KEHYSOHJELMISTO AJAX-PORTAALIN KÄYTTÖLIITTYMÄN TOTEUTUKSEEN

Case: Rosendahl Digital Networks Oy

LAHDEN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma

Ohjelmistotekniikan suuntautumisvaihtoehto

Opinnäytetyö

Kevät 2008

Mika Vallittu

Lahden ammattikorkeakoulu

Tietotekniikan koulutusohjelma

VALLITTU, MIKA: Kehysohjelmisto Ajax-portaalin käyttöliittymän toteutukseen
Case: Rosendahl Digital Networks Oy

Ohjelmistotekniikan opinnäytetyö, 75 sivua

Kevät 2008

TIIVISTELMÄ

Tämä opinnäytetyö käsittelee JavaScript-tekniikoita web-selaimen ohjelmoinnissa ja Rosendahl Digital Networks Oy:n ikkunoidun Ajax-portaalin toteutusta.

Teoriaosassa käsitellään sitä, kuinka JavaScriptiä käytetään yhdessä muiden web-tekniikoiden kanssa, kuten HTML, DOM, CSS ja XMLHttpRequest. Nämä tekniikat muodostavat pohjan Ajax-sovelluksille, joissa työpöytäohjelmista tuttuja toimintoja, kuten elementtien raahaaminen ja pudottaminen, voidaan toteuttaa selaimessa. Lisäksi teoriaosuudessa käydään läpi muutamia Ajax-työkaluja, jotka auttavat parantamaan selainohjelmoinnin tuottavuutta ja laatua.

Työn empiirinen osuus koostuu Rosendahl Digital Networks Oy:lle toteutetusta portaaliratkaisusta. Portaalin tarkoituksena on luoda helppo käyttöliittymäratkaisu, jota voidaan hyödyntää yrityksen selainpohjaisissa tuotteissa. Vaatimusmäärittelyn perusteella saatiin selville portaalin käyttötarkoitus ja tarvittavat perusominaisuudet. Toteutuksen perusteella pohdittiin myös selainohjelmointiin liittyviä suorituskyky- ja tietoturvakysymyksiä.

Työstä saatujen kokemusten perusteella voidaan todeta, että standardeilla selaintekniikoilla on mahdollista toteuttaa varsin rikkaita käyttöliittymiä ilman, että selaimeen tarvitsisi asentaa kolmannen osapuolen liitännäisiä, kuten esimerkiksi Adobe FlashPlayer. Tulevaisuudessa selaimista on selvästi muodostumassa oma ohjelmointialustansa. Raja työpöytäohjelmien ja web-sovellusten välillä hämärtyy, kun työpöytäohjelmista tehdään internet-versioita ja web-sovelluksia siirretään työpöydälle.

Asiasanat: Ajax, CSS, DOM, HTML, Internet, JavaScript, JSON, Portaali, Web-selainohjelmointi

Lahti University of Applied Sciences

Department of Engineering

VALLITTU, MIKA: Framework for creating graphical user interface for Ajax portal
Case: Rosendahl Digital Networks Oy

Bachelor's Thesis of Software Engineering, 75 pages

Spring 2008

ABSTRACT

This research deals with JavaScript programming techniques from the web developer's perspective and the Ajax portal implementation for Rosendahl Digital Networks Oy.

The aim of the theory is to study how JavaScript is being used together with the other standard web technologies such as HTML, CSS, DOM, and XMLHttpRequest. These technologies form the basis for Ajax applications where features familiar from the desktop, like drag-and-drop, can be moved to the web. Some tools helping to improve the development productivity and code quality are also scrutinized.

The empirical part consists of the Ajax portal implemented for Rosendahl Digital Networks Oy. The plan for the portal is to create an easy-to-use user interface solution that can be utilized by company's web applications. Through requirements gathering, were the basic features and needs for the portal found. Questions of performance and security for web applications are also considered.

Based on the experience gained from this project, it can be established, that it is possible to implement rich user interfaces by only using the standard web technologies natively supported by the modern browser, without the need for third party plugins such as Adobe FlashPlayer. In future, the browser is going to be an application platform of its own. The distinction between desktop and web applications is going to blur, as desktop applications migrate to the web and as web applications are being deployed to the desktop.

Key words: Ajax, CSS, DOM, HTML, Internet, JavaScript, JSON, Portal, Web development

SISÄLLYS

1 JOHDANTO

Monien sähköisten palvelujen ja sovellusten siirtyessä internettiin käytetään web-selainta yhä useammin interaktiivisen verkkosovelluksen käyttöliittymän toteutuksessa. Modernien web-selainten avulla voidaan toteuttaa varsin rikkaita käyttöliittymiä, jotka muistuttavat ominaisuuksiltaan yhä enemmän työpöytäohjelmia. Selainpohjaiset sovellukset tarjoavat käyttäjilleen liiketaloudellisia ja teknisiä hyötyjä, kuten sovellusten helpon jakelun ja päivittämisen.

Tässä työssä on tarkoitus tutkia JavaScriptin käyttöä selainpohjaisen käyttöliittymän toteutuksessa. Työn käytännön osuutena on Rosendahl Digital Networks Oy:lle (RDN) toteutettu ikkunoitu Ajax-portaali. Käytännön osuus sai alkunsa tarpeesta saada helposti käytettävä web-pohjainen käyttöliittymäratkaisu, jota voidaan hyödyntää yrityksen eri tuotteissa ja sisäisessä käytössä. Tavoitteena tässä työssä on käydä läpi tärkeimmät modernissa web-selainohjelmoinnissa käytettävät tekniikat ja valaista niitä portaalin tarjoaman käytännön esimerkin avulla.

Ajax-portaali on toteutettu työaikana RDN:llä yhdessä muiden yritykselle tekemieni ohjelmointitöiden kanssa. RDN on vuonna 2005 perustettu, Hollolassa sijaitseva yhteensä noin 30 hengen ohjelmisto- ja media-alan yritys. Yritys tuottaa ratkaisuja vaatetusteollisuudelle, tarjoten tuotteita ja palveluita alan yrityksille, jälleenmyyjille ja maahantuojille. Yritys jakautuu kahteen osastoon: RDN Software ja RDN Media. RDN Software on erikoistunut sovelluskehitykseen ja RDN Media keskittyy kuvan ja videon tuottamiseen sekä graafiseen suunnitteluun.

Työssä keskitytään moderneissa web-selaimissa natiivisti toimiviin tekniikoihin (HTML, JavaScript ja CSS). Työssä ei käsitellä vektorigrafiikkaan pohjautuvia teknologioita, kuten Adobe Flash tai SVG, eikä Java appletteja. Työssä ei

myöskään syvällisemmin tutkita palvelin- tai tietokantatekniikoita, vaan keskitytään selainpohjaisiin ratkaisuihin.

Seuraavassa luvussa (luku 2) tarkastellaan lyhyesti internetin historiallista kehitystä, ja sitä millainen internet on ohjelmointiympäristonä. Luvussa 3 perehdytään niihin yleisiin tekniikoihin, joita JavaScript-pohjaisen asiakassovelluksen toteutuksessa tarvitaan. Luvussa 4 tutkitaan mm. niitä työkaluja, joita tällä hetkellä on tarjolla Ajax-kehittäjille testausta ja debuggausta varten. Työn käytännön osuutena on RDN:lle toteutettu ikkunoitu Ajax-portaali, jota käsitellään luvussa 5. Lopuksi tarkastellaan vielä web-sovelluksiin liittyviä tietoturva- ja suorituskykynäkökulmia (luku 6).

2 WWW OHJELMOINTIYMPÄRISTÖNÄ

2.1 Internetin kehitys ja web-sovellukset

Internetin kehityksen alkuvaiheessa web-palvelimilta ladattiin lähinnä staattisia HMTL-dokumentteja, jotka eivät juuri sisältäneet toiminnallisuutta. Internetin laajetessa ja kehittyessä sivustojen dynaamisuutta lisättiin palvelinohjelmoinnin avulla. Web-selaimelta lähetettiin HTTP-kysely palvelimella olevalle skriptille prosessoitavaksi. Tuloksena saatiin uusi, lähetettyjen kyselyparametrien perusteella dynaamisesti generoitu HTML-sivu.

Standardin mukainen protokolla staattisten HTML-sivujen välittämiseen on HTTP. Data, joka lähetetään HTTP protokollalla, voidaan generoida dynaamisesti palvelimella. HTTP ja HTML ovat webin perusteknologioita. HTML on merkkauskieli, joka kuvaa dokumentin rakenteen ja sisällön. HTML koostuu staattisesta tekstistä ja kuvista. Sitä ei ole alunperin suunniteltu monimutkaisten käyttöliittymien ja vuorovaikutteisten sovellusten rakentamista varten. Jos käyttäjä haluaa uuden HTML sivun HTTP:n kautta, täytyy selaimeen ladata

kokonaan uusi sivu jostain tunnetusta URL-osoitteesta.

Palvelinteknologiat mahdollistavat huomattavasti monimutkaisempia web-sovelluksia kuin staattisten HTML sivujen tapauksessa. Palvelimella voidaan suorittaa vaativia laskutoimituksia, soveltaa olio-ohjelmointia, käyttää hyväksi tietokantoja ja paljon muuta. Web-palvelimien tarjoamat palvelut ovat saaneet aikaan www-vallankumouksen ja ovat suuri syy siihen, miksi internet on niin suosittu nykyisin (Darie & Brinzarea 2006, 11).

Palvelinteknologioita on olemassa useita. PHP on eräs suosittu palvelinteknologia. Sen kilpailijoita ovat mm. ASP.NET ja Java Server Pages (JSP). Palvelinpuolella käytetään yleisesti tietokantapalvelimia kuten Microsoft SQL Server ja MySQL.

Web-sovellukset eivät perinteisesti ole yltäneet käytettävyydeltään samalle tasolle kuin työasemaohjelmat. Toiminnot, kuten komponenttien raahaaminen ja pudottaminen ja rinnakkaiset toiminnot samassa ikkunassa, ovat olleet jo pitkään käytössä graafisissa työasemaohjelmissa. (Darie & Brinzarea 2006, 9.)

HTML sivut voivat sisältää muutakin kuin pelkkää HTML:ää, jolloin sivustoista saadaan dynaamisemmat kuin staattisen HTML:n tapauksessa. HTML sivujen käytettävyyttä ja interaktiivisuutta on pyritty parantamaan dynaamisilla selaimessa suoritettavilla teknologioilla. Tunnetuimpia näistä ovat JavaScript, Java Appletit ja Adobe Flash.

JavaScript luokitellaan skriptikieliin, joita käytetään erityisesti erilaisten sovellusten toimintojen laajentamiseen. Skriptin pituus voi vaihdella yhdestä rivistä aina täydelliseen sovellukseen saakka. Kaikissa tapauksissa JavaScript suoritetaan aina selaimessa tai muussa JavaScript-tulkin sisältävässä sovelluksessa. Uutena suosittuna interaktiivisuutta lisäävänä JavaScript-tekniikkana on AJAX (Asynkronisen JavaScriptin ja XML:n yhdistelmä).

Java appletit kirjoitetaan Java-ohjelmointikielellä ja ne suoritetaan Javan virtuaali-

koneessa, joka täytyy siis olla asennettuna koneelle. Java applettien avulla voidaan suorittaa vaativampaakin laskentaa, mutta ne ovat menettäneet sitä suosioita, joka niillä aikaisemmin oli. Tämä johtuu osittain applettien liiallisesta raskaudesta web-käyttöön, jonka takia esimerkiksi latausajat venyvät pitkiksi.

Adobe Flash on tehokas teknologia esittämään vektorigrafiikalla toteutettua animaatiota. Flashin käyttö edellyttää, että Adope FlashPlayer plugin on asennettuna. FlashPlayerin penetraation internet-työasema markkinoilla oli yli 98% vuonna 2007 (Adobe.com, Flash Player Penetration 2007). Animaatiota voidaan streamata eli katsoa ennenkuin koko tiedosto on latautunut ja Flashin avulla voidaan lisäksi toteuttaa vuorovaikutteisia käyttöliittymiä ja pelejä.

Web-sovellukset tarjoavat tiettyjä teknologisia etuja työasemaohjelmiin verrattuna. Darie ja Brinzarea (2006, 9) luettelevat tärkeimpinä hyötyinä seuraavat:

- Sovellusten jakelu asiakkaalle on helppoa ja halpaa. Web-sovellusten avulla ohjelmiston asennuskustannuksia voidaan radikaalisti alentaa. Tarvitaan vain tietokone, johon on asennettuna toimiva web-selain sekä internet yhteys.
- Sovellusten päivittäminen on helppoa ja taloudellista. Ohjelmistojen ylläpitokustannukset ovat aina olleet merkittäviä. Sovelluksen päivittäminen on pahimmillaan yhtä työlästä kuin uuden ohjelman asentaminen. Web-sovellusten tapauksessa riittää, että palvelimella oleva sivusto päivitetään, näin kaikki saavat käyttöönsä uusimman version.
- Loppukäyttäjän tietokoneelle ei aseta tiukkoja vaatimuksia. Koska hyvin tehty web-sovellus toimii samalla tavalla niin Mac, Windows tai Linux käyttöjärjestelmässä, ei loppukäyttäjä ole sidottuna tiettyyn teknologiaan. Riittää, että käytössä on jokin moderni selain, kuten Internet Explorer, Mozilla Firefox, Opera tai Safari.
- Tietokannat on helppo keskittää. Jokaiselle työasemalle ei tarvitse

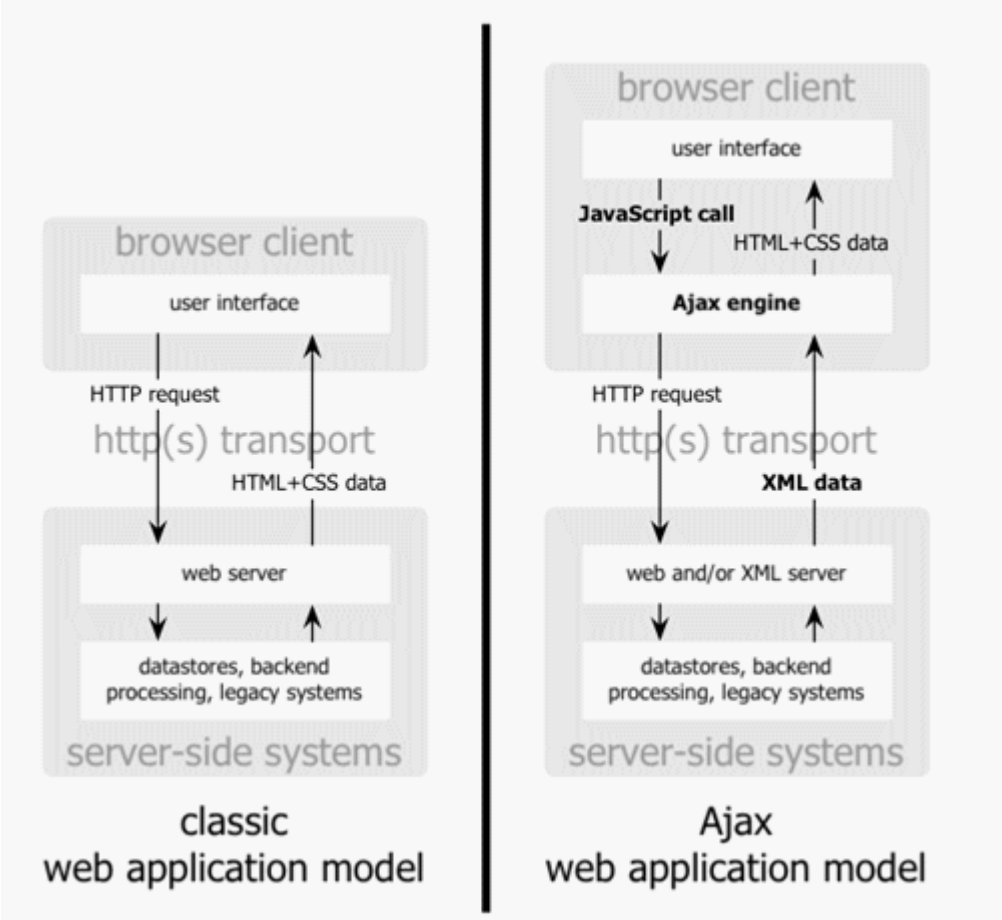
asentaa omaa tietokantaansa vaan kaikki tieto voidaan tallentaa samaan paikkaan. Tämä helpottaa tiedon synkronointia ja tietoturvan aikaansaamista.

- Pääsy kaikkialta maailmasta. Web-sovellusta voidaan käyttää missä päin maailmaan tahansa kunhan käytössä on riittävän nopea internetyhteys ja web-selain. Tämä on hyödyllistä esimerkiksi kansainväliselle yritykselle, jolla on toimipisteitä eri puolilla maapalloa.

2.2 Mitä on Ajax?

Ajax (Asynchronous JavaScript and XML) on kokoelma tekniikoita, joihin kuuluu HTML, CSS (Cascading Style Sheets), JavaScript, DOM (Document Object Model) ja XML. Ajax yhdistää siis useita jo olemassa olevia teknologioita. HMTL-merkkauskieltä ja CSS-tyylisivuja käytetään informaation esittämiseen ja muotoiluun selaimessa. JavaScript-koodi toimii sovelluksen ytimenä luoden dynaamisuutta käyttölittymään ja se kommunikoi palvelimen kanssa. DOMia käytetään HMTL/XML tiedostojen manipulointiin ja lukemiseen. XML on formaatti, jossa tietoa välitetään palvelimen ja selaimen välillä. Palvelimen ei tarvitse enää generoida HTML-koodia, vaan pelkkä datan käsittely riittää. (McLaughlin 2005.)

Ajax sovelluksessa selaimeen ladataan kertaalleen JavaScript-intesiivinen, joka pysyy käyttäjällä koko sovelluksen ajan, vaikka sen ulkoasu saattaakin muuttua huomattavasti. Koska sama dokumentti pysyy selaimessa koko käyttäjän istunnon ajan, voidaan sovelluksen tila tallentaa. Ajax sovelluksessa sivun ensimmäinen lataus kestää pitempään, koska koko JavaScript-moottori ladataan silloin selaimeen. Tämän jälkeen kommunikointi serverin kanssa on kuitenkin huomattavasti tehokkaampaa ja kuluttaa vähemmän tiedonsiirtokapasiteettia. Ajax mallin mukaisen web-sovelluksen eroja perinteiseen malliin havainnollistetaan alla (KUVIO 1). Yksi suuri muutos siirryttäessä perinteisestä mallista Ajax malliin on se, että käyttöliittymälogiikka erotetaan selvemmin palvelinlogiikasta.



KUVIO 1. Perinteinen malli web-sovelluksille verrattuna Ajax malliin (Garrett, 2005)

Perinteisessä web-sovelluksessa käyttäjävuorovaikutus koostuu yleensä hyperlinkkien painamisesta ja tekstilomakkeiden täyttämisestä. Ajaxin avulla voidaan hallita kehittyneempiä käyttöliittymäkomponentteja kuten drag-and-drop, puut ja dynaamiset gridit. Näin selaimen ja palvelimen yhteistyö muuttuu sujuvammaksi ja huomaamattomammaksi, kun toimintasekvenssiin ei tule katkoksia. Ajaxin avulla yhteys palvelimeen voidaan muodostaa esimerkiksi hiiren liikkeistä, raahauksesta tai napin painalluksesta, jolloin yhteistyö sovelluksen kanssa on saumattomampaa. Pyrkimyksenä on saada aikaan web-sovelluksia, jotka ovat ominaisuuksiltaan yhä lähempänä työasemaohjelmia.

Klassinen web-sovellus päivittää sivun sisällön kokonaan uudestaan siirryttäessä tilasta toiseen. Esimerkiksi web-kaupassa ostoskorin sisällön muuttuessa tarvitsi oikeastaan päivittää vain ostoskorin hinta. Sivun navigaatio, bannerit, tyylit ja kuvat pysyvät kutakuinkin samoina tilojen välillä. Ajax-sovelluksessa palvelin voi palauttaa selaimelle takaisin vain olennaisen datan. Tietoa selaimelle voidaan lähettää eri formaattien välityksellä. Vaihtoehtoina ovat mm. JSON (JavaScript Object Notation), puhdas teksti tai XML. Ajax-teknologian avulla oleellinen data voidaan erottaa selvemmin sovelluksen muusta sisällöstä.

Ajax sovelluksen koodaaminen on huomattavasti vaativampaa kuin pelkkien yksittäisten JavaScript-efektien luominen. Ajax sovelluksen täytyy pysyä toimintakykyisenä useita tunteja suurenkin kuormituksen alla. Hyvät koodaustavat ja ohjelmistosuunnittelu ovat tärkeitä Ajax-sovelluksen kehittämisessä. (Crane, Pascarello & James 2006, 23.)

