OULUN SEUDUN
AMMATTIKORKEAKOULU

Juha-Pekka Teirikangas

PROTOTYPEN JA SCRIPT.ACULO.UKSEN KÄYTTÖ WEB-PROJEKTISSA

Intopii Oy:n verkkokaupan laajennus

PROTOTYPEN JA SCRIPT.ACULO.UKSEN KÄYTTÖ WEB-PROJEKTISSA

Intopii Oy:n verkkokaupan laajennus

Juha-Pekka Teirikangas
Opinnäytetyö
Kevät 2011
Tietojenkäsittelyn koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä: Juha-Pekka Teirikangas
Opinnäytetyön nimi: Prototypen ja script.aculo.uksen käyttö web-projektissa
Työn ohjaaja: Jouni Juntunen
Työn valmistumislukukausi ja -vuosi: Kevät 2011 Sivumäärä: 46

Opinnäytetyön tavoitteena oli laajentaa toimeksiantajan eli Intopii Oy:n verkkokauppaa erillisellä lisäsivulla, jonka kautta ohjelmistokehittäjät voivat lähettää omia Intopiin tuotteisiin liittyviä tuotoksiaan hyväksyttäväksi verkkokauppaan ja saada korvauksen mahdollisista tuotoista. Kehittäjät voivat sivuston kautta muokata tuotteitaan ja omia tietojaan sekä seurata myynti- ja näyttötilastoja. Ylläpidon on vuorostaan kyettävä suorittamaan uusien ja muokattujen tuotteiden hyväksymisprosessi ja saatava tarkat tiedot myynnistä ja korvausten maksamisesta.

Työn toteutuksessa käytettiin Prototype JavaScript-kirjastoa helpottamaan koodin kirjoittamista ja script.aculo.us-kirjastoa sivuston käyttöliittymän tehosteiden tekoon. Sivuston tekoon kuului myös PHP-ohjelmointia ja MySQL-tietokannan laajentaminen, mutta niihin ei ole tämän raportin puitteissa keskitytty. Toteutuksesta kerrotaan JavaScriptin näkökulmasta ja käydään läpi Prototypeä hyödyntäviä valmiita ja vapaasti ladattavissa olevia käyttöliittymäkomponentteja.

Opinnäytetyön aikana havaittiin, että JavaScript-kirjastojen käyttö internet-sivujen teossa vähentää virhealtista kirjoitettavaa koodia, jolloin myös siirrettävän tiedon määrä vähenee jonkin verran. Monilla avoimen lähdekoodin kirjastoilla on jo laaja käyttäjäkunta, joka on testannut ja korjannut mahdollisia virheitä sekä optimoinut toiminnallisuudet. Näin ollen niiden käyttäjän tarvitsee huolehtia vain oman koodin testauksesta ja logiikasta.

Kirjastojen avulla selainyhteensopivuus paranee, koska niiden tekijät ovat jo huomioineet erot selaimissa, eikä niistä tarvitse itse huolehtia. Toisaalta myös uusien selainversioiden mahdolliset ongelmat voi helposti ratkaista päivittämällä kirjasto, jolloin kirjoitettu koodi voidaan säilyttää samanlaisena ja tarvittavat muutokset tulevat uuden version mukana. Päivittäminen tosin saattaa tuoda muita ongelmia esimerkiksi vanhentuneiden funktioiden tai muiden rakennemuutosten seurauksena.

Asiasanat: JavaScript, JavaScript-kirjasto, Prototype, Script.aculo.us, verkko-ohjelmointi

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Systems

Author: Juha-Pekka Teirikangas
Title of thesis: Using Prototype and Script.aculo.us in a Web Project
Supervisor: Jouni Juntunen
Term and year when the thesis was submitted: Spring 2011 Number of pages: 46

The primary objective of the thesis was to expand the Intopii Oy's e-commerce with an external page through which software developers can send their own Intopii related products for quality assurance and possible addition to the e-commerce site as well as receive possible revenues from sales. On the site developers can edit their products, their own information and follow the sale and view count history from diagrams. The administrators, in turn, are able to process new and edited products through a QA-interface and receive detailed reports about sales and revenues.

Prototype JavaScript library was used in the implementation of this thesis to ease the code writing and script.aculo.us library to produce user interface effects. PHP programming and expanding the MySQL database were also a part of the development process, but they are not discussed in this report. The implementation chapter focuses on JavaScript programming and the use of Prototype based complete widgets that can be downloaded for free from the internet.

The results seemed to indicate that using JavaScript libraries in web development reduces the amount of written code, which in turn reduces transferred data and programming errors. Many open source libraries have many users that have tested the product extensively, fixed possible errors and optimized execution. Thus the developer that uses these libraries has to only worry about testing and optimizing his or her own code.

Libraries can create better cross-browser support, because the developers have already taken the differences between browsers into account so the users do not have to. In addition, the problems caused by new browser versions can be easily fixed by updating the library. Then the written code can remain the same and the needed changes to fix the arisen issues come from the new version of the JavaScript library itself. On the other hand, updating can bring new issues such as deprecated functions or other structural changes.

Keywords: JavaScript, JavaScript library, Prototype, Script.aculo.us, web programming

SISÄLLYS

1 JOHDANTO 6
2 JAVASCRIPT 8
2.1 Dokumenttioliomalli eli DOM 9
2.1.1 DOM 1 10
2.1.2 DOM 2 ja 3 10
2.2 JavaScript-kirjastot 11
3 PROTOTYPE 13
3.1 Käyttöönotto 13
3.2 Elementin hakeminen id:n perusteella 14
3.3 Ehdot täyttävien elementtien hakeminen 15
3.4 Taulukot 16
3.5 Arvojoukot 17
3.6 Luokat ja perintä 18
4 SCRIPT.ACULO.US 20
4.1 Sisältö ja käyttöönotto 20
4.2 Tehosteet 21
4.3 Raahaa, pudota ja järjestele 22
4.4 Käyttöliittymäkomponentit 24
5 TOTEUTUS 26
5.1 Kirjautuminen 26
5.2 Tilastot 28
5.3 Tuotteet 31
5.3.1 Uuden tuotteen lisääminen 33
5.3.2 Tuotteen muokkaus 37
5.4 Kehittäjä 38
5.5 Tagit 40
6 POHDINTA 42
LÄHTEET 44

1 JOHDANTO

Tehtäessä perinteisellä JavaScriptillä web-sivuihin erilaisia toimintoja törmätään hyvin pian tilanteeseen, jossa jokin niistä tuottaa hieman erilaisen lopputuloksen toisessa selaimessa ja pahimmassa tapauksessa ei toimi lainkaan kolmannessa. Selaimet ovat ikään kuin erilaisia kulttuureita, joissa tietyn asian merkitys saattaa poiketa hyvin paljon eri maiden kesken. JavaScript-kirjastot ovat eräänlaisia tulkkeja, jotka auttavat selaimia ymmärtämään, mitä kirjoittaja tarkoitti, eikä synny toimintaa hankaloittavia väärinkäsityksiä. Ne myös nopeuttavat toistuvien rakenteiden käyttöä antamalla niihin käyttövalmiita ja optimoituja ratkaisuja.

Opinnäytetyön toimeksiantajalla Intopii Oy:llä on kuvan analysointiin, luokitteluun ja kuvioiden tunnistukseen liittyvien sovellusten tuotelisenssien verkkokauppa, jonka toimintaa haluttiin lähteä kehittämään edelleen. Intopiin verkkokauppa on rakennettu ilmaisen ja avoimen lähdekoodin OpenCart-verkkokauppasovelluksen päälle, minkä voi ladata osoitteesta http://www.opencart.com. OpenCart ei oletuksena sisällä tukea halutuille ominaisuuksille, joten ne päätettiin toteuttaa tämän opinnäytetyön puitteissa.

Tarkoituksena on vaatimusmäärittelyn kautta rakentaa tietokantaa lukuun ottamatta verkkokaupasta täysin erillinen sivusto, jonka kautta kolmannet osapuolet voivat lähettää tuotteitaan, esimerkiksi ohjelmia, lisäosia tai asetustiedostoja, ylläpidon hyväksyttäväksi. Mahdollisen hyväksymisen jälkeen heidän tulee voida seurata tuotteidensa myyntiä Intopiin verkkokaupassa sekä tehdä haluamiansa muutoksia lisäämiinsä kohteisiin. Sivustolla on kaksi hieman erilaista näkymää tavallisille käyttäjille ja ylläpidolle, jotka vaihtelevat riippuen siitä kumman tyyppisellä tunnuksella kirjaudutaan sisään. Tässä raportissa keskitytään lähinnä kuvaamaan toteutusta JavaScriptin näkökulmasta, mutta opinnäytetyön aikana rakennetaan sivusto kokonaisuudessaan käyttäen PHP:tä ja tehdään joitakin muutoksia OpenCartin MySQL-tietokantaan.

Opinnäytetyön tietoperustan lähtökohdaksi otettiin kahden JavaScript-kirjaston käyttö sivuston toteutuksessa. Suuri osa sivuston ohjelmoinnista keskittyi JavaScriptin kirjoittamiseen ja tämän työn helpottamiseen käytettiin yleisesti Intopiissä käytössä olevaa Prototypeä, joka tarjoaa lukuisia ohjelmoijan työtä nopeuttavia oikoteitä ja valmiita funktioita. Internetistä on myös saatavissa paljon Prototypeä hyväksikäyttäviä valmiita skriptejä, jolloin kehitystyössä voi keskittyä itse päätarkoitukseen, eikä aikaa kulu omien versioiden rakentamiseen yleisesti webissä käytössä ole-

vista toiminnoista. Käyttökokemukseen liittyy vahvasti myös ulkoasu ja script.aculo.us tarjoaa paljon tehosteita, joita käyttämällä saadaan sovelluksen toimintaan näyttävyyttä. Käyttäjille on myös tärkeää antaa palautetta toimintopainikkeiden painalluksissa, joissa voi olla huomattavia viiveitä ennen kuin näytöllä tapahtuu mitään palvelimelle tehtävien HTTP-pyyntöjen takia. Vaikka sovellus käyttää näitä Ajaxillä toteutettuja pyyntöjä paljon, ne eivät ole osa tietoperustaa, eikä niiden toimintaa selitetä tarkemmin tässä raportissa.

Verkkokaupan lisäosa käyttää kirjautumiseen muutamassa eri Intopiin palvelussa käytössä olevaa IntoID:tä, joka mahdollistaa samanaikaisen kirjautumisen kaikkiin Intopiin sivustoihin yhdellä tunnuksella kertakirjautumisena (Single sign-on). Kertakirjautuminen tarkoittaa käytännössä sitä, että kirjautuessaan sisään yhdelle sivustolle käyttäjä on kirjautuneena kaikkiin muihinkin IntoID-sivustoihin avatessaan ne. Uloskirjautuminen toimii myös vastaavalla tavalla. Näin erilliset palvelut toimivat käyttäjän kannalta helpommin, kun kirjautumista ei vaadita jokaisella sivulla erikseen.

2 JAVASCRIPT

JavaScript on de facto standardi selainpuolen komentosarjakielistä. Se mahdollistaa staattisten dokumenttien muuntamisen vuorovaikutteisiksi, mikä voi käytännössä tarkoittaa mitä tahansa dynaamisista lomakkeista selainpeleihin. PHP-koodi ajetaan palvelimella, mikä johtaa koko sivun uudelleen lataamiseen ja sitä kautta viiveeseen käyttökokemuksessa, jos käyttäjä esimerkiksi muuttaa tuoteluettelon lajittelua. JavaScript-toteutuksen hintana on kuitenkin se, että koodin suorittaminen jää käyttäjän selaimelle, eikä ole enää palvelimen hallitussa ympäristössä. Selaimia taas on useita erilaisia, mikä voi johtaa erilaisiin tulkintoihin ja ei-haluttuihin lopputuloksiin komentosarjoja ajettaessa.

JavaScript on oliopohjainen ohjelmointikieli ja tästä johtuen kaikilla olioilla voi olla ilmentymämetodeja. JavaScriptiin sisältyy useita eri tietotyyppejä: `Object`, `String`, `Array`, `RegExp`, `Boolean` ja `Date`. Kaikki tietotyypit perustuvat `Object`:n. Se on siis eräänlainen pohja, jonka päälle kaikki muut tyypit rakentavat. Yleiset alkeistyypit, kuten `String`, `Number`, ja `Boolean` voivat olla JavaScriptissä sekä alkeistyyppejä että olioita riippuen siitä, miten ne on luotu. Jos esimerkiksi asetetaan muuttujaan pelkkä numero 7, se on vain alkeistyyppi, mutta jos muuttujaksi asetetaan `new Number(7)`, se on olio. Tämä mahdollistaa sen, että niitä voidaan antaa eteenpäin pelkkänä arvona, eikä viitteenä, mutta ne voivat silti hyödyntää olio-ohjelmoinnin toimintoja, kuten esimerkiksi ilmentymämetodeja. (Dupont 2008, 4-5.)

Vaikka JavaScript onkin oliopohjainen kieli, siinä ei ole perinteisiä luokkia. Abstraktien luokkamääritysten sijaan kopioidaan jo olemassa oleva olio uudeksi olioksi. Toisaalta `Number`:lla ja `Number`:n ilmentymällä ei ole mitään eroa: molemmat ovat olioita. Toisin sanoen sekä `Number instanceof Object` että `new Number instanceof Object` palauttavat molemmat arvon `true`. Koska ei ole luokkia, ei voi myöskään käyttää perintää tavalliseen tapaan. Tämän korvaa olioiden prototype-ominaisuus, joka toimii kaikkien uusien ilmentymien pohjana. Se toimii sekä käyttäjän määrittelemissä olioissa että valmiissa. Omia ilmentymämetodeja on siis mahdollista lisätä mihin tahansa tietotyyppiin. Lisäämällä ilmentymämetodeja, funktioita ei tarvitse lisätä yleiseen nimiavaruuteen ja koodista tulee selkeämpää sekä nimien päällekkäisyyden riski pienenee. (Dupont 2008, 5-6.)

