuudessaan. Tämä saattaa aiheuttaa myöhemmin ongelmia tai joitain tehtäviä muutoksia sovelluksen toimintaan, jos pääohjelman suunniteltu toiminnallisuus muuttuu.

LÄHTEET

[1] Booch G., *Object-Oriented Analysis And Design With Applications*, Boston: Addison-Wesley, 1994

[2] Budd, T. A., *An Introduction to Object-Oriented Programming*, Boston: Addison-Wesley, 1991

[3] Reese, G., "Distributed Application Architecture", [online]. Saatavilla: http://java.sun.com/developer/Books/jdbc/ch07.pdf (Luettu: 2.3.2011)

[4] Richardson, L. ja Ruby, S., *RESTful Web Services*, Sebastopol: O'Reilly, 2007

[5] Fielding, R., *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California, 2000

[6] Reenskaug, T., "THING-MODEL-VIEW-EDITOR", *Xerox PARC technical note*, 1979

[7] Dijkstra. E. W., *Selected Writings on Computing: A Personal Perspective*, Berlin: Springer-Verlag, 1982

[8] Burbeck, S., "How to use Model-View-Controller (MVC)", [online]. Saatavilla: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html (Luettu: 2.3.2011)

[9] Koskela, M., *WWW-ohjelmistokehys - esittelyssä Zend Framework*, insinöörityö, Tampereen ammattikorkeakoulu, 2010

[10] Flanagan, D., *JavaScript: The Definitive Guide, Fifth Edition*, Sebastopol: O'Reilly, 2006

[11] Anupam, V. ja Mayer, A., "Security of Web Browser Scripting Languages: Vulnerabilities, Attacks, and Remedies", *7th USENIX Security Symposium*, Bell Laboratories, 1998

[12] Garrett, J., "Ajax: A New Approach to Web Applications", [online], Saatavilla: http://adaptivepath.com/ideas/e000385 (Luettu: 5.3.2011)

[13] Hayes, B., "Cloud Computing", *Communications of the ACM*, Vol. 51 No. 7, Sivut 9-11, heinäkuu 2008

[14] "PHP: Preface - Manual", [online], Saatavilla: http://www.php.net/manual/en/preface.php (Luettu: 2.3.2011)

[15] "PHP: What can PHP do? - Manual", [online], Saatavilla: http://www.php.net/manual/en/intro-whatcando.php (Luettu: 9.3.2011)

[16] "PHP: History of PHP - Manual", [online], Saatavilla: http://www.php.net/manual/en/history.php.php (Luettu: 6.3.2011)

[17] Yarger, R., Reese G. ja King, T., *MySQL & mSQL*, Sebastopol: O'Reilly, 1999

[18] "PHP: MySQL - Manual", [online], Saatavilla: http://fi2.php.net/manual/en/book.mysql.php (Luettu: 2.3.2011)

[19] "Document Object Model (DOM) Level 1 Specification", [online], Saatavilla: http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/ (Luettu: 2.3.2011)

[20] Kesteren, A. ja Pieters, S., "HTML5 differences from HTML4", [online], Saatavilla: http://www.w3.org/TR/html5-diff/ (Luettu: 12.3.2011)

[21] "jQuery Usage Statistics", [online], Saatavilla: http://trends.builtwith.com/javascript/JQuery (Luettu: 2.2.2011)

[22] “What is jQuery UI”, [online], Saatavilla: http://jqueryui.com/docs/Getting_Started/ (Luettu: 12.3.2011)

[23] “All About Smarty”, [online], Saatavilla: http://www.smarty.net/about_smarty (Luettu: 12.3.2011)

[24] “An Introduction to NetBeans”, [online], Saatavilla: http://netbeans.org/about/index.html (Luettu: 12.3.2011)

[25] “Cascading Style Sheets”, [online], Saatavilla: http://www.w3.org/Style/CSS/ (Luettu: 12.3.2011)