2.3 Esimerkkejä Ajax-sovelluksista

Ajax-tekniikkaa käytetään useissa suosituksi tulleissa web-sovelluksissa. Google on toiminut edelläkävijä Ajax-sovellusten tuomisessa suuren yleisön käyttöön. Yksi suosituimpia Googlen Ajax-pohjaisia palveluita on sähköpostiohjelma GMail, jonka beta versio julkaistiin jo vuonna 2004. Yhtenä palvelun suosion syynä oli ilmaiseksi tarjottava iso postilaatikon koko, mutta GMailin käyttöliittymä oli myös kehittyneempi kuin monissa klassisissa webmail-sovelluksissa (kts. KUVIO 2). GMailin avulla käyttäjä voi avata useita sähkopostiviestejä rinnakkain ja saapuneiden sähkopostiviestien listaa päivitettään automaattisesti, myös silloin kun käyttäjä on kirjoittamassa omaa viestiään. Gmailin avulla käyttäja voi myös suorittaa hakuja omasta sähköposti-historiastaan. Käytetty haku voi näyttää esimerkiksi seuraavalta: "to:markus has:attachment before:2008/01/01". Google on julkaissut myös muita vuorovaikutteisia palveluita kuten Google Suggest, joka täydentää automaattisesti syötettyjä hakuehtoja, sekä Google Maps, jonka avulla voi hakea kartalta

maantieteellisiä kohteita hakuehtojen perusteella.

KUVIO 2. Gmailin käyttöliittymä

3 JAVASCRIPT-TEKNIIKAT

3.1 JavaScript ohjelmointikielenä

JavaScript on tulkattava kieli, kuten kaikki muutkin skriptikielet. Sitä vastoin useimmat sovellusohjelmointikielet, kuten C ja C++, käännetään ennen suorittamista. JavaScript-tulkin voi upottaa mihin tahansa isäntäsovellukseen, kuten web-selaimeen tai web-palvelimeen. Tulkattavuus tuo mukanaan helpon ja nopean ohjelmistokehityksen. Kun JavaScript-ohjelmaa tulkataan, se suoritetaan rivi riviltä. Tulkattavuudessa on käännettyyn kieleen verrattuna sen etu, että JavaScript-ohjelmaa voi muokata yhtä helposti kuin tavallista HTML-dokumenttia. Muutokset tulevat voimaan ladattaessa dokumentti uudestaan selaimeen. (Wikipedia, JavaScript.)

JavaScriptissä merkkijonot, numerot ja funktiot ovat olioita. JavaScript perustuu prototyyppien käytölle, sillä kielessä ei ole luokkia, vaan olioiden luonnissa

voidaan käyttää toisia olioita niiden prototyyppeinä. Periytyminen toteutetaan perimällä ominaisuuksia toisista olioista, prototyyppiolioista. (Peltomäki & Nykänen 2006, 23.)

JavaScript muistuttaa syntaksiltaan C-perheeseen kuuluvia kieliä, joten esimerkiksi for-silmukka voidaan kirjoittaa seuraavalla C-ohjelmoinnista tutulla tavalla:

```
for (i = 0; i < data.length; i++) {
    if (data[i] == 3)
        return i;
}
```

Yksi JavaScriptin suurista vahvuuksista on sen mahdollistama funktionaalinen ohjelmointityyli, josta se on saanut vaikutteita mm. Scheme-ohjelmointikielestä (About ECMAScript, 2008). Funktioita voidaan tallentaa muuttujiin, välittää parametreina ja edelleen palauttaa funktioiden arvoina:

```
var averageDamp = function(fn){
          return function(x){
                    return (x+fn(x))/2;
          };
}
```

Yllä olevassa esimerkissä määritellään funktio nimeltä averageDamp, joka ottaa parametrikseen toisen funktion (fn). Kutsumalla funktiota antamalla sille parametriksi toinen funktio, kuten esimerkiksi fn = cos(x), saadaan tuloksena myös funktio, joka on tyyppiä:

```
function result(x){
    return (x + cos(x))/2;
}
```

Oliot muodostavat JavaScript-kielen perustan. Kielessä lähes kaikki arvot ovat olioita, joilla on omia julkisia metodeja:

```
var luku = (6500).toExponential()
var viesti = "Hello World!".toLowerCase()
```

JavaScript on saanut myös vaikutteita Self-ohjelmointi kielestä protyyppeihin perustuvan olio-ohjelmoinnin osalta. Oliot perivät ominaisuuksia prototyyppi-olioilta. Tarkemmin, prototyyppiolio on funktioille kuuluva muuttuja ja olion konstruktori-funktio voi periä ominaisuuksia toiselta oliolta prototyyppinsä kautta. Yksinkertainen luokkamäärittely prototyyppi-olion avulla näyttää seuraavalta:

```
function Henkilo(nimi){
          this.nimi = nimi;
}
Henkilo.prototype = {
          getNimi: function(){
                    return this.nimi;
          },
          setNimi: function(nimi){
                    this.nimi = nimi;
          }
}
```

JavaScript käyttää siis prototyyppeihin pohjautuvaa perintämekanismia, jossa oliot perivät ominaisuutensa suoraan toisilta olioilta. Menetelmä perustuu siihen, että olion konstruktori voi periä metodinsa yhdeltä toiselta oliolta, luoden prototyyppi-olion, josta kaikki uudet oliot rakennetaan (Resig 2006, 39). Javascriptissä jokaisella funktioilla on *prototype* kenttä (JavaScriptissä funktiot ovat itse asiassa Function-tyyppisiä olioita), johon voidaan lisätä haluttu määrä uusia jäseniä. Nämä jäsenet linkitetään automaattisesti olioihin, jotka luodaan kutsumalla konstruktori-funktiota *new*-operaattorilla. Tämän linkityksen avulla voidaan olioihin lisätä vakioita ja metodeja ilman, että itse alkuperäistä olioita tarvitsisi laajentaa. Seuraavan esimerkin avulla pyritään havainnollistamaan sitä, kuinka *prototype*-oliota voidaan käyttää yksinkertaiseen periyttämiseen.

```
// Määrittele konstruktori Henkilo olioille
function Henkilo(nimi){
    this.nimi = nimi;
}
// Lisää metodeja Henkilo olioille
Henkilo.prototype = {
    getNimi: function(){
        return this.nimi;
    },
    setNimi: function(nimi){
        this.nimi = nimi;
    }
}
```

```
// Konstruktori Kayttaja
function Kayttaja(nimi, salasana){
    // Huomaa, että isa-luokan kontruktoria ei voida kutsua
    // suoraan
    this.nimi = nimi;
    this.salasana = salasana;
}

// Kayttaja perii kaikki Henkilon metodit
Kayttaja.prototype = new Henkilo();

// Lisätään käyttäjälle oma metodi
Kayttaja.prototype.getSalasana = function(){
    return this.salasana;
}
```

Käskyllä new Henkilo() luodaan uusi Henkilo-olio käyttäen Henkilo-konstruktoria. Ohjelman osa saa aikaan sen, että uusi Kayttaja-olio saa kaikki metodit, jota Henkilo-oliolla oli, kun suoritettiin käsky new Henkilo(). Yllä olevassa esimerkissä luodut metodit ovat julkisia metodeja ja niitä voidaan uudelleenkäyttää toisten olioiden yhteydessä.

```
// määritellään yksinkertainen olio
var james = {nimi:"James Bond"};

// lisätään olioon metodeja
james.getNimi = Henkilo.prototype.getNimi;

james.slogan = function(){
    alert("My name is " + this.getNimi());
}
```

Olioiden luomiseen JavaScriptilla voidaan käyttää myös aaltosulkeita {...}. Aaltosulkeiden avulla voidaan määritellä olioita suoraan (*inline*) koodissa. Tämä on helppo ja nopea tapa luoda *singleton*eja (olioita, joista tarvitaan vain yksi instanssi):

```
// Määritellään singleton-olio
// (olion kaikki kentät julkisia)
var mika = {
    name: "Mika Vallittu",
    age: 28,
    maxAge: 120,
    helper = function(n, a){
        return "Person: " + n + ", " + a;
    },
    getName: function() {
        return name;
    },
```

```
    setName: function(sName) {
        name = sName;
    },
    getOlder: function(){
        if(age<this.maxAge) age++;
    },
    toString: function(){
        return helper(name, age);
    }
};
```

Module-suunnittelumalli on Douglas Crockfordin esittämä tapa luoda JavaScript-olioita, joilla on myös yksityisiä kenttiä. Menetelmä perustuu siihen, että funktion sisällä *var* avainsanalla määritellyt muuttujat ovat näkyviä vain kyseisessä funktiossa. Funktiota voidaan käyttää moduulina, joka rajaa muuttujia sisäänsä, pois globaalista nimiavaruudesta. Ohjelman modulaarisuutta voidaan parantaa myös käyttämällä nimiavaruuksia:

```
// Määritellään TEST nimiavaruus (tyhjä olio)
TEST = {};

// Määritellään Person-luokka TEST-nimiavaruudessa
// Module-suunnittelumallin avulla
TEST.Person = function (name, age) {
    // yksityiset (private) muuttujat
    var name = name;
    var age = age;
    // yksityiset (private) funktiot
    var helper = function(n, a){
        return "Person: " + n + ", " + a;
    };
    // julkinen (public) näkyvyys
    return {
        // julkinen (public) kenttä
        maxAge: 120,
        // julkisia (public) funktiota
        getName: function() {
            return name;
        },
        setName: function(sName) {
            name = sName;
        },
        getOlder: function(){
            if(age<this.maxAge) age++;
        },
        toString: function(){
            return helper(name, age);
        }
    };
};
```

Yksityiset (private) ja julkiset (public) jäsenet voidaan toteuttaa JavaScriptilla,

koska sisäfunktiolla on aina pääsy ulkofunktionsa parametreihin ja muuttujiin, myös sen jälkeen kun ulkofunktion on palauttanut (return) arvonsa. (Crockford 2001)

Koska jokaisella Person-luokasta luodulla oliolla on omat kopionsa funktioista, ei muistin käyttö yllä olevassa esimerkissä ole optimaalista. Yllä esitetty Module-rakenne onkin tarkoitettu lähinnä singletonien luomiseksi. Singletonin luominen onnistuu seuraavalla tavalla:

```
TEST.munModuuli= function(){
    // yksityiset (private) muuttujat
    var name = "Mika Vallittu";
    var age = 28;
    //...
    return {
        //...
    }
}(); // sulut suorittavat anonyymin funktion ja palauttavat arvon
```

Ideana on siis kutsua anonyymiä funktiota välittömästi, jolloin tuloksena saatu olio tallentuu suoraan muuttujaan `TEST.munModuuli`.

3.2 JavaScript ja CSS

JavaScriptin ja CSS:n yhdistelmä mahdollistaa dynaamisten ja vetovoimaisten käyttöliittymien toteutuksen web-selaimessa. Ohjelmoijat voivat rakentaa näyttäviä käyttöliittymiä käyttäen animaatiota ja valmiita käyttöliittymä-komponentteja. Lopputuloksena käyttäjä voi liikkua nopeammin ja säästää aikaa.

JavaScriptin avulla päästään tyyliominaisuuksiin käsiksi manipuloimalla elementin *style*-kenttää. Esimerkiksi, jos elementin *elem* korkeutta halutaan muuttaa, voidaan käyttää seuraavaa koodia:

```
elem.style.height = "333px";
```

Tyylitietoja käsitellessä saattaa tuloksissa esiintyä tiettyjä epäjohdonmukaisuuksia johtuen selainten välisistä eroavaisuuksista ja siitä, onko tyylin arvo määritelty erillisessä tyylitiedostossa vai elementin *style*-kentässä. Johdonmukaisten tulosten aikaansaamiseksi kannattaa käyttää jotakin JavaScript-kirjastoa (kuten Prototype tai jQuery), joka sisältää omat funktionsa tyylien manipulointia varten.

Dynaamiset HTML-elementit ovat elementtejä, joita manipuloidaan JavaScriptin ja CSS:n avulla. Yleisimpiä käyttäjävuorovaikutuksia voidaan simuloida selaimessa muuttamalla elementtien kolmea keskeistä tyyliominaisuutta: sijainti (*position*), koko (*size*) ja näkyvyys (*visibility*). (Resig 2006, 137.)

Elementtejä voidaan asemoida usealla CSS-standardin määrittelemällä tavalla: staattinen (*static*), suhteellinen (*relative*), absoluuttinen (*absolute*) ja kiinnitetty (*fixed*) asemointi.

Staattinen asemointi noudattaa yksinkertaisesti dokumentin normaalia kulkua (ylhäältä alas; vasemmalta oikealle). Tässä tapauksessa *top* ja *left* -ominaisuuksilla ei ole mitään vaikutusta elementtien asemointiin. Selaimissa elementtien asemointi on oletuksena staattinen. Suhteellinen asemointi on oletukseltaan varsin lähelle sama kuin staattinen asemointi, sillä poikkeuksella, että *top* ja *left* -ominaisuuksien asettaminen aiheuttaa elementin siirtymisen suhteessa elementin alkuperäiseen (staattiseen) sijaintiin nähden.

Absoluuttinen asemointi rikkoo dokumentin normaalin asettelun ja siirtää elementin haluttuun pisteeseen suhteessa ensimmäiseen isä elementtiin, joka ei ole staattisesti asemoitu. Jos isä-elementtiä ei ole olemassa asetetaan elementti suhteessa koko dokumenttiin.

Kiinteä asemointi pitää elementin koko ajan samassa kohdassa selaimen ikkunaan

nähden. Kiinteästi asemoidun elementin paikka ruudulla ei muutu selaimen ikkunaa vieritettäessä.

Erilaisten asemointitapojen käsittely on tärkeää siksi, että ymmärretään perusteet sille, kuinka elementtien sijaintia voidaan ohjelmallisesti manipuloida. Seuraavaksi käsitellään muutamaa funktiota, jotka ovat hyödyllisiä elementtien sijainnin laskemiseksi ja asettamiseksi.

Ensimmäiseksi tarkastellaan funktiota, jonka avulla voidaan saada halutun elementin tietyn tyyliominaisuuden arvo. On hyödyllistä käyttää tähän omaa funktiotaan, koska näin voidaan ottaa huomioon selainten väliset eroavaisuudet tyylin lasketun arvon saamiseksi (Resig 2006, 136).

```
// Hae tietyn elementin (elem) tyylin (name) arvo
function getStyle( elem, name ) {
    // Jos ominaisuus löytyy taulusta style[],
    // se on tuore, vasta asetettu arvo
    if (elem.style[name]){
        return elem.style[name];
    }
    // Muuten etsitään IE:n tavalla
    else if (elem.currentStyle){
        return elem.currentStyle[name];
    }
    // Tai W3C:n tavalla, mikäli löytyy
    else if (document.defaultView &&
                    document.defaultView.getComputedStyle) {
        // Tyylien syntaksi on tässä muotoa 'text-align'
        // eikä muotoa 'textAlign'
        name = name.replace(/([A-Z])/g,"-$1");
        name = name.toLowerCase();

        // Haetaan style-olio ja tyylin arvo
        // (mikäli kyseinen tyyli on olemassa)
        var s = document.defaultView.getComputedStyle(elem,"");
        return s && s.getPropertyValue(name);
    // Käytössä on jokin muu selain
    } else
        return null;
}
```

Tätä funktiota voidaan käyttää, kun aletaan rakentaa toisia funktioita tyyliominaisuuksien manipuloimiseksi. Voidaan esimerkiksi kirjoittaa apufunktion elementin left-attribuutin saamiseksi.

```
function getX(elem){
    // Palautetaan ominaisuuden left laskettu numeerinen arvo
    return document.parseInt( getStyle(elem, "left") );
}
```

Mikäli halutaan laskea elementin absoluuttinen sijainti koko dokumenttiin nähden, voidaan siihen käyttää seuraavaa funktiota.

```
// Etsitään x-koordinaatti koko dokumenttiin nähden
function pageX(elem) {
    var x = 0;

    // Jokaisen isä-elementin offset on lisättävä tulokseen
    while ( elem.offsetParent ) {
        // Lisätään offset nykyiseen arvon
        p += elem.offsetLeft;

        // ja siirrytään edeltävään elementtiin
        elem = elem.offsetParent;
    }

    return x;
}
```

Funktio käyttää hyödyksi kahta DOM-ominaisuutta:

- *offsetParent*: Elementti, jonka suora lapsi nykyinen elementti on (Firefox-selaimessa se viittaa koko dokumentin juureen).
- *offsetLeft*: Etäisyys vaakatasossa suhteessa edelliseen elementtiin.

Eräs visuaalisesti tärkeä elementin tyyliominaisuus on sen näkyvyys (visibility). On olemassa kaksi pääasiallista CSS-tekniikkaa, jolla elementin näkyvyyttä voidaan hallita.

- *visibility*: Ominaisuudelle voidaan antaa kaksi arvoa: 'visible' (oletus) tai 'hidden' (tämä piilottaa elementin). On huomattava, että asetuksella 'hidden' piilotettu elementti ei muuta dokumentin muitten elementtien asemointia, vaan säilyttää dokumentin normaalin etenemisen (normal flow). Tätä voidaan valaista seuraavalla esimerkillä:

  ```
  // Normaali teksti
  Hello world, are you warming?
  // Asetetaan sanalle 'world' visibility: hidden
  Hello      , are you warming?
  ```

- *display*: Tämä ominaisuus on monipuolisempi asetuksiltaan: sille voidaan antaa arvoksi 'inline' (käyttäytyy kuten <b> tai elementti), 'block'

(käyttäytyy kuten <p> tai elementti) tai none (piilottaa elementin). Jos elementti on piilotettu asetuksella 'hidden', käyttäytyy se samalla tavalla kuin, jos kyseinen elementti olisi kokonaan poistettu DOM-puusta. Piilotettu elementti saadaan takaisin näkyviin asettamalla kentän arvoksi uudelleen 'block' (tai 'inline'). Alla on esimerkki ominaisuuden käyttäytymisestä:

```
// Normaali teksti
Hello world, are you warming?
// Asetetaan sanalle 'world' display: none
Hello, are you warming?
```

Allaolevat funktiot ovat hyödyllisiä elementin näkyvyyden vaihtamisessa:

```
// Funktio elementin piilottamiseen
function hide( elem ) {
    // Haetaan display:n nykyinen arvo
    var curDisplay = getStyle( elem, 'display' );

    //  Laitetaan sen arvo muistiin
    if ( curDisplay != 'none' )
        elem.$oldDisplay = curDisplay;

    // Asetetaan display:n arvoksi 'none'
    elem.style.display = 'none';
}

// Funktio elementin näyttämiseen
function show( elem ) {
    // Asetetaan ominaisuudelle display sen vanha arvo tai
    // käytetään arvoa 'block'
    elem.style.display = elem.$oldDisplay || 'block';
}

// Funktio elementin näkyvyyden tilan vaihtamiseen
function toggle( elem ) {
    // Haetaan display:n nykyinen arvo
    var curDisplay = getStyle( elem, 'display' );

    // Piilotetaan tai näytetään elementti
    (curDisplay != 'none') ? hide(elem) : show(elem);
}
```

Kolmas tapa elementin näkyvyyden hallitsemiseksi on läpinäkyvyys (opacity). Läpinäkyvyyden avulla voidaan elementti tehdä esimerkiksi 30% näkyväksi, jolloin sen allaolevat elementit ovat myös nähtävissä. Kaikki selaimet eivät toteuta läpinäkyvyyden asettamista samalla tavalla. Alla esitetään funktio, joka toimii sekä IE:ssä että W3C-mukaisissa selaimissa:

```
// Asetetaan elementin läpinäkyvyys (level 0.0-1.0)
function setOpacity( elem, level ) {
    // Jos filters kenttä löytyy käytetään sitä (IE)
    if ( elem.filters )
        elem.filters.alpha.opacity = 100 * level;

    // Muuten käytetään W3C:n opacity ominaisuutta
    else
        elem.style.opacity = level;
}
```

3.3 DOM ja tapahtumankäsittely

3.3.1 DOM-ohjelmointirajapinta

DOM on HTML- ja XML-dokumenttien ohjelmointirajapinta. Se määrittelee tavan, jolla dokumentin puurakenteeseen päästään käsiksi. Puurakenne lähtee juurisolmusta ja haarautuu useaksi alisolmuksi. DOM-ohjelmistorajapinnan avulla voidaan lukea ja modifioida dokumenttipuun sisältöa. DOM liittää web-dokumentin ohjelmointikieleen, tässä tapauksessa JavaScriptiin, joka voi käyttää DOM-rakennetta DOM-liitoksen kautta. DOM ei ole ohjelmointikieli, mutta ilman sitä JavaScript-kielellä ei ole mitään mallia HTML- tai XML-dokumenteista. DOM on ohjelmointikieliriippumaton rajapinta, joka mahdollistaa HTML- ja XML-dokumenttien sisällön, rakenteen ja tyylin käsittelyn millä tahansa ohjelmointikielellä. Käytännössä dokumenttioliomallin on tarkoitus tarjota mahdollisuus päästä ohjelmoimalla käsiksi sivulla oleviin HTML-elementteihin. (Peltomäki & Nykänen 2006, 83.)