Monista muista ohjelmointikielistä poiketen JavaScriptissä myös funktio on olio. Tämä mahdollistaa sen, että funktio voidaan tallentaa muuttujaan, antaa argumenttina toiselle funktiolle tai funktio voi palauttaa funktion. Näin ollen myös funktioille, kuten kaikille muillekin olioille, voidaan asettaa tai poistaa ominaisuuksia. Luotaessa funktio muuttujaan käytetään hieman tavallisesta poikkeavaa syntaksia. Tavallisesti se tapahtuu:

```
function helloWorld()
{
     alert(”hello”);
}
```

Sama asia voidaan kuitenkin tehdä myös näin:

```
var helloWorld = function()
{
     alert(”hello”);
}
```

Kumpaakin voi kutsua samalla tavalla:

```
helloWorld();
```

(Dupont 2008, 7-8.)

2.1 Dokumenttioliomalli eli DOM

DOM (Document Object Model) on W3C:n määrittelemä ohjelmointikielistä riippumaton ohjelmointirajapinta HTML- ja XML-dokumentteihin. Sen avulla voidaan muokata dokumentissa olevia elementtejä, niiden järjestystä ja tyyliä. (Peltomäki 2000, 7.) DOM käyttää puumallista rakennetta, joka koostuu juurisolmusta ja useista haarautuvista alisolmuista (Peltomäki 2006, 175).

Ennen DOM:n standardointia käytettiin niin sanottua 0-tason DOM:a. Se oli luokkien joukko, jonka avulla skripti saattoi muokata HTML- tai XML-dokumenttia. Internet Explorer 4 oli ensimmäinen selain, jossa näytön näkymää ja tekstejä oli mahdollista muokata. Fontteja pystyi vaihtamaan ja elementtien sijaintia muuttamaan, jolloin IE osasi automaattisesti päivittää sivun. Sen

ajan suurien Netscape Navigator 4:n ja IE 4:n DOM-mallit eivät kuitenkaan olleet keskenään yhteensopivia, mikä teki JavaScript-ohjelmoinnista useaan eri ympäristöön hyvin vaikeaa. (Peltomäki 2000, 7.)

2.1.1 DOM 1

W3C:n tekemä dokumenttioliomallin standardoiminen yhtenäisti aiemmin hyvin hajanaista kenttää ja muodosti HTML-dokumenteista yhden hierarkkisen oliomallin. Tämä mahdollisti interaktiivisten internet-sivujen luomisen esimerkiksi JavaScriptin avulla. DOM määrittää HTML-elementeistä, kuten otsikoista, linkeistä tai kuvista koostuvan dokumentin. Näitä elementtejä ja niiden ominaisuuksia voidaan sitten muokata, poistaa tai lisätä. Tämän lisäksi DOM määrittelee sen, kuinka elementit viestivät keskenään ja miten niihin viitataan. (Peltomäki 2000, 8.)

DOM 1 -mallin mukaan dokumentista luodaan tietokoneen muistiin hierarkkinen dokumenttipuu (document tree), joka sisältää rajoittamattoman määrän solmuja (node), joita ovat esimerkiksi dokumentin elementti, attribuutti tai kommentti. Solmutyypit voidaan tässä mallissa tunnistaa erityisen tunnusnumeron avulla. Ylimmällä tasolla on dokumenttiolio, jonka alapuolella on juurisolmu eli HTML- tai XML-dokumentin juurielementti. Jokaisella solmulla voi olla lapsisolmuja (child node) ja kaikilla muilla paitsi juurisolmuilla on aina äitisolmu (parent node). Numeroihin ja mahdollisesti nimettyihin solmuihin voi DOM 1:n puitteissa viitata useilla tavoilla. (Peltomäki 2006, 176.) Esimerkki solmurakenteesta:

```
<html>
     <head>
     </head>
     <body>
          <h1>Otsikko</h1>
     </body>
</html>
```

2.1.2 DOM 2 ja 3

Vuonna 2000 alkoi uuden DOM 2 -standardin määrittely (World Wide Web Consortium 2000, hakupäivä 11.4.2011). Se on modulaarinen ja vain ydinmoduuli (Core), joka määrittelee doku-

mentin perusrakenteen sekä esimerkiksi Document-, Node-, Element- ja Text-rajapinnat, on pakollinen. Selainpäässä käytännössä kuitenkin tarvitaan useita perusmoduuleja, kuten HTML- ja CSS-moduulit ja erilaisten tapahtumakäsittelijöiden vaatima Event-moduuli. (Peltomäki 2006, 178.)

Tällä hetkellä uusin versio on DOM 3, joka julkaistiin huhtikuussa 2004 (World Wide Web Consortium 2004, hakupäivä 11.4.2011). Se sisältää viisi määritystä: Core, Load and Save, Validation, Events, ja XPath. Core-moduuliin on lisätty jotain uusia funktiota, Load and Save mahdollistaa DOM-dokumenttien serialisoinnin XML-dokumentiksi ja toisinpäin. Validation:n avulla dokumenttia ja sen rakennetta voidaan päivittää niin, että se säilyy validina tai muuttuu validiksi, kun taas Events-moduuliin on lisätty näppäimistön painallus -tapahtumankäsittelijöitä. (Mozilla Developer Network 2011, hakupäivä 28.11.2011.) XPathin avulla voi valita solmuja XML-dokumentissa kuten esimerkiksi DOM:n puurakenteessa (W3Schools 2011, hakupäivä 2.5.2011).

2.2 JavaScript-kirjastot

Aiemmin mainittu DOM mahdollistaa siis internet-sivujen muokkaamisen ohjelmallisesti. Selaimilla on kuitenkin erilainen toteutus DOM:sta, joita kutsutaan Javascript-moottoreiksi ja tämä johtaa usein käytettävyysongelmiin eri selainten välillä. Näiden moottorien tehtävänä on tulkita ja ajaa selaimen JavaScript-koodia. (Dupont 2008, 8.) Merkittävimpiä näistä ovat Mozilla Firefoxin JägerMonkey, Internet Explorerin Chakra, Operan Carakan, Safarin Nitro ja Chromen V8 (Apple 2011, hakupäivä 11.4.2011; Google 2011, hakupäivä 11.4.2011; Microsoft 2011, hakupäivä 11.4.2011; Mozilla 2011, hakupäivä 11.4.2011; Opera 2011, hakupäivä 11.4.2011).

Useimmat ohjelmointikielet on suunnattu tietylle tarkoin määritellylle kääntäjälle tai tulkille, joka toimii aina samalla tavalla. Esimerkiksi selaimilla on omat DOM:n tukitasot, joka johtaa ongelmiin haluttaessa käyttää samaa koodia useissa eri selaimissa. Mikä toimii toisessa selaimessa, saattaa toimia eritavalla toisessa, eikä ollenkaan kolmannessa. (Dupont 2008, 8-9.) Uudemmissa selainversioissa tämä ei ole enää niin merkittävä ongelma kuin ennen, mutta eroavaisuuksia löytyy silti.

Viime vuosina JavaScript-kielen ympärillä on ollut paljon erilaisia kirjasto-projekteja ja jotkin näistä ovat kehittyneet varteenotettaviksi vaihtoehdoiksi myös ammattilaistason sovellutuksiin. Tällä hetkellä suosituimpia JavaScript-kirjastoja ovat Prototype, script.aculo.us, jQuery, YUI

(Yahoo! UI Library), Ext JS, Dojo ja MooTools. Näistä Ext JS, Dojo ja YUI sisältävät laajan valikoiman erilaisia käyttöliittymäkomponentteja. (AjaxLine 2011, hakupäivä 11.4.2011.) JavaScript-kielen joustavuus ja dynaamisuus on johtanut siihen, että erilaiset kirjastot ovat voineet ottaa hyvin erilaisia näkökulmia samankaltaisten ongelmien ratkaisemiseen ja kaikissa niissä on sekä hyvä että huonoja puolia. (Orchard, Pehlivanian, Koon, & Jones 2009, XXV.)

3 PROTOTYPE

Prototype syntyi vuonna 2005, jolloin Sam Stephenson loi sen ensimmäisen version osana Ruby on Rails -projektin JavaScript tukea (Prototype JavaScript Framework 2011d, hakupäivä 27.1.2011). Sittemmin JavaScript on kehittynyt huomattavasti, sillä ensimmäinen Prototype sisälsi vain 335 riviä koodia. Siinä oli jo jotain perusfunktioita, jotka ovat säilyneet tähän päivään asti, mutta silloin se keskittyi vielä Ajaxin ja lomakkeiden väliseen vuorovaikutukseen. Kirjoitushetkellä koodia on jo reilut 6000 riviä, mutta se on edelleen vain yksi tiedosto, usein nimeltään `prototype.js`. (Dupont 2008, 10.)

Prototype ei ole käyttöliittymäkomponentti-, grafiikkakirjasto tai lomakkeiden tarkistus -työkalu, mutta sen päälle voi ja onkin rakennettu näitä. Merkittävin esimerkki on script.aculo.us, joka on Prototype pohjainen tehoste- ja käyttöliittymäkirjasto. Prototypen tarkoitus ei ole olla työkalu, jolla voi tehdä monimutkaisia asioita ilman laajaa JavaScript-tietämystä, vaan se vaatii enemmän perehtymistä ja osaamista. (Dupont 2008, 11.)

Prototype on avoimen lähdekoodin MIT-lisenssoitu projekti, mikä tarkoittaa että sitä voi hyödyntää kuten haluaa, kunhan nimeää tekijät. Kirjaston kehitys perustuu pitkälle yhteisöön ja siihen voi osallistua kuka vain. Koodin lisäksi kaivataan myös virheraportteja, dokumentaation täydennystä tai kehitysideoita. (Prototype JavaScript Framework 2011c, hakupäivä 29.1.2011). Tarkempia ohjeita projektiin osallistumisesta löytyy Prototypen internet-sivuilta, sähköpostilistalta tai IRC-kanavalta. (Dupont 2008, 11–12.)

3.1 Käyttöönotto

Siirtymällä osoitteeseen http://www.prototypejs.org, voi ladata Prototype-kirjaston etusivulta löytyvän latauslinkin kautta. Lataussivulta valitaan viimeisin versio, joka tätä kirjoitettaessa oli 1.7. JavaScript-tiedosto avautuu todennäköisimmin tekstinä selaimessa, josta sen voi tallentaa tietokoneelle haluttuun kansioon. Tämä tapahtuu, joko valitsemalla kaikki teksti ja siirtämällä se haluttuun tekstitiedostoon tai kopioimalla js-tiedosto valmiina palvelimelta. (Dupont 2008, 12–13.)

JavaScript-tiedoston lisääminen samaan sijaintiin web-sivuston tiedostojen kanssa ei vielä riitä, vaan se pitää ottaa käyttöön sivujen HTML-koodissa. Tämä tapahtuu lisäämällä dokumentin

head-osaan script-tagi, jonka src-attribuuttiin laitetaan Prototype-tiedoston polku tai jos HTML-tiedosto on samassa kansiossa sen kanssa, pelkkä tiedoston nimi riittää. Tämän jälkeen Prototype kaikkine funktioineen on käytettävissä kyseisellä sivulla. (Dupont 2008, 12–13.) Kokonaisuudessaan tagi voisi näyttää tältä:

```
<script type=”text/javascript” src=”prototype.js”></script>
```

3.2 Elementin hakeminen id:n perusteella

$-funktio on Prototypen käyttämä alias `getElementById`-metodille, joka hakee dokumentin ensimmäisen parametrina annetun id:n mukaisen elementin. Usein käytetyn metodin pitkän nimen kirjoittaminen joka ikinen kerta vie turhaa aikaa ja energiaa. $-merkki on käytössä siksi, että se on lyhyt, yksinkertainen, sallittu merkki funktioiden nimissä ja sille harvemmin on käyttöä muuten funktion nimenä. Se ei ole pelkkä alias vaan mahdollistaa myös useita asioita, joita ei voi tehdä käyttäen `getElementById`:tä. (Dupont 2008, 18.)

Tekstimuotoisen id:n lisäksi $ hyväksyy parametriksi myös elementin, jolloin se palauttaa samaisen elementin. Tämä mahdollistaa sen, että koodia kirjoitettaessa ei tarvitse ottaa huomioon annetaanko parametrina pelkkä id vai suoraan elementti ja näin koodista saadaan sekä joustavaa että tehokasta. Koska $-funktiota kutsutaan aina Prototypen omissa DOM:ia muokkaavissa funktioissa, voi kaikille näillekin antaa parametrina joko nimen tai elementin. Esimerkiksi alla esitetty Prototypen funktio `Element.remove` voi saada parametrina tekstimuotoisen elementin id:n tai suoraan itse elementin ja lopputulos on sama. Funktio kuitenkin palauttaa koko poistetun elementin, vaikka parametrina olisikin annettu pelkkä id. (Dupont 2008, 19.)

```
Element.remove = function(element) {
     element = $(element);
     element.parentNode.removeChild(element);
     return element;
};
```

On määritetty, että web-sivulla olevien elementtien id:t tulisivat olla uniikkeja, eli yhdellä id:llä ei voi löytää kuin yhden elementin. Tämän takia `getElementById` hyväksyy vain yhden id:n

parametrina. $-funktiolle voi kuitenkin antaa rajattoman määrän parametreja, mutta jos niitä on enemmän kuin yksi, niin se palauttaa elementin sijaan taulukon elementeistä. Sen sijaan, että haettaisiin jokainen elementti yksitellen ja sijoitettaisiin taulukkomuuttujaan, voidaan kaikki tämä hoitaa antamalla useampi parametri $-funktiolle. Prototypessä on paljon tapoja työskennellä kokoelmien kanssa, mikä tekee tästä erityisen hyödyllisen ominaisuuden. (Dupont 2008, 19–20.)