Elementtien etsiminen HTML-dokumentista eroaa XML-dokumentista etsimisestä sikäli, että HTML-dokumentin tapauksessa etsimisessä voidaan käyttää hyväksi CSS-luokkia ja CSS-syntaksia. Yksinkertaisin ja nopein tapa saada viite HTML-dokumentissa olevaan elementtiin on käyttää funktiota document.getElementById(). Funktio ottaa parametrinaan elementin id:n.

```
var content = document.getElementById("content")
```

Prototype kirjastossa on määritelty funktio getElementsByClassName(), joka nimensä mukaisesti hakee kaikki elementit, joilla on annettu CSS-luokka. Kirjastot kuten jQuery vievät tämän vielä pitemmälle, jQueryn avulla elementtejä voidaan hakea CSS 1-3 ja XPath syntaksin avulla.

```
// etsii kaikki <div> elementit, joilla on luokka 'links' ja
// joiden sisällä on <p> elementti
$("div.links[p]")
```

Mikäli kyseessä on puhdas XML-dokumentti, niin navigointiin voidaan silti käyttää kyseistä kirjastoa kunhan kyselyt muotoillaan XPath-syntaksin mukaisesti.(Resig 2006, 94)

Jokainen DOM-elementti voi sisältää tekstiä, muita elementtejä tai tekstin ja elementtien sekoituksen. Tarkastellaan esimerkkinä seuraavaa HTML koodia:

```
<p><b>Terve</b> maailma!</p>
```

Elementtien sisältöön päästään käsiksi pääasiallisesti kahdella menetelmällä. Oletetaan, että muuttuja *bold* on viite <b> elementtiin. Tällöin <b> elementin sisällä olevaan tekstiin päästään käsiksi seuraavasti:

```
// tapa 1
bold.innerHTML

//tapa 2
bold.firstChild.nodeValue
```

Ensimmäisessä tavassa innerHTML kenttä tarjoaa yksinkertaisen ja tehokkaan tavan päästä kiinni elementin sisältöön. Kenttä innerHTML palauttaa elementin sisällön merkkijonona. Tapauksessa, jossa `paragraph` on viite elementtiin <p>, saadaan seuraava merkkinojo:

```
// Palauttaa "<b>Terve</b> maailma!"
content = paragraph.innerHTML
```

Tällainen HTML-elementtien sisältöön kiinni pääseminen mahdollistaa esimerkiksi dynaamisen HTML-tekstieditorin toteuttamisen web-selaimessa. Uuden HTML koodin lisääminen dokumenttiin on myös yksinkertaista

innerHTML:n avulla:

```
// Saa aikaan <p><b><i>Moi</i></b> maailma!</p>
bold.innerHTML = "<i>Moi</i>";
```

Eräs innerHTML:n ongelma on se, että se tuhoaa täysin elementin vanhan sisällön ja korvaa sen kokonaan uudella. Mikäli elementin koko sisältöä ei haluta muuttaa, vaan lisätä jotain sen alkuun tai loppuun, on parempi käyttää DOM-metodeja insertBefore() tai appendChild().

Solmujen poistaminen DOM-puusta on lähes yhtä yleistä kuin niiden lisääminen. Tätä varten on olemassa funktio removeChild(). Esimerkkinä funktion käytöstä kirjoitetaan apufunktio, jolla voi poistaa haluttuja solmuja dokumentista:

```
// Funktio poistaa annetun elementin DOM puusta
function poista(elementti){
    if(elementti) elementti.parentNode.removeChild( elementti );
}
```

3.3.2 JavaScript tapahtumat

Tapahtumankäsittely on keskeinen osa selaimen JavaScript ohjelmointia. Tapahtumat ovat ikään kuin liimaa, jotka yhdistävät tietolähteen ja sen visuaalisen esittämisen dynaamiseksi kokonaisuudeksi. JavaScript käyttää tapahtumankäsittelyssä niin sanottuja asynkronisia callback-funktiota. Näitä funktioita kutsutaan silloin, kun selain tai käyttäjä aiheuttaa jonkin määritellyn tapahtuman. Havainnollistaan JavaScriptillä toteutettua tapahtumankäsittelyä seuraavalla yksinkertaisella esimerkillä.

```
<html>
<body>
    <ul id="lista">
        <li>Kouvola</li>
        <li>Lahti</li>
        <li>Tampere</li>
    </ul>
</body>
</html>
```

Kirjoitetaan nyt funktion, joka saa <li> elementin vaihtamaan taustaväriä kun hiirtä liikutetaan sen yläpuolella.

```
function setListEffect(){
    var lis = document.getElementsByTagName("li");
    for(var i=0;li=lis[i];i++){
        li.onmouseover = function(){
            this.style.background = "yellow";
        }
        li.onmouseout = function(){
            this.style.background = "white";
        }
    }
}
```

On olemassa myös toinen tapa kirjoittaa samanlainen funktio. Seuraava toteutusratkaisu käyttää hyväkseen Event-olion julkisia ominaisuuksia.

```
function setListEffect2(){
    var ul = document.getElementById("lista");
    var getTarget = function(e){
        // tapahtumankäsittelyyn kannattaa lisätä seuraavat
        // rutiinit alkuun IE yhteensopivuutta varten
        e = e || window.event;
        var target = e.target || e.srcElement;
        return target;
    }
    ul.onmouseover = function(e){
        getTarget(e).style.background = "yellow";
    }
    ul.onmouseout = function(e){
        getTarget(e).style.background = "white";
    }
}
```

Ylläolevassa funktiossa tapahtumankäsittelijät on liitetty ainoastaan kokoavaan elementtiin <ul>, jonka sisällä oleviin elementteihin päästään käsiksi hyödyntämällä ns. tapahtuman kuplimista. Event-olion *target* kenttä kertoo, mikä elementti oli tapahtuman alkuperäisenä lähteenä.

Tapahtumankäsittelijöiden liittäminen HTML-elementteihin voi tapahtua useammalla kuin yhdellä tavalla. Perinteinen tapa on liittää funktio suoraan DOM elementin ominaisuudeksi:

```
window.onload = function(){};
```

Tämä on hyvin yksinkertainen ja johdonmukainen tapa liittää elementteihin tapahtumankäsittelijöitä. Menetelmä toimii kaikissa moderneissa web-selaimissa. Haittapuolena menetelmässä on se, että elementtiin voidaa liittää vain yksi käsittelijä-funktio kerrallaan. Ongelmia aiheuttaa myös se, että IE ei välitä event-oliota funktiolle ensimmäisenä argumenttina vaan ainoastaan globaalin window.event-olion kautta.

Toinen tapa tapahtumankäsittelijöiden liittämiseksi elementteihin on käyttää W3C:n standardia. Koodi, jonka avulla uusi tapahtumankäsittelijä liitetään, on yksinkertaisesti funktio nimeltä addEventListener. Funktio on olemassa jokaiselle DOM elementille ja se ottaa parametreikseen kolme arvoa: tapahtuman nimen (esim. "click"), tapahtuman käsittelyn hoitavan funktion ja boolean tyyppisen parametrin, joka määrittelee sen, missä vaiheessa tapahtuma 'siepataan'.

```
window.addEventListener("load", function(){}, false);
```

Menetelmän etuna on mm. se, että elementtiin voidaan liittää useampia tapahtumankäsittelijöita ilman, että myöhemmin liitetty funktio ajaa yli aiemmin liitettyn funktion. Lisäksi *this* avainsana viittaa aina käsiteltävään DOM elementtiin. W3C:n standardissa event-olio välitään funktiolle automaattisesti ensimmäisenä argumenttina.

Internet Explorer ei valitettavasti tue W3C:n standardia, vaan tapahtumankäsittelijät liitetään attachEvent-funktion avulla:

```
window.attachEvent("onload", function(){});
```

Suurin hyöty tässä menetelmässä on se, että elementtiin voidaan liittää useita tapahtumankäsittelijöitä ilman, että ne ajavat yli edellisiä. IE:n tapauksessa *this* avainsana ei viittaa käsiteltävään elementtiin, vaan globaaliin window olioon. Ongelman on lisäksi se, että attachEvent-funktio ei toimi missään muissa selaimissa kuin Internet Explorerissa.

Nyt on käyty läpi kolme pääasiallista tapaa, joilla DOM-elementteihin voidaan lisätä tapahtumankäsittelylogiikkaa. Koska menetelmät poikkeavat hieman toisistaan eivätkä toimi samalla lailla yleisimmissä web-selaimissa, on käytännössä suositeltavaa käyttää tapahtumienkäsittelijöiden liittämiseen jotakin hyväksi havaittua JavaScript-kirjastoa, joka minimoi selainten väliset poikkeavuudet. Suositut JavaScript-kirjastot, kuten Prototype, jQuery ja Ext sisältävät kaikki addEventListener ja removeEventListener-tyyppiset funktiot, jotka abstrahoivat selainten väliset eroavuudet, tarjoten ohjelmoijalle yhtenäisen rajapinnan tapahtumien lisäämiseksi ja poistamiseksi.

3.4 XMLHttpRequest – Asynkroniset HTTP-pyynnöt

Kaikki modernit web-selaimet tukevat dynaamisten HTTP-yhteyksien luomista JavaScriptin avulla. Asynkroninen asiakas-palvelin-kommunikaatio voidaan toteuttaa käyttäen selaimen XMLHttpRequest-oliota. Olio mahdollistaa Ajax-toteutukset, joissa käyttäjä voi esimerkiksi suorittaa taustalla tietokantaoperaation palvelimella ja palauttaa tulokset web-selaimelle nähtäväksi ilman, että koko HTML-sivua tarvitsisi päivittää uudelleen.

Yhteyden avaaminen palvelimelle XMLHttpRequest-olion avulla vaati tietoa siitä, kuinka olio on toteutettu eri selaimissa. Internet Explorerin versiot 5 ja 6 toteuttavat yhteyden ActiveXObjectin avulla. Kaikki muut modernit web-selaimet (Firefox, Opera ja Safari) toteuttavat XMLHttpRequest-olion natiivisti. Seuraavassa kirjoitataan funktion, joka palauttaa XMLHttpRequest-olion myös Internet Explorerissa:

```
var AjaxRequest = function(){
    // Mikäli käytössä on IE
    if(typeof XMLHttpRequest == "undefined"){
        // IE käyttää ActiveXObjectia XMLHttpRequest
        // olion luomiseen
        return new ActiveXObject(
            // IE 5 käyttää eri oliota kuin IE 6
```

```
            navigator.userAgent.indexOf("MSIE 5") >= 0 ?
            "Microsoft.XMLHTTP" : "Msxml2.XMLHTTP"
        );
    } else {
        return new XMLHttpRequest();
    }
}
```

Yhteyden muodostaminen palvelimelle ja GET pyynnön lähettäminen onnistuu nyt seuraavasti:

```
// luodaan XMLHttpRequest olio
var request = AjaxRequest();

// Avataan socket
request.open("GET", "jokin/url.php?id=3", true);

// Luodaan yhteys ja lähetetään tiedot
request.send();
```

Palvelimelle voidaan lähettää myös POST pyyntö, jonka avulla voidaan lähettää dataa missä tahansa formaatissa ilman kokorajoitusta. Seuraavaksi luodaan POST pyyntö, joka lähettää palvelimelle XML-muotoista dataa:

```
// Luodaan request olio
var request = AjaxRequest();

// Avataan asynkroninen POST pyyntö
request.open("POST", "jokin/url.php", true);

//Asetetaan Content-Type header, jotta palvelin tietää, että
//sille ollaan lähettämässä XML-dataa
request.setRequestHeader("Content-Type", "text/xml");

//Varmistetaan, että selain lähettää datan oikean pituuden
if(request.overrideMimeType){
    request.setRequestHeader("Connection", "close");
}

// Lähetetään serialisoitu data
request.send("<henkilot><nimi>Mika</nimi><nimi>Mikko</nimi></henki
lot>");
```

Seuraavaksi katsotaan kuinka JavaScriptin avulla voidaan käsitellä palvelimelta tulevaa HTTP-vastausta. XMLHttpRequest-oliolla on kaksi kenttää responseXML ja responseText. Mikäli palvelimelta palautetaan XML dokumentti, niin responseXML sisältää DOM dokumentin, muussa tapauksessa vastaus on tallennettuna kentässä responseText.

```
// Luodaan request olio
var request = AjaxRequest();

// Avataan asynkroninen POST pyyntö
request.open("POST", "jokin/url.php", true);

// Seurataan, kun dokumentin tila päivittyy
request.onreadystatechange = function(){
    // Odotetaan kunnes kaikki tieto on ladattu
    if(requestreadyState == 4){
        alert(request.responseText);
        // Jos kyseessä XML
        // var xml = request.responseXML;
        // ...
    }
}
```

Käytännön web-selainohjelmoinnissa kannattaa käyttää jotakin valmista Ajax-kirjastoa. Alla on esimerkki siitä, kuinka Prototype kirjastolla voidaan kätevästi suorittaa Ajax-pohjainen GET pyyntö:

```
new Ajax.Request('/jokin/url.php?id=3', {
    method: 'get',
    onSuccess: function(request) {
        alert(request.responseText);
    }
});
```

3.5 Tiedonsiirtoformaatit (XML/HTML/JSON)

XML (Extensible Markup Language) on tekniikkaa rakenteisen tiedon koodaamiseksi tekstitiedostoon. XML muistuttaa HTML:ää siinä, että molemmat ovat merkkauskieliä, joissa käytetään tagien avulla merkittyjä elementtejä, sekä elementteihin liittyviä attribuutteja. XML-kielen tagit eivät ole ennalta määrättyjä, vaan ovat käyttäjän itse määriteltävissä. XML:ään liittyy joukko eri tekniikoita, jotka toimivat yhdessä. Tälläisiä tekniikoita ovat mm. DOM, XML Schema ja XSLT. Oleellisin ero XML:n ja HTML:n välillä on se, että XML on suunniteltu kuvaamaan datan rakennetta, kun taas HTML on suunniteltu datan esittämistä varten web-selaimessa. XML:n avulla voidaan mallintaa, varastoida ja välittää periaatteessa mitä tahansa informaatiota. Kyseessä on tekniseen toteutukseen ja suunnitteluun tarkoitettu tekniikkaa, joka ei näy loppukäyttäjälle välttämättä lainkaan.

XML-dokumentin rakenne voidaan siis merkata tarkoituksen mukaisilla tageillä. Esimerkkinä tästä olkoon dokumentti, johon voidaan tallentaa muistutuksia.

```
<?xml version="1.0"?>
<muistutukset>
    <muistutus id="1">
          <jatetty>1.1.2007</jatetty>
          <otsikko>Pyykki</otsikko>
          <runko>Pyykkivuoro varattu klo 10:30</runko>
    </muistutus>
</muistutukset>
```

JSON on yksinkertainen merkintäkieli, jota on helppo käyttää tiedonsiirtoon JavaScript-ohjelmissa. Se on verkossa käytettävä rakenteisen tiedon kevyt vaihtomuoto myös eri ohjelmien välillä, koska sitä voidaan käyttää myös muilla ohjelmointikielillä kuin JavaScriptillä. JSON on tekstipohjainen, ihmisen luettavissa ja sisältää olioita tai taulukoita. JSON on ohjelmointikielistä riippumaton käyttäen C-kieliperheen (C, C++, Java jne) käytäntöjä. (Wikipedia, JSON.)

JSONia käytetään Ajaxissa XML:ää yksinkertaisempana vaihtoehtona tiedon siirtämiseen palvelimen ja asiakkaan välillä. Monien eri ohjelmointikielien perustyypit ja tietorakenteet voidaan esittää JSON:illa, joten rakenteista tietoa voidaan siirtää näiden välillä. Tiedon JSON muotoon jäsentämiseen ja koodaamiseen onnistuu useimmilla ohjelmointikielillä kuten ActionScript, C, C#, Java, JavaScript, Objective-C, Perl, PHP, Python, Ruby ja Smalltalk. (Introducing JSON.)

Asiakas (web-selain) voi käyttää XMLHttpRequest –pyyntöä JSON olion saamiseksi palvelimelta. Rajoituksena on, että vain samalta isäntäpalvelimelta voidaan vastata pyyntöön mistä näytettävä sivukin saatiin. Esimerkkinä JSON datasta, jossa käytetään sekä olioita että taulukoita:

```
{muistutukset:[{
          jatetty:"1.1.2007",
          otsikko:"Pyykki",
          runko:"Pyykkivuoro varattu klo 10:30"
    },{
          jatetty:"2.1.2007",
          otsikko:"Hammaslääkäri",
          runko:"Hammaslääkäri klo 8:00"
}]}
```

Teoriassa palvelin voi lähettää selaimelle mitä tahansa tiedostomuotoa. XMLHttpRequest käsittelee kuitenkin vain tekstipohjaista dataa. Ajax tekniikan kannalta on olemassa muutama keskeinen tekstipohjainen dataformaatti, joista on esitetty yhteenveto alla:

- XML: Modernit web-selaimet tukevat natiivisti XML-formaattia. Selaimet muodostavat XML-dokumentista automaattisesti DOM-puun, jota voidaan käsitellä JavaScriptilla.

- HTML: Tässä formaatissa lähetetty tieto on rakenteista tekstiä. Suosittu Ajax-tekniikka on lisätä HTML:n osia suoraan dokumenttiin (välttäen datan parsimista tai läpikäyntiä) käyttäen hyväksi innerHTML-ominaisuutta.

- JSON: Serialisoitu JSON data tarjoaa kevyen, mutta tehokkaan vaihtoehdon XML formaatille. Etuna on mm. se, että JavaScript tukee JSON-formaattia natiivisti, joten erillistä ohjelmallista datan parsimista ja läpikäymistä ei tarvitse suorittaa.

- JavaScript: Suoritettavaa JavaScript-koodia voidaan myös välittää tekstinä selaimelle. Menetelmä mahdollistaa monimutkaisiakin sovelluksia, koska selaimella vastaanotettu JavaScript-koodi voidaan haluttaessa dynaamisesti generoida palvelimmalla (code generation).

Seuraavassa käsitellään esimerkkinä XML-dokumentin lukemista Ajax-kutsussa. Kysymys on tilanteesta, jossa palvelimelta haetaan yksinkertainen RSS-syöte. Esimerkissä käytetään hyödyksi Prototype-kirjastoa.

```
<html>
<head>
    <title>RSS Syöte</title>
    <!-- Ladataan Prototype JavaScript-kirjasto -->
    <script src="prototype.js"></script>
    <script>
        // Odotetaan kunnes dokumentti on latautunut
        window.onload = function(){
            // Ladataan RSS syöte Ajaxilla
            new Ajax.Request(
                // RSS-syötteen URL
                "rss.xml", {

                // Haetaan GET metodilla
                method: "get",

                // Tätä funktiota kutsutaan, kun HTTP-pyyntö
                 // on suoritettu
                onSuccess: function( request ) {
                     // Viite RSS XML dokumentin DOM-
                     // esitykseen
                     var rss = request.responseXML;

                     // Lisätään syötteen otsikot elementtiin
                     // <ol>, jolla on id nimeltä "feed"
                     var feed =
                           document.getElementById("feed");

                     // Haetaan otsikot RSS XML dokumentista
                     var titles =
                           rss.getElementsByTagName("title");

                    // Käydään title-elementit läpi
                    for(var i=0;i<titles.length;i++){
                        // Luodaan <li> elementti
                        // sisältämään tiedon
                        var li =
                             document.createElement("li");
                        // Asetetaan sisällöksi otsikon
                         // teksti
                        li.innerHTML =
                             titles[i].firstChild.nodeValue;

                        // Lisätään otsikko DOM:iin, <ol>
                        // elementin sisään
                        feed.appendChild( li );
                    }
                }
            });
        };
    </script>
</head>
<body>
    <h1>Dynaaminen RSS syöte</h1>
    <p>Lue RSS:</p>
    <!-- Tänne RSS syöte lisätään -->
    <ol id="feed"></ol>
</body>
</html>
```

Esimerkistä huomataan se, kuinka selain tekee XML-dokumentin läpikäynnistä helpompaa, tarjoamalla responseXML:n kautta käyttöön valmiiksi parsitun DOM-puun.

HTML:n tapauksessa tilanne on vielä yksinkertaisempi, koska palvelimelta saatu teksti voidaan suoraan lisätä dokumenttiin käyttämällä innerHTML ominaisuutta. Kyseessä on nopea ja helppo tapa päivittää haluttuja osia web-sivusta.

```
<html>
<head>
    <title>Uutisten lataaminen Ajaxilla</title>
    <!-- Ladataan Prototype-kirjasto -->
    <script src="prototype.js"></script>
    <script>
        // Odotetaan kunnes dokumentti on latautunut
        window.onload = function(){
            // Ladataan uutiset sisältävä HTML Ajaxilla
            new Ajax.Request(
                // Uutisten URL
                "news.html", {

                // Haetaan GET metodilla
                method: "get",

                // Tätä funktiota kutsutaan, kun HTTP-pyyntö on
                 // suoritettu
                onSuccess: function( request ) {
                   // Otetaan talteen saatu HTML
                   var html = request.responseText;

                   // Pistetään uutiset tähän div elementtiin
                   var news = document.getElementById("news");

                   // Lisätään saatu HTML dokumenttiin
                   news.innerHTML = html;
                }
            });
        };
    </script>
</head>
<body>
    <h1>HTML uutiset</h1>
    <!-- Uutiset tänne -->
    <div id="news"></div>
</body>
</html>
```

Selaimelta saatu koodi olisi voitu myös generoida dynaamisesti, esimerkiksi kutsumalla skriptiä `news.php?id=3`. Menetelmässä palvelinta voidaan siis käyttää perinteiseen tapaan HTML:n generoimisessa.

Mikäli palvelimelta haetaan suoritettavaa JavaScript-koodia tekstinä, täytyy teksti muistaa evaluoida. Tämä voi tapahtua käyttäen JavaScriptin eval()-funktiota. Funktion avulla voidaan kutsua ohjelmallisesti selaimen JavaScript-tulkkia, joka suorittaa eval()-funktiolle välitetyssä merkkijonossa olevat JavaScript-lauseet. Toinen tapa saada tekstinä palautettu JavaScript suoritettua, on kääriä se <script> tagin sisään ja lisätä saatu tagi dokumenttiin, tällöin selain suorittaa automaattisesti koodin evaluoinnin. Tarkastellaan esimerkkinä tilannetta, jossa palvelimelta halutaan hakea tiedosto rivers.js, joka sisältää seuraavat rivit:

```
var rivers = ['Tornionjoki', 'Oulunjoki', 'Siikajoki'];
for(var i=0;river=rivers[i];i++){
    alert(river);
}
```

Tämän JavaScript-koodin hakeminen selaimella suoritettavaksi onnistuu alla esitettävällä tavalla:

```
<html>
<head>
    <!-- Ladataan Prototype-kirjasto -->
    <script src="prototype.js"></script>
    <script>
        // Odotetaan kunnes dokumentti on latautunut
        window.onload = function(){
            // Ladataan JS Ajaxilla
            new Ajax.Request(
                // JavaScript tiedoston URL
                url: "rivers.js", {

                // Haetaan GET metodilla
                method: "get",

                // Tätä funktiota kutsutaan, kun HTTP-pyyntö
                // on suoritettu
                onSuccess: function( request ) {
                   // Suoritetaan teksti JavaScriptinä
                   eval( request.responseText );
                }
            });
        };
    </script>
</head>
<body></body>
</html>
```

Menetelmän yksi hyöty piilee siinä, että kaikkea Ajax-sovelluksessa tarvittavaa JavaScript-koodia ei tarvitse ladata kerralla selaimeen. Osa koodista voidaan jättää ladattavaksi vain, jos sitä todella tarvitaan (just-in-time).

4 AJAX-TYÖKALUJA

4.1 JavaScript-kirjastot

4.1.1 Yleistä

JavaScript-intensiivisten Ajax-sovellusten yleistyminen on luonut tarpeen kirjastoille, jotka pyrkivät normalisoimaan selainten välisiä eroavaisuuksia sekä tarjoamaan ohjelmoijille valmiit rutiinit mm. tapahtumankäsittelyä, DOM-manipulointia, animaatiota ja Ajax-kutsuja varten. Suosituimmat JavaScript-kirjastot ovat avoimen lähdekoodin projekteja, joiden ympärille on muodostunut oma aktiivinen yhteisönsä ja joista on saatavilla kattava dokumentaatio. JavaScript-kirjastot luovat pohjan, jonka päälle voi rakentaa omia web-sovelluksia ilman, että kaikkea perustoiminnallisuutta ja infrastruktuuria tarvitsee kirjoittaa itse.

Kirjastot eivät poista tarvetta JavaScriptin ja selainohjelmoinnin perusteiden tuntemiselle. Selainten eroavaisuudet ja niiden erilaiset JavaScript-toteutukset johtavat aina tilanteisiin, jossa kehittäjän on itse kyettävä ymmärtämään virhetilanteisiin ja ongelmiin johtaneet syyt. Kuitenkin sopivan kirjaston (tai kirjastojen) käyttö lisää huomattavasti ohjelmoinnin tuottavuutta ja siksi JavaScript-kirjastoihin tutustuminen ja niiden soveltaminen on kannattava investointi. Erilaisia hyviä JavaScript-kirjastoja on olemassa lukuisa määrä, eikä niitä kaikkia voida käydä tässä läpi. Seuraavassa tarkastellaan hiukan lähemmin suosittua Prototype kirjastoa.

4.1.2 Prototype

Prototype on olio-suuntautunut JavaScript-kirjasto, joka on saanut vaikutteita mm. Ruby ohjelmointikielestä. Prototype on varsin elegantti ja helppokäyttöinen kirjasto. Yksi Prototypen yleisimmin käytetyistä funktiosta on operaattori $, jonka avulla voidaan hakea elementtejä niiden id:n mukaan:

```
var element = $('content');
```

Funktio toimii aliaksena DOM-funktiolle document.getElementById(). Funktiolle $ voidaan välittää parametrina joko elementin id tai itse DOM-elementti, ja se palauttaa arvonaan DOM-elementin. Funktio myös laajentaa palauttamansa elementin uusilla Prototypen määrittelemillä DOM-funktioilla. Tämä mahdollistaa sen, että tuloksia voidaan ketjuttaa:

```
$('content').addClassName('panel').hide();
```

Prototype sisältää myös tehokkaan $$ funktion, jonka avulla voidaan valita useita elementtejä perustuen CSS-syntaksiin:

```
// taulukko, joka sisältää kaikki input elementit, jotka ovat
// 'content' elementin sisällä
$$("#content input");
```

Yksi Prototypen keskeisistä suunnitteluratkaisuista on se, että kirjasto laajentaa JavaScriptin natiiveja olioita. Esimerkiksi JavaScriptin omaa String-oliota on laajennettu joukolla uusia funktioita lisäämällä niitä String.prototype kentän kautta.

```
"alussa Jumala loi".capitalize(); // -> "Alussa jumala loi"
```

Prototype laajentaa myös JavaScriptin natiiveja Array-olioita lukuisilla jäsenfunktiolla. Erityisesti hyödyllisiä ovat Prototypen lisäämät iteraatorit, kuten each:

```
["yksi", "kaksi", "kolme"].each(function(luku){
    alert(luku);
});
```

Seuraavassa tarkastelellaan miten Prototype toteuttaa olio-ohjelmoinnin

mekanismeja. Prototype simuloi olio-ohjelmointia kahden funktionsa Class.create() ja Object.extend() avulla (Resig 2006, 46):

```
// Luodaan globaali olio nimeltä 'Class'
var Class = {
    // funktio, joka luo uuden konstruktorin
    create: function() {
        // luodaan anonyymi konstruktori funktio
        return function() {
            // funktio kutsuu omaa initialize metodiaan
            this.initialize.apply(this, arguments);
        }
    }
}

// Lisätään staattinen metodi Object olioon, joka kopio
// ominaisuudet yhdeltä oliolta toiselle
Object.extend = function(destination, source) {
    // Käydään läpi kaikki lähteen ominaisuudet
    for (property in source) {
        // ja lisätään ne kohteen ominaisuuksiksi
        destination[property] = source[property];
    }

    // palautetaan modattu olio
    return destination;
}
```

Funktio Class.create() palauttaa arvonaan anonyymin funktion, jota voidaan käyttää konstruktorina. Tämä funktio kutsuu automaattisesti omaa initialize metodiaan, mikäli sellainen on määritelty. Muussa tapauksessa funktio aiheuttaa poikkeuksen.

Funktio Object.extend() yksinkertaisesti kopio olion kaikki jäsenet toiselle oliolle. Prototype käyttää funktiota simuloimaan (varsin staattista) periytymistä kopioimalla sen avulla uusia jäseniä olioiden prototype kenttään:

```
// Luodaan Person olion, jolla tyhjä konstruktori
var Person = Class.create();

// Kopioidaan seuraavat metodit kohteeseen Person.prototype
Object.extend( Person.prototype, {

    // Tätä metodia kutsutaan heti kun olio luodaan new-
    // operaatiolla
    initialize: function( name ) {
        this.name = name;
    },
```

```
    // Yksinkertainen jäsenfunktio
    getName: function() {
        return this.name;
    }

});

// Luodaan User olio
var User = Class.create();

// User olio perii kaikki Person olion ominaisuudet
User.prototype = Object.extend( new Person(), {

    // Korvataan initialize metodi uudella
    initialize: function( name, password ) {
        this.name = name;
        this.password = password;
    },

    // Lisätään uusi jäsenfunktio
    getPassword: function() {
        return this.password;
    }

});
```

Seuraavassa esimerkissä demonstroidaan sitä, kuinka Prototype itse käyttää Object.extend() funktiota laajentamaan JavaScriptin natiiveja olioita uudella toiminnallisuudella:

```
// Lisätään uusia metodeja String prototype kenttään
Object.extend(String.prototype, {
    // Muuttaa merkkijonon merkeistä koostuvaksi taulukoksi
    toArray: function() {
        return this.split('');
    },

    // Merkkijonon ensimmäinen kirjain isolla muut pienellä
    capitalize: function() {
        return this.charAt(0).toUpperCase() +
                            this.substring(1).toLowerCase();
    }
});
```

4.2 Debuggaus

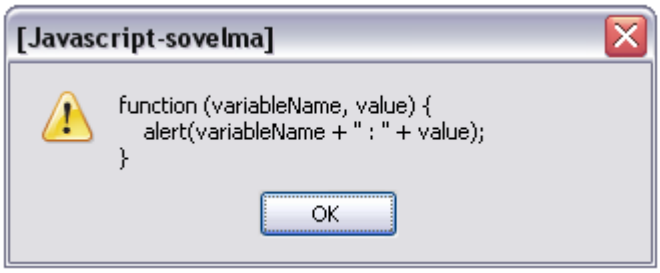
JavaScript sovellusten kehittämisessä huomattava aika kuuluu testaukseen ja debuggaukseen. Tilanteen tekee erityisen haastavaksi se, että eri

selainvalmistajien JavaScript-toteutukset eivät ole identtisiä, jolloin kattavan testauksen ja debuggauksen suorittaminen voi olla hyvin työlästä. Seuraavassa käsitellään muutamaa yleistä menetelmää ja työkalua, jolla virheiden löytämistä koodista voidaan tehostaa.

Yksinkertaisin tapa debugata JavaScript-ohjelmia on käyttää alert() funktiota. Funktiolle voidaan antaa parametrina merkkijonoja tai minkä tahansa muuttujien arvoja. Jos alert() funktiolle annetaan funktio-arvoinen parametri, tulostaa se funktion lähdekoodin. Esimerkiksi koodi

```
var test = function(variableName, value){
    alert(variableName + " : " + value);
}
alert(test);
```

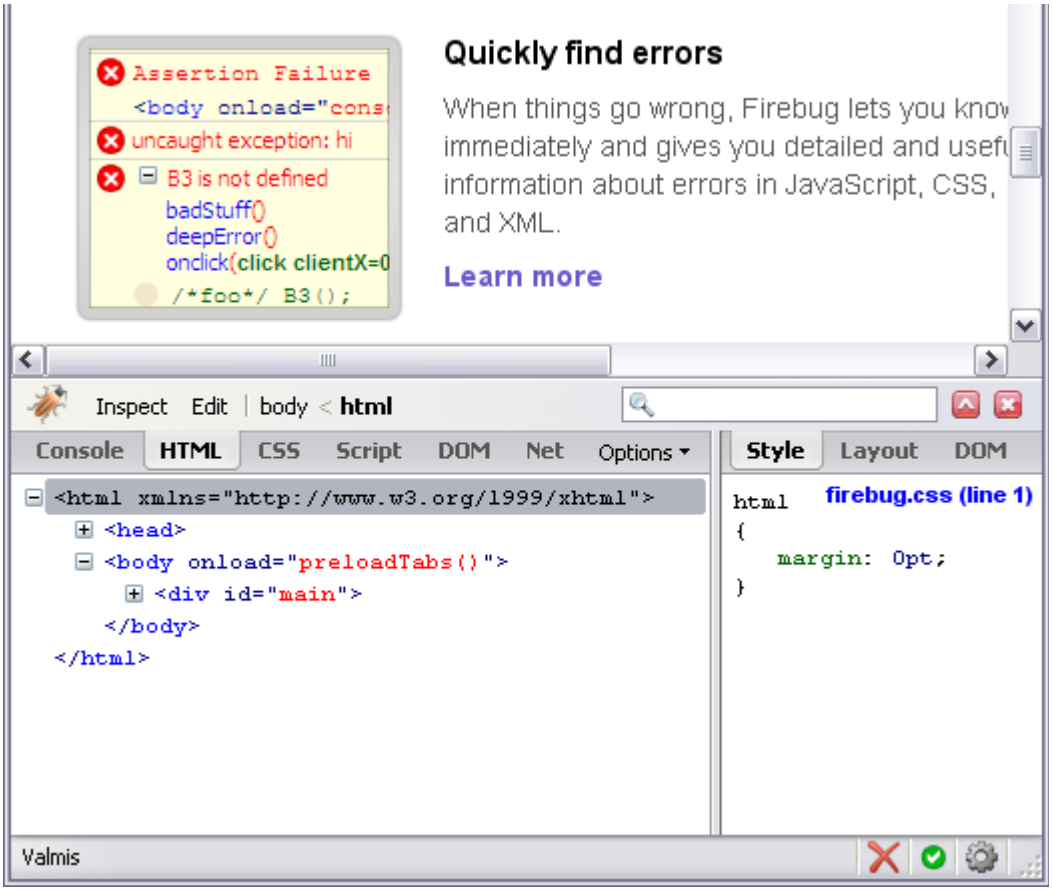
avaa seuraavan näköisen alert-laatikon (KUVIO 3)



KUVIO 3. Alertin käyttö debuggauksessa

Funktion käytössä täytyy huomioida myös se, että alert() pysäyttää ohjelman suorituksen, kunnes alert-laatikko on käyttäjän toimesta suljettu. Tämä saattaa aiheuttaa ongelmia tilanteessa, jossa halutaan tutkia muuttujien arvoja (mahdollisesti päättymättömän) silmukan sisällä. Pahimmillaan voidaan ajautua tilanteeseen, jossa koko selainprosessi on pakko keskeyttää, koska alert-ikkunoiden jatkuva sulkeminen ei ole enää mahdollista tai järkevää.

Huomattavasti kehittyneemmät työkalut JavaScript debuggaukseen tarjoaa Mozilla Firefoxin selainlaajennos Firebug. Laajennus sisältää mm. virhekonsolin, debuggerin, DOM-näkymän, CSS-monitorin ja paljon muuta. Firebugissa voidaan suorittaa JavaScript-koodia konsolilta. Erityisen hyödyllisiä debuggauksessa ovat Firebugin loki-funktiot, kuten console.log(), jolla konsolille voidaan tulostaa informaatiota ilman, että ohjelman suoritus pysähtyy. Firebugin konsolille tulostamat virheviestit sisältävät linkit JavaScript-tiedoston riville, jossa virhe tapahtui. Riveille voidaan asettaa keskeytyskohtia ja ohjelman etenemistä ja muuttujien arvoja voidaan seurata debuggerin avulla siirryttäessä ohjelman riviltä toiselle. Firebug on tällä hetkellä selvästi paras työkalu JavaScript-kehityksessä ja debuggauksessa. Mozillan Firefox selain yhdessä Firebug-laajennoksen kanssa onkin monen web-selainohjelmoijan oletus JavaScript-kehitysympäristönä (KUVIO 4).



KUVIO 4. Kuva Mozilla Firefoxin Firebug laajennoksesta

4.3 Testaus

4.3.1 Yksikkötestauksesta

Testauksen tavoitteena on ylläpitää koodin laatua ja pitää koodissa olevien virheiden määrä mahdollisimman alhaisena. Mitä aikaisemmassa vaiheessa ongelmakohdat löydetään, sitä edullisempaa niiden korjaaminen on. Testauksessa tarkoituksena ei ole pelkästään virheellisen koodin löytäminen, vaan myös sen varmistaminen, että tehdyt korjaukset eivät ole rikkoneet koodin toimintaa muualla.

Yksikkötesti on automatisoitu joukko toimintoja, joilla pyritään varmistamaan tietyn rajoitetun koodin osan (esimerkiksi yksittäinen funktio, luokka, tiedosto, moduuli tai pakkaus) validisuus. Strategiana on jakaa koodi erillisiin moduuleihin ja testata, että ohjelman yksittäiset osat toimivat. Halutessaan testaaja voi kirjoittaa testejä jokaisesta moduulin rajapinnan funktiosta. Testitapaukset muodostavat sopimuksen, jota moduulin täytyy noudattaa. Koodiin tehtyjen korjausten ja muutosten jälkeen testitapaukset ajetaan, jotta selviäisi toimiiko koodi yhä niinkuin on sovittu. Automatisoitu ja itseään tarkistava koodi vähentää manuaalisen debuggauksen tarvetta.

4.3.2 Test.Simple

Yksinkertaisen tavan aloittaa yksikkötestaus JavaScript-ohjelmissa tarjoaa kirjasto Test.Simple. Kirjaston perustana on funktion `ok()`. Funktiolle annetaan parametriksi lauseke, jonka tulee evaluoitua boolean tyyppiseksi arvoksi.

```
ok( foo == bar, description );
ok( foo == bar );
```

Funktio tulostaa joko "ok" tai "not ok", riippuen testin tuloksesta. Mikäli funktiolle annetaan toiseksi parametriksi merkkijono (description), tulostetaan lisäksi vielä tämä merkkijonon sisältämä viesti.

Testitapaukset tulee kirjoittaa elementin sisään, jonka id on "test". Seuraavassa on esimerkki kirjaston käytöstä.

```
<head>
    <script src="Test.Builder.js"></script>
    <script src="Test.Simple.js"></script>
    <script src="Person.js"></script>
</head>
<body>
    <pre id="test">
    <script type="text/javascript">
        plan({ tests: 4 });
        var mika = Person('Mika J. Vallittu', 28);

        ok(mika && typeof mika=='object', 'Konstruktori toimii');
        ok(mika.getName()=='Mika J. Vallittu', 'getName()');

        mika.getOlder();

        ok(mika.getAge()==29, 'getOlder() + getAge()');
        ok(mika.maxAge == 100, 'maksimi-ikä 100 vuotta');
    </script>
    </pre>
</body>
```

Ohjelma tulostaa seuraavan

```
1..4
ok 1 - Konstruktori toimii
ok 2 - getName()
ok 3 - getOlder() + getAge()
not ok 4 - maksimi ikä 100 vuotta
#     Failed test
```

Test.Simple on nimensä mukaisesti mahdollisimman yksinkertainen yksikkötestauksen toteutus. Mikäli halutaan käyttää muitakin funktioita kuin ok(), voidaan siirtyä käyttämään moduulia Test.More, joka sisältää monipuolisempia testifunktiota, kuten is(), isnt(), like(), unlike(), pass() ja fail().

5 CASE: ROSENDAHL DIGITAL NETWORKS OY:N AJAX-PORTAALI

5.1 Vaatimusmäärittely

5.1.1 Portaalin käyttötarkoitus

Monet yritykset ja yhteisöt käyttävät web-portaalia sijoittaakseen kaiken olennaisen tiedon yhteen paikkaan. Portaalin avulla käyttäjän ei tarvitse navigoida selaimella useaan eri kohteeseen. Portaaliin voidaan koota yhteen joukko erilaisia toimintoja ja palveluita kuten webmail, hakukone, muistiinpanot, uutissyötteet, jne. Ajax-teknologian avulla portaalin käytettävyyttä voidaan parantaa; useita eri sivustoja voidaan ladata rinnakkain eikä käyttäjän tarvitse odottaa, että koko sivu päivitetään. Esimerkkejä suosituista portaaleista ovat mm. Googlen henkilökohtainen etusivu ja pageflakes.com.

Pageflakes.com sivusto tarjoaa käyttäjilleen rikkaan valikoiman valmiita komponentteja ('flakes'), joita voidaan sijoitella usealle välilehdelle. Ajax-tekniikkaa hyödynnetään mm. siinä, että uutissyötteet päivittyvät omassa ikkunassaan automaattisesti.

Portaaliratkaisu, jota tullaan käyttämään Rosendahl Digital Networks Oy:lla (RDN) tulee muistuttamaan yllä mainittuja ratkaisuja. Portaali on tarkoitettu usealle käyttäjälle siten, että kukin käyttäjä voi kustomoida oman näkymänsä niin kuin haluaa. Järjestelmä tallentaa käyttäjän asetukset ja muistaa ne seuraavaa käyttökertaa varten, vaikka käyttäjä kirjautuisi eri koneelta. Ikkunoiden liikuttelu on RDN:n portaaliratkaisussa vapaata, lisäksi ikkunoiden kokoa voidaan muuttaa. RDN:n portaalin muistuttaakin varsin paljon ikkunoitua käyttöliittymää.