Jos haluaa suoraan elementin sisältämän tekstin (value) eikä itse elementtiä, voi oikaista käyttämällä funktiota `$F`. Se on alias `Form.Element.getValue`-metodille ja palauttaa siis parametrina annetun id:n mukaisen kentän arvon (Prototype JavaScript Framework 2011a, hakupäivä 6.2.2011). Lomakkeen elementteihin viitataan joskus name-attribuutilla, mutta tässä yhteydessä se ei toimi. Kentän arvon hakeminen tekstikentistä on yksinkertainen tehtävä, mutta valintanappien, -ruutujen ja listojen kanssa `$F` tarjoaa helpon ja varman ratkaisun. Listoissa, joissa voi valita useamman kohteen funktio palauttaa taulukon valituista arvoista, muuten palautuu aina yksittäinen arvo ja valitsemattomat valintanapit ja -ruudut palauttavat null-arvon. Arvot palautetaan tekstimuodossa, ja jos value-attribuuttia ei ole asetettu palautuu elementin teksti. (Porteneuve 2007, 38-39.)

Web-sivulla voisi olla valintanappi sukupuolen valintaan tähän tapaan, jossa mies on valittuna:

```
<input name="rdbSukupuoli" id="rdbMies" value="0"
checked="checked" type="radio"> Mies
<input name="rdbSukupuoli" id="rdbNainen" value="1"
type="radio"> Nainen
```

`$F('rdbMies');` palauttaisi `"0"`, kun taas `$F('rdbNainen');` arvon `null`, koska se ei ole valittu.

3.3 Ehdot täyttävien elementtien hakeminen

JavaScriptin perus DOM-funktioilla ei voi hakea tiettyjä ehtoja täyttäviä elementtejä, niin kuin esimerkiksi CSS:n (Cascading Style Sheets) tyylimäärityksissä voi. Prototypessä voi hakea elementtejä luokan nimen, attribuuttien ja sijainnin mukaan käyttämällä $$-funktiota, joka palauttaa taulukon ehdot täyttävistä kohteista tai kohteesta.

```
$$('a'); // Kaikki a eli linkkielementit
$$('a.selected'); // Kaikki linkit, joille on annettu luokannimi "selected"
$$('#linkit a'); // Kaikki linkit jonkin elementin, jonka id on "linkit" sisällä
```

Toki myös perus `document.getElementsByTagName('a')` ajaisi saman asian kuin `$$('a')`, mutta jälleen säästetään paljon merkkejä ja nopeutetaan siten koodin kirjoitusta. Monimutkaisempien hakujen tekeminen vaatii huomattavasti enemmän tekstiä, kuten vaikka esimerkin `$$('#linkit a')`. Jotta tämän voisi toteuttaa ilman Prototypeä, pitäisi ensin esimerkiksi hakea kaikki dokumentin linkit ja tehdä silmukka, joka kävisi jokaisen läpi tarkistaen onko sen yläpuolella elementtiä, jonka id on "linkit". Muutamalla merkillä voidaan siis tehdä, jotain joka vaatisi muuten useita rivejä koodia. (Dupont 2008, 26–27.)

$$-valitsimena voi periaatteessa käyttää mitä vain tekstimuotoista CSS3-syntaksin mukaista valitsinta. Elementtejä voi muun muassa hakea attribuuttien perusteella, esimerkiksi `$$('a[href=www.oamk.fi]')` hakisi kaikki linkit, jotka johtavat osoitteeseen "www.oamk.fi". Suoraan elementin, jonka id on "kappaleet" alapuolella olevat p eli kappale elementit: `$$('#kappaleet > p')`. Jos "kappaleet" elementin sisällä olisi div, jonka sisällä olisi lisää p-elementtejä, eivät ne enää tässä tapauksessa kuuluisi palautettaviin elementteihin. $$-funktiossa voidaan myös sulkea elementtejä pois käyttämällä edellä kuvattuja valitsimia. Lauseke `$$('a:not(.valittu)')` hakisi kaikki dokumentin linkit paitsi ne, joilla on luokka "valittu", kun taas `$$('a:not([href=www.google.fi])')` hakisi kaikki linkit paitsi ne, jotka viittaavat osoitteeseen "www.google.fi". (Dupont 2008, 27–28.)

3.4 Taulukot

$A-funktion avulla voi muuttaa minkä tahansa taulukkoyhteensopivan kohteen `Array`-olioksi. DOM-funktiot saattavat palauttaa NodeList- tai HTMLCollection-olioita, joilla on `length`-ominaisuus ja kohteet on listamaisesti numeroitu kokonaisluvuilla nollasta ylöspäin. Toisin sanoen ne ovat iteroitavissa, mutta eivät ole `Array`-luokan ilmentymiä. Prototype laajentaa Array:tä merkittävästi lisäämällä uusia metodeja, jotka helpottavat taulukkojen käsittelyä. (Porteneuve 2007, 36.)

$w helpottaa tekstimuotoisten vakiotaulukoiden tekemistä poistamalla tarpeen käyttää pilkkuja ja lainausmerkkejä jokaisen eri kohteen välissä. Se on mahdollista, koska taulukon kohteiden erottajana käytetään välilyöntiä, mikä tosin johtaa siihen, etteivät yksittäiset kohteet voi sisältää välejä, koska ne tulkittaisiin erillisiksi. Koska parametrina annetaan välilyönnein eroteltua tekstiä, ovat numerotkin tekstimuodossa taulukossa. Esimerkki käytöstä:

```
// Vanhalla tavalla
var henkilot = ['Kalle', 'Satu', 'Antti', 'Jaakko'];
// Uudella tavalla
var henkilot = $w('Kalle Satu Antti Jaakko'); (Porteneuve 2007, 33.)
```

Prototype tuo mukanaan uudenlaisen olion Hash:n (Porteneuve 2007, 39). $H luo olion Hash-luokasta, joka on Prototypen versioon 1.6 kirjoitettu kokonaan uudelleen. Se on käytännössä assosiatiivinen taulukko, jossa on uniikki avain ja siihen liitetty arvo. JavaScriptissä on itsessäänkin mahdollista luoda olioita, joiden ominaisuuksia voi lisätä, poistaa, muokata ja käydä läpi for…in-silmukalla. Hash tarjoaa näiden perusasioiden lisäksi käteviä metodeja esimerkiksi avainten tai arvojen hakemiseen sekä kahden kokoelman yhdistämiseen. Hash:n tekeminen onnistuu, joko käyttämällä syntaksia `new Hash()` tai `$H()`. Esimerkki käytöstä:

```
var henkilo = $H({ nimi: "Matti", ika: 47 });
```
(Porteneuve 2007, 215–216.)

3.5 Arvojoukot

$R on puhtaasti vain alias ObjectRange-alustajalle. Se ottaa parametrinaan joukon ensimmäisen ja viimeisen kohteen, tavallisimmin kokonaisluvun:

```
$A($R(2, 9))
// [2, 3, 4, 5, 6, 7, 8, 9]
```
(Porteneuve 2007, 39.)

Joukon voi tehdä myös käyttäen kirjaimia, esimerkiksi antamalla parametrit ”a” ja ”e”, jolloin saadaan aakkoset kirjaimesta a kirjaimeen e. Tekstimuotoisten joukkojen kanssa pitää kuitenkin olla tarkkana:

```
$A($R('ay', 'bb'))
// ["ay", "az", "a{", "a|", "a}", "a~"…]
```

Voisi olettaa, että lopputulos olisi muotoa `["ay" , "az" , "ba" , "bb"]`, mutta sen sijaan saadaan monta kymmentätuhatta kohdetta sisältävä taulukko sisältäen mitä mielivaltaisempia merkkejä. (Porteneuve 2007, 220–221.)

3.6 Luokat ja perintä

Kuten mainittu, JavaScript on prototyyppeihin perustuva ohjelmointikieli, jossa ei ole varsinaisia luokkia. Perinteisten luokkien toimintaa voi kuitenkin jäljitellä, mitä Prototype helpottaa jonkin verran. Prototypessä luokka tehdään käyttämällä `Class.create()` -funktiota. (Porteneuve 2007, 385–386.) Yliluokan määrittelyn sisältävä syntaksi on seuraavanlainen:

```
Class.create([superclass][, methods...]) → Class
```

(Prototype JavaScript Framework 2011b, hakupäivä 19.4.2011.)

Kaikki parametrit voi jättää pois, jolloin syntyy tyhjä luokka, johon metodit pitää lisätä jälkeenpäin. Vielä Prototypen versiossa 1.5 tämä oli ainoa tapa, mutta nyt metodit voi määritellä suoraan. Myös luokan alustajan voi uusimmissa Prototypen versioissa jättää pois ja se korvataan automaattisesti tyhjällä funktiolla: `Prototype.emptyFunction()`. Työntekijä-luokka voisi olla seuraavan esimerkin mukainen, jossa alustaja eli `initialize`-funktio saa parametrina nimen ja jäsenfunktiot ovat erota ja erotusilmoitus. Luokasta luodaan olio `var juhaPekka = new Tyontekija("Juha-Pekka");` ja metodikutsulla `juhaPekka.erota("yt-neuvottelut");` palautuisi ”Juha-Pekka on erotettu. Syy: yt-neuvottelut”. (Porteneuve 2007, 386–387.)

```
var Tyontekija = Class.create(
{
```

```
     initialize: function(nimi)
     {
          this.nimi = nimi;
     },
     erota: function(syy)
     {
          return this.erotusilmoitus(syy);
     },
     erotusilmoitus: function(syy)
     {
          return this.nimi + ” on erotettu. Syy: ” + syy;
     }
});
```

Luokkien perintä on perinteisessä JavaScriptissä monimutkaista, mutta Prototype piilottaa tämän kaiken, jolloin lopputulos on hyvin yksinkertainen: annetaan vain yliluokka parametrina ja kutsutaan sen funktioita käyttämällä merkintää `$super`. Kaikki yliluokan funktiot ovat käytettävissä aliluokassa mukaan lukien alustaja, vain korvattavat kirjoitetaan uudelleen. Luokkiin voi lisätä metodeja käyttämällä funktiota `addMethods()`, johon funktiot määritellään samalla tavalla kuin luomisenkin yhteydessä. (Porteneuve 2007, 388–389.) Työntekijän aliluokka Johtaja voisi olla seuraavanlainen:

```
var Johtaja = Class.create(Tyontekija,
{
     erotusilmoitus: function($super, syy)
     {
          return this.nimi + ” lähtee yrityksestä ja saa
          suuret bonukset, koska: ” + syy;
     }
});
```

4 SCRIPT.ACULO.US

Thomas Fuchs kehitti script.aculo.us-kirjaston erillisenä projektina, mutta tiiviissä yhteistyössä Prototypen kanssa. Molempien julkaisuaikataulut kulkevat samaa tahtia ja Fuchs onkin osa Prototypen kehitystiimiä. Script.aculo.us on Prototypen päälle rakennettu käyttöliittymäkirjasto, johon kuuluu erilaisia käyttöliittymäkomponentteja ja raahaus-pudotus -järjestelmä. Prototype ratkaisee yleisiä ongelmia, kun taas script.aculo.us keskittyy tiettyjen asioiden tekemiseen. Script.aculo.us on myös MIT-lisenssoitu, joten sitä voi käyttää sekä kaupallisissa että ei-kaupallisissa projekteissa. (Dupont 2008, 208.)

4.1 Sisältö ja käyttöönotto

Toisin kuin Prototype, script.aculo.us koostuu useista eri tiedostoista, joilla jokaisella on oma tarkoituksensa. Moduulitiedostot eivät ole kaikki välttämättömiä ja käyttäjä voi itse valita, mitä ottaa mukaan omaan työhönsä. Latauspaketti, joka on ilmaiseksi ladattavissa osoitteesta http://script.aculo.us/downloads, sisältää `lib`-, `src`- ja `test`-kansiot sekä `CHANGELOG`-, `MIT-LICENSE`- ja `README`-tiedostot. Src-kansio sisältää varsinaiset JavaScript-tiedostot, jotka tulee ottaa mukaan web-projektiin. (Dupont 2008, 209–210.)

Script.aculo.us 1.7 kuuluvista JavaScript-tiedostoista `scriptaculous.js` sisältää versionumeron ja tarkistaa, että Prototype on otettu käyttöön ja on yhteensopivaa versiota. `Effects.js` hoitaa käyttöliittymän animaatiot, `dragdrop.js` raahaus-pudotus-ominaisuudet, `controls.js` taas erilaisia käyttöliittymäkomponentteja, `slider.js` sisältää vierityspalkkimaisen liukusäätimen, `sounds.js` nimensä mukaisesti mahdollistaa äänien toistamisen dokumenteissa, `builder.js` on DOM-elementtien luomista varten ja `unittest.js` on tarpeellinen yksikkötestauksessa. (Dupont 2008, 210.)

Pelkän `scriptaculous.js` -tiedoston lisääminen dokumenttiin tuo automaattisesti muut tiedostot mukanaan, mutta lisäämällä haluttujen tiedostojen nimet hakulausekkeeseen (query string) voi valita vain osan niistä ladattavaksi. Tällöin script-tagi näyttäisi seuraavalta:

```
<script type=”text/javascript”
```

```
src="scriptaculous.js?load=effects,controls"></script>
```

Vain yhden tiedoston liittäminen on kehittäjän kannalta helpompi ratkaisu, mutta varsinaiseen julkaisuversioon kannattaa jokaiseen tiedostoon viitata suoraan. Tällöin ei siis tarvita `scriptaculous.js`-tiedostoa lainkaan. (Dupont 2008, 212.)

4.2 Tehosteet

Tehosteet (effects) ovat script.aculo.uksen yhteydessä määritelty jonkin elementin muokkaukseksi tietyn ajan kuluessa. Näin ollen mikä tahansa tehoste tarvitsee ainakin kolme asiaa: kohteen eli elementin, alku- ja lopputilanteen ja asteittaisen siirtymisen näiden kahden välillä eli funktion. Voidaan esimerkiksi valita tietty elementti sivulla ja piilottaa se. Näin ollen on kohde sekä alkutilanteena elementti täysin näkyvissä ja lopputilanteena kokonaan piilossa. `fade`-funktio taas piilottaa elementin asteittain muokkaamalla sen opacity-tyylimäärittelyä. Lopputuloksena on tehoste, jossa elementti häipyy näkyvistä. (Dupont 2008, 215.)

Tehosteet eivät ole sama asia kuin animaatiot, joita näkee usein käytettävän web-sivuilla. Jokainen on varmasti törmännyt johonkin sivustoon, jossa on alussa monien kymmenien sekuntien mittainen flash-intro, joka vain parhaassakin tapauksessa ärsyttää ensimmäisen katselukerran jälkeen. Tehosteiden tarkoitus on antaa käyttäjälle palautetta toiminnasta ja sen onnistumisesta. Harva työpöytäympäristön käyttäjä kiinnittää huomiota siihen, mitä tapahtuu pienennettäessä ikkuna tehtäväpalkkiin: ikkuna ikään kuin loikkaa alas hyvin nopeasti. Ensikertalaiselle tällaiset vihjeet ovat erittäin tärkeitä, muuten käyttäjä voisi luulla ikkunan sulkeutuneen kokonaan, eikä olisi ymmärtänyt pienennysnapin toimintoa oikealla tavalla. (Dupont 2008, 216.)

Funktioille voidaan antaa parametrina objekteja, joilla voi määritellä esimerkiksi efektin alku- ja lopputilanne käyttämällä `To`- ja `From`-avaimia tai antaa takaisinkutsu (callback) käyttäen `afterFinish`-avainta. Kaikille tehosteille voi antaa omia takaisinkutsu-funktioita, jotka ajetaan automaattisesti suorituksen jälkeen. Käytännössä tämä tarkoittaa toimintojen suorittamista sen jälkeen, kun käyttäjälle on näytetty jokin tehoste. Esimerkiksi aiemmin mainitun `fade`-funktion takaisinkutsuna voidaan poistaa elementti, sitten kun se on kokonaan piilotettu:

```
$('elementinId').fade(
{
```

```
     afterFinish: function ()
     {
          $('elementinId').remove();
     }
});
```

4.3 Raahaa, pudota ja järjestele

Raahaaminen ja pudottaminen ovat olleet osa tietokoneen käyttöliittymää siitä lähtien, kun hiiri otettiin käyttöön, mutta se on vasta viimeaikoina saanut sijaa web-sovelluksissa. Kyse ei ole teknologisista rajoitteista, sillä DOM-skriptauksessa voidaan ottaa kiinni hiiren painallus pohjaan (mousedown), hiiren siirtäminen (mousemove) sekä hiiren napin vapauttaminen (mouseup). Tapahtumien lisäksi, elementin CSS-määritysten mukaisen sijainnin muuttaminen hiiren tahdissa on toteutettavissa. (Dupont 2008, 257.)

Script.aculo.uksessa elementeistä voi tehdä hiirellä liikuteltavia tekemällä uuden ilmentymän luokasta Draggable, joka vaatii parametriksi kohteen id:n tai suoraan itse elementin: `new Draggable('elementinId');`. Tämän jälkeen elementtiä voi liikutella vapaasti, mutta se ei tee mitään muuta. Raahauksen asetuksia, kuten liikkumisen sujuvuutta (snap), palautumista takaisin lähtöpisteeseen (revert), sitä mikä kohta elementistä liikuttaa sitä (handle) ja takaisinkutsu-funktiot voidaan määrittää antamalla ne toisena parametrina ilmentymää luotaessa. (Dupont 2008, 257–261.)

Raahaamisen toinen osuus eli pudottaminen voidaan toteuttaa käyttämällä Droppablesia. Se havaitsee, milloin Draggable-luokan elementti raahataan siihen ja tarkistaa, onko se yhteensopiva. Pudotuskohteen määrittäminen eroaa hieman aiemmasta siten, että se ei ole luokka, eikä siitä siten luoda ilmentymää. Tämän sijaan käytetään `Droppables.add`-metodia: `Droppables.add('elementinId');`. Jotta pudottamisessa tapahtuisi jotain, täytyy vielä määritellä takaisinkutsut, jotka ovat `onHover` ja `onDrop`. Ensimmäinen ajetaan raahatun elementin ollessa pudotusalueen päällä ja jälkimmäinen, kun se pudotetaan siihen. (Dupont 2008, 264–265.) Takaisinkutsut määritellään samalla tavalla, kuin Draggable-luokan käytön yhteydessä:

```
Droppables.add('elementinId',
```

```
{
     onDrop: function()
     {
          // Pudotettaessa suoritettava koodi
     }
});
```

Pudotettavan elementin siirtämiseksi pudotusalueelle voidaan tehdä yksinkertainen funktio, joka saa parametrina raahattavan elementin, kohteen ja tapahtuman:

```
function lisaaPudotettaessa(draggable, droppable, event)
{
     droppable.appendChild(draggable);
}
```

DOM-metodi `appendChild` lisää siirretyn elementin alkuperäisestä paikastaan uuteen kohteeseen, eikä erillistä poistoa ja lisäystä tarvita. Viittaamalla aiemmin määritellystä `onDrop`-takaisinkutsusta uuteen funktioon, on kokonaisuus valmis. Koska `lisaaPudotettaessa` on tallennettu funktiona muuttujaan, voidaan `Droppables.add`-vaiheessa jättää funktiomäärittelyt pois ja antaa suoraan muuttuja: `onDrop: lisaaPudotettaessa`. (Dupont 2008, 266–267.)

Jos halutaan pelkkiä hiirellä järjestettäviä kokonaisuuksia, kuten järjestää listan (ul) elementit (li) tiettyyn järjestykseen, on yksinkertaisempaa käyttää Sortablesia. Sortable vastaa käytöltään hyvin paljon edellistä droppablea, sillä on vain yksi `Sortable`-olio, joka huolehtii kaikista sivun lajitteluista. Kutsumalla `Sortable.create("elementinId")` aiemmin mainitun listan tapauksessa niin, että parametrina on ul-elementin id, script.aculo.us tekee automaattisesti kaikki li-elementit lajiteltaviksi. Antamalla toisen parametrin, voi toimintaa jälleen hienosäätää haluttuun suuntaan tai määritellä `onChange`- ja `onUpdate`-takaisinkutsut. (Dupont 2008, 270–272.)

4.4 Käyttöliittymäkomponentit

Script.aculo.uksessa on mukana myös joitain valmiita käyttöliittymäkomponentteja, jotka on suunniteltu tiettyjä käyttötapauksia varten. Sen lisäksi, että tietää miten niitä käytetään, on hyvä tietää, milloin niitä käytetään. Tässä yhteydessä esitellään lyhyesti esimerkin vuoksi `Autocompleter`, joka esittää täydennettyjä vaihtoehtoja käyttäjän kirjoittamiin syötteisiin, `InPlaceEditor`, minkä avulla voi web-sivulla olevaa tekstiä muokata ja tallentaa sekä `Slider` eli liukusäädinelementti. (Dupont 2008, 277.)

`Autocompleter` on periaatteessa sama toiminto, kuin yleisimpien internet-selainten osoiterivi, joka näyttää karsiutuvan listan aiemmin vierailluista osoitteista. Karsiutuminen tarkoittaen sitä, että lista esittää täydentäviä mahdollisuuksia siihen, mitä käyttäjä on jo kirjoittanut. `Autocompleter` ei kuitenkaan listaa aiemmin syötettyjä tietoja, vaan käyttää kehittäjän määrittelemiä arvoja. Se soveltuu parhaiten käytettäväksi silloin, jos vaihtoehtona sille on tekstikenttä eikä esimerkiksi pudotusvalikko. Pudotusvalikkoon käyttäjä ei voi kirjoittaa omaa vaihtoehtoa, joten se ei toiminnaltaan vastaa `Autocompleteria`. Toisaalta, jos käyttäjä on poistanut JavaScriptin käytöstä selaimessaan, `Autocompeter` muuttuu tavalliseksi tekstikentäksi muutenkin. (Dupont 2008, 277–278.)

Täydennysvaihtoehdot voidaan hakea Ajaxin avulla palvelimelta tai sitten suodattaa taulukosta selaimessa. Ensimmäinen eli `Autocompleter.Local` on yleensä nopeampi vaihtoehto, koska palvelinkutsu vie oman aikansa. Jos vaihtoehtojen määrä on valtava tai näytettävien vaihtoehtojen valintaprosessi on monimutkainen, voi `Ajax.Autocompleter` olla parempi valinta. Syntaksi ei kummassakaan tapauksessa paljoa eroa toisistaan: paikallisessa annetaan taulukko arvoista, kun taas Ajax-vaihtoehdossa osoite, josta taulukko saadaan. `Autocompleter` luodaan seuraavasti: `new Autocompleter.Local( tekstikentanId, listanId, taulukko, asetukset);`. (Dupont 2008, 279.)

`Ajax.InPlaceEditor` voi muokata haluttuja tekstejä sivulla, valitsemalla ne hiirellä ja korjaamalla teksti ilmestyneeseen tekstikenttään. Tekstikentän lisäksi sivulle tulee myös linkit tallennukseen ja peruuttamiseen. Erityisen hyödyllinen tällainen ominaisuus voi olla ylläpitokäyttöliittymässä, jossa sivujen sisältöä on yleensä tarpeen muokata jossain vaiheessa. Kuten ni-

mestäkin voi päätellä, tallennuksen yhteydessä tehdään Ajax-pyyntö palvelimelle, johon muokattu teksti tallennetaan. Kaikki tämä onnistuu näin: `new Ajax.InPlaceEditor( elementinId, tallennusUrl, asetukset);`. (Dupont 2008, 287–288.)

Liukusäädin on kaikille tietokoneen käyttäjille tuttu sivujen vierityksestä, mutta sille on helppo keksiä käyttötarkoituksia myös itse web-sivulla, kuten zoomaaminen, jonkin kohteen koon muuttaminen jne. Jostain syystä sitä ei kuitenkaan ole oletuksena toteutettu selaimissa, mutta Script.aculo.us pyrkii korjaamaan tilanteen omalla versiollaan. `Slider` vaatii elementit kiskolle ja vetokahvalle, mutta perinteisiä ylös- ja alaspäin osoittavia nuolia ei oletuksena ole mukana. Syntaksi on siis: `new Control.Slider(kahvaelementtienIdt, kiskoelementinId, asetukset);`. Takaisinkutsuja ovat `onSlide`, jota kutsutaan kokoajan liu'utettaessa ja `onChange`, joka kutsutaan hiiren irrotessa eli liu'un päättyessä. (Dupont 2008, 293–296.)

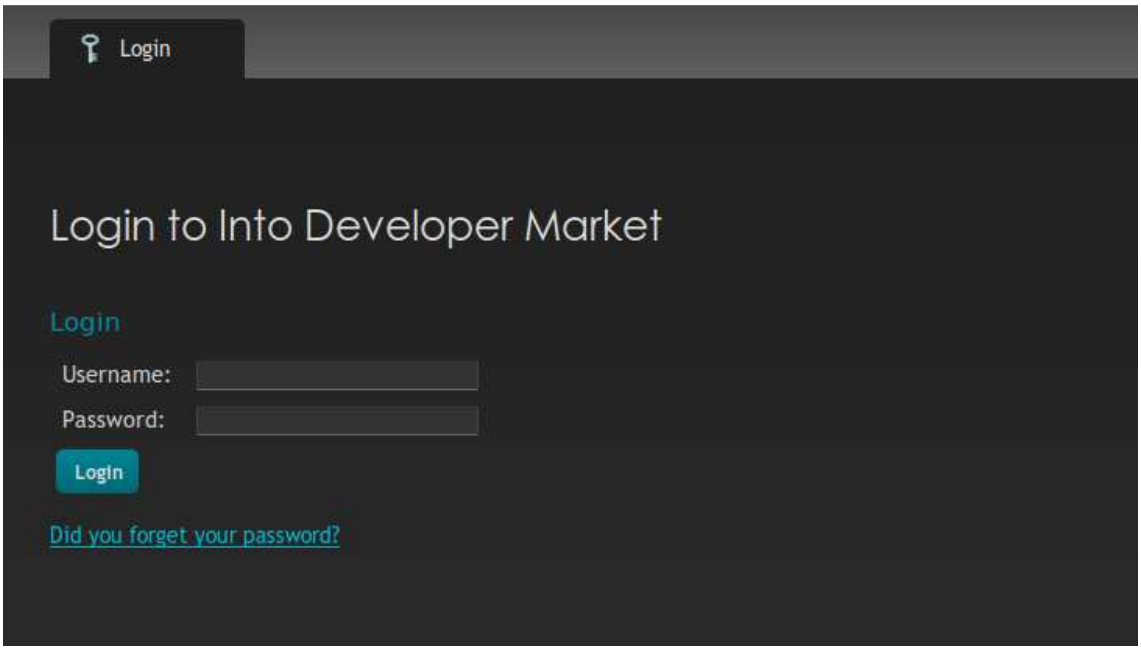
5 TOTEUTUS

Tässä luvussa esitellään lopputuloksena syntyneen www-sivuston välilehtiä sekä niiden tekemiseen käytettyjä JavaScript-koodeja ja valmiita skriptejä. Ohjelmointiesimerkit liittyvät Prototypen ja script.aculou.uksen käyttöön, mutta työn aikana tehtiin paljon muutakin, mitä ei tässä esitellä. JavaScriptin osalta ne olivat pääasiassa pieniä yksityiskohtia, kuten lomakkeiden syötteiden tarkastaminen tai sivuston ulkoasun asetteleminen, joka tehdään vasta selainpäässä, jotta sivua ei tarvitse ladata uudelleen vaihdettaessa välilehteä. Palvelimen ohjelmoinnista taas on jätetty käytännössä kaikki käsittelemättä, kuten Ajax-pyyntöjä varten tehty oma PHP-luokka ja tietokannan rajapinta.

Varsinaisen lähdekoodin käyttäminen raportissa ei ollut mahdollista, koska kysymyksessä on Intopii Oy:n liiketoimintaan liittyvä sivu. Esimerkit vastaavat todellista toteutusta syntaksiltaan, mutta sisältö on täysin mielikuvituksellinen ja usein yksinkertaistettu luettavuuden helpottamiseksi. Kuvankaappaukset ovat kehitysympäristössä käytössä olleesta versiosta ja sen sisältö esimerkiksi tuotteiden osalta ei vastaa todellisuutta.