Portaalin tarkoituksena on toimia sekä itsenäisenä intranet sivustona, että tarjota pohja RDN:n Business-järjestelmän käyttöliittymälle. Business-järjestelmässä tulee olemaan useita moduuleja, joita käyttäjä voi asetella ruudulle haluamallaan tavalla. Lisäksi portaalin ulkoasu on muokattavissa taustakuvan ja ikkunakomponentin osalta. Käyttäjä voi kategorisoida omat näkymänsä valikon avulla siten, että jokainen valikon solmu avaa oman näkymän, jossa on haluttu määrä ikkunoita. Valikon avulla portaaliin voi kategorisoida suuren määrän toiminnallisuutta. Tarkoituksena on luoda joukko valmiita komponentteja, joita käyttäjät voivat lisäillä omiin näkymiinsä. Business-järjestelmässä valittavien komponenttien määrä suodattuu käyttäjän profiilin mukaan.

Käyttäjäprofiilit mahdollistavat sen, että samaa sivustoa voivat käyttää useat erityyppiset toimijat. Esimerkiksi portaalissa voi olla yksi profiili myyjälle, toinen profiili varastomiehelle ja kolmas sovelluskehittäjälle. Kaikilla voi olla pääsy esimerkiksi yrityksen kalenteriin, mutta myyntilukuja, varastosaldoja tai ohjelmistovaatimuksia ei tarvitse näyttää kaikille. Rajoittamalla pääsyä parannetaan yrityksen tietoturvaa, mutta samalla tehdään mahdolliseksi keskitetty pääsyn informaatioon.

Portaalin hallintaa voidaan tehostaa Ajax-tekniikalla siten, että konfigurointi tapahtuu samalla sivulla itse sovelluksen kanssa. Käyttäjän ei tarvitse navigoida kokonan uudelle sivulle, vaan käyttäjä voi napsauttaa käyttöliittymän oliota ja tehdä haluamansa asetukset suoraan samalla sivulla.

5.1.2 Tärkeimmät toiminnot

Portaalin pääominaisuudet ovat käyttäjän id:n mukaan tallentuva profiili, muokattava valikkorakenne ja kustomoitava ikkunakomponentti. Keskeisiä toimintoja ovat ikkunoiden lisäys, poisto ja muokkaus sekä valikkorakenteen määrittely editorin avulla. Lisäksi käyttäjä voi valita portaalin ulkoasun teeman.

Alla on luettelo keskeisistä ominaisuuksista:

- kustomoitava ikkunakomponentti (ikkunoiden asema, koko ja ulkoasun teema muutettavissa + ikkunoiden lisäys, poisto ja muokkaus portaalin käyttöliittymän kautta)
- päävalikko (3-portainen)
- kontekstivalikko
- valikkoeditori (henkilö-/ryhmäkohtaisen valikkorakenteen määrittely)
- käyttäjäprofiilit (käyttäjän id:n mukaan tallentuva)
- kustomoitava ulkoasu (ikkunan + portaalin teemat)
- kieliversiot.

5.2 Käyttöliittymä

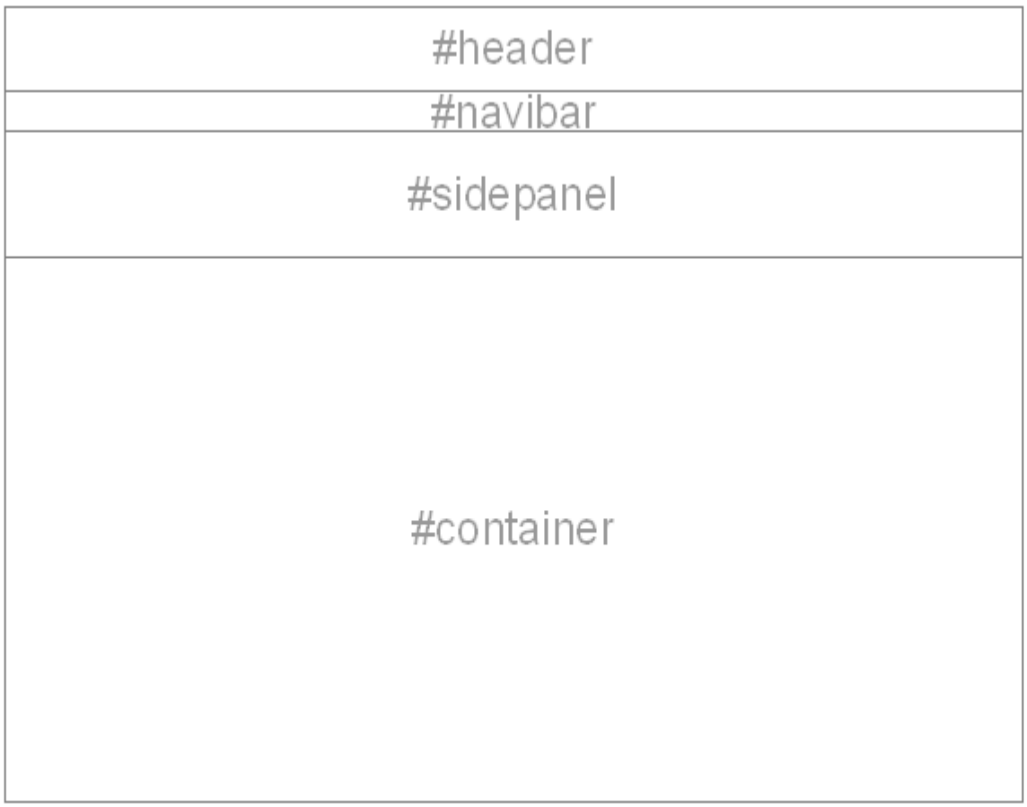
5.2.1 Vaatimukset

Portaalin käyttöliittymästä on tarkoitus tehdä helposti muokattava. Eri asiakkaat saattavat haluta erilaisia ulkoasuja, joten portaaliin tulisi olla mahdollisuus määritellä ulkoasultaan erilaisia teemoja. Käyttöliittymän osalta toivottavia ominaisuuksia ovat etenkin selkeys, käytettävyys, esteettisyys ja ylläpidettävyys.

Ylläpidettävyys on tärkeää sen takia, että voidaan suorittaa mahdollisimman selkeä työnjako graafikoiden ja ohjelmoijien välillä. Tämä täytyy otta huomioon teknisessä toteutuksessa siten, että teemoja voitaisiin vaihtaa vain muutaman tyylitiedoston avulla.

5.2.2 Sivun layout

Portaalin layout on esitetty kuviossa 5. Käyttöliittymän neljä pääaluetta ovat: otsikko (#header), valikko (#navibar), paneeli (#sidepanel) ja sisältö (#container).



KUVIO 5. Portaalin käyttöliittymä (wireframe layout)

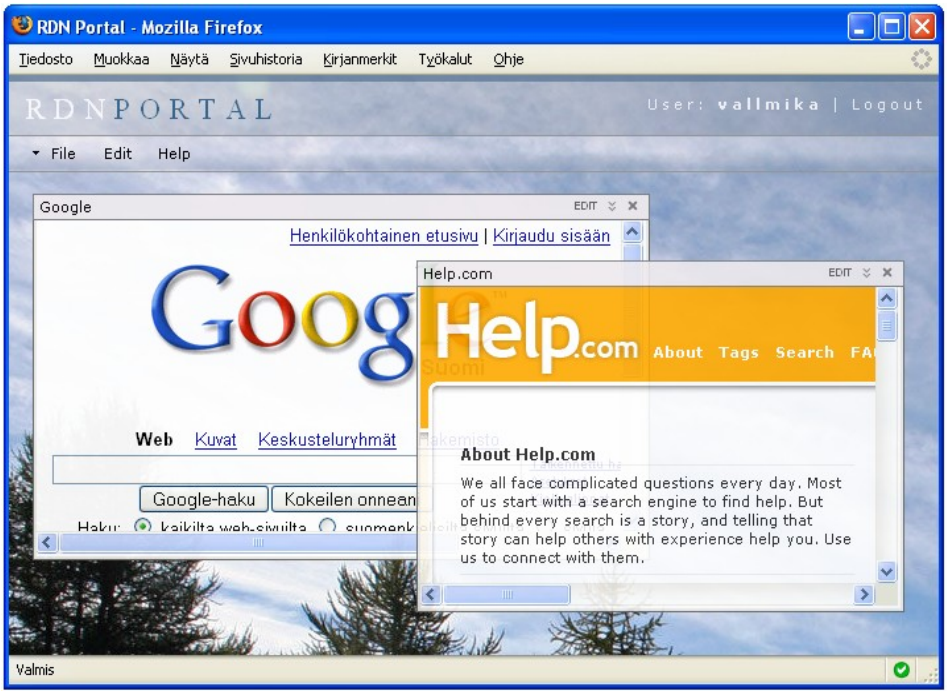
Otsikkoalue sisältää ainakin logon ja pikanäppäimiä. Otsikon ulkoasua voi helposti muuttaa vaihtamalla logoa tai taustakuvaa. Tarvittaessa otsikkoalueen voi piilottaa klikkaamalla. Valikolle on varattu oma vaakasuora alue, jonka sisään valikko luodaan. Paneelialueeseen voidaan myös sijoittaa pikanäppäimiä eri toiminnoille. Sisältöalue on tarkoitettu ikkunakomponenteille työpöytäalueeksi. Sivun layoutin yleiset ominaisuudet määritellään tiedostossa template.css. Yllä oleva layout on pääosin kaikille teemoille sama, pieniä eroja voi olla eri alueiden dimensioissa.

5.2.3 Ajax-käyttöliittymä

Tarkoituksena oli luoda portaalille dynaamisesti päivittyvä ja interaktiivinen käyttöliittymä JavaScriptin ja PHP:n avulla. Käyttöliittymän alueet ovat koodissa div-elementtejä, jotka on sijoiteltu CSS-tyylien avulla. Firebugin HTML-näkymässä sovelluksen pääsivu `index.php` näyttää `body`-osuudessaan seuraavalta:

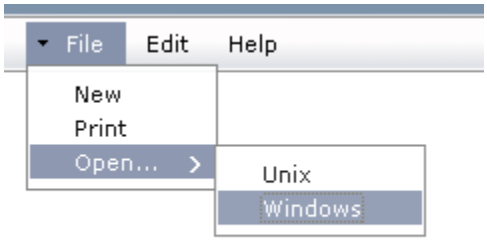
KUVIO 6. Portaalin etusivun HTML-näkymä Firebugissa

Aloitussivu `index.php`:n `<body>`-lohko sisältää `<div>`-elementtejä, joiden `id`:t ovat `header_toggle`, `header`, `navibar`, `sidepanel`, `container` ja `footer`. Sovelluksen varsinainen sisältö ladataan dynaamisesti JavaScriptin avulla pääasiassa `navibar`- ja `container`-elementtien sisään, johon sijoitetaan valikko ja ikkunat. Kuten yllä on nähtävissä (KUVIO 6), niin elementit `sidepanel` ja `footer` eivät ole aktiivisia. Tarvittaessa elementit ovat kuitenkin valmiita käyttöön. Kun layoutiin lisätään grafiikat sekä valikko- ja ikkunakomponentit, näyttää tulos seuraavalta (KUVIO 7).



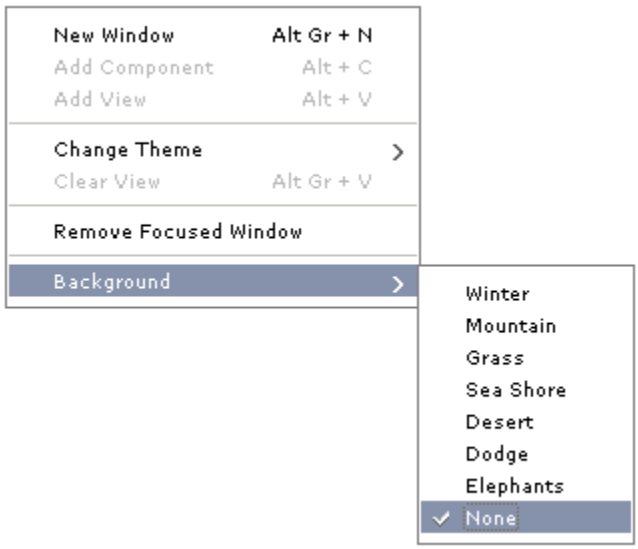
KUVIO 7. Portaalin layout grafiikkoineen

Portaalissa käytettävä valikko on kolmiportainen (KUVIO 8), ja sitä voidaan muokata erilisen menueditorin avulla. Jokainen valikon päättyvä solmu määrittelee oman näkymän, joka voi sisältää mielivaltaisen määrän ikkunoita.



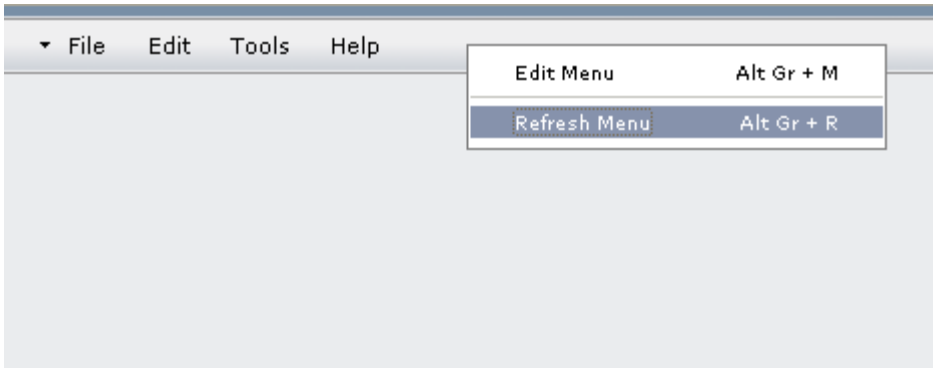
KUVIO 8. Kolmiportainen valikkorakenne

Lisäksi sovellus sisältää kontekstivalikon, joka avautuu hiiren oikeata nappia painamalla (KUVIO 9). Kontekstivalikon avulla voidaan käyttää portaalin omia toimintoja, joita ovat mm. uuden ikkunan luominen, ikkunan poistaminen, ikkunoiden ulkoasun teeman määrääminen ja taustakuvan muuttaminen.



KUVIO 9. Kontekstivalikko

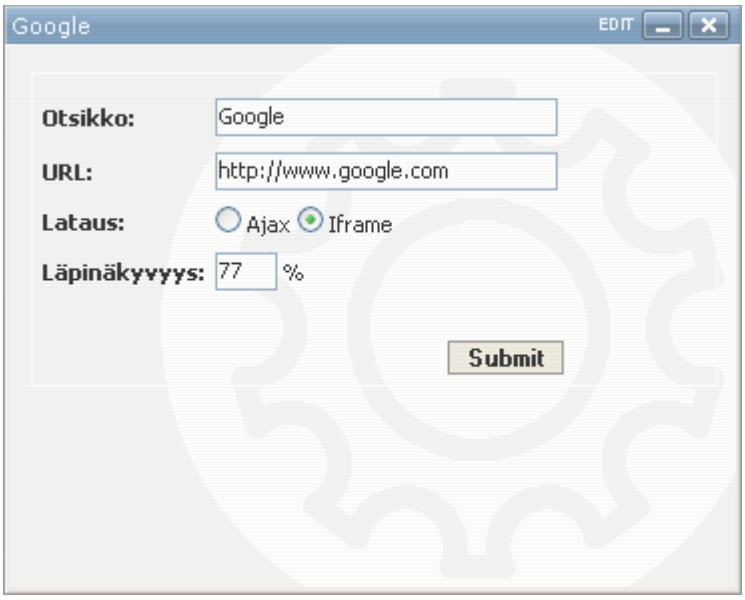
Mikäli hiiren oikeaa näppäintä painetaan navikaatiopalkin päällä saadaan esille seuraavan näköinen kontekstivalikko (KUVIO 10).



KUVIO 10. Kontekstivalikko navigaatiopalkin päällä

5.2.4 Käyttöliittymäkomponentit

Hyödyntämällä valmiita käyttöliittymäkomponentteja säästettiin aikaa ja vaivaa. Useimissa web-selaimissa samalla lailla ja oikein toimivien käyttöliittymäkomponenttien kirjoittaminen JavaScriptillä on varsin aikaavievää ja työlästä.



KUVIO 11. Portaali-ikkuna editointi tilassa

Ikkunan toteutuksessa käytettiin Protype Window –kirjastoa, joka avulla luotiin siirrettävä ja skaalattava ikkunakomponentti (KUVIO 11). Kirjasto koostuu `windows.js` tiedostosta, lisäksi tarvitaan tyylitiedosto(ja) ja kuvia ikkunan ulkoasua varten. Kirjasto on avoimen lähdekoodin projekti ja siihen voitiin tehdä tarvittavat muutokset portaaliin integrointia varten.

Portaalissa käytettävä ikkunakomponentti mahdollistaa käyttäjän määrittelemään ikkunan koon, sijainnin ja ulkoasun. Ikkunakomponenttiin lisättiin myös ominaisuus, jolla ikkunan tila voidaan tallentaa. Käyttäjän muuttaessa ikkunan paikkaa tai kokoa suoritetaan päivitys tietokantaan Ajax-tekniikalla ilman, että käyttäjä huomaa tallennusta tapahtuvan.

Valikkona on käytetty Yahoon YUI-kirjaston menukomponenttia (KUVIO 8). Yahoon YUI-kirjasto on julkaistu BSD-lisenssillä ja on ilmainen kaikkiin käyttötarkoituksiin. Samaa valikkokomponenttia voidaan käyttää myös kontekstivalikkona (context menu), joka aktivoituu kun elementin päällä painetaan hiiren oikeaa painiketta. Yahoon käyttöliittymäkirjasto on varsin monipuolinen kokonaisuus. Valikon käyttöönotto edellyttää, että otetaan käyttöön monia JavaScript-tiedostoja, joista tiedoston `menu.js` toiminta riippuu. Tällaisia

ovat mm. `yahoo.js`, `event.js`, `dom.js` ja `animation.js`.

Myös käyttäjäkohtainen valikkorakenne tallennetaan tietokantaan. Valikkoa voi muokata valikkoeditorin avulla (KUVIO 12), jolla kukin käyttäjä voi luoda profiilinsa mukaisen valikkorakenteen.

KUVIO 12. Valikkoeditori ladattuna ikkunakomponentin sisään

5.2.5 Teemat ja profiilit

Käyttöliittymään on tarkoitus luoda eri teemoja, jotka voidaan määritellä mahdollisimman pitkälle CSS-tyylitiedostojen avulla. Teknisesti teeman määrittelyyn tarvitaan (tällä hetkellä) kolme css-tiedostoa. Ikkunalle vaaditaan oma tyylitiedostonsa, portaalin teemalle omansa. Lisäksi ikkunoiden sisään avattavalle sisällölle tulee tehdä oma css-määrityksensä. Näiden tiedostojen avulla voidaan määritellä sovellukselle yhtenäinen ulkoasu eli teema.

Koko systeemin teema voidaan vaihtaa JavaScriptin avulla, kunhan css-määritykset ovat olemassa. Ikkunoiden tyyli voidaan muutta dynaamisesti käyttämällä funktiota `ikkuna.changeClassName()`. Portaalin teema muutetaan

kutsumalla funktiota `PORTAL.setTheme()`. Funktio vaihtaa css-tiedostot ja -luokat toisiksi. Tämä tehdään dynaamisesti siten, että annetaan linkitetylle css-tiedostolle oma id, jolla siihen voidaan viitata JavaScriptin kautta. Tämän jälkeen voidaan helposti käydä muuttamassa link-elementin href-attribuuttia uuden teeman mukaiseksi. Portaalin yleinen ulkoasu määritellään tyylitiedoston avulla seuraavasti

```
<link id="portal_custom_css" rel="stylesheet" type="text/css"
href="css/theme_sloth.css" />
```

Tyylimäärityksen sisältävään link-elementtiin voidaan viitata id-attribuutin avulla ja sen href-attribuutin arvoa voidaan muuttaa. Ikkunoiden tyylien määrittämiselle on olemassa oma mekanisminsa. Ikkunoille voi luoda uuden teeman tekemällä ikkunaan tarvittavat kuvat (9 kappaletta), asettamalla ne omaan kansioonsa, ja yhdellä tyylitiedostolla. Jokaisella ikkunalla voi periaatteessa olla oma tyylinsä ja ulkoasunsa.

Mikäli ikkunoiden sisältö on ladattu Ajax-tekniikalla, voidaan myös ikkunoiden sisällön tyylejä manipuloida samassa nimiavaruudessa kuin portaalin tyylejäkin. Mikäli sisällön tyylejä halutaan muuttaa dynaamisesti, riittää käydä JavaScriptin avulla muuttamassa tyylitiedostoa.