5.1 Kirjautuminen

Kirjautumissivulla (KUVIO 1) on hyvin vähän mitään sisältöä ja se näytetään ainoana välilehtenä vain silloin, kun käyttäjä ei ole kirjautunut sisään. Kirjautuminen tallennetaan sessio-muuttujaan kirjautumispalvelimelle, johon sivusto tekee kyselyn tarkistaakseen onko käyttäjä jo kirjautunut tällä tai jollain muulla samoja tunnuksia käyttävällä sivulla. Jos sessiota ei löydy tai käyttäjän IP-osoite on vaihtunut, vaaditaan kirjautuminen.



KUVIO 1. Kirjautumissivu

Sivulle tultaessa käyttäjätunnuskenttä on oletuksena valittu, koska sivun latauduttua suoritetaan: `$('txtKayttajatunnus').focus()`. Käyttäjä voi tämän jälkeen alkaa välittömästi kirjoittamaan tunnustaan ilman tarvetta valita kenttää hiirellä. Käyttäjätunnuksen lisäksi on kenttä salasanalle ja kirjautumisnappi, jota painamalla suoritetaan syötteiden tarkistus. Jos molempiin kenttiin on annettu vähimmäisvaatimuksen ylittävä määrä merkkejä, tehdään Ajaxilla HTTP-pyyntö kirjautumispalvelimelle, joka tarkistaa käyttäjän tiedot. Salasana muutetaan yksisuuntaisella salauksella tiivisteeksi, jota verrataan tietokantaan tallennettuun vastaavaan arvoon.

Sivustolla on yleisesti käytössä Ajax-pyyntöjen yhteydessä työn aikana tehty LoadingSpinner JavaScript-luokka, joka lisää halutun elementin jälkeen pyörivän latauskiekon (KUVIO 2), joka antaa käyttäjälle palautteen napin painalluksesta. Näin ei synny tilannetta, jossa pyynnön aiheuttama viive antaisi vaikutelman, että painalluksella ei ollut mitään vaikutusta. Luokalle voi antaa jo mainitun kohde-elementin lisäksi halutun lataus-, onnistumis- ja virheviestin, jotka näytetään käyttäjälle. On myös mahdollista valita näytetäänkö palauteviestejä laisinkaan, jos esimerkiksi ladataan sisältöä ja latausanimaation korvaa ladattu kohde.

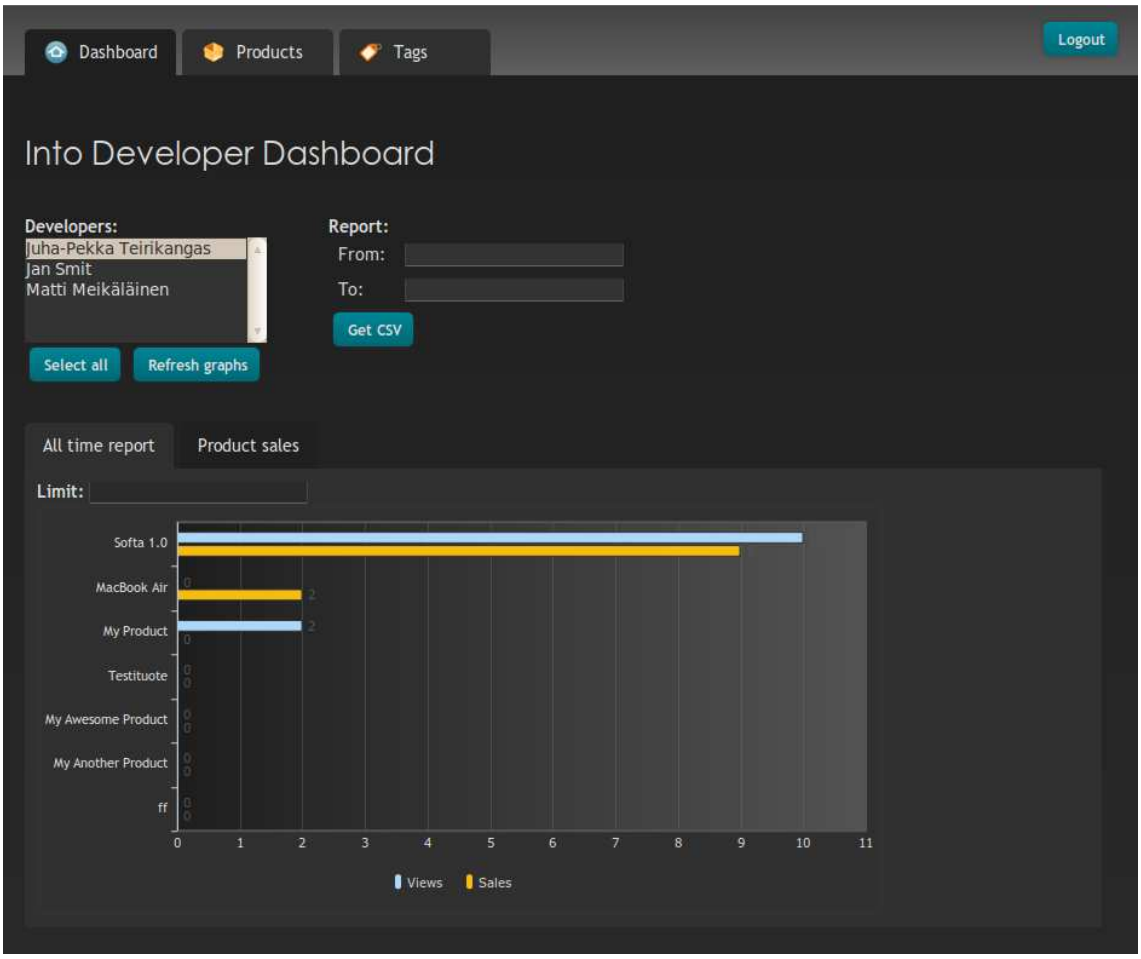
KUVIO 2. Kaavion latausanimaatio ja -teksti

LoadingSpinner luo latauselementin, joka lisätään kohteen eli tässä tapauksessa kirjautumisnapin perään ja käyttäen script.aculo.usin `appear`-tehostetta ilmestyy näkyviin asteittain, mutta hyvin nopeasti. Vaikka pyyntö suoritettaisiin millisekunneissa, on luokka määritelty näyttämään latausanimaation ja mahdollisen tekstin määritellyn ajan, esimerkiksi puoli sekuntia, jolloin käyttäjä ehtii havainnoida sen, eikä se jää vain turhaksi vilahdukseksi. Tässä tapauksessa palauteviesti näytetään ja latauskiekko häivytetään näkyvistä `fade`-tehosteella. Tilalle ilmestyy samaa tehostetta käyttäen kuin aikaisemminkin symboli sen mukaan, oliko pyyntö onnistunut vai ei, tekstin sijainnin ollessa sama kokoajan. Näin ollen näyttää kuin latausanimaatio ikään kuin muuttuisi palauteikoniksi.

Onnistunut pyyntö korostetaan vihreällä V-symbolilla ja epäonnistunut punaisella X:llä. Palauteteksti ja -ikoni häipyvät näkyvistä määritellyn ajan jälkeen, joka tässä tapauksessa on 10 sekuntia. LoadingSpinner tekee yhdestä napista vain yhden pyynnön kerrallaan, joten kirjautumispainikkeen painaminen useampaan kertaan ei tulvi näytölle lataustekstejä, mutta palauteviestejä voi olla useampiakin näkyvissä samanaikaisesti viiveen takia. Onnistuneen kirjautumisen jälkeen sivu ladataan uudelleen ja koska sessio nyt löytyy palvelimelta, ei kirjautumista enää näytetä vaan näkyviin tulevat uudet välilehdet, riippuen siitä onko tavallinen käyttäjä vai ylläpitäjä.

5.2 Tilastot

Tilastosivulle (KUVIO 3) on koottu myynti- ja näyttötilastoja koko tuotteiden elinkaarelta tai myyntiraportteja halutulta ajanjaksolta. Näyttömäärät tarkoittavat montako kertaa tuotteen sivua on katsottu verkkokaupassa, mutta sitä ei voi ryhmitellä ajan mukaan, koska käytetty OpenCart-verkkokauppa tallentaa tietokantaan määrän vain kokoajalta. Taulukoiden toteuttamisessa hyödynnetään FusionChartsia, joka on valmis työkalu taulukkomuotoisen tiedon esittämiseen. Toimeksiantajan toiveesta käytetään yleisempien flash-kaavioiden sijaan JavaScriptillä toteutettuja kuvioita, jotta sivusto näyttäisi yhdenmukaiselta kaikilla alustoilla.



KUVIO 3. Tilastosivu ylläpitäjän näkemänä

FusionChartin mukana tulee kolme JavaScript-tiedostoa: `FusionCharts.js`, `highcharts.js` ja `jquery.min.js`. Jos tiedostot ovat samassa kansiossa, vain ensimmäinen tiedosto täytyy ottaa mukaan sivulle ja muut tulevat automaattisesti tarvittaessa. JavaScript-koodissa taulukko tehdään luomalla olio `FusionCharts`-luokasta: `var kaavio = new FusionCharts("Column3D.swf", "kaavionId", "400", "300")`. Riippumatta siitä halutaanko kaavio piirtää sivulle käyttäen flash- tai JavaScript-toteutusta, annetaan ensimmäisenä parametrina käytettävän kaaviotyypin swf-tiedosto. Se on myös ainoa pakollinen parametri. KaavionId on uudelle kaaviolle annettava id, jonka avulla siihen voi myöhemmin viitata, 400 ja 300 taas ovat leveys ja korkeus. (FusionCharts 2011b, hakupäivä 12.5.2011.)

Olion luomisen jälkeen kaavion tiedot haetaan tietokannasta ja muutetaan JSON (JavaScript Object Notation) muotoon, jota FusionCharts ymmärtää. JSON rakentuu avain ja arvo -pareista, joissa kaksoispiste on avaimen jälkeen ja erilliset parit erotetaan toisistaan pilkuilla. JavaScriptin

olion mukaisesti nämä kokonaisuudet laitetaan aaltosulkeiden sisälle, esimerkiksi: `{ ”nimi”: ”Jaakko”, ”ammatti”: ”metsuri” }`. Kaavion tiedot voisi myös antaa XML-muodossa, mutta tässä yhteydessä sitä ei käytetty. (FusionCharts 2011a, hakupäivä 1.3.2011.) Data asetetaan kutsumalla funktiota `kaavio.setJSONData(jsonMuuttuja)` (FusionCharts 2011c, hakupäivä 1.3.2011).

Kaavion luominen sivulle voidaan tämän jälkeen tehdä funktiolla `kaavio.render(”kohdeId”)`, jossa annetaan parametrina sen elementin id, jonka sisälle kaavio halutaan piirtää. Koska sekä flash- että JavaScript-versiot luodaan samalla tavalla, tekisi edellä esitelty koodi oletuksena flash-kaavion tai jos sitä ei tueta, JavaScript-kaavion. Tässä toteutuksessa ei haluttu käyttää flashia lainkaan, joten ennen `kaavio`-olion luontia täytyy tämä kertoa FusionChartsille: `FusionCharts.setCurrentRenderer("javascript")`, jonka jälkeen käytetään aina pelkkää JavaScript-renderöintiä. (FusionCharts 2011b, hakupäivä 1.3.2011.)

Halutun ajanjakson myyntiä on mahdollista selata käyttäen valmiita määrityksiä eli tämän vuoden, kuukauden ja viikon tilastot tai valitsemalla haluttu ajanjakso päivämääräkentillä. Näiden lisäksi on mahdollista valita näytetäänkö taulukossa ostojen määrä vai ostojen arvo. Päivämääräkenttiin voi itse kirjoittaa halutun päivän tai käyttää kenttää klikkaamalla avautuvaa kalenteria, joka on toteutettu avoimen lähdekoodin CalendarView JavaScript-käyttöliittymäkomponentilla. CalendarViewiin kuuluu kaksi tiedostoa `calendarview.js` ja `calendarview.css`, joista ainakin js-tiedosto täytyy olla mukana, jotta kalenteri toimisi. Ne ovat ladattavissa ilmaiseksi osoitteesta http://calendarview.org.

Sivun latauduttua suoritetaan koodi, joka avaa kalenterin sivun päälle. Popup-kalenterin lisäksi CalendarViewin voi myös upottaa sivulle antamalla asetuksissa kohde-elementti eli `parentElement`. Tässä yhteydessä sen sijaan tarvitaan `triggerElement`, joka määrittelee elementin, jota klikkaamalla kalenteri aukeaa eli sama tekstikenttä, johon päivämäärä kirjoitetaan eli `dateField`. Näiden lisäksi kalenterille voidaan määritellä myös kaksi takaisinkutsua `selectHandler` ja `closeHandler`, jotka suoritetaan valittaessa päivämäärä ja suljettaessa kalenteri. Kalenterin alustusesimerkki:

```
Calendar.setup(
{
```

```
    dateField: 'txtAlkupvm',
    triggerElement: 'txtAlkupvm'
});
```

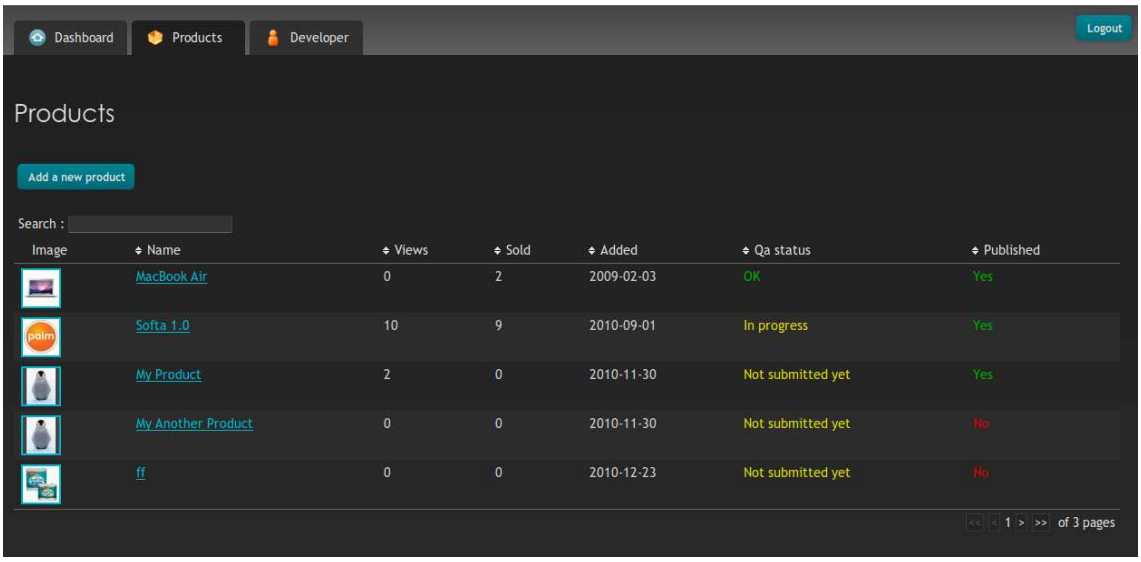
(CalendarView 2011, hakupäivä 4.5.2011.)

Myyntitilastot ovat myös ladattavissa CSV-tiedostona (comma-separated values), jonka avulla tilastotietojen siirtäminen esimerkiksi taulukkolaskentaohjelmaan on helppoa. CSV on tiedosto, joka sisältää puolipisteellä tai pilkulla erotettuja arvoja rivitettynä, ikään kuin taulukkona. Ensimmäinen rivi voi olla otsikkorivi, mutta se ei ole välttämätöntä. Tässä tapauksessa käytettiin puolipisteitä ja otsikkoriviä. Latauspainiketta painettaessa ohjataan selain PHP-tiedostoon, joka luo tarvittavan CSV-tiedoston ja lähettää sen käyttäjälle takaisin HTTP-headerin avulla, jolloin käyttäjälle avautuu normaaliin tapaan tiedoston latausikkuna.

Ylläpitäjän näkymä on täällä muuten vastaava, mutta koska tällöin voidaan näyttää kaikkien käyttäjien tuotteiden tiedot, on lisätty näytettävien tuotteiden määrän rajoituskenttä ja käyttäjien valintalista. Ylläpitäjä voi valita näytettäväksi kaikki tuotteet jättämällä rajoituskentän tyhjäksi, mutta muuten näytetään aina annettua lukumäärää vastaava määrä tuotteita tai niin monta kuin niitä löytyy. Kokoajalta oleva myynti- ja näyttötilasto on oletuksena lajiteltu suurimmasta myyntimäärästä pienimpään tai jos nämä ovat samat niin näyttöjen mukaan, joten ylläpitäjä voi siis valita näytettäväksi haluamansa määrän verkkokaupan myydyimpiä tuotteita. Vastaava lajittelu löytyy myös myyntitilastoista, siellä vain ei oteta näyttökertoja huomioon. Käyttäjälistasta on mahdollista valita kaikki käyttäjät tai vain tietyt, jolloin tilastoissa näkyvät vain näiden käyttäjien tuotteet.

5.3 Tuotteet

Tuotteet sivulla (KUVIO 4) kehittäjä voi selata vanhoja ja lisätä uusia tuotteita, lähettää niitä hyväksyttäväksi verkkokauppaan lisäystä varten sekä muokata ja poistaa aiemmin lisättyjä kohteita. Ylläpidon käyttämä laadunvarmistus- eli QA-prosessi (quality assurance) tapahtuu myös tällä sivulla. Oletuksena sivulla näkyy uuden tuotteen lisäysnappi sekä taulukko aiemmin lisätyistä, johon on käytetty internetistä löytyvää valmista TableOrderer-skriptiä. (Projektin osoite: http://prototools.negko.com/demo/tableorderer) Se on Prototypen päälle rakennettu dynaaminen taulukko, jonka avulla tietoja voidaan sivuttaa, lajitella ja hakea. Tietokannasta löytyvät tuotetiedot syötetään käyttäen JSON-formaattia tai oliotaulukkoa.



KUVIO 4. Tuotesivun oletusnäkymä

Taulukon luominen sivulle tapahtuu antamalla kohde-elementti, jonka sisälle taulukko luodaan ja näytettävät tiedot: `new TableOrderer('sailionId', { data: tiedot });`. Valitsemalla hiirellä taulukossa oleva tuotekuva, se avataan suurempana lightbox-tehostetta käyttäen. Tällöin näytön tausta himmenee ja suurempi kuva ilmestyy näytölle sulkemispainikkeen ja halutun selitteen kera. Tässä yhteydessä on käytetty Lightbox 2 -nimistä skriptiä, joka on toteutettu käyttäen Prototypeä ja script.aculo.usta. Se on vapaasti ladattavissa osoitteesta: http://www.huddletogether.com/projects/lightbox2.

Lightbox 2 tarvitsee toimiakseen Prototypen lisäksi myös script.aculo.us-kirjaston ja luonnollisesti mukana tulevat JavaScript- ja CSS-tiedostot. Latauspaketin mukana tulee myös joitain kuvia, kuten sulkemisikoni. Kun tarvittavat tiedostot ovat mukana, saa tehosteen toimimaan lisäämällä mihin tahansa sivun linkkiin `rel="lightbox"`-attribuutin. Linkin `title` näytetään kuvatekstinä. Lisäämällä useita kuvia samaan ryhmään, voi kuvan katselunäkymässä selata ryhmän kuvia edellinen ja seuraava -ikoneita painamalla. Ryhmä määritellään hakasulkeisiin `rel`-attribuutin lightbox-sanan perään, esimerkiksi: `rel="lightbox[kuvat]"`. (Dhakar 2011, hakupäivä 4.5.2011.)

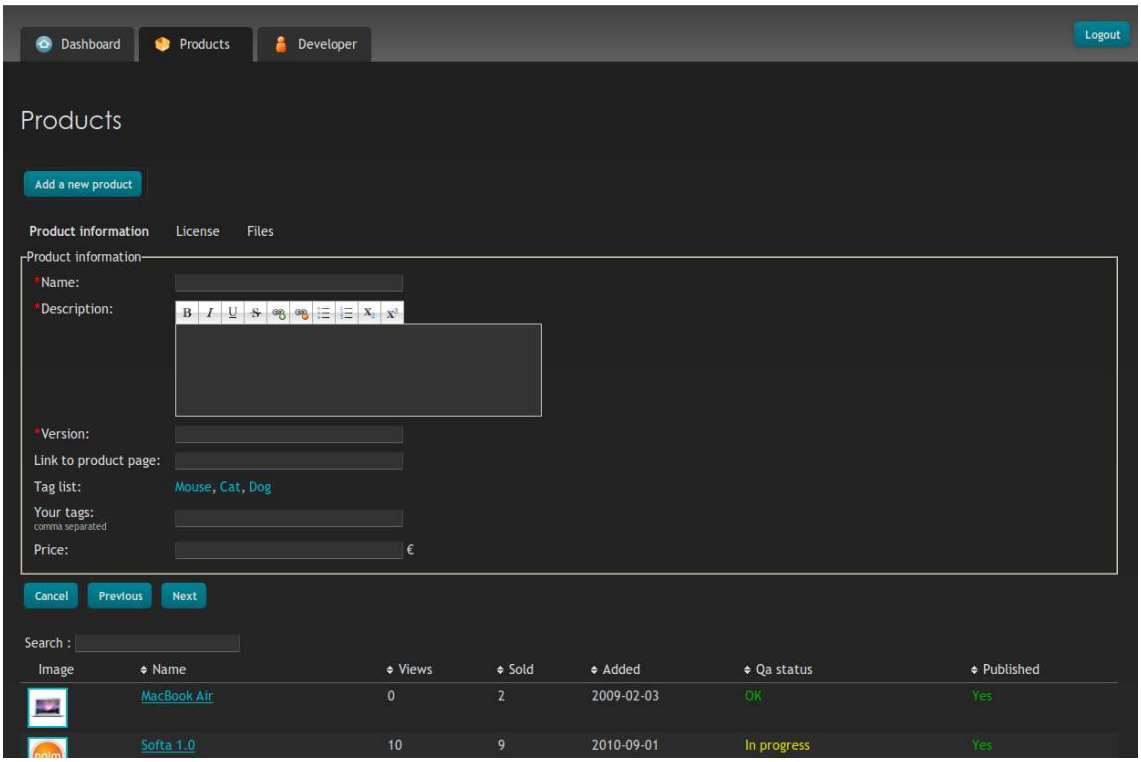
Klikkaamalla taulukossa tuotteen nimeä, avautuu sen rivin alle muokkausnäkymä, jonka avulla kehittäjä tai ylläpitäjä voi tehdä tarvittavia muutoksia tuotteeseensa, lähettää se hyväksyttäväksi, piilottaa tuote verkkokaupasta tai poistaa se kokonaan. Tuotteen hyväksymisprosessin käynnistäminen lukitsee muokkauksen, kunnes ylläpito on hyväksynyt tuotteen ja siten lisännyt sen verk-

kokauppaan. Muokkaamalla hyväksymisen jälkeen tuotetta uudelleen se pitää uudelleen hyväksyttää, ennen kuin se voidaan taas julkaista.

Tuotelistauksen perustietoihin kuuluu tuotekuvan ja -nimen lisäksi tuotteen näyttö- ja myyntikertojen lukumäärä, lisäyspäivämäärä sekä hyväksymisprosessin ja julkaisun statukset. Hyväksymisstatuksia on neljä erilaista: ei vielä lähetetty (Not submitted yet), hyväksytty (OK), hylätty (Failed) ja hyväksyminen kesken (In progress). Statustekstit on myös värikoodattu siten, että hyväksytty ja julkaistu on kirjoitettu vihreällä tekstillä, kun taas hylätty ja piilotettu on punaisella. Ei lähetetyt ja kesken olevan hyväksymisen tila taas on esitetty keltaisella. Näin kehittäjä voi yhdellä vilkaisulla nähdä tuotteidensa tilanteen lukematta tekstejä.

5.3.1 Uuden tuotteen lisääminen

Tuotteen lisäys on mahdollista vain tavalliselle käyttäjälle, koska verkkokaupassa myytävät Intopii Oy:n tuotteet lisätään OpenCartin oman käyttöliittymän kautta ja hallitaan sieltä käsin. Tuotteen lisäysnappia painettaessa tehdään palvelimelle Ajax-pyyntö, jolla haetaan lisäykseen käytetty lomake ja sijoitetaan se piilotettuna tuotetaulukon (KUVIO 5) yläpuolella olevaan `div`-elementtiin. JavaScript-luokka FormWizard erottelee haetun lomakkeen fieldset-elementit omiksi sivuikseen, siirtää legendin otsikot välilehdiksi ja lisää peruuta, edellinen, seuraava ja lähetä -napit sivujen alle.



KUVIO 5. Uuden tuotteen lisääminen

Ensimmäisellä sivulla on tuotteen perustiedot: nimi, kuvaus, versio, linkki ulkoiselle tuotesivulle, tagit ja hinta. Vain kolme ensimmäistä ovat pakollisia kenttiä, mistä kertoo punainen tähti kenttien selitetekstien edessä. Hintaa ei sinänsä ole pakko antaa, mutta jos sitä ei ole, oletetaan hinnaksi nolla euroa. Tagilistan kohteita klikkaamalla, ne siirtyvät automaattisesti tagien tekstikenttään (Your tags) tai vastaavasti, jos se löytyy jo sieltä, se poistetaan. JavaScriptin-koodi osaa ottaa huomioon pilkkujen sijainnit tekstikentässä lisäten ne vain tagien väliin, eikä enää viimeisen perään.

Tuotekuvauksessa on rich text eli WYSIWYG (What You See Is What You Get) -kenttä, mikä tarkoittaa sitä, että tekstiin on mahdollista tehdä muotoiluja, kuten lihavointia, kursiivia tai listoja. Tekstikenttä myös piilottaa kaikki HTML-elementti tagit, jolloin esimerkiksi kappaletta tarkoittava <p> on piilotettu. Tavallisen textarean muuttamiseen rich text -muokkaimeksi on käytetty Prototypeä hyödyntävää Webadmin Rich Text Editoria, joka on ladattavissa osoitteesta: http://labs.fivebyfivedigital.com/users/rumble/webadmin-richtexteditor.

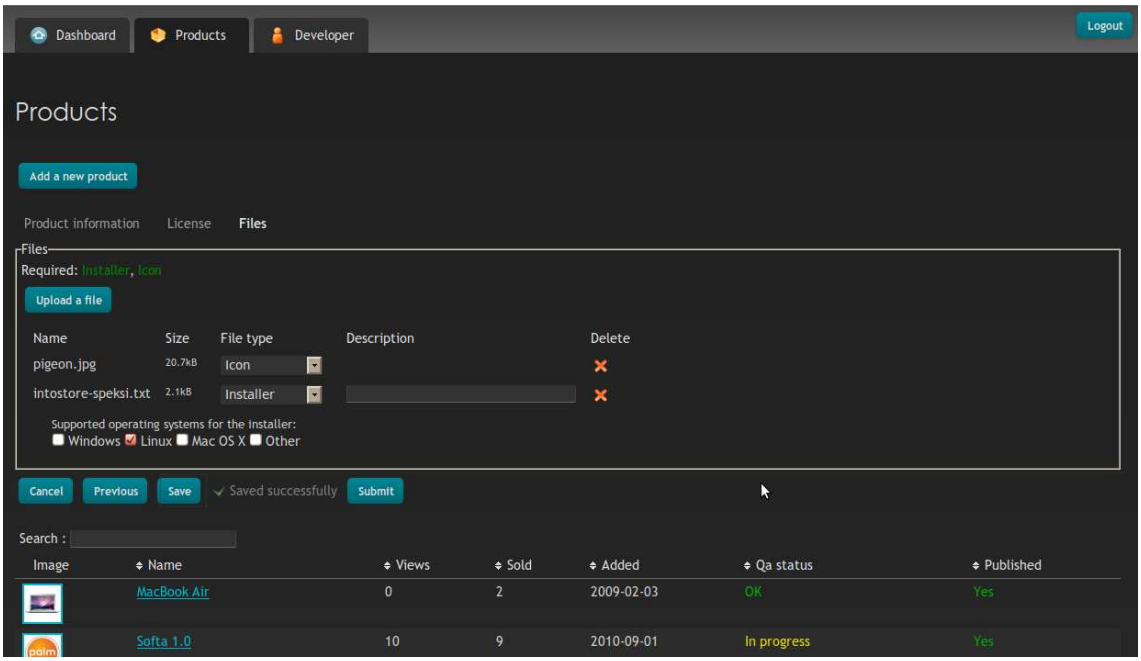
Tarvittava koodi on hyvin lyhyt ja se vaatii yksinkertaisimmillaan vain yhden parametrin eli tekstikentän id:n. `new Webadmin.RichTextEditorForm( "tekstikentanId", {asetukset});` Jos haluaa käyttää takaisinkutsuja tai määritellä, mitkä fonttimuokkaimet

ovat käytössä, täytyy ottaa käyttöön asetukset-parametri. Fonttimuokkaimet, kuten vaikka lihavoinnin käyttöönotto tapahtuu avain-arvo -parilla, joissa avain on ”bold” ja arvo joko true tai false. (Rumble 2011, hakupäivä 3.5.2011.)