Esimerkiksi teema nimeltä Sloth voidaan määritellä seuraavien tiedostojen avulla: `theme_sloth.css` (portaalin tyylit), `manage_organizations_sloth.css` (ikkunoiden sisällön tyylit) ja `sloth.css` (ikkunan tyylit). Lisäksi ikkunalle on oma tyylien yhteydessä oleva kuvakansionsa nimeltä `sloth`.

5.3 Ohjelmistoarkkitehtuuri

5.3.1 Suunnitteluratkaisut

Projektin tarkoituksena on rakentaa kustomoitava, usean käyttäjän Ajax-portaali. Projektin toteutukseen tarvitaan palvelinpuolen koodia, asiakaspuolen koodia ja tietokanta. Selaimessa suoritettava JavaScript-koodi on vastuussa käyttöliittymän toiminnasta, kuten ikkunoiden siirtämisestä ja tiedon lähettämisestä takaisin palvelimelle Ajax-tekniikalla. Palvelimella käytetään PHP:ta käsittelemään käyttäjäistunnot, siirtämään dataa takaisin selaimelle ja kommunikoimaan tietokannan kanssa. Portaalin arkkitehtuuri olettaa, että varmistettu käyttäjätunnus lähetetään POST-parametrina aloitussivulle (index.php).

Portaalin kkunoihin voi ladata informaatiota joko iframe-elementin sisään tai suoraan ikkunan div-elementin sisään Ajax-kutsulla. Ikkunakirjaston avulla voidaan luoda ikkunoita, joita voidaan sijoittaa mihin tahansa näytölle sekä liikuttaa, pudottaa, skaalata, avata ja sulkea. Näihin komponentteihin lisätään Ajax-teknologiaa saaden ne vuorovaikuttamaan palvelimen kanssa automaattisesti.

Portaalin JavaScript-koodissa on pyritty mukailemaan MVC-arkkitehtuuria. Portaaliin liittyvä koodia jakautuu kolmeen JavaScript-nimiavaruuteen: GUI, OBS ja XHR. Käyttöliittymän hallitsemiseen tarvittava koodi on sijoitettu nimiavaruuteen GUI. Olio GUI on agregaatti ja sisältää omat oliot eri käyttöliittymä komponenteille: ContextMenu, Footer, Header, Menu, View ja Window. Näiden olioiden avulla voidaan hallita kutakin käyttöliittymän komponenttia. Luokat ContextMenu, Menu ja Window käärivät sisäänsä valmiit avoimen lähdekoodin käyttöliittymäkomponentit.

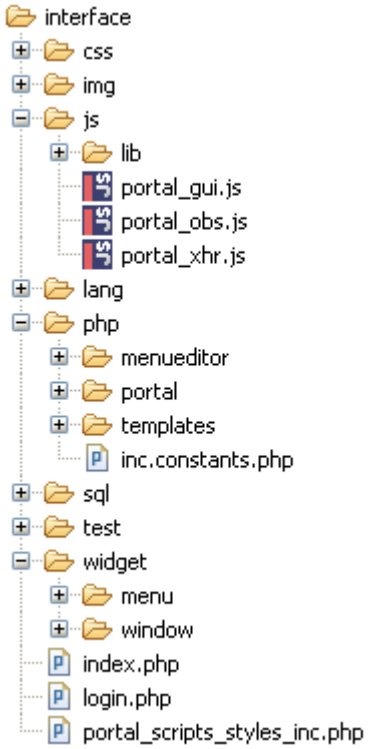
Tapahtumankäsittelijät on kerätty omaan luokkaansa. Tämä tämä ei aina onnistu valikon ja ikkunan (Menu ja Window) osalta, koska ne sisältävät omat sisäiset tapahtumankäsittelymekanisminsa. Tapahtumankäsittelijät on kuitenkin pyritty siirtämään aina pois itse html elementeistä erillisiin JavaScript-tiedostoihin.

Portaalin totetukseksessa on päädytty käyttämään valmiita DHTML- ja JavaScript-komponentteja valikon ja ikkunan osalta. Syynä on sekä ajan säästäminen että selainten välisen yhteensopivuuden takaaminen. Käyttämällä valmiita, testattuja komponentteja tämä onnistuu kivuttomammin.

Kaksi keskeistä tarvittavaa käyttöliittymäkomponenttia ovat valikko ja ikkuna. Valmiiden komponenttien käyttäminen vapauttaa meidät niistä teknisistä haasteista, jotka liittyvät dynaamisten käyttöliittymäkomponenttien toteutukseen niin, että ne toimivat suosituimmilla web-selaimilla. Palvelinpuolen tekniikkana käytetty PHP sopii tarkoitukseensa hyvin. Portaalin tarvitsema palvelinpuolen toiminnallisuus on varsin yksinkertaista, joten sen toteuttamiseen ei tarvita raskasta arkkitehtuuria. PHP:n tehtävänä on lähinnä toimia rajapintana tietokantaan.

5.3.2 Projektin rakenne

Portaalin staattinen rakenne tiedostojen osalta selviää kuviosta 13.



KUVIO 13. Projektin kansiorakenne

Kansio css sisältää tyylitiedostoja. Kansiosta löytyy tiedosto template.css, joka määrittelee pääsivun (index.php) yleiset tyyliasetukset sekä lisäksi 'theme_' -alkuisia teemakohtaisia tyylitiedostoja. Tyylitiedosto reset.css normalisoi HTML elementtien esittämisen selaimissa, esimerkiksi se asettaa kaikki marginaalit ja reunat nollaksi sekä fontit normaaliin oletuskokoonsa. Tällä pyritään minimoimaan selainten väliset eroavaisuudet elementtien oletustyyleissä. Varsinainen portaalin layout ja ulkonäkö määritellään tiedostossa template.css. Portaalin kuvat on tallennettu kansioon img, joka sisältää lähinnä etusivun taustakuvat, gradientit ja joitakin ikoneja. Portaalin yleisistä toiminnoista vastaavat JavaScript-tiedostot on sijoitettu kansioon js. Näitä selainpuolen toimintoja ovat käyttöliittymän ohjelmointi (portal_gui.js), tapahtuman käsittelijät

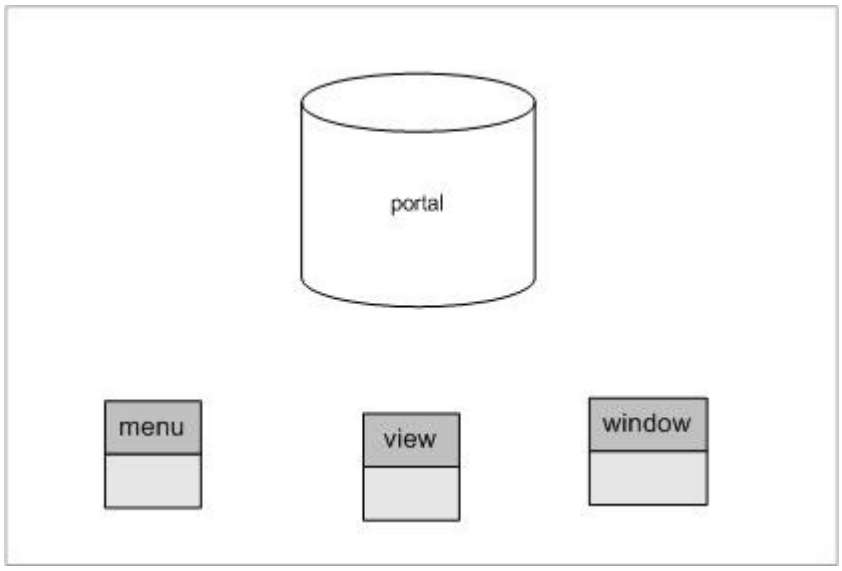
(portal_obs.js) ja Ajax-kommunikaatio palvelimen kanssa (portal_xhr.js).

Palvelinpuolen koodit on sijoitettu kansioon php. Se sisältää skriptit, joiden avulla voidaan hallita portaalin valikkoa (menueditor), skriptit jotka ovat vastuussa portaalin ikkunoiden sijainnin ja koon tallentamisesti tietokantaan (portal), sekä ne php-tiedostot, jotka toimivat templateina (templates). Kansion php juuressa sijaitsee keskeinen tiedosto inc.constants.php, jossa määritellään tietokantayhteyden muodostamiseen tarvittavat parametrit. Ilman tätä tiedostoa sovellus ei käynnisty oikein. Kansio sql sisältää vain sql-dumpin tietokannasta. Valmiit, projektissa hyödynnettävät JavaScript-käyttöliittymäkomponentit on sijoitettu kansioon widget. Tärkeimpiä komponentteja ovat valikko- (menu) ja ikkunakomponentti (window). Portaalin pohjana toimii juuressa sijaitseva tiedosto index.php. Se sisältää HTML koodin, joka määrittelee portaalin perusrakenteen. Sivulle index.php generoidaan sitten latauksen jälkeen dynaamisesti sisältöä pääasiassa JavaScriptin, mutta myös PHP:n avulla.

Sivulle index.php saavutaan sivun login.php kautta, joka välittää index.php:lle userid-parametrin jonka avulla sivun näkymä personoidaan. Personointi tarkoittaa sitä, että sivun valikkorakenne on käyttäjäkohtainen. Kaikki toiminnallisuus – sisäänkirjautumista lukuunottamatta – toteutetaan index.php sivulla JavaScriptin avulla, joten sivun index.php lataaminen lataa itse sovelluksen.

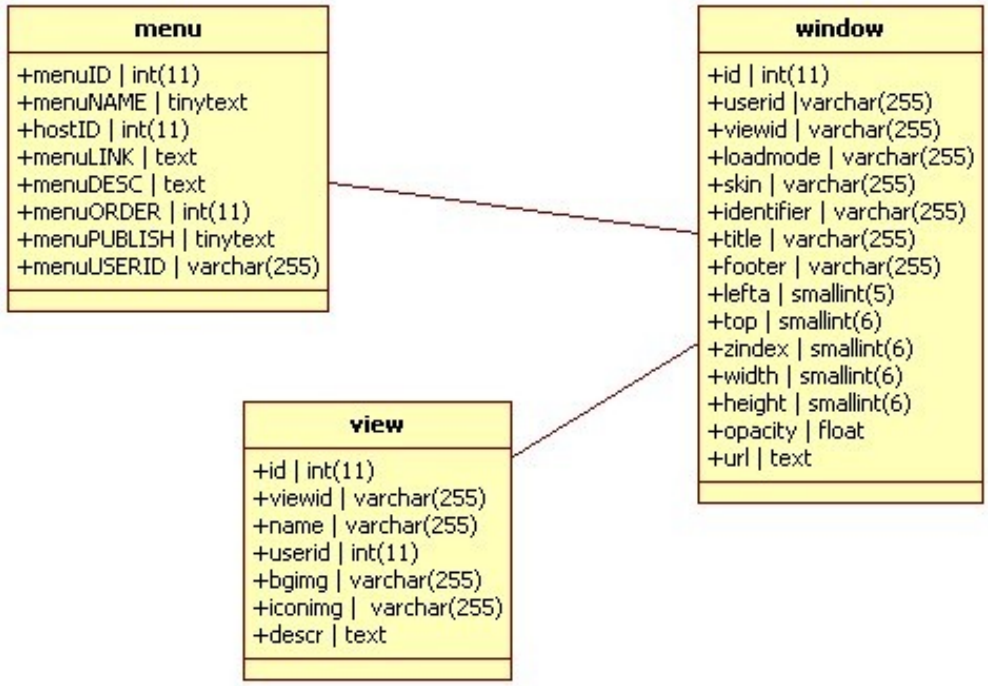
5.3.3 Tietokantarakenne

Seuraavassa tarkastellaan lähemmin portaalin tietokannan rakennetta, joka on varsin yksinkertainen (KUVIO 14). Kanta koostuu kolmesta taulusta: menu, view ja window. Taulu menu tallentaa käyttäjän valikkorakenteen, view-taulu sisältää avatun näkymän tiedot (mm. nimi, taustakuva) ja window-taulu ikkunan tiedot (mm. ikkunan url:n, dimensiot ja koordinaatit).



KUVIO 14. Tietokannan portal taulut

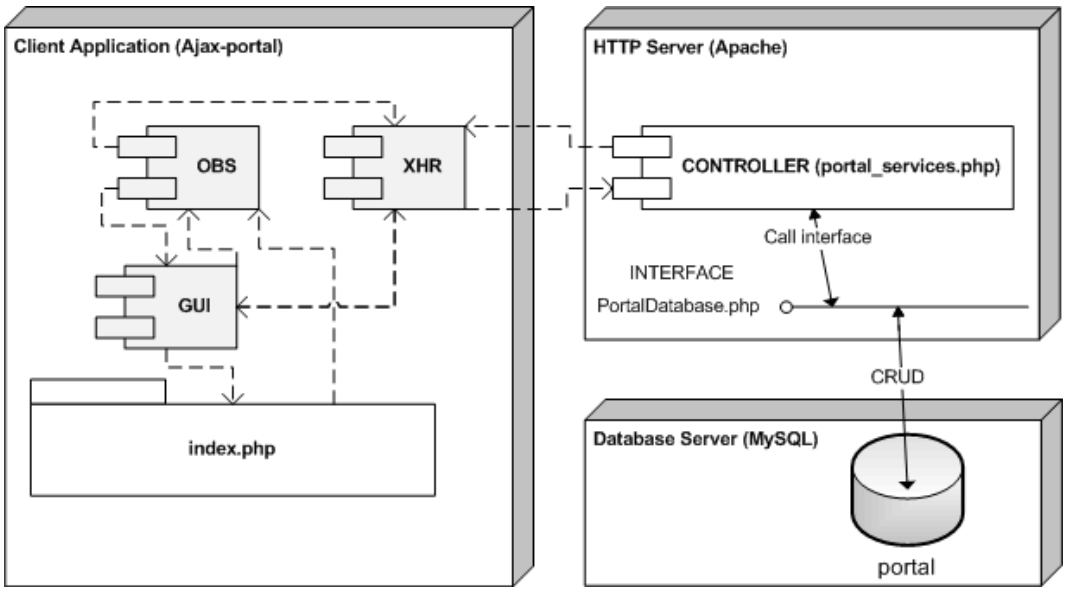
Taulut `menu` ja `window` liittyvät toisiinsa kenttien `menuID` ja `viewid` avulla. Jos `menu`-taulussa rivillä on `menuId`:n arvo `123`, niin `window` taulussa `viewid` on arvoltaan vastaavasti '`view_123`'. Taulut `view` ja `window` linkittyvät suoraan `viewid`:n kautta. Tällä hetkellä taulua `view` ei varsinaisesti käytetä, mutta sinne voidaan tarvittaessa tallentaa esimerkiksi näkymän taustakuvan nimi tai näkymän kuvaus (kts. KUVIO 15).



KUVIO 15. Taulujen rakenne

5.3.4 Asiakas ja palvelin työnjako

Kuviossa 16 on abstrahoitu sovelluksen keskeiset osat. Kuvassa olevat nuolet tarkoittavat, että moduuli kutsuu toisen moduulin funktioita.



KUVIO 16. Selaimen ja palvelimen välinen työnjako

Kuten kuviosta 16 voidaan nähdä, on moduuli XHR (portal_xhr.js) vastuussa kommunikoinnista palvelimen kanssa. Tämän skriptin kautta välittyy viesti ikkunoiden tilan muutoksista palvelinpuolen kontrollerille (portal_services.php) ja edelleen tietokantaan PortalDatabase.php rajapinnan kautta. Kun kontrolleri palauttaa dataa, niin datan jatkokäsittely suoritetaan XHR:n sisällä callback-funktiossa. Callback-funktio voi edelleen kutsua moduulia GUI ja tätä kautta päivittää käyttöliittymän tilaa.

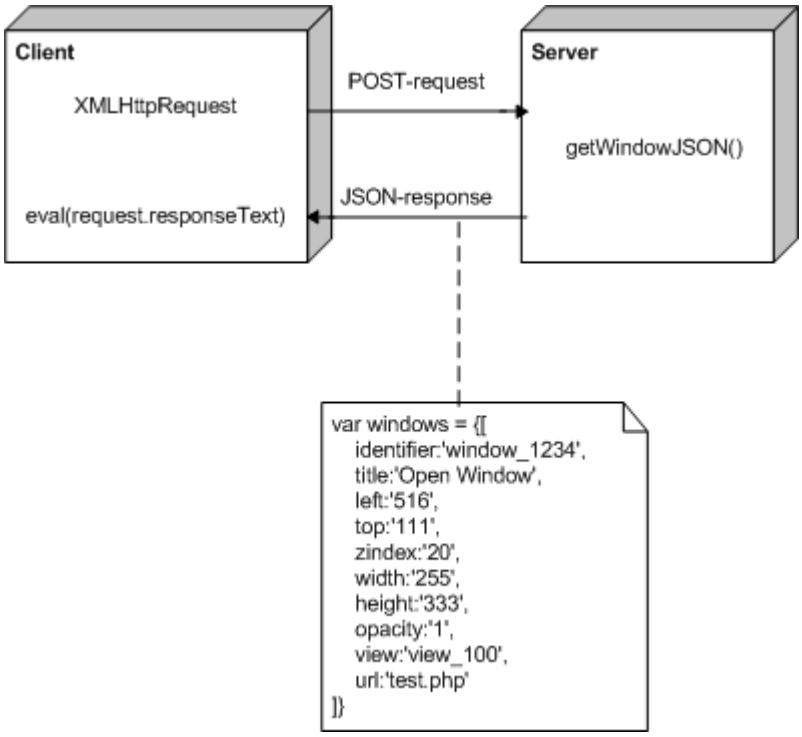
Käyttöliittymäkomponentteja hallintaan moduulin GUI (portal_gui.js) avulla. Moduulilla on funktiot käyttöliittymän hallitsemista varten. Moduuliin OBS (portal_obs.js) on kerätty tapahtumankäsittelyyn liittyvää koodia. Kaikki käyttäjävuorovaikutus tapahtuu sivulla index.php. Sivun avaaminen lataa Ajax-sovelluksen (index.php sisältää kaikki tarvittavat JavaScript-ohjelmamoduulit ja CSS-tyylitiedostot).

5.3.5 JSON tiedonsiirto

JavaScript-ohjelmointikielellä luonteva tapa kommunikoida selaimen ja palvelimen välillä on käyttää tiedonsiirrossa suoritettavaa, mobiilia JavaScript-koodia. JSON on tällainen yksinkertainen merkintäkieli, jota on helppo käyttää JavaScript-ohjelmissa. JavaScript-kielessä on `eval()` funktio, jonka avulla voidaan tulkata ja suorittaa mielivaltainen JavaScriptiä sisältävä merkkijono. Tässä sovelluksessa hyödynnetään tätä JavaScriptin ominaisuutta, kun `eval()` funktiolla suoritetaan palvelimen palauttamaa koodia.

Asiakkaalta palvelimelle tiedot lähetetään perinteisesti serialisoituina kyselymerkkijonona, joko `GET`- tai `POST`-metodilla. Ikkunan tiedot saadaan palvelimen PHP funktiolta `getWindowJSON()` suoraan JSON-muodossa.

Seuravassa on esitetty tapaus (KUVIO 17), jossa palvelimelta haetaan ikkunan luomiseen tarvittavat tiedot.



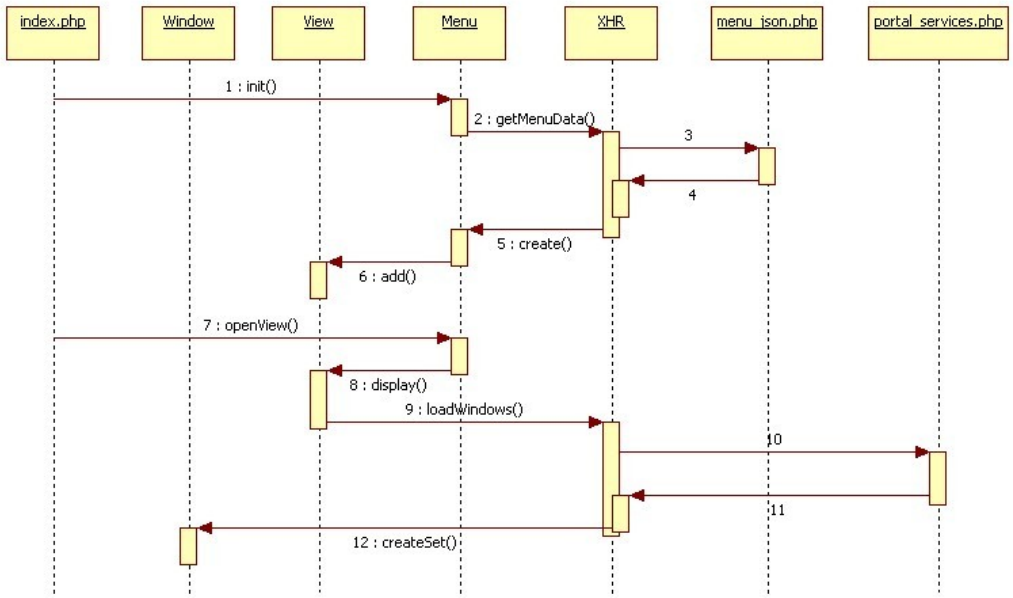
KUVIO 17. JSON-datan haku palvelimelta

Tiedot palautetaan suoraan muodossa, jota JavaScript voi ymmärtää. Jotta tiedot saataisiin käyttöön, täytyy koodi ensin suorittaa `eval()`-käskyllä. Koodin teksti saadaan luettua `XMLHttpRequest`-olion `responseText`-kentän avulla.

```
eval( request.responseText );
```

Tämän jälkeen JavaScript-olio on suoraan käytettävissä web-selaimessa.