Seuraavalla välilehdellä on tuotteen lisenssitiedot, joka sisältää vain valintaruutulistan käytettävissä olevista lisensseistä. Jos tuotteelle on annettu jokin muu hinta kuin nolla euroa, näytetään kaupallisten lisenssien lista ja ilmaisella tuotteella taas ei-kaupalliset lisenssit. Kaupallisissa lisensseissä on myös mahdollisuus tehdä oma lisenssi, joka on sen luomisesta lähtien aina käytettävissä kyseisen kehittäjän lisenssilistassa. Lisenssien otsikko ja teksti on tallennettu tietokantaan. josta ne tulostetaan sivulle, mutta ensimmäisen sivun hinnasta riippuen piilotetaan kaupalliset tai ei-kaupalliset vaihtoehdot.

Viimeisellä lisäämistoiminnon välilehdellä (KUVIO 6) määritellään kaikki tuotteeseen liittyvät tiedostot, kuten logo, kuvankaappaukset, lähdetiedostot, asennuspaketit jne. Lisättyihin tuotteisiin voidaan määritellä pakolliset tiedostotyypit, jotka tässä esimerkissä ovat logo ja asennustiedosto. Vaaditut tiedostotyypit listataan tiedoston lisäys -napin yläpuolelle ja korostetaan punaisella värillä, jos sen tyyppistä tiedostoa ei ole lisätty tai vihreällä, jos se on olemassa. JavaScript-koodi huolehtii tästä, käymällä lisätyt tiedostot läpi jokaisen tiedostotyyppimuutoksen yhteydessä.

Tiedostotyypeille on määritelty myös sallitut tiedostomuodot. Esimerkiksi kuvatiedosto voi olla vain logo tai jokin muu kuva, eikä esimerkiksi asennustiedosto. Näin ollen vain nämä ovat valittavissa pudotusvalikosta tiedoston kohdalla. OpenCartin tietokannan rakenteen johdosta, logolla ei voi olla kuvausta, mutta kaikille muille tiedostotyypeille on mahdollista antaa haluttu kuvausteksti. Asennustiedostojen yhteydessä kysytään myös kohdekäyttöjärjestelmät, jotka valitaan valintaruuduilla. Valitsemalla vaihtoehto jokin muu eli other, ilmestyy alle tekstikenttä omien pilkulla erotettujen käyttöjärjestelmien nimien syöttämistä varten.



KUVIO 6. Uuden tuotteen tiedostojen lisäys

Tiedostoja voi lisätä valitsemalla lisäysnappi (Upload a file) tai raahaamalla tiedostoja sivulle. Ajax Upload -skriptiin kuuluu etenemispalkki ja latauksen peruuttamismahdollisuus, niitä tukevissa selaimissa. Se löytyy ilmaiseksi osoitteesta: http://valums.com/ajax-upload. Lisättyjä tiedostoja voi poistaa latauksen jälkeen painamalla punaista x-symbolia tiedostorivin perässä, mikä heittää käyttäjälle varmennusikkunan. Tuotteen lähettäminen hyväksyttäväksi onnistuu tiedostovälilehdeltä vain, jos tarvittavat kentät on täytetty.

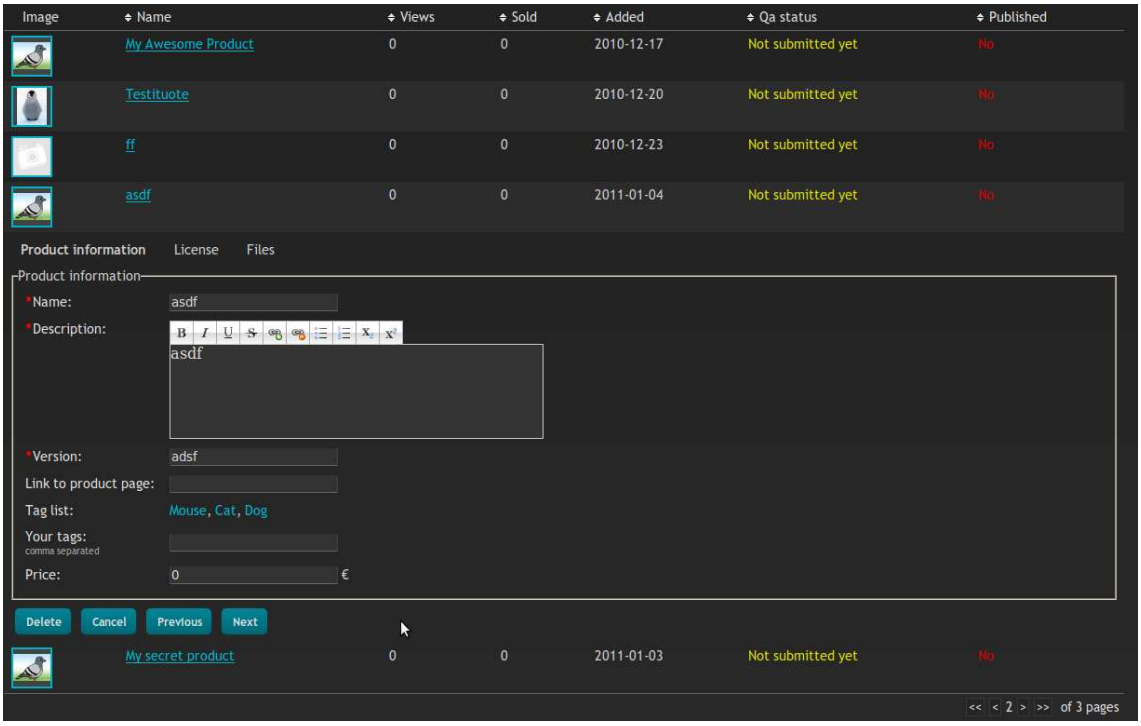
Ajax Upload käyttää tiedostojen siirtoon XMLHttpRequestia (XHR) ja sitä voisi käyttää myös salatun yhteyden (HTTPS) yli. Latauspaketissa on esimerkkikoodit ja tarvittavat tiedostot sekä asiakkaan että palvelimen päähän. `Fileuploader.js`-tiedostossa on kaksi suoraan käytettävää luokkaa: `qq.FileUploader` ja `qq.FileUploaderBasic`. Ensimmäinen on valmis ratkaisu, jonka voi ottaa suoraan käyttöön sivulla, kun taas basic-vaihtoehto sisältää vain perusominaisuudet, jonka päälle voi itse tehdä haluamansa ulkoasun ja toiminnallisuuden. (Valums 2010, hakupäivä 14.5.2011.) Tässä on käytetty valmista luokkaa, mutta siihen kuuluva HTML-template eli ladattujen tiedostojen esitystapa on tehty uudelleen muistuttamaan aiemmin lisättyjen tiedostojen listausta.

5.3.2 Tuotteen muokkaus

Prototypen avulla voidaan sivun latauduttua kokonaan ajaa haluttua JavaScript-koodia tarttumalla Prototypen omaan dom:loaded tapahtumaan. Se on erittäin hyödyllinen tilanteissa, joissa sivuun tehdään muutoksia selainpäässä esimerkiksi JavaScript-koodin ajon jälkeen. Tapahtuman käsittely näyttää tältä:

```
document.observe("dom:loaded", function()
{
     // Ajettava koodi
});
```

Tuotteen muokkauksessa on hyödynnetty tätä toimintoa, koska tuotelistaus tehdään JavaScriptin avulla ja tarvittu muokkaustoiminto voidaan lisätä taulukkoon vasta kun se on valmis. Toisin sanoen ensin tehdään PHP:llä sivu palvelimella, lähetetään se selaimelle, luodaan tuotelista tietokannasta saaduilla tiedoilla ja lopuksi lisätään toimintoja tuotelistan riveihin. Tuotteen muokkaus on tällainen toiminto, joka hakee tarvittavat tiedot palvelimelta Ajax-pyynnöllä ja lisää taulukon tuoterivin alle tuotteen lisäyksestä tutun sivutetun lomakkeen tuotteen tiedoilla täytettynä (KUVIO 7).



KUVIO 7. Tuotteen tietojen muokkaus

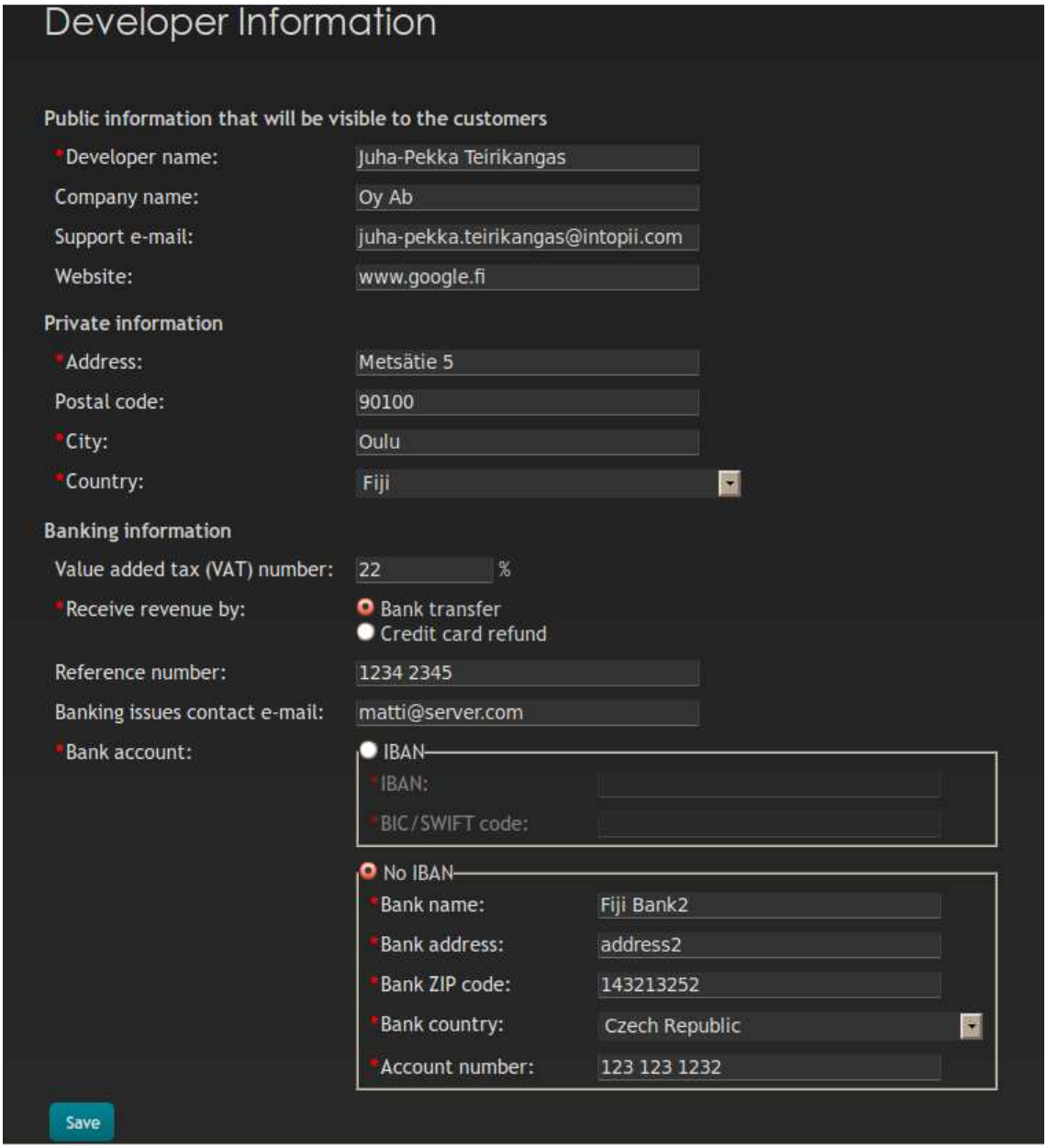
Käytännössä lisättäessä muokkausmahdollisuus tuotelistaan, käydään kaikki taulukon rivit läpi ja lisätään tuotteen nimiin `onClick`-tapahtuma, joka liu'uttaa script.aculo.uksen `slide`-tehostetta käyttäen haetun muokkauslomakkeen näkyviin. Koodi voisi näyttää vaikka tältä:

```
$$("#tuoteLista tr td.nimi").each(function (nimiSolu)
{
     nimiSolu.observe("click", function (tapahtuma)
     {
          // Tässä olisi koodi jolla lomake haetaan
     });
});
```

Muokkauslomake ei eroa lisäyslomakkeesta mitenkään muuten, kuin että tiedostot-välilehdellä on aiemmin lisätyt tiedostot listattuna oman alaotsikkonsa alle ilman tiedostojen lisäykseen käytettyä Ajax Uploaderia. Tiedostot on ryhmitelty tyypin mukaan, eikä esimerkiksi kuvatiedostoa voi enää vaihtaa asennustiedostoksi. Uusia tiedostoja voi lisätä normaaliin tapaan ja kaikki listatut tiedostot ovat poistattavissa ja rajoitetusti muokattavissa. Lisättäessä uusia tiedostoja niitä ei liitetä aiemmin lisättyjen tiedostojen listaan vaan ne listataan Ajax Uploaderin avulla samalla tavalla kuin uuden tuotteen lisäyksessäkin.

5.4 Kehittäjä

Vain tavallisille käyttäjille näkyvä kehittäjä sivu (KUVIO 8) sisältää tarvittavat nimi- ja yhteystiedot sekä pankki- tai luottokorttitiedot mahdollisten myyntitulojen maksua varten. Nimi, sähköposti ja www-sivujen osoite ovat julkisia tietoja, jotka näkyvät verkkokaupan asiakkaille tuotteiden myyntisivuilla. Postiosoite ja pankki- tai luottokorttitiedot taas ovat luonnollisesti vain Intopiin omaan käyttöä varten, eivätkä siten ole julkisesti saatavilla.



KUVIO 8. Kehittäjäsivu pankkitiedoilla

Maalistan valtiot haetaan OpenCartin tietokannasta johon ne, sekä kyseisten maiden postinumeroiden pakollisuudet, ovat tallennettu. Näin postinumero voidaan merkitä pakolliseksi kentäksi vain sellaisten maiden yhteyteen, joissa se on käytössä. Aina vaihdettaessa pudotuslistassa valittua maata, sivusto tekee Ajaxilla pyynnön palvelimelle, joka palauttaa tiedon, onko tässä maassa postinumero pakollinen. Vastauksen mukaan joko merkitään punainen tähti postinumerokentän selitteen eteen tai poistetaan se, jos se on jo olemassa.

Korvauksen maksutapavalinnan (Receive revenue by) mukaan näytetään käyttäjälle pankkiin tai luottokorttiin liittyvät kentät. Kaikki kohteet ovat aina sivulla, mutta vain valitun kategorian kentät

ovat näkyvissä. Vaihdettaessa valintanappia piilotetaan näkyvissä olevat kentät käyttäen script.aculo.uksen `hide`-funktiota ja näytetään asteittain uudet `appear`-tehosteella. Luotto-korttivaihtoehdossa on vain yksi kenttä kortin numeroa varten, mutta pankkitiedoissa tarvitaan enemmän, koska halutaan tarjota mahdollisuus käyttää myös ulkomaisia pankkeja. Pankkitilitie-toihin voidaan syöttää joko suurimmassa osassa Eurooppaa ja joissakin sen ulkopuolisissa maissa käytössä oleva IBAN (International Bank Account Number) -tilinumero tai antaa sekä tili- että pankin tiedot (Nordea 2011, hakupäivä 11.4.2011).

IBAN-valintanappien mukaan poistetaan ei-valitun fieldsetin tekstikentät käytöstä ja piilotetaan osittain kaikki sen sisältämät elementit legendia eli otsikkoa lukuun ottamatta. Tämä tehdään hakemalla ensin fieldsetin legend ja kutsumalla Prototypen funktiota `siblings`, joka palauttaa kaikki elementin sisarukset eli saman äitielementin sisällä olevat kohteet. Löytyneet elementit häivytetään osittain script.aculo.ouksen `fade`-tehosteella antamalla `from`- ja `to`-arvot para-metrina annetun kokoelman mukana. Toteutus voisi näyttää tältä:

```
$$('#fsId > #legId').first().siblings().each(function (el)
{
     el.fade({ from: 1, to: 0.5 });
});
```

5.5 Tagit

Tagi on web-sovelluksissa käytetty eräänlainen avainsana, joka kuvaa tiettyä kohdetta ja jonka avulla voi löytää kaikki tiettyyn asiaan liittyvät kohteet. Ne ovat siis metatietoa eli tietoa tiedosta. Verkkokaupassa tuotteilla voisi olla tagina esimerkiksi valmistajan nimi, käyttötarkoitus tai tuote-ryhmä. Näin tuotteen tietoja luettaessa voi valita tagin ja saada näkyviin kaikki saman valmistajan tuotteet tai etsiä hakutoimintoa käyttäen tietyn valmistajan tietyn tyyppiset tuotteet, hyödyntäen tageihin tallennettua tietoa.

Tuotteiden tagien hallinta on vain ylläpitäjille näkyvä sivu, jonka avulla voi lisätä, poistaa ja kate-gorisoida valmiiksi asetettuja tageja (KUVIO 9). Näiden lisäksi sivulla näkee myös verkkokaupan yleisimmät tagit joko asetetuista, käyttäjien itse lisäämistä tai kaikista. Tämän avulla voidaan esi-merkiksi huomata, jos useat käyttäjät lisäävät jonkin tagin tuotteeseensa, mutta sitä ei ole lisätty valmiiden tagien joukkoon. Tagikategorioiden lisääminen ei ole pakollista ja jos niitä ei ole, käyt-

täjä näkee vain listan tageista ilman kategorioiden nimiä. Hallintapuolella kategoriattomat tagit näkyvät Misc-nimikkeen alla.

KUVIO 9. Tagi-sivu ilman lisättyjä kategorioita

Tagit tulostetaan kategorioidensa alle ja raahaamalla tagi listasta toiseen, voidaan vaihtaa sen kategoriaa. Tämä on toteutettu käyttämällä script.aculo.uksesta löytyvää `Sortable`-luokkaa, jonka `create`-funktiolla saadaan listan elementit raahattua ja järjesteltyä uudelleen sekä antamalla parametrina taulukko, joka sisältää molempien listojen id:t, voidaan siirtää kohteita myös listojen välillä. Jos listaelementtien nimet olisivat lista1 ja lista2, koodi näyttäisi tältä:

```
Sortable.create('lista1',
{
     containment: ['lista1', 'lista2'],
     dropOnEmpty: true
});
Sortable.create('lista2',
{
     containment: ['lista1', 'lista2'],
     dropOnEmpty: true
});
```

6 POHDINTA

Tavoitteena oli luoda toimeksiantajalle sivusto, joka muuttaisi heidän verkkokauppansa yleiseksi kauppapaikaksi ja mielestäni tässä tavoitteessa onnistuttiin. Työssä oli kaikki projektin alussa tehdyn ja myöhemmin täydennetyn vaatimusmäärittelyn mukaiset toiminnallisuudet, vaikkakin testaus olisi voinut olla perusteellisempaa. Toki kaikki uudet ominaisuudet sekä mahdolliset sivuvaikutukset pyrittiin testaamaan ja mahdolliset ongelmat korjaamaan, mutta huomasin pitäytyväni tiettyihin kaavoihin testityössä. Tästä johtuen oman testaukseni jälkeen toimivaksi todettu ominaisuus saattoi toisen käyttäjän testaamana olla täysin käyttökelvoton, koska hän lähestyi tehtävää erilaisesta näkökulmasta ja teki siten asioita eri tavalla ja eri järjestyksessä kuin minä.

Toinen testauksen kompastuskivi oli selainyhteensopivuus. Vaikka käytössä olikin juuri yhteensopivuutta ajatellen suunniteltu JavaScript-kirjasto, liittyivät ongelmat osittain perinteiseen JavaScript-koodin, jota oli myös mukana ja ulkoasuseikkoihin. Mozilla Firefox oli kehitystyössä pääselain ja kaikki testattiin sillä, kun taas muita vaihtoehtoja tarkasteltiin vain silloin tällöin. Toisaalta sivuston toteutuksessa käytettiin Ubuntu Linuxia, johon Internet Exploreria ei ole suunniteltu asennettavaksi. Käytössä oli kyllä etätyöpöytäyhteys Windows-koneeseen, mutta koska IE:stä voi asentaa vain yhden version kerrallaan ja eri versiot poikkeavat huomattavasti toisistaan, testaus ei ehkä ollut tarpeeksi kattavaa tältä osin.

Sivuston tilastokaaviot piirrettiin FusionChartsilla, mutta koska toimeksiantaja halusi käyttää JavaScript-pohjaista esitysmuotoa Adobe Flashin sijaan, jouduin tekemään paljon ylimääräistä työtä saadakseni ulkoasun toivotun mallin mukaiseksi. Tämä johtui lähinnä siitä, että JavaScript-versio on vasta kehitysasteella ja se sisälsi paljon bugeja ja muita epäloogisuuksia. Ulkoasun määritteitä täytyi tehdä laskelmieni mukaan kolmeen eri paikkaan, koska jostain syystä tietyt määrittelyt toimivat vain tietyssä paikassa, eikä keskittäminen siksi ollut mahdollista. Keskustelupalstat olivat käytännössä ainoa paikka, josta saattoi löytää vihjeitä, koska viralliset dokumentaatiot olivat vielä kesken.

Käytettäessä useita erilaisia valmiita JavaScript-tiedostoja huomasi nopeasti, että eri kirjastoja hyödyntäviä komponentteja on hankala käyttää yhdessä. Esimerkiksi Prototypen ja jQueryn käyttäminen yhtä aikaa vaatii joidenkin asioiden huomioonottamista. Näissä molemmissa on $-funktio, jolloin koodia ajettaessa ei voi tietää kumman kirjaston funktiota yritetään kutsua. Tämä

voidaan korjata kutsumalla funktiota `jQuery.noConflict()`, jonka jälkeen jQueryn $-funktioon voi viitata vain nimellä jQuery ja Prototypen $-funktion pitäisi toimia normaalisti. Yleisesti voi kuitenkin sanoa, että on helpompaa käyttää vain yhtä kirjastoa kerrallaan.

Vaikka projektin alussa tehty vaatimusmäärittely oli kattava ja säilyi melko muuttumattomana läpi koko prosessin, tuli katselmointivaiheissa aina jonkin verran muutoksia olemassa oleviin toteutuksiin ja ehdotuksia uusiin. Pääasiassa nämä olivat pieniä kosmeettisia muutoksia tai muita epäloogisuuksien korjauksia, mutta tagi-sivu on esimerkki suuremmasta lisäyksestä, joka otettiin mukaan vasta hyvin loppuvaiheessa. Välillä syntyi jopa pieniä ristiriitoja siitä, mikä olisi paras ratkaisu johonkin ongelmaan, mutta pääasiassa kehitystyö eteni sujuvasti ja palautetta sai säännöllisesti. Lopussa tehty hyväksymistestaus ja siihen liittynyt koodin läpikäynti oli hyödyllinen kokemus myös tekijän kannalta, kun sai kuulla kokeneempien mielipiteitä käytetyistä menetelmistä ja ohjelmointitavoista.

LÄHTEET

AjaxLine. 2011. Javascript library. Hakupäivä 13.2.2011 http://www.ajaxline.com/10-most-popular-javascript-frameworks.

Apple. 2011. Safarin ominaisuudet. Hakupäivä 11.4.2011 http://www.apple.com/fi/safari/features.html.

CalendarView. 2011. CalendarView for Prototype. Hakupäivä 4.5.2011 http://calendarview.org.

Dhakar, L. 2011. Lightbox 2. Hakupäivä 4.5.2011 http://www.huddletogether.com/projects/lightbox2.

Dupont, A. 2008. Practical Prototype and script.aculo.us. New York: Apress.

FusionCharts. 2011a. FusionCharts and JSON. Hakupäivä 1.3.2011 http://www.fusioncharts.com/docs/DataFormats/JSON/Overview.html.

FusionCharts. 2011b. Using JavaScript (HTML5) renderer. Hakupäivä 12.5.2011 http://www.fusioncharts.com/docs/FirstChart/UsingPureJS.html.

FusionCharts. 2011c. Using JSON as data source. Hakupäivä 12.5.2011 http://www.fusioncharts.com/docs/FirstChart/JSONData.html.

Google. 2011. Google Chrome ja nopeus. Hakupäivä 11.4.2011 http://www.google.com/chrome/intl/fi/more/speed.html.

Microsoft. 2011. Internet Explorer 9 Guide for Developers. Hakupäivä 11.4.2011 http://msdn.microsoft.com/en-us/ie/ff468705.

Mozilla Developer Network. 2011. DOM Levels. Hakupäivä 28.11.2011 https://developer.mozilla.org/en/DOM_Levels.

Mozilla. 2011. Performance. Hakupäivä 11.4.2011 http://www.mozilla.com/en-US/firefox/performance.

Nordea. 2011. IBAN maat. Hakupäivä 11.4.2011 http://www.nordea.com/Palvelut/Kansainv%C3%A4liset+tuotteet+ja+palvelut/Cash+Management/IBAN+maat/908492.html.

Opera. 2011. Features. Hakupäivä 11.4.2011 http://www.opera.com/browser/features.

Orchard, M., Pehlivanian, A., Koon, S. & Jones, H. 2009. Professional JavaScript Frameworks: Prototype,YUI, ExtJS, Dojo and MooTools. Indianapolis: Wiley Publishing.

Peltomäki, J. 2000. JavaScript: JavaScriptin peruskirja. Jyväskylä: Docendo.

Peltomäki, J. 2006. Web-selainohjelmointi. Jyväskylä: Docendo.

Porteneuve, C. 2007. Prototype and script.aculo.us: You Never Knew JavaScript Could Do This! Odessa: The Pragmatic Programmers.

Prototype JavaScript Framework. 2011a. API Docs. Hakupäivä 6.2.2011 http://api.prototypejs.org/dom/dollar-F.

Prototype JavaScript Framework. 2011b. API Docs. Hakupäivä 19.4.2011 http://api.prototypejs.org/language/Class/create.

Prototype JavaScript Framework. 2011c. Contribute. Hakupäivä 29.1.2011 http://www.prototypejs.org/contribute.

Prototype JavaScript Framework. 2011d. The Prototype Core Team. Hakupäivä 27.1.2011 http://www.prototypejs.org/core.

Rumble, M. 2011. Webadmin Rich Text Editor. Hakupäivä: 3.5.2011 http://labs.fivebyfivedigital.com/users/rumble/webadmin-richtexteditor.

Valums, A. 2010. Ajax Upload; A file upload script with progress-bar, drag-and-drop. Hakupäivä 14.5.2011 http://valums.com/ajax-upload.

W3Schools. 2011. XPath Introduction. Hakupäivä 2.5.2011 http://www.w3schools.com/xpath/xpath_intro.asp.

World Wide Web Consortium. 2000. Document Object Model (DOM) Level 2 Core Specification. Hakupäivä 11.4.2011 http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113.

World Wide Web Consortium. 2004. Document Object Model (DOM) Level 3 Core Specification. Hakupäivä 11.4.2011 http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407.