5.3.6 Toimintasekvenssi



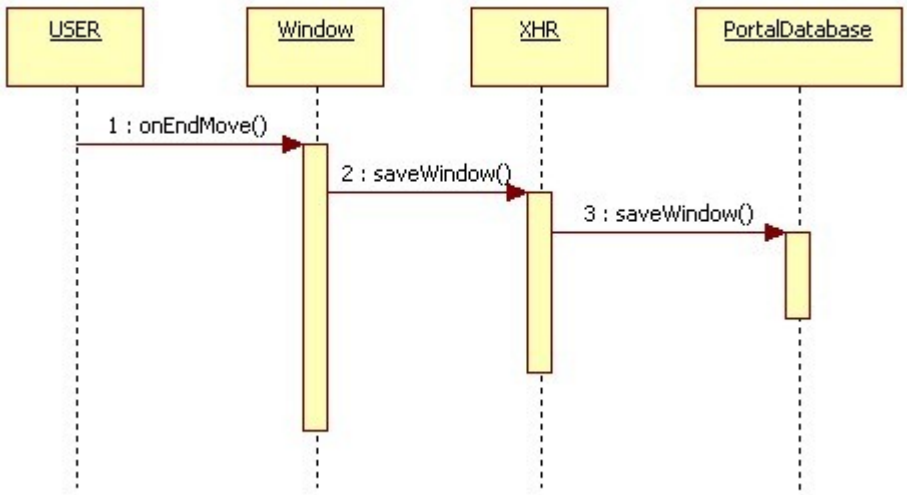
KUVIO 18. Sekvenssidiagrammi funktiokutsuista portaalia avattaessa

Yllä olevassa sekvenssissä (KUVIO 18) pyritään selventämään sitä, mitä tapahtuu kun sivu `index.php` ladataan, valikko alustetaan ja näkymä avataan. Kun sivu `index.php` ladataan (`window.onload`), niin kutsutaan automaattisesti Menun `init()`-funktiota (kuviossa 18 kohta 1), joka kutsuu olion `XHR` funktiota `getMenuData()` (2). Vaiheissa (3) ja (4) käydään Ajax-tekniikalla hakemassa tieto valikkoa varten. Valikkorakenne palautetaan (4) JSON formaatissa takaisin `XHR`-oliolle, joka edelleen kutsuu Menun `create()`-funktiota, joka varsinaisesti generoi valikon käyttäen PHP-skriptin palauttamaa tietoa. Jokaista valikon lehteä kohti luodaan uusi näkymä kutsumalla `View`-olion `add()`-funktiota (6). Uuden näkymän luonti tarkoittaa käytännössä sitä, että `index.php`:n container-osioon luodaan uusi `<div>`-elementti, johon näkymän ikkunat sijoitetaan.

Kun valikkorakenne on generoitu, on sivusto valmis käyttäjävuorovaikutukseen. Käyttäjän painaessa valikosta jotain näkymää (7), kutsutaan Menun `openView()`-funktiota, joka aktivoi halutun näkymän ja piilottaa muut näkymät kutsumalla

View:n `display()`-funktiota. Kun näkymä on aktivoitu, voidaan sinne avata ikkunoita. Tämä tapahtuu kohdassa (9), jossa `loadWindows()`-funktio käy hakemassa Ajaxilla palvelimelta näkymän ikkunoiden tiedot. Nämäkin tiedot toimitetaan JSON muodossa. Kun tiedot on haettu - kohdat (10) ja (11) - niin voidaan ikkunat luoda Window-olion `createSet()`-funktiolla. Nyt näkymän ikkunat on luotu ja portaali on käyttövalmis.

Käyttäjän siirrellessä ja skaalatessa ikkunoita portaali päivittää automaattisesti tietokannan tilaa (KUVIO 19). Aina käyttäjän liikuttaessa ikkunaa tapahtuu tietokantaan tallennus.



KUVIO 19. Ikkunan tilan automaattinen tallennus

Käyttäjän lopettaessa ikkunan siirtämisen kutsutaan ikkunan `onEndMove()`-tapahtumankäsittelijää (kuviossa 19 osa 1). Tapahtumankäsittelijässä kutsutaan `saveWindow()`-funktiota (2), joka hakee ikkunan tiedot, serialisoi ne, ja lähettää tiedot palvelimelle tietokannan päivittämistä varten (3).

5.4 Toteutusratkaisuja

5.4.1 Ikkunoiden tilan automaattinen tallennus tietokantaan

Portaalissa käytettävä ikkunakomponentti mahdollistaa käyttäjän itse asettaa ikkunan koon, sijainnin ja ulkoasun. Käyttäjän muuttaessa ikkunan tilaa, suoritetaan päivitys tietokantaan Ajaxilla ilman, että käyttäjä huomaa mitään. Ikkunoiden tiedot tallennetaan omaan tauluun, jossa on kenttiä, kuten left, top, width, height ja zindex. Nämä parametrit lähetetään PHP:lle, joka suorittaa päivityksen tietokantaan.

Taulu, johon tallennetaan kaikkien käyttäjien ikkunoiden tiedot tulee sisältämään yhden rivin jokaista käyttäjän ikkunaa kohden (KUVIO 20). Taulusta voidaan hakea ikkunan viimeisin paikka ja koko, kun käyttäjä kirjautuu järjestelmään. Käyttäjän tehdessä muutoksia tietokannan arvoja päivitetään, joten layout pysyy ajan tasalla.

```
CREATE TABLE window (
  id int(11) primary key auto_increment,
  userid varchar(255),
  viewid varchar(255),
  loadmode varchar(255),
  skin varchar(255),
  identifier varchar(255),
  title varchar(255),
  footer varchar(255),
  left smallint(6),
  top smallint(6),
  zindex smallint(6),
  width smallint(6),
  height smallint(6),
  opacity float,
  url text
);
```

KUVIO 20. Taulun window SQL-määritys

Jokaisella käyttäjällä tulee todennäköisesti olemaan taulussa useampi rivi, yksi jokaista ikkunaa kohti. Kenttä userid määrittelee sen, kenen profiiliin ikkuna kuuluu. Ikkunalla on juokseva id:n lisäksi yksilöllinen identifier-kenttä, jolla

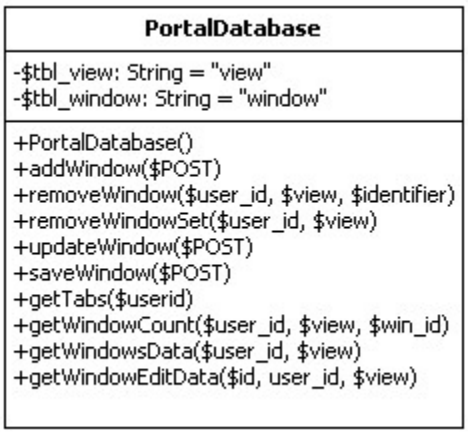
ikkunaan viitataan JavaScript-koodissa. Kahdella ikkunalla voi olla sama identifier, jos ikkunat kuuluvat eri käyttäjille. Kenttä viewid määrittelee sen, mihin näkymään ikkuna kuuluu. Samassa näkymässä voi olla useampia ikkunoita. Ikkunan sijainti ja koko on tallennettu kenttiin lefta, top, zindex, width, height. Taulun viimeinen kenttä määrittelee ikkunan sisällön URL:n, joka voi olla absoluuttinen osoite muodossa `http://www.jotain.com` tai suhteellinen osoite muotoa `../jotain/index.html`. Suhteellista osoitetta käytetään silloin, kun ikkunaan ladataan sisältöä paikalliselta palvelimelta.

Alla on informaatio käyttäjän `vallmika` kuuden eri ikkunan esittämiseksi (KUVIO 21). Ikkunat ovat jakautuneet neljälle eri näkymälle. Kuviosta paljastuu myös se, että ikkunoiden `identifier`-kentät on generoitu selaimenpuolella ohjelmallisesti.

id	userid	viewid	identifier	title
63	vallmika	view_63	window_1176467780762	Google
64	vallmika	view_88	window_1176737877497	Hotbot
65	vallmika	view_88	window_1176789940267	Google
66	vallmika	view_100	window_1177330337563	Open Window
67	vallmika	view_99	window_1177604288025	Ikkuna
68	vallmika	view_99	window_1177640731784	Google

KUVIO 21. Osa window-tauluun tallennetuista riveistä

Kun käyttäjä on kirjautunut sisään, täytyy ikkunoiden tiedot käydä hakemassa tietokannan `window`-taulusta JavaScriptille, joka sitten generoi käyttäjän näkymän. Tähän tarkoitukseen on luotu PHP-luokka `PortalDatabase`:



KUVIO 22. PortalDatabase-luokan rakenne

Varsinainainen rajapinta palvelinpuolen toiminnallisuuteen on tiedostossa `portal_services.php`, joka käyttää luokan `PortalDatabase` palveluita (KUVIO 22). Skriptille `portal_services.php` lähetetään `POST`-kyselymerkkijonona tieto siitä, mikä operaatio halutaan suorittaa ja lisäksi operaatioon tarvittavat parametrit. Asiakkaan puolella skriptin `portal_services.php` palveluita kutsutaan oliosta `XHR` käsin. Alla kontrollerina toimivan skriptin `portal_services.php` lähdekoodi:

```
include "PortalDatabase.php";
$winDB = new PortalDatabase();        // db object

$oper = $_POST['operation']; // defines what operation is to be
                             // performed by this script

switch ($oper) {
    case 'remove_window':
         $winDB->removeWindow($_POST['userid'],
                              $_POST['view'], $_POST['winid']);
         break;
    case 'save_window':
         $winDB->saveWindow($_POST);
         break;
    case 'get_json':
         getWindowJSON($_POST['userid'], $_POST['view']);
         break;
}
```

Ikkunakomponenttina on siis käytetty DHTML kirjastoa, joka voidaan ladata ilmaiseksi internetistä (Prototype Window 2008). Kaikki ikkunaa varten tarvittava

JavaScript-koodi on sijoitettu tiedostoon `window.js`. Lisäksi tarvitaan perustaksi Protype-kirjasto, josta tiedoston `window.js` toiminta riippuu. Ikkunan käyttöönottamiseksi tarvitaan seuraavat `script`-määrittelyt:

```
<script language="javascript" src="prototype.js"></script>
<script language="javascript" src="window.js"></script>
```

lisäksi mukaan tulee ottaa ikkunan tyylit

```
<link rel="stylesheet" type="text/css" href="/window/skins/default.css"/>
```

Alkuperäiseen tiedostoon `window.js` tehdään muutamia muutoksia, jotta se saadaan integroitua portaaliin. Automaattinen tallennus lisätään ikkunaan Ajax-tekniikalla. Näin vältetään se, että käyttäjän tarvitsisi painaa nappia, joka lähettää tiedot palvelimelle, aina kun hän haluaa tallentaa ikkuna-asetelmansa. Automaattinen tallentaminen tullaan lisäämään tapahtumiin, jossa ikkuna luodaan, sitä siirretään, tai sen kokoa muutetaan. Ikkunaan voidaan lisätä tapahtumankäsittelijöitä `Windows.addObserver()`-funktiolla.

```
// Windows autosave
Windows.addObserver({
    onShow: function() {
        _win_ = arguments[1];
        if( _win_.options.autoSave )
            XHR.saveWindow( _win_.getId() );
    },
    onEndMove: function() {
        _win_ = arguments[1];
        if( _win_.options.autoSave )
            XHR.saveWindow( _win_.getId() );
    },
    onEndResize: function() {
        _win_ = arguments[1];
        if( _win_.options.autoSave )
               XHR.saveWindow( _win_.getId() );
    },
    onClose: function() {
        _win_ = arguments[1];
        if ( confirm("Remove this window also from database?") )
            XHR.removeWindow( _win_.getId() );
    }
});
```

Funktio `XHR.saveWindow()` lähettää ikkunan tiedot palvelimelle päivitettäväksi tietokantaan. `Window`-luokkaan on lisätty boolean-tyyppinen kenttä

`options.autoSave`. Tämän kentän avulla voidaan määrittää, halutaanko luotu ikkuna automaattisen tallennuksen piiriin vai ei. Ikkunan tiedot saadaan haettua funktiolla `GUI.Window.getParams(win_id)`. Funktion `XHR.saveWindow()` toteutus on alla:

```
saveWindow: function( win_id ) {
    params = GUI.Window.getParams(win_id);
    view_id = GUI.View.getActive();
    var myAjax = new Ajax.Request (
         this._server, {
             method: 'post',
             parameters: 'userid=' + this.userid + '&view=' +
             view_id + '&' + params + '&operation=save_window
         }
    );
}
```

Käyttäjän kannalta huomaamaton tietojen päivittäminen parantaa portaalin käytettävyyttä, koska se poistaa yhden vaiheen (manuaalisen tallentamisen) käyttäjävuorovaikutuksesta. Näkymän automaattinen tallennus on portaalin ydintoimintoja yhdessä muokattavan valikon kanssa.

5.4.2 Valikon rakenteen tallennus tietokantaan

Valikon puurakenne tallennetaan tauluun menu (KUVIO 23). Esittämällä puun juuri- ja lapsisolmujen välinen linkitys kenttien menuID ja hostID avulla, voidaan tauluun tallentaa mielivaltainen puurakenne.

Sarake	Tyyppi
menuID	int(11)
menuNAME	tinytext
hostID	int(11)
menuLINK	text
menuDESC	text
menuORDER	int(11)
menuPUBLISH	tinytext
menuUSERID	varchar(255)

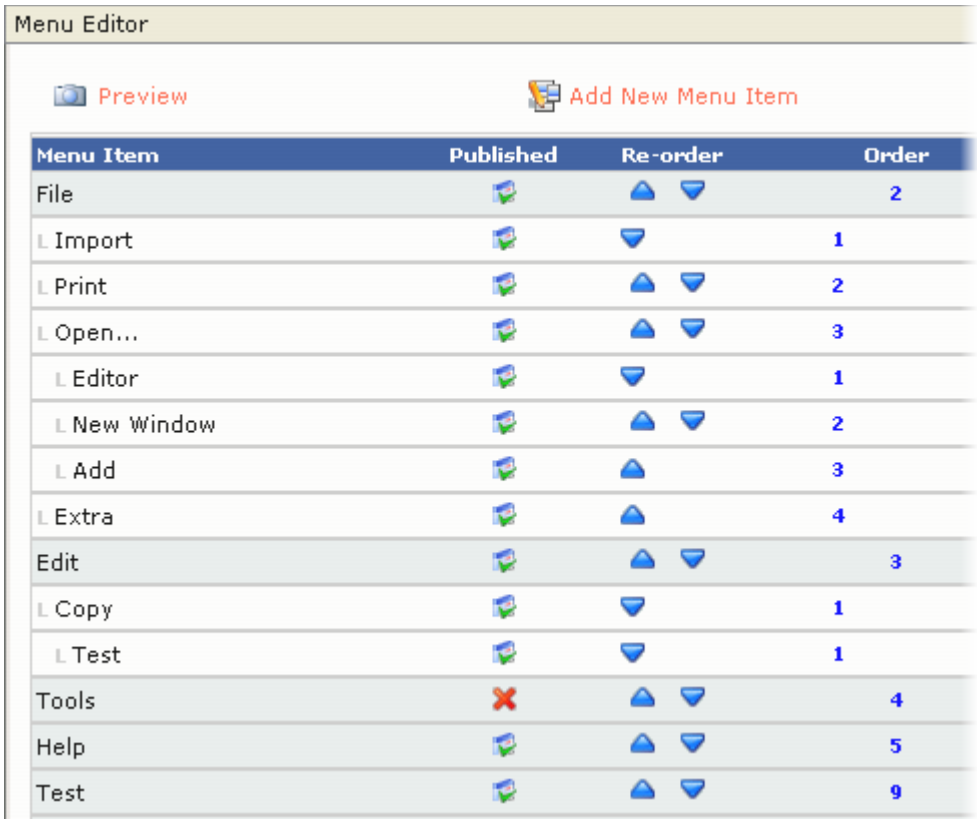
KUVIO 23. Taulun menu rakenne

Kenttien menuID ja hostID välinen linkitys näkyy selvemmin kun katsotaan taulun menu sisältöä (KUVIO 24). Taulun menu rivit voivat näyttää esimerkiksi seuraavilta.

menuID	menuNAME	hostID	menuLINK	menuDESC	menuORDER
89	Edit	0			3
85	Open...	82			3
86	Editor	85			1
87	New Window	85			2
99	Tools	0		*NULL*	4
88	Help	0			5
84	Print	82			2
83	Import	82			1
82	File	0			2
100	Test	0		*NULL*	9
101	Extra	82		*NULL*	4
102	Copy	89		*NULL*	1
103	Theme	0		*NULL*	10
104	Add	85		*NULL*	3
105	Test	102		*NULL*	1

KUVIO 24. Taulun menu rivejä

Menu Editorilla käyttäjä voi muokata haluamansa valikkorakenteen portaaliin. Kuvion 24 menu-taulun sisällön esitys Menu Editorissa on nähtävissä kuviosta 25.



KUVIO 25. Menu Editorin käyttöliittymä

Menun rakenne lähetetään selaimelle JSON formaatissa (KUVIO 26). Palvelimen skripti `menu_json.php` antaa seuraavan tulosteen:

```
var aItemData = [
        { vid:'82', text:'File', submenu:{id:'sub_82', zIndex:9999, itemdata: [
                { vid:'83', text:'Import', onclick: { fn:GUI.Menu.openView, obj:'83' } },
                { vid:'84', text:'Print', onclick: { fn:GUI.Menu.openView, obj:'84' } },
                { vid:'85', text:'Open...', submenu:{id:'sub_85', zIndex:9999, itemdata: [
                { vid:'86', text:'Editor', onclick: { fn:GUI.Menu.openView, obj:'86' } },
                { vid:'87', text:'New Window', onclick: { fn:GUI.Menu.openView, obj:'87' } },
                { vid:'104', text:'Add', onclick: { fn:GUI.Menu.openView, obj:'104' } }
]}},
                { vid:'101', text:'Extra', onclick: { fn:GUI.Menu.openView, obj:'101' } }
]}},
        { vid:'89', text:'Edit', submenu:{id:'sub_89', zIndex:9999, itemdata: [
                { vid:'102', text:'Copy', submenu:{id:'sub_102', zIndex:9999, itemdata: [
                { vid:'105', text:'Test', onclick: { fn:GUI.Menu.openView, obj:'105' } }
]}
]}},
        { vid:'88', text:'Help', onclick: { fn:GUI.Menu.openView, obj: '88' }},
        { vid:'100', text:'Test', onclick: { fn:GUI.Menu.openView, obj: '100' }},
        { vid:'103', text:'Theme', onclick: { fn:GUI.Menu.openView, obj: '103' }}
]
```

KUVIO 26. Menun rakenne palautettuna JSON formaatissa

Saadun JSON datan (KUVIO 26) pohjalta voidaan luoda dynaaminen JavaScript-valikko. Valikossa on käytetty hyödyksi Yahoo YUI:n menua (YUI Menu 2008), joka osaa luoda (JavaScriptin, HTML:n ja CSS:n avulla) JSON datan perustella dynaamisen käyttöliittymäkomponentin loppukäyttäjälle.

5.4.3 Kieliversiot

Portaalin kieliversiot on toteutettu yksinkertaisesti tiedostojen `inc.lang.js` ja `inc.lang.php` avulla. Tiedostoihin on määritelty assosiatiivinen taulukko nimeltä LANG, johon on tallennettu kaikki tarvittavat kielivakiot, joita ei portaalin osalta ole edes paljon. Tiedostoja tarvitaan kaksi sen takia, että osa tiedoista tulostetaan PHP:llä ja osa JavaScriptillä. Alla tiedoston `inc.lang.js` sisältöä:

```
LANG = [];

LANG["fi"] = [];
LANG["fi"]["new_window"] = "Uusi ikkuna";
LANG["fi"]["show_modules"] = "Näytä moduulit";
LANG["fi"]["edit_menu"] = "Muokkaa valikkoa";
LANG["fi"]["change_theme"] = "Vaihda teema";
LANG["fi"]["change_lang"] = "Vaihda kieli";
LANG["fi"]["remove_focused_window"] = "Poista aktiivinen ikkuna";
LANG["fi"]["refresh_menu"] = "Päivitä valikko";

LANG["en"] = [];
LANG["en"]["new_window"] = "New Window";
LANG["en"]["show_modules"] = "Show Modules";
LANG["en"]["edit_menu"] = "Edit Menu";
LANG["en"]["change_theme"] = "Change Theme";
LANG["en"]["change_lang"] = "Change Language";
LANG["en"]["remove_focused_window"] = "Remove Focused Window";
LANG["en"]["refresh_menu"] = "Refresh Menu";
```

PHP-tiedosto `inc.lang.php` näyttää taas vastaavasti seuraavalta:

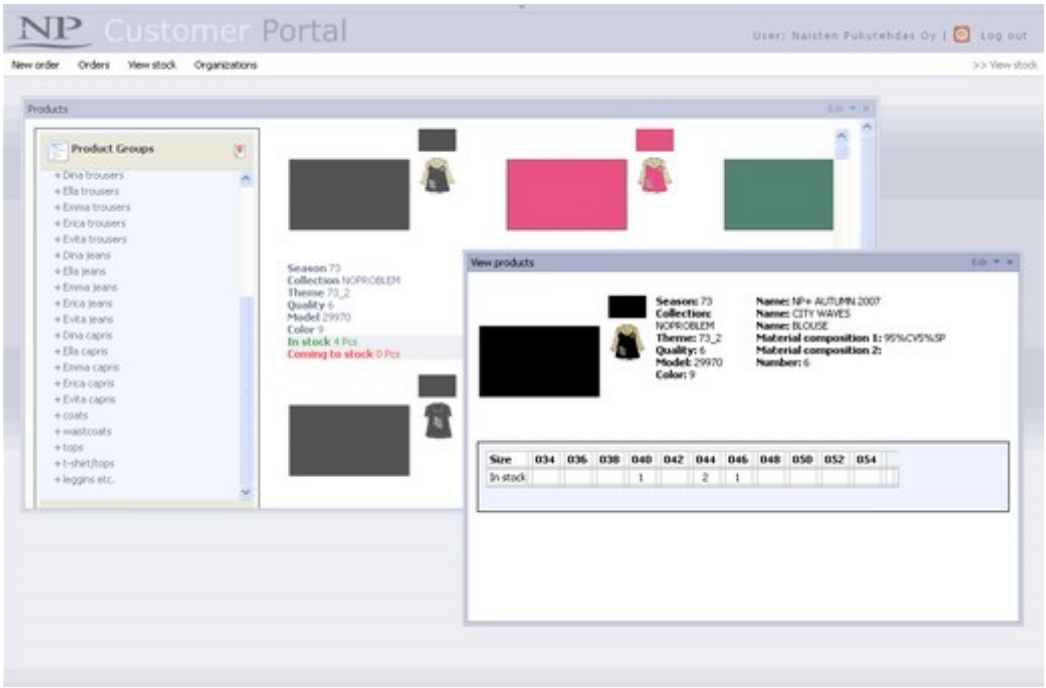
```
<?php
$_LANG = array();

$_LANG['fi'] = array();
$_LANG['fi']['logout'] = 'Kirjaudu ulos';
$_LANG['fi']['new_window'] = 'Uusi ikkuna';
$_LANG['fi']['edit_menu'] = 'Muokkaa valikkoa';
$_LANG['fi']['add_module'] = 'Lisää moduuli';
?>
```

Kielen tila on tallennettu sivulla `index.php` globaaliin muuttujaan `$LANGUAGE`, joka oletetaan välitettäväksi sivulle ennen laatamista. JavaScript-koodissa on myös määritelty globaali muuttuja `PORTAL.lang`, johon kielen tilatieto on tallennettu.

5.4.4 Esimerkki: NP Customer Portal

Esimerkkinä Ajax-portaalin käytöstä RDN:n ohjelmistotuotteissa on Naisten Pukutehdas Oy:lle (NP) toteutettu Customer Portal ratkaisu. Järjestelmän avulla asiakas voi tehdä ja hallita tekemiään tilauksia NP:n tuotteista. Lisäksi käyttäjä voi selata tuotteiden saatavuutta varastosaldojen avulla ja listata omia asiakkaitaan. Kuviossa 27 on nähtävissä tilanne, jossa tutkitaan tuotteen saatavuutta varastosta. Products-ikkunaan on listattu tietyn sesongin tuotteita, joiden varastosaldoja halutaan tutkia. Painamalla hiirellä tuotteen ikonia avautuu portaaliin View products –ikkuna, jossa on tuotteen yksityiskohtaiset tiedot, sekä varastosaldot jaoteltuna eri kokojen mukaan.



KUVIO 27. NP Customer Portal

Portaalin ulkoasun osalta NP:lle on toteutettu oma (sinertävä) teema. Ikkunoiden sisällöt on ladattu Ajax-tekniikalla palvelimelta. Käyttäjä voi sijoitella ikkunat haluamallaan tavalla näytölle ja muutokset tallentuvat automaattisesti tietokantaan. Lisäksi portaalin navigaatiopalkin jäsenet voitaisiin organisoida erilaiseen järjestykseen, uusia jäseniä voitaisiin lisätä tai tekstien sisältöä muuttaa helposti Menu Editorin avulla. Portaaliin kielenä on NP:n tapauksessa käytetty englantia.

6 TIETOTURVA JA SUORITUSKYKY

6.1 Tietoturvariskit ja kolmannen osapuolen web-palvelut

Ajax-sovellusta avattaessa palvelimelta siirtyy selaimelle huomattava määrä suoritettavaa JavaScript-koodia. Käyttäjän salliessa tämän, asettaa hän samalla luottamuksensa sovelluksen kehittäjien hyväntahtoisuuteen, koska vastaanotettu koodi voi myös suorittaa luvattomia ja käyttäjälle haitallisia komentoja. Mobiili, internetin yli lähetettävä koodi, sisältää täten aina potentiaalisia turvallisuusriskejä.

Turvallisuusriskien pienentämiseksi selaimet suorittavat vastaanotetun JavaScript-koodin ns. hiekkalaatikossa (sandbox), eli suljetussa ympäristössä, jolla on vain rajoitettu pääsy tietokoneen resursseihin, kuten sen tiedostojärjestelmään. Selainten tietoturvamallissa pyritään estämään se, että JavaScriptin avulla ladattaisiin tietoa muualta kuin samalta palvelimelta mistä suoritettava JavaScript-tiedosto oli itse haettu (Same Origin Policy). Esimerkiksi iframe elementin avulla voidaan ladata sisältöä myös ulkoisten domainien alta, mutta iframessa oleva koodi ei voi kommunikoida isä-dokumentissa määritellyn koodin kanssa. Mikäli iframen koodi ladataan samalta palvelimelta kuin isä-dokumentti, niin yhteistyö skripten välillä onnistuu. Selainten käyttämä JavaScript-tietoturvamalli estää tilanteen, jossa haitallinen sivusto lataa kaiken asiakaspuolen Ajax-koodinsa

toiselta palvelimelta ilman, että sivuston käyttäjä kuitenkaan huomaa vuorovaikuttavansa vieraan (mahdollisesti haitallisen) palvelimen kanssa.

Yksi JavaScriptin tietoturvamallin rajoituksista on se, että samalta palvelimelta, mutta eri alidomainista tulevat skriptit eivät voi vuorovaikuttaa. Valitettavasti tietoturvamalli estää sellaisia tilanteita, joissa on hyvä syy hakea tietoa eri palvelimilta. Esimerkiksi, mikäli Ajax-portaalin tapauksessa halutaan dynaamisesti generoida ikkunaan RSS-syötteestä parsittua tietoa, joka on saatavilla eri osoitteesta kuin missä portaali itse sijaitsee, niin törmätään ongelmiin selaimen tietoturvarajoitteiden kanssa.

Eräs kätevä tapa kiertää tällaisia selaimen tietoturvarajoitteita on käyttää (dynaamisesti generoituja) <script> tageja ja JSON dataformaattia. Esimerkiksi monet Yahoon web-palvelut tarjoavat mahdollisuuden datan saamiseen JSON formaatissa. Yahoo kuvahaku voidaan suorittaa seuraavalla kyselymerkkijonolla:

```
http://search.yahooapis.com/ImageSearchService/V1/imageSearch?appi
d=YahooDemo&query=Madonna&output=json&callback=getPictures
```

Jotta tuloste saataisiin JSON formaatissa tulee kyselymerkkijonossa käyttää parametria `output=json`. Käyttämällä parametria `callback=getPictures` saadaan palautettu JSON data vielä käärittynä funktion getPictures yhteyteen inline parametrina seuraavaan tapaan:

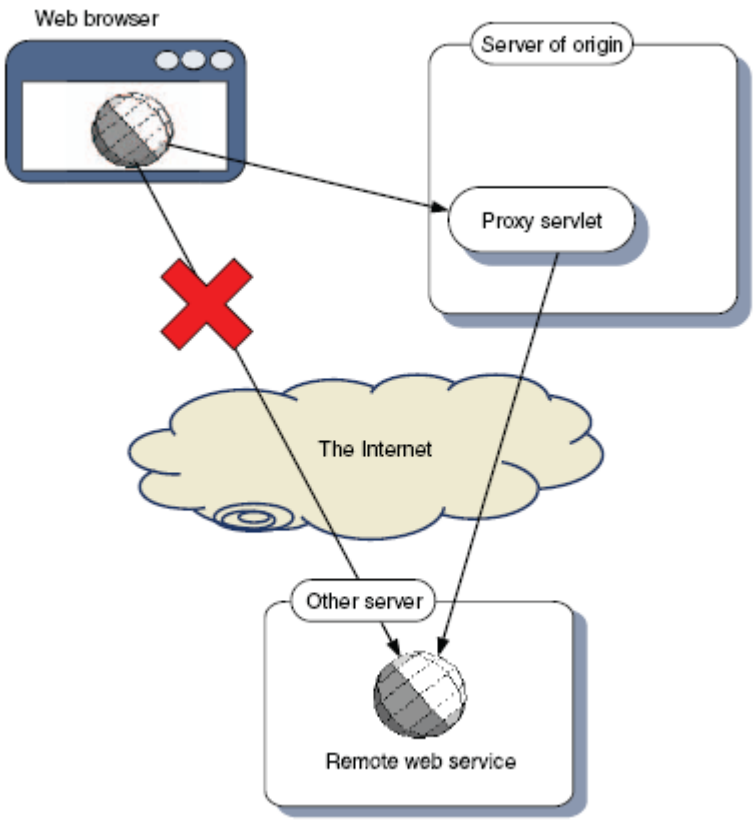
```
getPictures({"ResultSet":{"totalResultsAvailable":"767252","totalR
esultsReturned":10}})
```

Koska JSON data on natiivia JavaScriptia, ei dataa tarvise erikseen parsia. Data saadaan selaimella kätevästi käyttöön asettamalla kysely <script> tagin sisään. (Using JSON with Yahoo! Web Services, 2008.)

```
<html>
<head>
<title>How Many Pictures Of Madonna Do We Have?</title>
</head>
</body>
<script type="text/javascript">
```

```
    function getPictures(json) {
        alert(json.ResultSet.totalResultsAvailable);
    }
</script>
<script type="text/javascript"
src="http://search.yahooapis.com/ImageSearchService/V1/imageSearch
?appid=YahooDemo&query=Madonna&output=json&callback=getPictures">
</script>
<body></body>
</html>
```

Mikäli data olisi haettu käyttäen XMLHttpRequest oliota olisi turvallisuusrajoitteesta johtuen jouduttu käyttämään palvelimella proxya, joka ensin hakee datan kolmannen osapuolen (tässä Yahoon) palvelimelta isäntäpalvelimelle ja sitten uudelleen lähettää datan selaimelle (KUVIO 28).



KUVIO 28. Isäntäpalvelimen käyttäminen proxyna (Crane ym. 2006, 252).

Dynaamisten script-tagien käyttö sisältää myös tietoturvariskejä. Script-tagiin ladatulla koodilla on täysi pääsy sivun DOM-rakenteeseen ja skriptillä on samat

oikeudet kuin omalta palvelimelta saadulla skriptillä. Teoriassa skriptin avulla voidaan varastaa evästetietoja (cookies) tai hakea tietoja palvelimelta. (Crockford, 2006.)

6.2 Suorituskyvyn optimoinnista

Portaalin kehitysvaiheessa ei ole käytetty juuri hyväksi JavaScript-tiedostojen minimointia. Sovelluksen latausaikaa voidaan vähentää JavaScript-tiedostojen minimoinnilla, jossa JavaScript-tiedostoista poistetaan kommentit, ylimääräiset välilyönnit ja (algoritmista riippuen) lyhennetään myös yksityisten muuttujien nimet. Verkon yli siirrettävien tiedostojen kokoa voidaan lisäksi huomattavasti pienentää palvelimella tapahtuvalla gzip-pakkauksella. Minimoinnin piiriin voidaan myös ottaa selaimelle lähetettävät css-tyylitiedostot. Kuvien pakkaaminen ei juurikaan hyödytä, koska esimerkiksi jpeg-muotoiset kuvat ovat jo itsessään pakattuja.

Toinen varsin tehokas tapa suorituskyvyn lisäämiseksi on tiedostojen kokoaminen yhteen. HTTP-pyyntöjen lukumäärää voidaan vähentää sillä, että kootaan kaikista tarvittavista JavaScript-tiedostoista yksi iso tiedosto. Samoin kaikki tyylitiedostot voidaan koota yhteen yhdeksi isoksi CSS-tiedostoksi.

Myös selaimen välimuistin (cachen) tehokkaalla käyttöllä voidaan nopeuttaa sivujen latautumista huomattavasti, koska tietoja ei tarvitse hakea palvelimelta. Kun tiedoston Expires Header on asetettuna kauas tulevaisuuteen, tiedosto tallentuu selaimen välimuistiin tulevaa käyttöä varten. (Exceptional Performance, 2008.)

7 YHTEENVETO

Tämän työn kautta Javascript-pohjaiset perustekniikat ja menetelmät tulivat tutuiksi. Hyödyllistä oppimisen kannalta oli myös Ajax-portaalissa käytettyjen valmiiden JavaScript-kirjastojen ja -komponentteihin lähdekoodin lukeminen. JavaScript osoittautui varsin selkeäksi, joustavaksi ja ilmaisuvoimaiseksi kieleksi, jolla voidaan rakentaa olio-suuntautuneita ratkaisuja. Kaikkia teoriaosan tekniikoita ei täysimittaisesti käytetty Ajax-portaalin toteutuksessa, joten sovelluksen lähdekoodin tasoa ja toimivuutta voidaan teoriaosan tiedoilla vielä hyvin kehittää.

JavaScript-koodi voidaan mukauttaa eri ohjelmointi paradigmoihin (Prototype muistuttaa Rubya, kun taas YUI enemmän Javaa). JavaScript-maailmaan voidaan soveltaa monia tekniikoita, joita on jo aiemmin kehitetty työpöytäohjelmien ja palvelinohjelmoinnin yhteydessä. Tälläisia ovat mm. yksikkötestaus, olio-ohjelmointi, suunnittelumallit ja käyttöliittymäkirjastot. Ajax-kirjastojen kehitys on ollut varsin nopeaa ja monet toteutusratkaisut portaalin osalta olisi voinut tehdä uusien kirjastojen avulla toisin. Nyt portaaliin on yhdistelty komponentteja ja koodia eri projekteista. Nykysin on tarjolla laajoja JavaScript-pohjaisia käyttöliittymäkirjastoja - esimerkiksi Ext JS - joiden avulla saataisiin toteutettua yhtenäisempi tekninen ratkaisu.

Portalin toteutuksessa onnistuttiin saamaan RDN Oy:lle toimiva ratkaisu halutuilla perusominaisuuksilla. Portaali on tällä hetkellä tuotantokäytössä, joten projektista saatiin hyödyllisiä tuloksia myös liiketoiminnan kannalta. Jatkokehitysideoitakin on käyttäjiltä jo tullut: käyttäjäryhmä kohtaiset menuprofiilit, uusia ulkoasun teemoja ja laajempi kielituki.

Kun katsotaan internetin tulevaisuuten, on selaimista selvästi muodostumassa oma

ohjelmointialustansa. On olemassa jo useita sekä kaupallisia että avoimen lähdekoodin projekteja, jotka tekevät työpöytätyyppisten sovellusten kehittämisen mahdolliseksi web-selaimessa. On mielekiintoista nähdä, saako jokin kilpailevista JavaScript-käyttöliittymäkirjastoista hallitsevan aseman. Tälläisesta sovelluskehyksestä voisi kehittyä uusi selainpohjanen Windows. Internet ympäristönä on kuitenkin avoin, ja web-kehittäjät eivät todennäisesti ajaudu yhden ainoan tekniikan tai kirjaston käyttäjiksi.

On myös nähtävissä, että yhä useimpi työpöytäohjelma ja palvelu on siirtymässä verkkoon, tai siitä tehdään ainakin riisuttu versio internettiä varten. Esimerkiksi Adoben PhotoShopista on lupailtu ilmaista nettiversiota. Myös Google tarjoaa ilmaiseksi selainpohjaisia toimisto-ohjelmia (Google Docs). Rikkaiden internet-sovellusten tekemiseen on nousemassa muitakin, erityisesti vektoripohjaisia tekniikoita, kuten Adoben Flex, Microsoftin Silverlight ja Sunin JavaFX. Toivottavaa olisi, että W3C:n avoin vektorigrafiikkastandardi SVG ei menettäisi asemaansa suljetuille formaateille.

Toinen mielenkiintoinen kehityssuunta on työpöytäohjelmien ja web-sovellusten yhdistyminen. Adoben kehittämällä Air-teknologialla voidaan sekä Ajax- että Flash-tyyppiset web-sovellukset tuoda työpöydälle ja taata niille pääsy mm. triedostojärjestelmään ja lokaaleihin tietokantoihin. JavaScript-pohjaisten tekniikkojen käyttö tulee mitä todennäköisimmin lisääntymään sekä verkossa että työpöydällä.

LÄHTEET

About ECMAScript, 2008 [verkkojulkaisu] ECMAScript The Language of the Web [viitattu 22.2.2008]. Saatavissa: http://www.ecmascript.org/about.php

Adobe.com, FlashPlayer Penetration 2007 [verkkojulkaisu]. Saatavissa: http://www.adobe.com/products/player_census/flashplayer/

Crane, D., Pascarello, E. & James, D. 2006. Ajax in Action. Greenwich: Manning Publications.

Crockford, D. 2006. JSONRequest [verkkojulkaisu, viitattu 2.3.2008]. Saatavissa: http://www.json.org/JSONRequest.html

Crockford, D. 2001. Private Members in JavaScript [verkkojulkaisu, viitattu 9.2.2008]. Saatavissa: http://javascript.crockford.com/private.html

Darie, C. & Brinzarea, B. 2006. AJAX and PHP: Building Responsive Web Applications. Packt Publishing.

Exceptional Performance, 2008 [verkkojulkaisu]. Yahoo Developer Network [viitattu 22.1.2008]. Saatavissa: http://developer.yahoo.com/performance/

Garrett, J. 2005. Ajax: A New Approach to Web Applications [verkkojulkaisu]. Adaptive Path Essay Archives [viitattu 24.2.2208]. Saatavissa: http://adaptivepath.com/ideas/essays/archives/000385.php

Introducing JSON, 2008 [verkkojulkaisu] Saatavissa: http://www.json.org/

McLaughlin, B. 2005. Mastering Ajax, Part 1: Introduction to Ajax [verkkojulkaisu]. IBM developerWorks [viitattu: 29.5.2007]. Saatavissa: http://www-128.ibm.com/developerworks/web/library/wa-ajaxintro1.html

Peltomäki, J. & Nykänen, O. 2006. Web-selainohjelmointi. Jyväskylä. Docendo.

Prototype Window, 2008 [verkkojulkaisu] Prototype Window Class: Introduction. [viitattu 1.3.2008] Saatavissa: http://prototype-window.xilinus.com/index.html

Resig, J. 2006. Pro JavaScript Techniques. Berkeley. Apress.

Using JSON with Yahoo! Web Services, 2008 [verkkojulkaisu]. Yahoo Developer Network [viitattu 2.3.2008]. Saatavissa: http://developer.yahoo.com/common/json.html

Wikipedia, Gmail 2008. [verkkojulkaisu] Wikipedia, the free encyclopedia [viitattu: 9.2.2008]. Saatavissa: http://en.wikipedia.org/wiki/GMail

Wikipedia, JavaScript 2008 [verkkojulkaisu] Wikipedia, the free encyclopedia [viitattu 22.2.2008]. Saatavissa: http://en.wikipedia.org/wiki/JavaScript

Wikipedia, JSON 2008 [verkkojulkaisu] Wikipedia, the free encyclopedia [viitattu 1.3.2008]. Saatavissa: http://en.wikipedia.org/wiki/JSON

YUI Menu, 2008 [verkkojulkaisu]. The Yahoo! User Interface Library (YUI) [viitattu 1.3.2008]. Saatavissa: http://developer.yahoo.com/yui/